Vision Soil Analyzer

Product design of a vision based soil analyzer

Jelle Spijker

PUBLISHED BY ROYAL IHC

BOOK-WEBSITE.COM

First printing, September 2015

Contents

1. Introduction

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Design

2. Functional Design

2.1 Global Input-Proces-Output

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt

ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

2.2 Specifications

This statement requires citation [2]; this one is more specific [1, page 122].

2.2.1 Functional requirements

Name Description
Word Definition
Comment Elaboration

2.2.2 Technical requirements

Name Description
Word Definition
Comment Elaboration

3. Technical Design

3.1 Theorems

This is an example of theorems.

3.2 Several equations

This is a theorem consisting of several equations.

Theorem 3.2.1 — Name of the theorem. In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$\big|\,||\mathbf{x}|| - ||\mathbf{y}||\,\big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{3.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{3.2}$$

3.3 Single Line

This is a theorem consisting of just one line.

Theorem 3.3.1 A set $\mathscr{D}(G)$ in dense in $L^2(G), |\cdot|_0$.

4. Vision design

4.1 Definitions

This is an example of a definition. A definition could be mathematical or it could define a concept.

Definition 4.1.1 — Definition name. Given a vector space E, a norm on E is an application, denoted $||\cdot||$, E in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \Rightarrow \mathbf{x} = \mathbf{0} \tag{4.1}$$

$$||\lambda \mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \tag{4.2}$$

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \tag{4.3}$$

4.2 Notations

Notation 4.1. *Given an open subset G of $\mathbb{R}^n$, the set of functions φ are:*

1. *Bounded support G;*
2. *Infinitely differentiable;*

a vector space is denoted by $\mathscr{D}(G)$.

II Realisation

5. Technical Realisation

6. Vision realisation

6.1 Image Acquisition

This is an example of a remark.

R The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

6.2 Corollaries

This is an example of a corollary.

Corollary 6.2.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

6.3 Propositions

This is an example of propositions.

6.3.1 Several equations

Proposition 6.3.1 — Proposition name. It has the properties:

$$\big|\,||\mathbf{x}|| - ||\mathbf{y}||\,\big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{6.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{6.2}$$

6.3.2 Single Line

Proposition 6.3.2 Let $f, g \in L^2(G)$; if $\forall \varphi \in \mathscr{D}(G)$, $(f, \varphi)_0 = (g, \varphi)_0$ then $f = g$.

6.4 Examples

This is an example of examples.

6.4.1 Equation and Text

■ **Example 6.1** Let $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ and denoted by: $x^0 = (1,1)$; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \tag{6.3}$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \varepsilon\}$ for all $\varepsilon \in]0; 5/2 - \sqrt{2}[$. ■

6.4.2 Paragraph of Text

■ **Example 6.2 — Example name.** Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

■

6.5 Exercises

This is an example of an exercise.

Exercise 6.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

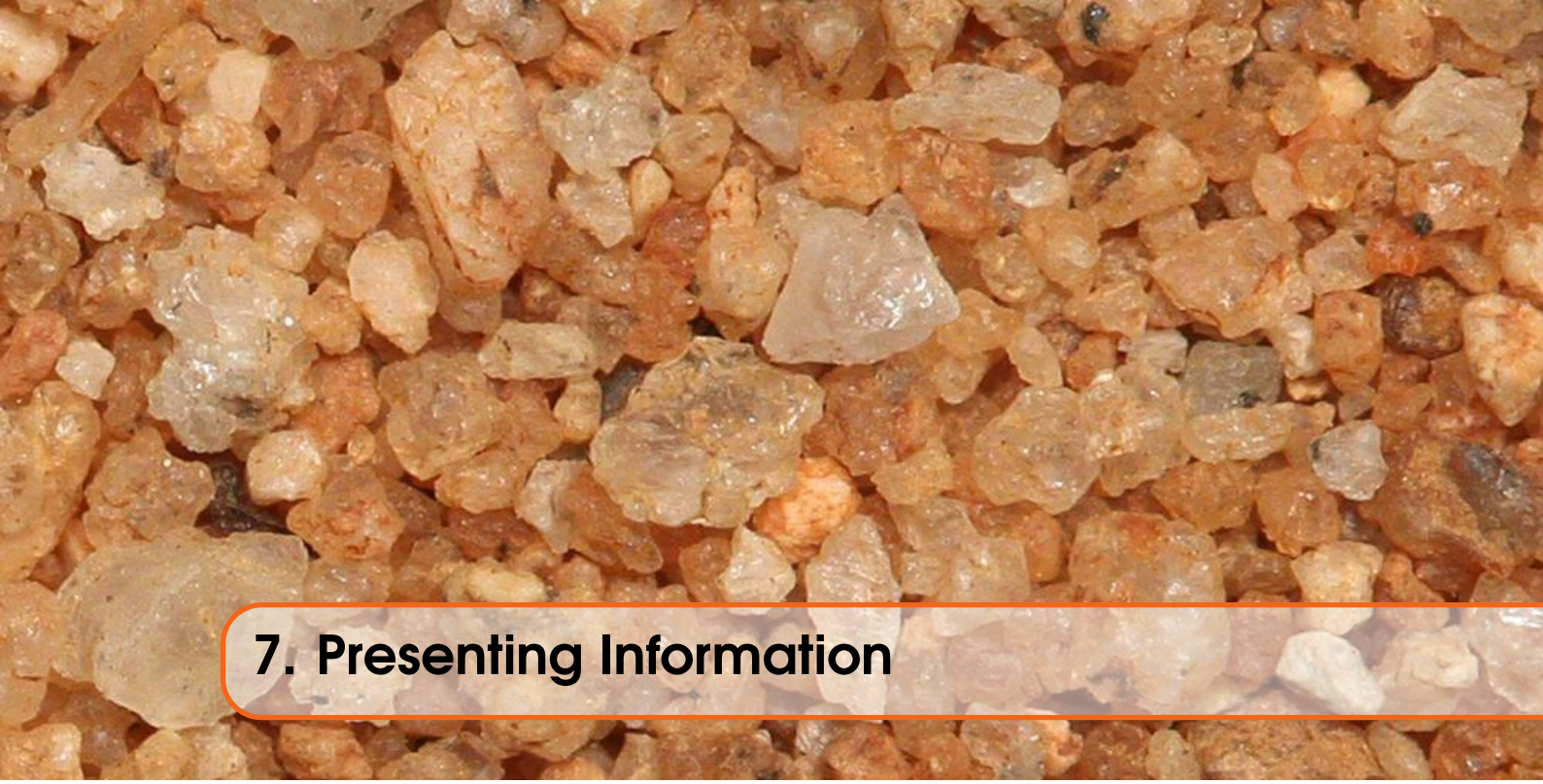
6.6 Problems

Problem 6.1 What is the average airspeed velocity of an unladen swallow?

6.7 Vocabulary

Define a word to improve a students' vocabulary.
Vocabulary 6.1 — Word. Definition of word.

III Verification



7. Presenting Information

7.1 Table

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Table 7.1: Table caption

7.2 Figure

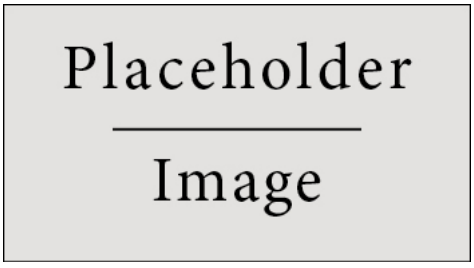
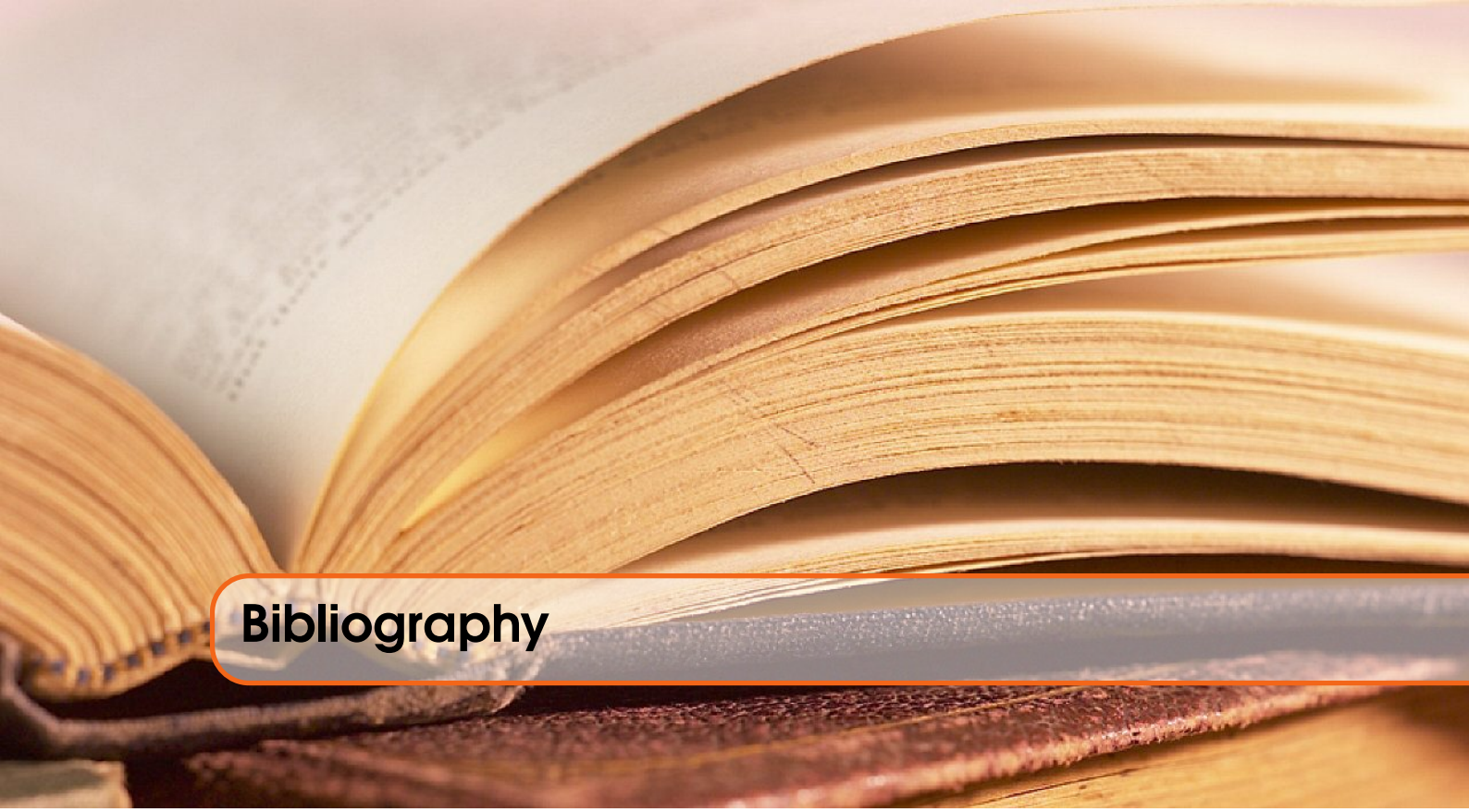


Figure 7.1: Figure caption

IV Addenda



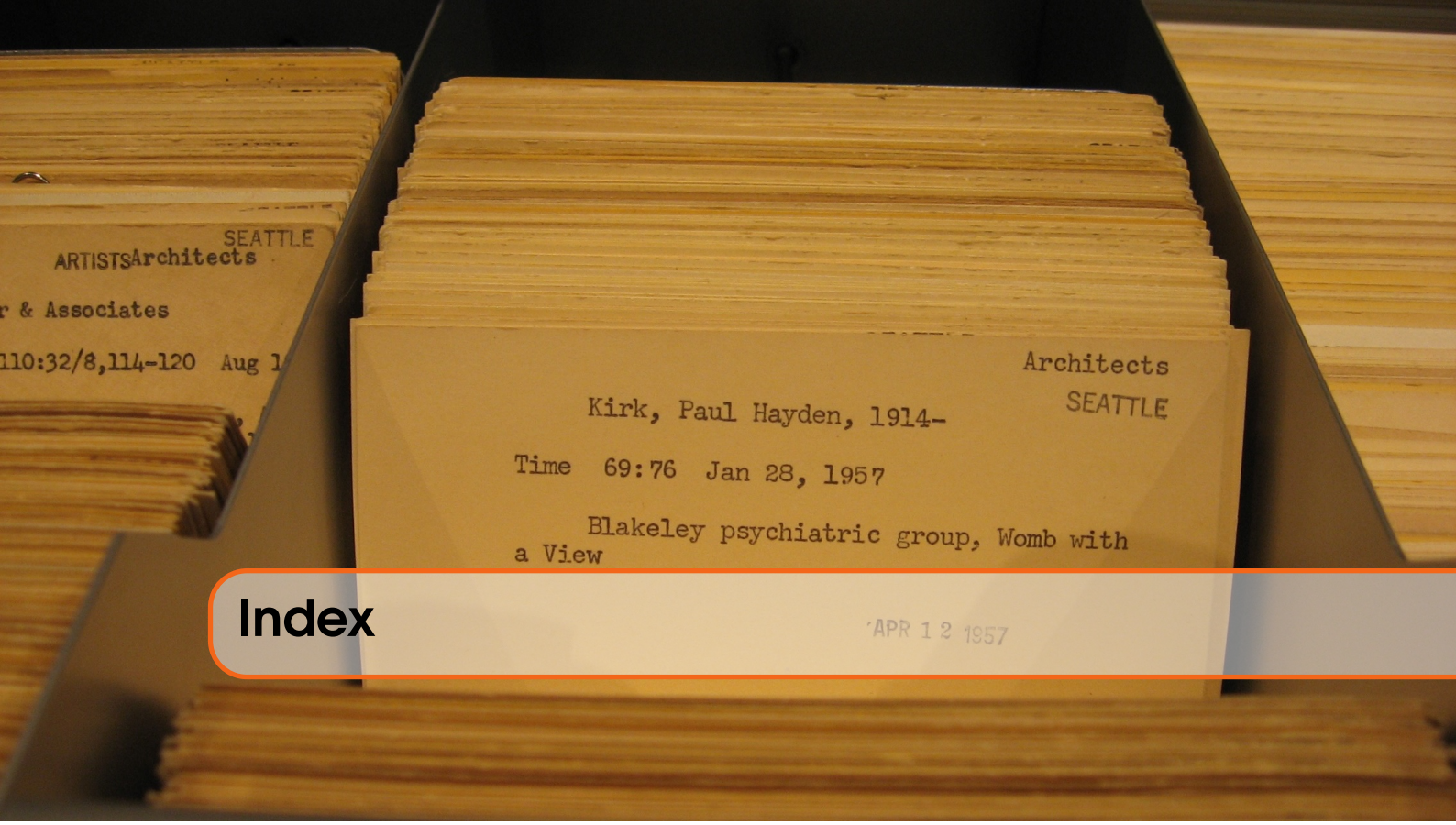
Bibliography

Books

[Smi12] John Smith. *Book title*. 1st edition. Volume 3. 2. City: Publisher, Jan. 2012, pages 123–200 (cited on page 10).

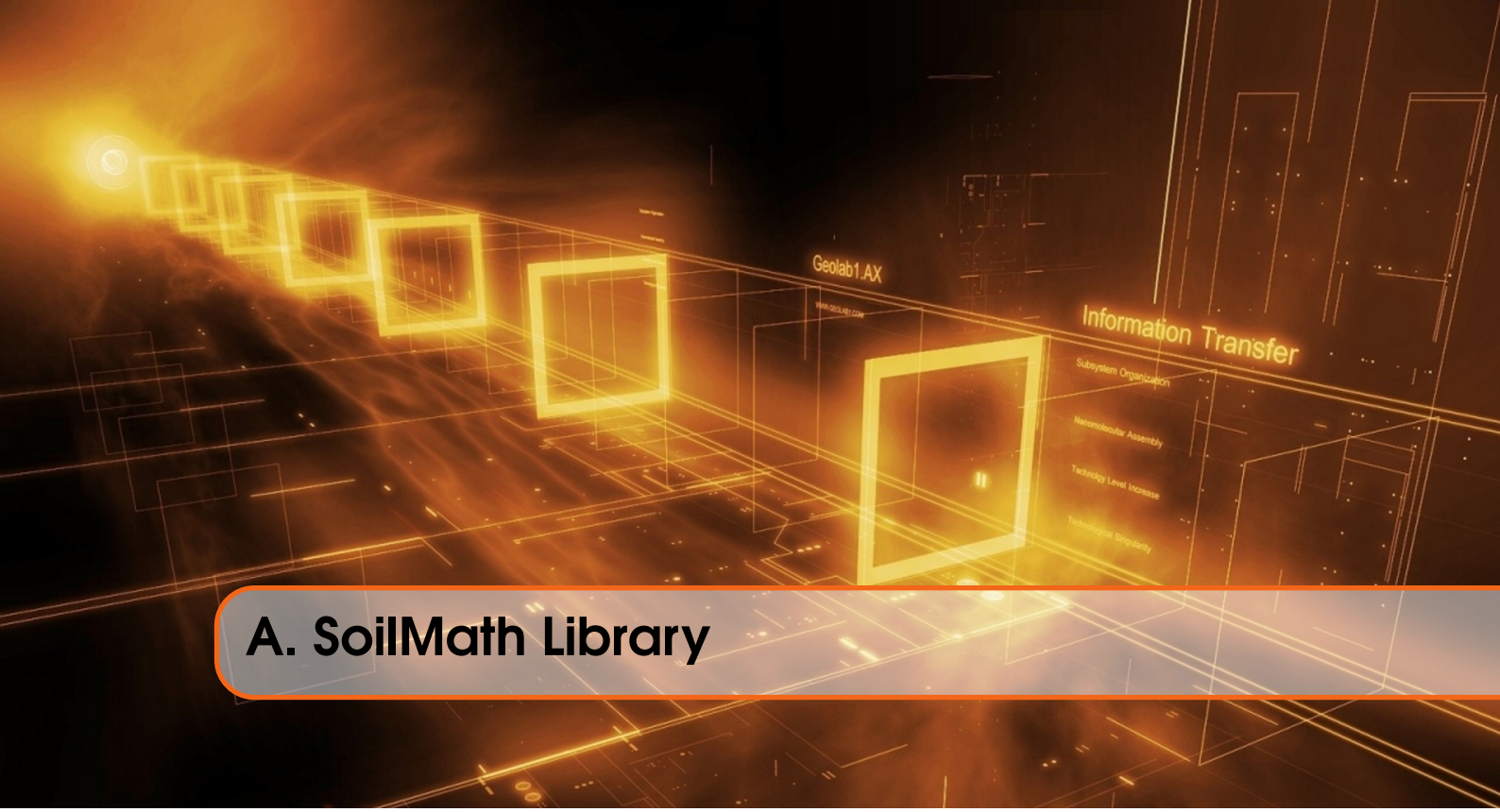
Articles

[Smi13] James Smith. “Article title”. In: 14.6 (Mar. 2013), pages 1–8 (cited on page 10).



Index

V



A. SoilMath Library

A.0.1 Genetic Algorithm Class

```
/* 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
 */

//! Genetic Algorithmes used for optimization problems
/*!
  * Use this class for optimization problems. It's currently optimized for
 * Neural Network optimzation
  */
#pragma once

#include <bitset>
#include <random>
#include <string>
#include <algorithm>
#include <chrono>
#include <math.h>
#include <list>

//#include "NN.h"
#include "SoilMathTypes.h"
#include "MathException.h"

#include <QtCore/QObject>
#include <QDebug>
#include <QThread>
#include <QtConcurrent>
```

```
#include <boost/bind.hpp>

namespace SoilMath {

class GA : public QObject {
  Q_OBJECT

public:
  float MutationRate = 0.075f; /**< mutation rate*/
  uint32_t Elitisme = 4;       /**< total number of the elite bastard*/
  float EndError = 0.001f;     /**< acceptable error between last itteration*/
  bool Revolution = true;

  /*!
 * \brief GA Standard constructor
 */
  GA();

  /*!
   * \brief GA Construction with a Neural Network initializers
   * \param nnfunction the Neural Network prediction function which results will
   * be optimized
   * \param inputneurons the number of input neurons in the Neural Network don't
   * count the bias
   * \param hiddenneurons the number of hidden neurons in the Neural Network
   * don't count the bias
   * \param outputneurons the number of output neurons in the Neural Network
   */
  GA(NNfunctionType nnfunction, uint32_t inputneurons, uint32_t hiddenneurons,
     uint32_t outputneurons);

  /*!
   * \brief GA standard de constructor
   */
  ~GA();

  /*!
   * \brief Evolve Darwin would be proud!!! This function creates a population
   * and itterates
   * through the generation till the maximum number off itterations has been
   * reached of the
   * error is acceptable
   * \param inputValues complex vector with a reference to the inputvalues
   * \param weights reference to the vector of weights which will be optimized
   * \param rangeweights reference to the range of weights, currently it doesn't
   * support indivudal ranges
   * this is because of the crossing
   * \param goal target value towards the Neural Network prediction function
   * will be optimized
   * \param maxGenerations maximum number of itterations default value is 200
   * \param popSize maximum number of population, this should be an even number
   */
  void Evolve(const InputLearnVector_t &inputValues, Weight_t &weights,
              MinMaxWeight_t rangeweights, OutputLearnVector_t &goal,
              uint32_t maxGenerations = 200, uint32_t popSize = 30);
signals:
```

```
  void learnErrorUpdate(double newError);

private:
  NNfunctionType NNfuction; /**< The Neural Net work function*/
  uint32_t inputneurons;    /**< the total number of input neurons*/
  uint32_t hiddenneurons;   /**< the total number of hidden neurons*/
  uint32_t outputneurons;   /**< the total number of output neurons*/

  MinMaxWeight_t rangeweights;
  InputLearnVector_t inputValues;
  OutputLearnVector_t goal;

  float minOptim = 0;
  float maxOptim = 0;
  uint32_t oldElit = 0;
  float oldMutation = 0.;
  std::list<double> last10Gen;
  uint32_t currentGeneration = 0;
  bool revolutionOngoing = false;

  /*!
   * \brief Genesis private function which is the spark of live, using a ra
   * seed
   * \param weights a reference to the used Weight_t vector
   * \param rangeweights pointer to the range of weights, currently it does
   * support indivudal ranges
   * \param popSize maximum number of population, this should be an even nu
   * \return
   */
  Population_t Genesis(const Weight_t &weights, uint32_t popSize);

  /*!
   * \brief CrossOver a private function where the partners mate with each
   * The values or PopMember_t are expressed as bits or ar cut at the point
   * CROSSOVER
   * the population members are paired with the nearest neighbor and new me
   * are
   * created pairing the Genome_t of each other at the CROSSOVER point.
   * Afterwards all
   * the top tiers partners are allowed to mate again.
   * \param pop reference to the population
   */
  void CrossOver(Population_t &pop);

  /*!
   * \brief Mutate a private function where individual bits from the Genome
   * are mutated
   * at a random uniform distribution event defined by the MUTATIONRATE
   * \param pop reference to the population
   */
  void Mutate(Population_t &pop);

  /*!
   * \brief GrowToAdulthood a private function where the new population mem
   * serve as the
   * the input for the Neural Network prediction function. The results are
```

```
 * weight against
 * the goal and this weight determine the fitness of the population member
 * \param pop reference to the population
 * \param inputValues a InputLearnVector_t with a reference to the inputvalues
 * \param rangeweights pointer to the range of weights, currently it doesn't
 * support indivudal ranges
 * \param goal a Predict_t type with the expected value
 * \param totalFitness a reference to the total population fitness
 */
void GrowToAdulthood(Population_t &pop, float &totalFitness);

/*!
 * \brief SurvivalOfTheFittest a private function where a battle to the death
 * commences
 * The fittest population members have the best chance of survival. Death is
 * instigated
 * with a random uniform distibution. The elite members don't partake in this
 * desctruction
 * The ELITISME rate indicate how many top tier members survive this
 * catastrophic event.
 * \param inputValues a InputLearnVector_t with a reference to the inputvalues
 * \param totalFitness a reference to the total population fitness
 * \return
 */
bool SurvivalOfTheFittest(Population_t &pop, float &totalFitness);

/*!
 * \brief PopMemberSort a private function where the members are sorted
 * according to
 * there fitness ranking
 * \param i left hand population member
 * \param j right hand population member
 * \return true if the left member is closser to the goal as the right member.
 */
static bool PopMemberSort(PopMember_t i, PopMember_t j) {
  return (i.Fitness < j.Fitness);
}

/*!
 * \brief Conversion of the value of type T to Genome_t
 * \details Usage: Use <tt>ConvertToGenome<Type>(type, range)</tt>
 * \param value The current value wich should be converted to a Genome_t
 * \param range the range in which the value should fall, this is to have a
 * Genome_t
 * which utilizes the complete range 0000...n till 1111...n
 */
template <typename T>
inline Genome_t ConvertToGenome(T value, std::pair<T, T> range) {
  uint32_t intVal = static_cast<uint32_t>(
      (UINT32_MAX * (range.first + value)) / (range.second - range.first));
  Genome_t retVal(intVal);
  return retVal;
}

/*!
 * \brief Conversion of the Genome to a value
```

```
   * \details Usage: use <tt>ConvertToValue<Type>(genome, range)
   * \param gen is the Genome which is to be converted
   * \param range is the range in which the value should fall
   */
  template <typename T>
  inline T ConvertToValue(Genome_t gen, std::pair<T, T> range) {
    T retVal =
        range.first +
        (((range.second - range.first) * static_cast<T>(gen.to_ulong())) /
         UINT32_MAX);
    return retVal;
  }
};
}
```

```
/* 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 "GA.h"

namespace SoilMath {
GA::GA() {}

GA::GA(NNfunctionType nnfunction, uint32_t inputneurons, uint32_t hiddenneurons,
       uint32_t outputneurons) {
  this->NNfuction = nnfunction;
  this->inputneurons = inputneurons;
  this->hiddenneurons = hiddenneurons;
  this->outputneurons = outputneurons;
}

GA::~GA() {}

void GA::Evolve(const InputLearnVector_t &inputValues, Weight_t &weights,
                MinMaxWeight_t rangeweights, OutputLearnVector_t &goal,
                uint32_t maxGenerations, uint32_t popSize) {
  minOptim = goal[0].OutputNeurons.size();
  minOptim = -minOptim;
  maxOptim = 2 * goal[0].OutputNeurons.size();
  oldElit = Elitisme;
  oldMutation = MutationRate;
  this->inputValues = inputValues;
  this->rangeweights = rangeweights;
  this->goal = goal;

  // Create the population
  Population_t pop = Genesis(weights, popSize);
  float totalFitness = 0.0;
  for (uint32_t i = 0; i < maxGenerations; i++) {
    CrossOver(pop);
    Mutate(pop);
    totalFitness = 0.0;
    GrowToAdulthood(pop, totalFitness);
    if (SurvivalOfTheFittest(pop, totalFitness)) {
      break;
    }
  }
  weights = pop[0].weights;
}

Population_t GA::Genesis(const Weight_t &weights, uint32_t popSize) {
  if (popSize < 1)
    return Population_t();

  Population_t pop;
  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
```

```
  std::default_random_engine gen(seed);
  std::uniform_real_distribution<float> dis(rangeweights.first,
                                            rangeweights.second);

  for (uint32_t i = 0; i < popSize; i++) {
    PopMember_t I;
    for (uint32_t j = 0; j < weights.size(); j++) {
      I.weights.push_back(dis(gen));
      I.weightsGen.push_back(
          ConvertToGenome<float>(I.weights[j], rangeweights));
    }
    pop.push_back(I);
  }
  return pop;
}

void GA::CrossOver(Population_t &pop) {
  Population_t newPop; // create a new population
  PopMember_t newPopMembers[2];
  SplitGenome_t Split[2];

  for (uint32_t i = 0; i < pop.size(); i += 2) {

    for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
      // Split A
      Split[0].first = std::bitset<CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(0, CROSSOVER));
      Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(CROSSOVER,
                                                  GENE_MAX - CROSSOVER));

      // Split B
      Split[1].first = std::bitset<CROSSOVER>(
          pop[i + 1].weightsGen[j].to_string().substr(0, CROSSOVER));
      Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i + 1].weightsGen[j].to_string().substr(CROSSOVER,
                                                      GENE_MAX - CROSSOVER)

      // Mate A and B to AB and BA
      newPopMembers[0].weightsGen.push_back(
          Genome_t(Split[0].first.to_string() + Split[1].second.to_string()
      newPopMembers[1].weightsGen.push_back(
          Genome_t(Split[1].first.to_string() + Split[0].second.to_string()
    }
    newPop.push_back(newPopMembers[0]);
    newPop.push_back(newPopMembers[1]);
    newPopMembers[0].weightsGen.clear();
    newPopMembers[1].weightsGen.clear();
  }

  // Allow the top tiers population partners to mate again
  uint32_t halfN = pop.size() / 2;
  for (uint32_t i = 0; i < halfN; i++) {
    for (uint32_t j = 0; j < pop[i].weights.size(); j++) {
      Split[0].first = std::bitset<CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(0, CROSSOVER));
```

```
      Split[0].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i].weightsGen[j].to_string().substr(CROSSOVER,
                                                  GENE_MAX - CROSSOVER));

      Split[1].first = std::bitset<CROSSOVER>(
          pop[i + 2].weightsGen[j].to_string().substr(0, CROSSOVER));
      Split[1].second = std::bitset<GENE_MAX - CROSSOVER>(
          pop[i + 2].weightsGen[j].to_string().substr(CROSSOVER,
                                                      GENE_MAX - CROSSOVER));

      newPopMembers[0].weightsGen.push_back(
          Genome_t(Split[0].first.to_string() + Split[1].second.to_string()));
      newPopMembers[1].weightsGen.push_back(
          Genome_t(Split[1].first.to_string() + Split[0].second.to_string()));
    }
    newPop.push_back(newPopMembers[0]);
    newPop.push_back(newPopMembers[1]);
    newPopMembers[0].weightsGen.clear();
    newPopMembers[1].weightsGen.clear();
  }
  pop = newPop;
}

void GA::Mutate(Population_t &pop) {
  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
  std::default_random_engine gen(seed);
  std::uniform_real_distribution<float> dis(0, 1);

  std::default_random_engine genGen(seed);
  std::uniform_int_distribution<int> disGen(0, (GENE_MAX - 1));

  QtConcurrent::blockingMap<Population_t>(pop, [&](PopMember_t &P) {
    for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
      if (dis(gen) < MutationRate) {
        P.weightsGen[j][disGen(genGen)].flip();
      }
    }
  });
}

void GA::GrowToAdulthood(Population_t &pop, float &totalFitness) {

  QtConcurrent::blockingMap<Population_t>(pop, [&](PopMember_t &P) {
    // std::for_each(pop.begin(), pop.end(), [&](PopMember_t &P) {
    for (uint32_t j = 0; j < P.weightsGen.size(); j++) {
      P.weights.push_back(ConvertToValue<float>(P.weightsGen[j], rangeweights));
    }
    Weight_t iWeight(P.weights.begin(),
                     P.weights.begin() + ((inputneurons + 1) * hiddenneurons));
    Weight_t hWeight(P.weights.begin() + ((inputneurons + 1) * hiddenneurons),
                     P.weights.end());

    for (uint32_t j = 0; j < inputValues.size(); j++) {
      Predict_t results = NNfuction(inputValues[j], iWeight, hWeight,
                                    inputneurons, hiddenneurons, outputneurons);
      // See issue #85
```

```
      bool allGood = true;
      float fitness = 0.0;
      for (uint32_t k = 0; k < results.OutputNeurons.size(); k++) {
        bool resultSign = std::signbit(results.OutputNeurons[k]);
        bool goalSign = std::signbit(goal[j].OutputNeurons[k]);
        fitness += results.OutputNeurons[k] / goal[j].OutputNeurons[k];
        if (resultSign != goalSign) {
          allGood = false;
        }
      }
      fitness += (allGood) ? results.OutputNeurons.size() : 0;
      P.Fitness += fitness;
    }
  });

  for_each(pop.begin(), pop.end(), [&](PopMember_t &P) {
    P.Fitness /= inputValues.size();
    totalFitness += P.Fitness;
  });
}

bool GA::SurvivalOfTheFittest(Population_t &pop, float &totalFitness) {
  bool retVal = false;
  uint32_t decimationCount = pop.size() / 2;

  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count
  std::default_random_engine gen(seed);

  std::sort(pop.begin(), pop.end(),
            [](const PopMember_t &L, const PopMember_t &R) {
              return L.Fitness < R.Fitness;
            });

  float maxFitness = pop[pop.size() - 1].Fitness * pop.size();
  uint32_t i = Elitisme;
  while (pop.size() > decimationCount) {
    if (i == pop.size()) {
      i = Elitisme;
    }
    std::uniform_real_distribution<float> dis(0, maxFitness);
    if (dis(gen) > pop[i].Fitness) {
      totalFitness -= pop[i].Fitness;
      pop.erase(pop.begin() + i);
    }
    i++;
  }

  std::sort(pop.begin(), pop.end(),
            [](const PopMember_t &L, const PopMember_t &R) {
              return L.Fitness > R.Fitness;
            });

  float learnError = 1 - ((pop[0].Fitness - minOptim) / (maxOptim - minOpti

  // Viva la Revolution
  if (currentGeneration > 9) {
```

```
    double avg = 0;
    for_each(last10Gen.begin(), last10Gen.end(), [&](double &G) { avg += G; });
    avg /= 10;
    double minMax[2] = {avg * 0.98, avg * 1.02};
    if (learnError > minMax[0] && learnError < minMax[1]) {
      if (!revolutionOngoing) {
        qDebug() << "Viva la revolution!";
        oldElit = Elitisme;
        Elitisme = 0;
        oldMutation = MutationRate;
        MutationRate = 0.25;
        revolutionOngoing = true;
      }
    } else if (revolutionOngoing) {
      qDebug() << "Peace has been restort";
      Elitisme = oldElit;
      MutationRate = oldMutation;
      revolutionOngoing = false;
    }
    last10Gen.pop_front();
    last10Gen.push_back(learnError);
  } else {
    last10Gen.push_back(learnError);
  }
  currentGeneration++;
  emit learnErrorUpdate(static_cast<double>(learnError));
  if (learnError < EndError) {
    retVal = true;
  }
  return retVal;
}
}
```

A.0.2 Fast Fourier Transform Class

```
/* 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 <vector>
#include <complex>
#include <cmath>
#include <valarray>
#include <array>
#include <deque>
#include <queue>
#include <iterator>
#include <algorithm>
#include <stdint.h>
#include <opencv2/core.hpp>
#include "SoilMathTypes.h"
#include "MathException.h"

namespace SoilMath {
/*!
 * \brief Fast Fourier Transform class
 * \details Use this class to transform a black and white blob presented as
 * cv::Mat with values 0 or 1 to a vector of complex values representing th
 * Descriptors.
 */
class FFT {
public:
  /*!
 * \brief Standard constructor
 */
  FFT();

  /*!
  * \brief Standard deconstructor
  */
  ~FFT();

  /*!
   * \brief Transforming the img to the frequency domain and returning the
   * Fourier Descriptors
   * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
   * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img
   * 1 \} \f$
   * \return a vector with complex values, represing the contour in the
   * frequency domain, expressed as Fourier Descriptors
   */
  ComplexVect_t GetDescriptors(const cv::Mat &img);

private:
```

```
  ComplexVect_t
      fftDescriptors; /**< Vector with complex values which represent the
                         descriptors*/
  ComplexVect_t
      complexcontour; /**< Vector with complex values which represent the
                         contour*/
  cv::Mat Img;        /**< Img which will be analysed*/

  /*!
   * \brief Contour2Complex a private function which translates a continous
   * contour image
   * to a vector of complex values. The contour is found using a depth first
   * search with
   * extension list. The alghorithm is based upon <a
   * href="http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/
   * opencourseware
   * 6-034-artificial-intelligence lecture 4</a>
   * \param img contour in the form of a cv::Mat type CV_8UC1. Which should
   * consist of a continous contour. \f$ \{ img \in \mathbb{Z} | 0 \leq img \leq
   * 1 \} \f$
   * \param centerCol centre of the contour X value
   * \param centerRow centre of the contour Y value
   * \return a vector with complex values, represing the contour as a function
   */
  ComplexVect_t Contour2Complex(const cv::Mat &img, float centerCol,
                                float centerRow);
  /*!
   * \brief Neighbors a private function returning the neighboring pixels which
   * belong to a contour
   * \param O uchar pointer to the data
   * \param pixel current counter
   * \param columns total number of columns
   * \param rows total number of rows
   * \return
   */
  iContour_t Neighbors(uchar *O, int pixel, uint32_t columns, uint32_t rows);

  /*!
   * \brief fft a private function calculating the Fast Fourier Transform
   * let \f$ m \f$ be an integer and let \f$ N=2^m \f$ also
   * \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional complex vector
   * let \f$ \omega=\exp({-2\pi i\over N}) \f$
   * then \f$ c_k={\frac{1}{N}}\sum_{j=0}^{j=N-1}CA_j\omega^{jk} \f$
   * \param CA a \f$ CA=[x_0,\ldots,x_{N-1}] \f$ is an \f$ N \f$ dimensional
   * complex vector
   */
  void fft(ComplexArray_t &CA);

  /*!
   * \brief ifft
   * \param CA
   */
  void ifft(ComplexArray_t &CA);
};
}
```

```
/* 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 "FFT.h"

namespace SoilMath {
FFT::FFT() {}

FFT::~FFT() {}

ComplexVect_t FFT::GetDescriptors(const cv::Mat &img) {
  if (!fftDescriptors.empty()) {
    return fftDescriptors;
  }

  complexcontour = Contour2Complex(img, img.cols / 2, img.rows / 2);

  // Supplement the vector of complex numbers so that N = 2^m
  uint32_t N = complexcontour.size();
  double logN = log(static_cast<double>(N)) / log(2.0);
  if (floor(logN) != logN) {
    // Get the next power of 2
    double nextLogN = floor(logN + 1.0);
    N = static_cast<uint32_t>(pow(2, nextLogN));

    uint32_t i = complexcontour.size();
    // Append the vector with zeros
    while (i++ < N) {
      complexcontour.push_back(Complex_t(0.0, 0.0));
    }
  }

  ComplexArray_t ca(complexcontour.data(), complexcontour.size());
  fft(ca);
  fftDescriptors.assign(std::begin(ca), std::end(ca));
  return fftDescriptors;
}

iContour_t FFT::Neighbors(uchar *0, int pixel, uint32_t columns,
                          uint32_t rows) {
  long int LUT_nBore[8] = {-columns + 1, -columns, -columns - 1, -1,
                           columns - 1,  columns,  1 + columns,
1};
  iContour_t neighbors;
  uint32_t pEnd = rows * columns;
  uint32_t count = 0;
  for (uint32_t i = 0; i < 8; i++) {
    count = pixel + LUT_nBore[i];
    while (count >= pEnd && i < 8) {
      count = pixel + LUT_nBore[++i];
    }
```

```
    if (i >= 8) {
      break;
    }
    if (O[count] == 1)
      neighbors.push_back(count);
  }
  return neighbors;
}

ComplexVect_t FFT::Contour2Complex(const cv::Mat &img, float centerCol,
                                   float centerRow) {
  uchar *O = img.data;
  uint32_t pEnd = img.cols * img.rows;

  std::deque<std::deque<uint32_t>> sCont;
  std::deque<uint32_t> eList;

  // Initialize the queue
  for (uint32_t i = 0; i < pEnd; i++) {
    if (O[i] == 1) {
      std::deque<uint32_t> tmpQ;
      tmpQ.push_back(i);
      sCont.push_back(tmpQ);
      break;
    }
  }

  if (sCont.front().size() < 1) {
    throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
                                   EXCEPTION_NO_CONTOUR_FOUND_NR);
  } // Exception handling

  uint32_t prev = -1;

  // Extend path on queue
  for (uint32_t i = sCont.front().front(); i < pEnd;) {
    iContour_t nBors =
        Neighbors(O, i, img.cols, img.rows); // find neighboring pixels
    std::deque<uint32_t> cQ = sCont.front(); // store first queue;
    sCont.erase(sCont.begin());              // erase first queue from beginning
    if (cQ.size() > 1) {
      prev = cQ.size() - 2;
    } else {
      prev = 0;
    }
    // Loop through each neighbor
    for (uint32_t j = 0; j < nBors.size(); j++) {
      if (nBors[j] != cQ[prev]) // No backtracking
      {
        if (nBors[j] == cQ.front() && cQ.size() > 8) {
          i = pEnd;
        } // Back at first node
        if (std::find(eList.begin(), eList.end(), nBors[j]) ==
            eList.end()) // Check if this current route is extended elsewhere
        {
          std::deque<uint32_t> nQ = cQ;
```

```
          nQ.push_back(nBors[j]); // Add the neighbor to the queue
          sCont.push_front(nQ);   // add the sequence to the front of the c
        }
      }
    }
    if (nBors.size() > 2) {
      eList.push_back(i);
    } // if there are multiple choices put current node in extension List
    if (i != pEnd) {
      i = sCont.front().back();
    } // If it isn't the end set i to the last node of the first queue
    if (sCont.size() == 0) {
      throw Exception::MathException(EXCEPTION_NO_CONTOUR_FOUND,
                                     EXCEPTION_NO_CONTOUR_FOUND_NR);
    }
  }

  // convert the first queue to a complex normalized vector
  Complex_t cPoint;
  ComplexVect_t contour;
  float col = 0.0;
  // Normalize and convert the complex function
  for_each(
      sCont.front().begin(), sCont.front().end(),
      [&img, &cPoint, &contour, &centerCol, &centerRow, &col](uint32_t &e)
        col = (float)((e % img.cols) - centerCol);
        if (col == 0.0) {
          cPoint.real(1.0);
        } else {
          cPoint.real((float)(col / centerCol));
        }
        cPoint.imag((float)((floorf(e / img.cols) - centerRow) / centerRow)
        contour.push_back(cPoint);
      });

  return contour;
}

void FFT::fft(ComplexArray_t &CA) {
  const size_t N = CA.size();
  if (N <= 1) {
    return;
  }

  //!< Divide and conquor
  ComplexArray_t even = CA[std::slice(0, N / 2, 2)];
  ComplexArray_t odd = CA[std::slice(1, N / 2, 2)];

  fft(even);
  fft(odd);

  for (size_t k = 0; k < N / 2; ++k) {
    Complex_t ct = std::polar(1.0, -2 * M_PI * k / N) * odd[k];
    CA[k] = even[k] + ct;
    CA[k + N / 2] = even[k] - ct;
  }
```

```
}

void FFT::ifft(ComplexArray_t &CA) {
  CA = CA.apply(std::conj);
  fft(CA);
  CA = CA.apply(std::conj);
  CA /= CA.size();
}
}
```

A.0.3 Neural Network Class

```
/* 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 <stdint.h>
#include <vector>
#include <string>
#include <fstream>

#include <boost/archive/xml_iarchive.hpp>
#include <boost/archive/xml_oarchive.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/version.hpp>

#include "GA.h"
#include "MathException.h"
#include "SoilMathTypes.h"
#include "FFT.h"

#include <QtCore/QObject>

namespace SoilMath {
/*!
 * \brief The Neural Network class
 * \details This class is used to make prediction on large data set. Using
 * learning algoritmes
 */
class NN : public QObject {
  Q_OBJECT

public:
  /*!
 * \brief NN constructor for the Neural Net
 * \param inputneurons number of input neurons
 * \param hiddenneurons number of hidden neurons
 * \param outputneurons number of output neurons
 */
  NN(uint32_t inputneurons, uint32_t hiddenneurons, uint32_t outputneurons)

  /*!
   * \brief NN constructor for the Neural Net
   */
  NN();

  /*!
   * \brief ~NN virtual deconstructor for the Neural Net
   */
  virtual ~NN();
```

```
/*!
 * \brief Predict The prediction function.
 * \details In this function the neural net is setup and the input which are
 * the complex values descriping the contour in the frequency domein serve as
 * input. The absolute value of these im. number because I'm not interrested
 * in the orrientation of the particle but more in the degree of variations.
 * \param input vector of complex input values, these're the Fourier
 * descriptors
 * \return a real valued vector of the output neurons
 */
Predict_t Predict(ComplexVect_t input);

/*!
 * \brief PredictLearn a static function used in learning of the weights
 * \details It starts a new Neural Network object and passes all the
 * paramaters in to this newly created object. After this the predict function
 * is called and the value is returned. This work around was needed to pass
 * the neural network to the Genetic Algorithm class.
 * \param input a complex vector of input values
 * \param inputweights the input weights
 * \param hiddenweights the hidden weights
 * \param inputneurons the input neurons
 * \param hiddenneurons the hidden neurons
 * \param outputneurons the output neurons
 * \return
 */
static Predict_t PredictLearn(ComplexVect_t input, Weight_t inputweights,
                              Weight_t hiddenweights, uint32_t inputneurons,
                              uint32_t hiddenneurons, uint32_t outputneurons);

/*!
 * \brief SetInputWeights a function to set the input weights
 * \param value the real valued vector with the values
 */
void SetInputWeights(Weight_t value) { iWeights = value; }

/*!
 * \brief SetHiddenWeights a function to set the hidden weights
 * \param value the real valued vector with the values
 */
void SetHiddenWeights(Weight_t value) { hWeights = value; }

/*!
 * \brief SetBeta a function to set the beta value
 * \param value a floating value ussualy between 0.5 and 1.5
 */
void SetBeta(float value) { beta = value; }
float GetBeta() { return beta; }

/*!
 * \brief Learn the learning function
 * \param input a vector of vectors with complex input values
 * \param cat a vector of vectors with the know output values
 * \param noOfDescriptorsUsed the total number of descriptos which should be
 * used
 */
```

```
  void Learn(InputLearnVector_t input, OutputLearnVector_t cat,
             uint32_t noOfDescriptorsUsed);

  /*!
   * \brief SaveState Serialize and save the values of the Neural Net to di
   * \details Save the Neural Net in XML valued text file to disk so that a
   * object can
   * be reconstructed on a latter stadia.
   * \param filename a string indicating the file location and name
   */
  void SaveState(std::string filename);

  /*!
   * \brief LoadState Loads the previouse saved Neural Net from disk
   * \param filename a string indicating the file location and name
   */
  void LoadState(std::string filename);

  Weight_t iWeights; /**< a vector of real valued floating point input weig
  Weight_t hWeights; /**< a vector of real valued floating point hidden wei

  uint32_t MaxGenUsedByGA = 200;
  uint32_t PopulationSizeUsedByGA = 30;
  float MutationrateUsedByGA = 0.075f;
  uint32_t ElitismeUsedByGA = 4;
  float EndErrorUsedByGA = 0.001;
  float MaxWeightUsedByGA = 50;
  float MinWeightUSedByGa = -50;

  uint32_t GetInputNeurons() { return inputNeurons; }
  void SetInputNeurons(uint32_t value);

  uint32_t GetHiddenNeurons() { return hiddenNeurons; }
  void SetHiddenNeurons(uint32_t value);

  uint32_t GetOutputNeurons() { return outputNeurons; }
  void SetOutputNeurons(uint32_t value);

  bool studied =
      false; /**< a value indicating if the weights are a results of a
                learning curve*/

signals:
  void learnErrorUpdate(double newError);

private:
  GA *optim = nullptr;
  std::vector<float> iNeurons; /**< a vector of input values, the bias is
                                  included, the bias is included and
                                    is the first value*/
  std::vector<float>
      hNeurons; /**< a vector of hidden values, the bias is included and
                   is the first value*/
  std::vector<float> oNeurons; /**< a vector of output values*/

  uint32_t hiddenNeurons = 50; /**< number of hidden neurons minus bias*/
```

```
  uint32_t inputNeurons = 20;  /**< number of input neurons minus bias*/
  uint32_t outputNeurons = 18; /**< number of output neurons*/
  float beta; /**< the beta value, this indicates the steepness of the sigmoid
                function*/

  friend class boost::serialization::access; /**< a private friend class so the
                                                  serialization can access all
                                                  the needed functions*/
  /*!
   * \brief serialization function
   * \param ar the object
   * \param version the version of the class
   */
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
      ar &BOOST_SERIALIZATION_NVP(inputNeurons);
      ar &BOOST_SERIALIZATION_NVP(hiddenNeurons);
      ar &BOOST_SERIALIZATION_NVP(outputNeurons);
      ar &BOOST_SERIALIZATION_NVP(iWeights);
      ar &BOOST_SERIALIZATION_NVP(hWeights);
      ar &BOOST_SERIALIZATION_NVP(beta);
      ar &BOOST_SERIALIZATION_NVP(studied);
      ar &BOOST_SERIALIZATION_NVP(MaxGenUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(PopulationSizeUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MutationrateUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(ElitismeUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(EndErrorUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MaxWeightUsedByGA);
      ar &BOOST_SERIALIZATION_NVP(MinWeightUSedByGa);
    }
  }
};
}
BOOST_CLASS_VERSION(SoilMath::NN, 0)
```

```
/* 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 "NN.h"
using namespace std;

namespace SoilMath {
NN::NN() { beta = 0.666; }

NN::NN(uint32_t inputneurons, uint32_t hiddenneurons, uint32_t outputneuron
  // Set the number of neurons in the network
  inputNeurons = inputneurons;
  hiddenNeurons = hiddenneurons;
  outputNeurons = outputneurons;
  // Reserve the vector space
  iNeurons.reserve(inputNeurons + 1);  // input neurons + bias
  hNeurons.reserve(hiddenNeurons + 1); // hidden neurons + bias
  oNeurons.reserve(outputNeurons);     // output neurons

  beta = 0.666;
}

NN::~NN()
{
  if (optim != nullptr) {
      delete optim;
    }
}

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

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

Predict_t NN::PredictLearn(ComplexVect_t input, Weight_t inputweights,
                           Weight_t hiddenweights, uint32_t inputneurons,
                           uint32_t hiddenneurons, uint32_t outputneurons)
  NN neural(inputneurons, hiddenneurons, outputneurons);
  neural.studied = true;
  neural.SetInputWeights(inputweights);
  neural.SetHiddenWeights(hiddenweights);
  return neural.Predict(input);
}
```

```
Predict_t NN::Predict(ComplexVect_t input) {
  if (input.size() != inputNeurons) {
    throw Exception::MathException(EXCEPTION_SIZE_OF_INPUT_NEURONS,
                                   EXCEPTION_SIZE_OF_INPUT_NEURONS_NR);
  }
  if (!studied) {
    throw Exception::MathException(EXCEPTION_NEURAL_NET_NOT_STUDIED,
                                   EXCEPTION_NEURAL_NET_NOT_STUDIED_NR);
  }

  iNeurons.clear();
  hNeurons.clear();
  oNeurons.clear();

  // Set the bias in the input and hidden vector to 1 (real number)
  iNeurons.push_back(1.0f);
  hNeurons.push_back(1.0f);

  Predict_t retVal;
  uint32_t wCount = 0;

  // Init the network
  for (uint32_t i = 0; i < inputNeurons; i++) {
    iNeurons.push_back(static_cast<float>(abs(input[i])));
  }
  for (uint32_t i = 0; i < hiddenNeurons; i++) {
    hNeurons.push_back(0.0f);
  }
  for (uint32_t i = 0; i < outputNeurons; i++) {
    oNeurons.push_back(0.0f);
  }

  for (uint32_t i = 1; i < hNeurons.size(); i++) {
    wCount = i - 1;
    for (uint32_t j = 0; j < iNeurons.size(); j++) {
      hNeurons[i] += iNeurons[j] * iWeights[wCount];
      wCount += hNeurons.size() - 1;
    }
    hNeurons[i] = 1 / (1 + pow(2.71828f, (-hNeurons[i] * beta)));
  }

  for (uint32_t i = 0; i < oNeurons.size(); i++) {
    wCount = i;
    for (uint32_t j = 0; j < hNeurons.size(); j++) {
      oNeurons[i] += hNeurons[j] * hWeights[wCount];
      wCount += oNeurons.size();
    }
    oNeurons[i] =
        (2 / (1.0f + pow(2.71828f, (-oNeurons[i] * beta)))) -
        1; // Shift plus scale so the learning function can be calculated
  }

  retVal.OutputNeurons = oNeurons;
  retVal.ManualSet = false;
  return retVal;
}
```

```
void NN::Learn(InputLearnVector_t input, OutputLearnVector_t cat,
               uint32_t noOfDescriptorsUsed __attribute__((unused))) {
  if (optim == nullptr) {
      optim = new SoilMath::GA(PredictLearn, inputNeurons, hiddenNeurons, o
    }
  connect(optim, SIGNAL(learnErrorUpdate(double)), this, SIGNAL(learnErrorU

  optim->Elitisme = ElitismeUsedByGA;
  optim->EndError = EndErrorUsedByGA;
  optim->MutationRate = MutationrateUsedByGA;

  ComplexVect_t inputTest;
  std::vector<Weight_t> weights;
  Weight_t weight(((inputNeurons + 1) * hiddenNeurons) +
                      ((hiddenNeurons + 1) * outputNeurons),
                  0);
  // loop through each case and adjust the weights
  optim->Evolve(input, weight,
               MinMaxWeight_t(MinWeightUSedByGa, MaxWeightUsedByGA), cat,
               MaxGenUsedByGA, PopulationSizeUsedByGA);

  this->iWeights = Weight_t(
      weight.begin(), weight.begin() + ((inputNeurons + 1) * hiddenNeurons)
  this->hWeights = Weight_t(
      weight.begin() + ((inputNeurons + 1) * hiddenNeurons), weight.end());
  studied = true;
}

void NN::SetInputNeurons(uint32_t value) {
  if (value != inputNeurons) {
    inputNeurons = value;
    iNeurons.clear();
    iNeurons.reserve(inputNeurons + 1);
    studied = false;
  }
}

void NN::SetHiddenNeurons(uint32_t value) {
  if (value != hiddenNeurons) {
    hiddenNeurons = value;
    hNeurons.clear();
    hNeurons.reserve(hiddenNeurons + 1);
    studied = false;
  }
}

void NN::SetOutputNeurons(uint32_t value) {
  if (value != outputNeurons) {
    outputNeurons = value;
    oNeurons.clear();
    oNeurons.reserve(outputNeurons);
    studied = false;
  }
}
}
```

A.0.4 Statistical Class

```
/* 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 MAX_UINT8_VALUE 256
#define VECTOR_CALC 1

#include <stdint.h>
#include <utility>
#include <vector>
#include <cstdlib>
#include <cmath>
#include <limits>
#include <typeinfo>
#include <string>

#include <fstream>

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/version.hpp>
#include <boost/math/distributions/students_t.hpp>

#include "MathException.h"
#include "SoilMathTypes.h"
#include "CommonOperations.h"

namespace SoilMath {

/*!
 * \brief Stats class
 * \details Usage Stats<type1, type2, type3>Stats() type 1, 2 and 3 shoudl
 * the same value and concecuative in size
 */
template <typename T1, typename T2, typename T3> class Stats {
public:
  bool isDiscrete = true; /**< indicates if the data is discrete or real*/

  T1 *Data = nullptr;       /**< Pointer the data*/
  uint32_t *bins = nullptr; /**< the histogram*/
  double *CFD = nullptr;    /**< the CFD*/
  bool Calculated = false;  /**< indication if the data has been calculated
  float Mean = 0.0;         /**< the mean value of the data*/
  uint32_t n = 0;           /**< number of data points*/
  uint32_t noBins = 0;      /**< number of bins*/
  T1 Range = 0;             /**< range of the data*/
  T1 min = 0;               /**< minimum value*/
  T1 max = 0;               /**< maximum value*/
  T1 Startbin = 0;          /**< First bin value*/
  T1 EndBin = 0;            /**< End bin value*/
```

```
  T1 binRange = 0;          /**< the range of a single bin*/
  float Std = 0.0;          /**< standard deviation*/
  T3 Sum = 0;               /**< total sum of all the data values*/
  uint16_t Rows = 0;        /**< number of rows from the data matrix*/
  uint16_t Cols = 0;        /**< number of cols from the data matrix*/
  bool StartAtZero = true;  /**< indication of the minimum value starts at zero
                               or could be less*/
  double *BinRanges = nullptr;
  double HighestPDF = 0.;

  uint32_t *begin() { return &bins[0]; }    /**< pointer to the first bin*/
  uint32_t *end() { return &bins[noBins]; } /**< pointer to the last + 1 bin*/

  /*!
   * \brief WelchTest Compare the sample using the Welch's Test
   * \details (source:
   * http://www.boost.org/doc/libs/1_57_0/libs/math/doc/html/math_toolkit/stat_tut
   * \param statComp Statiscs Results of which it should be tested against
   * \return
   */
  bool WelchTest(SoilMath::Stats<T1, T2, T3> &statComp) {
    double alpha = 0.05;
    // Degrees of freedom:
    double v = statComp.Std * statComp.Std / statComp.n +
               this->Std * this->Std / this->n;
    v *= v;
    double t1 = statComp.Std * statComp.Std / statComp.n;
    t1 *= t1;
    t1 /= (statComp.n - 1);
    double t2 = this->Std * this->Std / this->n;
    t2 *= t2;
    t2 /= (this->n - 1);
    v /= (t1 + t2);
    // t-statistic:
    double t_stat = (statComp.Mean - this->Mean) /
                    sqrt(statComp.Std * statComp.Std / statComp.n +
                         this->Std * this->Std / this->n);
    //
    // Define our distribution, and get the probability:
    //
    boost::math::students_t dist(v);
    double q = cdf(complement(dist, fabs(t_stat)));

    bool rejected = false;
    // Sample 1 Mean == Sample 2 Mean test the NULL hypothesis, the two means
    // are the same
    if (q < alpha / 2)
      rejected = false;
    else
      rejected = true;
    return rejected;
  }

  /*!
   * \brief Stats Constructor
   * \param rhs Right hand side
```

```
 */
Stats(const Stats &rhs)
    : bins{new uint32_t[rhs.noBins]{0}}, CFD{new double[rhs.noBins]{}},
      BinRanges{new double[rhs.noBins]{}} {
  this->binRange = rhs.binRange;
  this->Calculated = rhs.Calculated;
  this->Cols = rhs.Cols;
  this->EndBin = rhs.EndBin;
  this->isDiscrete = rhs.isDiscrete;
  this->max = rhs.max;
  this->Mean = rhs.Mean;
  this->min = rhs.min;
  this->n = rhs.n;
  this->noBins = rhs.noBins;
  this->n_end = rhs.n_end;
  this->Range = rhs.Range;
  this->Rows = rhs.Rows;
  this->Startbin = rhs.Startbin;
  this->Std = rhs.Std;
  this->Sum = rhs.Sum;
  std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
  std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
  std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins, this->BinRanges);
  this->Data = rhs.Data;
  this->StartAtZero = rhs.StartAtZero;
  this->HighestPDF = rhs.HighestPDF;
}

/*!
 * \brief operator = Assigmnet operator
 * \param rhs right hand side
 * \return returns the right hand side
 */
Stats &operator=(Stats const &rhs) {
  if (&rhs != this) {
    Data = rhs.Data;

    if (bins != nullptr) {
      delete[] bins;
      bins = nullptr;
    }
    if (CFD != nullptr) {
      delete[] CFD;
      CFD = nullptr;
    }
    if (BinRanges != nullptr) {
      delete[] BinRanges;
      BinRanges = nullptr;
    }

    bins = new uint32_t[rhs.noBins];     // leak
    CFD = new double[rhs.noBins];        // leak
    BinRanges = new double[rhs.noBins]; // leak
    this->binRange = rhs.binRange;
    this->Calculated = rhs.Calculated;
    this->Cols = rhs.Cols;
```

```
      this->EndBin = rhs.EndBin;
      this->isDiscrete = rhs.isDiscrete;
      this->max = rhs.max;
      this->Mean = rhs.Mean;
      this->min = rhs.min;
      this->n = rhs.n;
      this->noBins = rhs.noBins;
      this->n_end = rhs.n_end;
      this->Range = rhs.Range;
      this->Rows = rhs.Rows;
      this->Startbin = rhs.Startbin;
      this->Std = rhs.Std;
      this->Sum = rhs.Sum;
      this->Data = &rhs.Data[0];
      std::copy(rhs.bins, rhs.bins + rhs.noBins, this->bins);
      std::copy(rhs.CFD, rhs.CFD + rhs.noBins, this->CFD);
      std::copy(rhs.BinRanges, rhs.BinRanges + rhs.noBins, this->BinRanges);
      this->StartAtZero = rhs.StartAtZero;
      this->HighestPDF = rhs.HighestPDF;
    }
    return *this;
  }

  /*!
   * \brief Stats Constructor
   * \param noBins number of bins with which to build the histogram
   * \param startBin starting value of the first bin
   * \param endBin end value of the second bin
   */
  Stats(int noBins = 256, T1 startBin = 0, T1 endBin = 255) {
    min = std::numeric_limits<T1>::max();
    max = std::numeric_limits<T1>::min();
    Range = std::numeric_limits<T1>::max();
    Startbin = startBin;
    EndBin = endBin;
    this->noBins = noBins;
    bins = new uint32_t[noBins]{0};    // leak
    CFD = new double[noBins]{};        // leak
    BinRanges = new double[noBins]{}; // leak

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
      binRange = static_cast<T1>((EndBin - Startbin) / noBins);
    } else {
      isDiscrete = true;
      binRange = static_cast<T1>(round((EndBin - Startbin) / noBins));
    }
  }

  /*!
   * \brief Stats constructor
   * \param data Pointer to the data
   * \param rows Number of rows
   * \param cols Number of Columns
   * \param noBins Number of bins
```

```
 * \param startBin Value of the start bin
 * \param startatzero bool indicating if the bins should be shifted from
 */
Stats(T1 *data, uint16_t rows, uint16_t cols, int noBins = 256,
      T1 startBin = 0, bool startatzero = true) {
  min = std::numeric_limits<T1>::max();
  max = std::numeric_limits<T1>::min();
  Range = max - min;

  Startbin = startBin;
  EndBin = startBin + noBins;
  StartAtZero = startatzero;

  if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
      typeid(T1) == typeid(long double)) {
    isDiscrete = false;
  } else {
    isDiscrete = true;
  }

  Data = data;
  Rows = rows;
  Cols = cols;
  bins = new uint32_t[noBins]{0};
  CFD = new double[noBins]{};
  BinRanges = new double[noBins]{};
  this->noBins = noBins;
  if (isDiscrete) {
    BasicCalculate();
  } else {
    BasicCalculateFloat();
  }
}

/*!
 * \brief Stats Constructor
 * \param data Pointer the data
 * \param rows Number of rows
 * \param cols Number of Columns
 * \param mask the mask should have the same size as the data a value of
 * indicates that the data pointer doesn't exist. A 1 indicates that the
 * pointer is to be used
 * \param noBins Number of bins
 * \param startBin Value of the start bin
 * \param startatzero indicating if the bins should be shifted from zero
 */
Stats(T1 *data, uint16_t rows, uint16_t cols, uchar *mask, int noBins = 2
      T1 startBin = 0, bool startatzero = true) {
  min = std::numeric_limits<T1>::max();
  max = std::numeric_limits<T1>::min();
  Range = max - min;

  Startbin = startBin;
  EndBin = startBin + noBins;
  StartAtZero = startatzero;
```

```
    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
    } else {
      isDiscrete = true;
    }

    Data = data;
    Rows = rows;
    Cols = cols;
    bins = new uint32_t[noBins]{0};
    CFD = new double[noBins]{};
    BinRanges = new double[noBins]{};
    this->noBins = noBins;
    if (isDiscrete) {
      BasicCalculate(mask);
    } else {
      BasicCalculateFloat(mask);
    }
  }

  /*!
   * \brief Stats Constructor
   * \param binData The histogram data
   * \param startC start counter
   * \param endC end counter
   */
  Stats(T2 *binData, uint16_t startC, uint16_t endC) {
    noBins = endC - startC;
    Startbin = startC;
    EndBin = endC;
    uint32_t i = noBins;

    if (typeid(T1) == typeid(float) || typeid(T1) == typeid(double) ||
        typeid(T1) == typeid(long double)) {
      isDiscrete = false;
      throw Exception::MathException(EXCEPTION_TYPE_NOT_SUPPORTED,
                                     EXCEPTION_TYPE_NOT_SUPPORTED_NR);
    } else {
      isDiscrete = true;
    }

    bins = new uint32_t[noBins]{0};
    CFD = new double[noBins]{};
    BinRanges = new double[noBins]{};
    while (i-- > 0) {
      bins[i] = binData[i];
      n += binData[i];
    }
    BinCalculations(startC, endC);
  }

  ~Stats() {
    Data == nullptr;
    if (bins != nullptr) {
      delete[] bins;
```

```
    bins = nullptr;
  }
  if (CFD != nullptr) {
    delete[] CFD;
    CFD = nullptr;
  }
  if (BinRanges != nullptr) {
    delete[] BinRanges;
    BinRanges = nullptr;
  }
}

/*!
 * \brief BasicCalculateFloat execute the basic float data calculations
 */
void BasicCalculateFloat() {
  float sum_dev = 0.0;
  n = Rows * Cols;
  for (uint32_t i = 0; i < n; i++) {
    if (Data[i] > max) {
      max = Data[i];
    }
    if (Data[i] < min) {
      min = Data[i];
    }
    Sum += Data[i];
  }
  binRange = (max - min) / noBins;
  uint32_t index = 0;
  Mean = Sum / (float)n;
  Range = max - min;

  if (StartAtZero) {
    for (uint32_t i = 0; i < n; i++) {
      index = static_cast<uint32_t>(Data[i] / binRange);
      if (index == noBins) {
        index -= 1;
      }
      bins[index]++;
      sum_dev += pow((Data[i] - Mean), 2);
    }
  } else {
    for (uint32_t i = 0; i < n; i++) {
      index = static_cast<uint32_t>((Data[i] - min) / binRange);
      if (index == noBins) {
        index -= 1;
      }
      bins[index]++;
      sum_dev += pow((Data[i] - Mean), 2);
    }
  }
  Std = sqrt((float)(sum_dev / n));
  getCFD();
  Calculated = true;
}
```

```
/*!
 * \brief BasicCalculateFloat execute the basic float data calculations with a
 * mask
 * \param mask uchar mask type 0 don't calculate, 1 calculate
 */
void BasicCalculateFloat(uchar *mask) {
  float sum_dev = 0.0;
  n = Rows * Cols;
  uint32_t nmask = 0;
  for (uint32_t i = 0; i < n; i++) {
    if (mask[i] != 0) {
      if (Data[i] > max) {
        max = Data[i];
      }
      if (Data[i] < min) {
        min = Data[i];
      }
      Sum += Data[i];
      nmask++;
    }
  }
  binRange = (max - min) / noBins;
  uint32_t index = 0;
  Mean = Sum / (float)nmask;
  Range = max - min;
  if (StartAtZero) {
    for (uint32_t i = 0; i < n; i++) {
      if (mask[i] != 0) {
        index = static_cast<uint32_t>(Data[i] / binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((Data[i] - Mean), 2);
      }
    }
  } else {
    for (uint32_t i = 0; i < n; i++) {
      if (mask[i] != 0) {
        index = static_cast<uint32_t>((Data[i] - min) / binRange);
        if (index == noBins) {
          index -= 1;
        }
        bins[index]++;
        sum_dev += pow((Data[i] - Mean), 2);
      }
    }
  }
  Std = sqrt((float)(sum_dev / nmask));
  getCFD();
  Calculated = true;
}

/*!
 * \brief BasicCalculate execute the basic discrete data calculations
 */
```

```
void BasicCalculate() {
  double sum_dev = 0.0;
  n = Rows * Cols;
  for (uint32_t i = 0; i < n; i++) {
    if (Data[i] > max) {
      max = Data[i];
    }
    if (Data[i] < min) {
      min = Data[i];
    }
    Sum += Data[i];
  }
  binRange = static_cast<T1>(ceil((max - min) / static_cast<float>(noBins
  if (binRange == 0) {
    binRange = 1;
  }
  Mean = Sum / (float)n;
  Range = max - min;

  uint32_t index;
  if (StartAtZero) {
    std::for_each(Data, Data + n, [&](T1 &d) {
      index = static_cast<uint32_t>(d / binRange);
      if (index == noBins) {
        index -= 1;
      }
      bins[index]++;
      sum_dev += pow((d - Mean), 2);
    });
  } else {
    std::for_each(Data, Data + n, [&](T1 &d) {
      index = static_cast<uint32_t>((d - min) / binRange);
      if (index == noBins) {
        index -= 1;
      }
      bins[index]++;
      sum_dev += pow((d - Mean), 2);
    });
  }
  Std = sqrt((float)(sum_dev / n));
  getCFD();
  Calculated = true;
}

/*!
 * \brief BasicCalculate execute the basic discrete data calculations wit
 * mask
 * \param mask uchar mask type 0 don't calculate, 1 calculate
 */
void BasicCalculate(uchar *mask) {
  double sum_dev = 0.0;
  n = Rows * Cols;
  uint32_t nmask = 0;
  uint32_t i = 0;
  std::for_each(Data, Data + n, [&](T1 &d) {
    if (mask[i++] != 0) {
```

```
        if (d > max) {
          max = d;
        }
        if (d < min) {
          min = d;
        }
        Sum += d;
        nmask++;
      }
    });
    binRange = static_cast<T1>(ceil((max - min) / static_cast<float>(noBins)));
    Mean = Sum / (float)nmask;
    Range = max - min;

    uint32_t index;
    if (StartAtZero) {
      i = 0;
      std::for_each(Data, Data + n, [&](T1 &d) {
        if (mask[i++] != 0) {
          index = static_cast<uint32_t>(d / binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((d - Mean), 2);
        }
      });
    } else {
      i = 0;
      std::for_each(Data, Data + n, [&](T1 &d) {
        if (mask[i++] != 0) {
          index = static_cast<uint32_t>((d - min) / binRange);
          if (index == noBins) {
            index -= 1;
          }
          bins[index]++;
          sum_dev += pow((d - Mean), 2);
        }
      });
    }
    Std = sqrt((float)(sum_dev / nmask));
    getCFD();
    Calculated = true;
  }

  /*!
   * \brief BinCalculations excute the cacluations with the histogram
   * \param startC start counter
   * \param endC end counter
   */
  void BinCalculations(uint16_t startC, uint16_t endC __attribute__((unused))) {
    float sum_dev = 0.0;
    // Get the Sum
    uint32_t i = 0;
    for_each(begin(), end(), [&](uint32_t &b) { Sum += b * (startC + i++); });
```

```
  // Get Mean
  Mean = Sum / (float)n;

  // Get max
  for (int i = noBins - 1; i >= 0; i--) {
    if (bins[i] != 0) {
      max = i + startC;
      break;
    }
  }

  // Get min
  for (uint32_t i = 0; i < noBins; i++) {
    if (bins[i] != 0) {
      min = i + startC;
      break;
    }
  }

  // Get Range;
  Range = max - min;

  // Calculate Standard Deviation
  i = 0;
  for_each(begin(), end(), [&](uint32_t &b) {
    sum_dev += b * pow(((i++ + startC) - Mean), 2);
  });
  Std = sqrt((float)(sum_dev / n));
  getCFD();
  Calculated = true;
}

uint32_t HighestFrequency() {
  uint32_t freq = 0;
  std::for_each(begin(), end(), [&](uint32_t &B) {
    if (B > freq) {
      freq = B;
    }
  });
  return freq;
}

void GetPDFfunction(std::vector<double> &xAxis, std::vector<double> &yAxi
                    double Step, double start = 0, double stop = 7) {
  uint32_t resolution;
  resolution = static_cast<uint32_t>(((stop - start) / Step) + 0.5);

  xAxis.push_back(start);
  double yVal0 = (1 / (Std * 2.506628274631)) *
                 exp(-(pow((start - Mean), 2) / (2 * pow(Std, 2))));
  yAxis.push_back(yVal0);
  HighestPDF = yVal0;
  for (uint32_t i = 1; i < resolution; i++) {
    double xVal = xAxis[xAxis.size() - 1] + Step;
    xAxis.push_back(xVal);
    double yVal = (1 / (Std * 2.506628274631)) *
```

```
                    exp(-(pow((xVal - Mean), 2) / (2 * pow(Std, 2))));
      yAxis.push_back(yVal);
      if (yVal > HighestPDF) {
        HighestPDF = yVal;
      }
    }
  }

protected:
  uint32_t n_end = 0; /**< data end counter used with mask*/

  /*!
   * \brief getCFD get the CFD matrix;
   */
  void getCFD() {
    uint32_t *sumBin = new uint32_t[noBins];
    sumBin[0] = bins[0];
    CFD[0] = (static_cast<double>(sumBin[0]) / static_cast<double>(n)) * 100.;
    for (uint32_t i = 1; i < noBins; i++) {
      sumBin[i] = (sumBin[i - 1] + bins[i]);
      CFD[i] = (static_cast<double>(sumBin[i]) / static_cast<double>(n)) * 100.;
      if (CFD[i] > HighestPDF) {
        HighestPDF = CFD[i];
      }
    }
    delete[] sumBin;
  }

  friend class boost::serialization::access; /**< Serialization class*/

  /*!
   * \brief serialize the object
   * \param ar argument
   * \param version
   */
  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
      ar &isDiscrete;
      ar &n;
      ar &noBins;
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &bins[dc];
      }
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &CFD[dc];
      }
      for (size_t dc = 0; dc < noBins; dc++) {
        ar &BinRanges[dc];
      }
      ar &Calculated;
      ar &Mean;
      ar &Range;
      ar &min;
      ar &max;
      ar &Startbin;
```

```
      ar &EndBin;
      ar &binRange;
      ar &Std;
      ar &Sum;
      ar &Rows;
      ar &Cols;
      ar &StartAtZero;
      ar &HighestPDF;
    }
  }
};
}

typedef SoilMath::Stats<float, double, long double>
    floatStat_t; /**< floating Stat type*/
typedef SoilMath::Stats<uchar, uint32_t, uint64_t>
    ucharStat_t; /**< uchar Stat type*/
typedef SoilMath::Stats<uint16_t, uint32_t, uint64_t>
    uint16Stat_t; /**< uint16 Stat type*/
typedef SoilMath::Stats<uint32_t, uint32_t, uint64_t>
    uint32Stat_t; /**< uint32 Stat type*/
BOOST_CLASS_VERSION(floatStat_t, 0)
BOOST_CLASS_VERSION(ucharStat_t, 0)
BOOST_CLASS_VERSION(uint16Stat_t, 0)
BOOST_CLASS_VERSION(uint32Stat_t, 0)
```

```
/* 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 "Stats.h"
#include <boost/serialization/base_object.hpp>

namespace SoilMath {
class PSD : public SoilMath::Stats<double, double, long double> {
private:
  uint32_t DetBin(float value) {
    uint32_t i = noBins - 1;
    while (i > 0) {
      if (value > BinRanges[i]) {
        return i;
      }
      i--;
    }
    return 0;
  }

  void BasicCalculatePSD() {
    float sum_dev = 0.0;
    n = Rows * Cols;
    for (uint32_t i = 0; i < n; i++) {
      if (Data[i] > max) {
        max = Data[i];
      }
      if (Data[i] < min) {
        min = Data[i];
      }
      Sum += Data[i];
    }
    uint32_t index = 0;
    Mean = Sum / (float)n;
    Range = max - min;
    for (uint32_t i = 0; i < n; i++) {
      index = DetBin(Data[i]);
      bins[index]++;
      sum_dev += pow((Data[i] - Mean), 2);
    }
    Std = sqrt((float)(sum_dev / n));
    getCFD();
    Calculated = true;
  }
  friend class boost::serialization::access;

  template <class Archive>
  void serialize(Archive &ar, const unsigned int version) {
    if (version == 0) {
```

```
      ar &boost::serialization::base_object<
          SoilMath::Stats<double, double, long double>>(*this);
    }
  }

public:
  PSD() : SoilMath::Stats<double, double, long double>() {}

  PSD(double *data, uint32_t nodata, double *binranges, uint32_t nobins,
      uint32_t endbin)
      : SoilMath::Stats<double, double, long double>(nobins, 0, endbin) {
    std::copy(binranges, binranges + nobins, BinRanges);
    Data = data;
    Rows = nodata;
    Cols = 1;

    BasicCalculatePSD();
  }
};
}
BOOST_CLASS_VERSION(SoilMath::PSD, 0)
```

A.0.5 General project file

```
/* 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
 */

/*! \brief Collection of the public SoilMath headers
 * Commonpractice is to include this header when you want to add Soilmath
 * routines
 */
#pragma once

#include "Stats.h"
#include "Sort.h"
#include "FFT.h"
#include "NN.h"
#include "GA.h"
#include "CommonOperations.h"
#include "SoilMathTypes.h"
#include "psd.h"
#include "Mat_archive.h"
#include "predict_t_archive.h"
```

```
/* 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 COMMONOPERATIONS_VERSION 1

#include <algorithm>
#include <stdint.h>
#include <math.h>
#include <vector>

namespace SoilMath {
inline uint16_t MinNotZero(uint16_t a, uint16_t b) {
  if (a != 0 && b != 0) {
    return (a < b) ? a : b;
  } else {
    return (a > b) ? a : b;
  }
}

inline uint16_t Max(uint16_t a, uint16_t b) { return (a > b) ? a : b; }

inline uint16_t Max(uint16_t a, uint16_t b, uint16_t c, uint16_t d) {
  return (Max(a, b) > Max(c, d)) ? Max(a, b) : Max(c, d);
}

inline uint16_t Min(uint16_t a, uint16_t b) { return (a < b) ? a : b; }

inline uint16_t Min(uint16_t a, uint16_t b, uint16_t c, uint16_t d) {
  return (Min(a, b) > Min(c, d)) ? Min(a, b) : Min(c, d);
}

static inline double quick_pow10(int n) {
  static double pow10[19] = {1, 10, 100, 1000, 10000, 100000, 1000000, 1000
                             100000000, 1000000000, 10000000000, 1000000000
                             1000000000000, 10000000000000, 100000000000000
                             1000000000000000, 10000000000000000,
                             100000000000000000, 1000000000000000000};
  return pow10[(n >= 0) ? n : -n];
}


// Source:
// http://martin.ankerl.com/2012/01/25/optimized-approximative-pow-in-c-and
static inline double fastPow(double a, double b) {
  union {
    double d;
    int x[2];
  } u = {a};
  u.x[1] = (int)(b * (u.x[1] - 1072632447) + 1072632447);
  u.x[0] = 0;
```

```
  return u.d;
}

static inline double quick_pow2(int n) {
  static double pow2[256] = {
      0,     1,     4,     9,     16,    25,    36,    49,
64,    81,
      100,   121,   144,   169,   196,   225,   256,   289,
324,   361,
      400,   441,   484,   529,   576,   625,   676,   729,
784,   841,
      900,   961,   1024,  1089,  1156,  1225,  1296,  1369,
1444,  1521,
      1600,  1681,  1764,  1849,  1936,  2025,  2116,  2209,
2304,  2401,
      2500,  2601,  2704,  2809,  2916,  3025,  3136,  3249,
3364,  3481,
      3600,  3721,  3844,  3969,  4096,  4225,  4356,  4489,
4624,  4761,
      4900,  5041,  5184,  5329,  5476,  5625,  5776,  5929,
6084,  6241,
      6400,  6561,  6724,  6889,  7056,  7225,  7396,  7569,
7744,  7921,
      8100,  8281,  8464,  8649,  8836,  9025,  9216,  9409,
9604,  9801,
      10000, 10201, 10404, 10609, 10816, 11025, 11236, 11449, 11664, 11881,
      12100, 12321, 12544, 12769, 12996, 13225, 13456, 13689, 13924, 14161,
      14400, 14641, 14884, 15129, 15376, 15625, 15876, 16129, 16384, 16641,
      16900, 17161, 17424, 17689, 17956, 18225, 18496, 18769, 19044, 19321,
      19600, 19881, 20164, 20449, 20736, 21025, 21316, 21609, 21904, 22201,
      22500, 22801, 23104, 23409, 23716, 24025, 24336, 24649, 24964, 25281,
      25600, 25921, 26244, 26569, 26896, 27225, 27556, 27889, 28224, 28561,
      28900, 29241, 29584, 29929, 30276, 30625, 30976, 31329, 31684, 32041,
      32400, 32761, 33124, 33489, 33856, 34225, 34596, 34969, 35344, 35721,
      36100, 36481, 36864, 37249, 37636, 38025, 38416, 38809, 39204, 39601,
      40000, 40401, 40804, 41209, 41616, 42025, 42436, 42849, 43264, 43681,
      44100, 44521, 44944, 45369, 45796, 46225, 46656, 47089, 47524, 47961,
      48400, 48841, 49284, 49729, 50176, 50625, 51076, 51529, 51984, 52441,
      52900, 53361, 53824, 54289, 54756, 55225, 55696, 56169, 56644, 57121,
      57600, 58081, 58564, 59049, 59536, 60025, 60516, 61009, 61504, 62001,
      62500, 63001, 63504, 64009, 64516, 65025};
  return pow2[(n >= 0) ? n : -n];
}

static inline long float2intRound(double d) {
  d += 6755399441055744.0;
  return reinterpret_cast<int &>(d);
}

/*!
 * \brief calcVolume according to ISO 9276-6
 * \param A
 * \return
 */
static inline float calcVolume(float A) {
  return (pow(A, 1.5)) / 10.6347f;
```

```
}

static inline std::vector<float> makeOutput(uint8_t value, uint32_t noNeuro
  std::vector<float> retVal(noNeurons, -1);
  retVal[value - 1] = 1;
  return retVal;
}

/*!
 * \brief calcDiameter according to ISO 9276-6
 * \param A
 * \return
 */
static inline float calcDiameter(float A) {
  //return sqrt((4 * A) / M_PI);
  return 1.1283791670955 * sqrt(A);
}
}
```

```
/* 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 GENE_MAX 32  /**< maximum number of genes*/
#define CROSSOVER 16 /**< crossover location*/

#include <stdint.h>
#include <bitset>
#include <vector>
#include <complex>
#include <valarray>
#include <array>

typedef unsigned char uchar;   /**< unsigned char*/
typedef unsigned short ushort; /**< unsigned short*/
typedef unsigned int uint32_t;

typedef std::complex<double> Complex_t;       /**< complex vector of doubles*/
typedef std::vector<Complex_t> ComplexVect_t; /**< vector of Complex_t*/
typedef std::valarray<Complex_t> ComplexArray_t; /**< valarray of Complex_t*/
typedef std::vector<uint32_t> iContour_t;        /**< vector of uint32_t*/
typedef std::bitset<GENE_MAX> Genome_t; /**< Bitset repressenting a genome*/
typedef std::pair<std::bitset<CROSSOVER>, std::bitset<GENE_MAX - CROSSOVER>>
    SplitGenome_t; /**< a matted genome*/

typedef std::vector<float> Weight_t;     /**< a float vector*/
typedef std::vector<Genome_t> GenVect_t; /**< a vector of genomes*/
typedef struct PopMemberStruct {
  Weight_t weights;       /**< the weights the core of a population member*/
  GenVect_t weightsGen;   /**< the weights as genomes*/
  float Calculated = 0.0; /**< the calculated value*/
  float Fitness = 0.0;    /**< the fitness of the population member*/
} PopMember_t;            /**< a population member*/
typedef std::vector<PopMember_t> Population_t; /**< Vector with PopMember_t*/
typedef std::pair<float, float>
    MinMaxWeight_t; /**< floating pair weight range*/

typedef struct Predict_struct {
  uint8_t Category = 1; /**< the category number */
  float RealValue = 1.;  /**< category number as float in order to estimate how
                        precise to outcome is*/
  float Accuracy = 1.;   /**< the accuracy of the category*/
  std::vector<float> OutputNeurons; /**< the output Neurons*/
  bool ManualSet = true;
} Predict_t;                        /**< The prediction results*/
typedef Predict_t (*NNfunctionType)(
    ComplexVect_t, Weight_t, Weight_t, uint32_t, uint32_t,
    uint32_t); /**< The prediction function from the Neural Net*/

typedef std::vector<ComplexVect_t>
```

```
    InputLearnVector_t; /**< Vector of a vector with complex values*/
typedef std::vector<Predict_t> OutputLearnVector_t; /**< vector with result
```

```
/* 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-bo
#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 __attrib
  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];
  }
}
}
}
```

```
/* 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-
#pragma once

#include <boost/archive/binary_iarchive.hpp>
#include <boost/archive/binary_oarchive.hpp>
#include <boost/serialization/access.hpp>
#include <boost/serialization/vector.hpp>
#include <boost/serialization/complex.hpp>
#include "SoilMathTypes.h"

namespace boost {
namespace serialization {
/*!
 * \brief serialize Serialize the openCV mat to disk
 */
template <class Archive>
inline void serialize(Archive &ar, Predict_t &P, const unsigned int version
  ar &P.Accuracy;
  ar &P.Category;
  ar &P.OutputNeurons;
  ar &P.RealValue;
}
}
}
```

```
/* 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
 */

#define EXCEPTION_MATH "Math Exception!"
#define EXCEPTION_MATH_NR 0
#define EXCEPTION_NO_CONTOUR_FOUND
\
  "No continuous contour found, or less then 8 pixels long!"
#define EXCEPTION_NO_CONTOUR_FOUND_NR 1
#define EXCEPTION_SIZE_OF_INPUT_NEURONS
\
  "Size of input unequal to input neurons exception!"
#define EXCEPTION_SIZE_OF_INPUT_NEURONS_NR 2
#define EXCEPTION_NEURAL_NET_NOT_STUDIED "Neural net didn't study exception!"
#define EXCEPTION_NEURAL_NET_NOT_STUDIED_NR 3
#define EXCEPTION_TYPE_NOT_SUPPORTED
\
  "Type not supported for operation exception!"
#define EXCEPTION_TYPE_NOT_SUPPORTED_NR 4

#pragma once
#include <exception>
#include <string>

namespace SoilMath {
namespace Exception {
class MathException : public std::exception {
public:
  MathException(std::string m = EXCEPTION_MATH, int n = EXCEPTION_MATH_NR)
      : msg(m), nr(n){};
  ~MathException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
  const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }

private:
  std::string msg;
  int nr;
};
}
}
```

```
/* 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 <stdint.h>

namespace SoilMath {
/*!
 * \brief The Sort template class
 */
class Sort {
public:
  Sort() {}
  ~Sort() {}

  /*!
   * \brief QuickSort a static sort a Type T array with i values
   * \details Usage: QuickSort<type>(*type , i)
   * \param arr an array of Type T
   * \param i the number of elements
   */
  template <typename T> static void QuickSort(T *arr, int i) {
    if (i < 2)
      return;

    T p = arr[i / 2];
    T *l = arr;
    T *r = arr + i - 1;
    while (l <= r) {
      if (*l < p) {
        l++;
      } else if (*r > p) {
        r--;
      } else {
        T t = *l;
        *l = *r;
        *r = t;
        l++;
        r--;
      }
    }
    Sort::QuickSort<T>(arr, r - arr + 1);
    Sort::QuickSort<T>(l, arr + i - l);
  }

  /*!
   * \brief QuickSort a static sort a Type T array with i values where the
   * are also changed accordingly
   * \details Usage: QuickSort<type>(*type *type , i)
   * \param arr an array of Type T
   * \param key an array of 0..i-1 representing the index
```

```
   * \param i the number of elements
   */
  template <typename T> static void QuickSort(T *arr, T *key, int i) {
    if (i < 2)
      return;

    T p = arr[i / 2];

    T *l = arr;
    T *r = arr + i - 1;

    T *lkey = key;
    T *rkey = key + i - 1;

    while (l <= r) {
      if (*l < p) {
        l++;
        lkey++;
      } else if (*r > p) {
        r--;
        rkey--;
      } else {
       if (*l != *r) {
          T t = *l;
          *l = *r;
          *r = t;

          T tkey = *lkey;
          *lkey = *rkey;
          *rkey = tkey;
        }

        l++;
        r--;

        lkey++;
        rkey--;
      }
    }
    Sort::QuickSort<T>(arr, key, r - arr + 1);
    Sort::QuickSort<T>(l, lkey, arr + i - l);
  }
};
}
```

B. Hardware Library

B.0.1 Microscope Class

```
/* 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
 */

/*! \class Microscope
Interaction with the USB 5 MP microscope
*/

#pragma once

#include <stdint.h>
#include <vector>
#include <string>
#include <utility>
#include <algorithm>

#include <sys/stat.h>
#include <sys/utsname.h>
#include <sys/ioctl.h>
#include <fstream>
#include <fcntl.h>

#include <linux/videodev2.h>
#include <linux/v4l2-controls.h>
#include <linux/v4l2-common.h>

#include <boost/filesystem.hpp>
#include <boost/regex.hpp>
```

```
#include <opencv2/photo.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/core.hpp>

#include "MicroscopeNotFoundException.h"
#include "CouldNotGrabImageException.h"

namespace Hardware {
class Microscope {
public:
  enum Arch { ARM, X64 };

  enum PixelFormat { YUYV, MJPG };

  struct Resolution_t {
    uint16_t Width = 2048;
    uint16_t Height = 1536;
    PixelFormat format = PixelFormat::MJPG;
    std::string to_string() {
      std::string retVal = std::to_string(Width);
      retVal.append(" x ");
      retVal.append(std::to_string(Height));
      if (format == PixelFormat::MJPG) {
          retVal.append(" - MJPG");
        }
      else {
          retVal.append(" - YUYV");
        }
      return retVal;
    }
    uint32_t ID;
  };

  struct Control_t {
    std::string name;
    int minimum;
    int maximum;
    int step;
    int default_value;
    int current_value;
    uint32_t ID = V4L2_CID_BASE;
    bool operator==(Control_t &rhs) {
      if (this->name.compare(rhs.name) == 0) {
        return true;
      } else {
        return false;
      }
    }
    bool operator!=(Control_t &rhs) {
      if (this->name.compare(rhs.name) != 0) {
        return true;
      } else {
        return false;
      }
```

```
  }
};

typedef std::vector<Control_t> Controls_t;

struct Cam_t {
  std::string Name;
  std::string devString;
  uint32_t ID;
  std::vector<Resolution_t> Resolutions;
  uint32_t delaytrigger = 1;
  Resolution_t *SelectedResolution = nullptr;
  Controls_t Controls;
  int fd;
  bool operator==(Cam_t const &rhs) {
    if (this->ID == rhs.ID || this->Name == rhs.Name) {
      return true;
    } else {
      return false;
    }
  }
  bool operator!=(Cam_t const &rhs) {
    if (this->ID != rhs.ID && this->Name != rhs.Name) {
      return true;
    } else {
      return false;
    }
  }
};

std::vector<Cam_t> AvailableCams;
Cam_t *SelectedCam = nullptr;
Arch RunEnv;

Microscope();
Microscope(const Microscope &rhs);

~Microscope();

Microscope operator=(Microscope const &rhs);

bool IsOpened();
bool openCam(Cam_t *cam);
bool openCam(int &cam);
bool openCam(std::string &cam);

bool closeCam(Cam_t *cam);

void GetFrame(cv::Mat &dst);
void GetHDRFrame(cv::Mat &dst, uint32_t noframes = 3);

Control_t *GetControl(const std::string name);
void SetControl(Control_t *control);

Cam_t *FindCam(std::string cam);
Cam_t *FindCam(int cam);
```

```
private:
  cv::VideoCapture *cap = nullptr;

  std::vector<cv::Mat> HDRframes;

  std::vector<Cam_t> GetAvailableCams();
  Arch GetCurrentArchitecture();
  int fd;
};
}
```

```
/* 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 "Microscope.h"

namespace Hardware {

Microscope::Microscope() {
  RunEnv = GetCurrentArchitecture();
  AvailableCams = GetAvailableCams();
  for_each(AvailableCams.begin(), AvailableCams.end(), [](Cam_t &C) {
    C.SelectedResolution = &C.Resolutions[C.Resolutions.size() - 1];
  });
}

Microscope::Microscope(const Microscope &rhs) {
  std::copy(rhs.AvailableCams.begin(), rhs.AvailableCams.end(),
            this->AvailableCams.begin());
  this->RunEnv = rhs.RunEnv;
  this->SelectedCam = rhs.SelectedCam;
  this->cap = rhs.cap;
  this->fd = rhs.fd;
  this->HDRframes = rhs.HDRframes;
}

Microscope::~Microscope() { delete cap; }

Microscope::Arch Microscope::GetCurrentArchitecture() {
  struct utsname unameData;
  Arch retVal;
  uname(&unameData);
  std::string archString = static_cast<std::string>(unameData.machine);
  if (archString.find("armv7l") != string::npos) {
    retVal = Arch::ARM;
  } else {
    retVal = Arch::X64;
  }
  return retVal;
}

std::vector<Microscope::Cam_t> Microscope::GetAvailableCams() {
  const string path_ss = "/sys/class/video4linux";
  const string path_ss_dev = "/dev/video";
  std::vector<Cam_t> retVal;
  struct v4l2_queryctrl queryctrl;
  struct v4l2_control controlctrl;

  // Check if there're videodevices installed
  // Itterate through the cams
  for (boost::filesystem::directory_iterator itr(path_ss);
       itr != boost::filesystem::directory_iterator(); ++itr) {
```

```
string videoln = itr->path().string();
videoln.append("/name");
if (boost::filesystem::exists(videoln)) {
  Cam_t currentCam;
  std::ifstream camName;
  camName.open(videoln);
  std::getline(camName, currentCam.Name);
  camName.close();
  currentCam.ID =
      std::atoi(itr->path().string().substr(28, std::string::npos).c_str());

  // Open Cam
  currentCam.devString = path_ss_dev + std::to_string(currentCam.ID);
  if ((currentCam.fd = open(currentCam.devString.c_str(), O_RDWR)) == -1) {
    throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
                                         EXCEPTION_NOCAMS_NR);
  }

  // Get controls
  memset(&queryctrl, 0, sizeof(queryctrl));
  memset(&controlctrl, 0, sizeof(controlctrl));
  for (queryctrl.id = V4L2_CID_BASE; queryctrl.id < V4L2_CID_LASTP1;
       queryctrl.id++) {

    if (ioctl(currentCam.fd, VIDIOC_QUERYCTRL, &queryctrl) == 0) {
      if (!(queryctrl.flags & V4L2_CTRL_FLAG_DISABLED)) {
        Control_t currentControl;
        currentControl.ID = queryctrl.id;
        currentControl.name = (char *)queryctrl.name;
        currentControl.minimum = queryctrl.minimum;
        currentControl.maximum = queryctrl.maximum;
        currentControl.default_value = queryctrl.default_value;
        currentControl.step = queryctrl.step;
        controlctrl.id = queryctrl.id;
        if (ioctl(currentCam.fd, VIDIOC_G_CTRL, &controlctrl) == 0) {
          currentControl.current_value = controlctrl.value;
        }
        currentCam.Controls.push_back(currentControl);
      }
    } else {
      if (errno == EINVAL)
        continue;
      throw Exception::MicroscopeException(EXCEPTION_QUERY,
                                           EXCEPTION_QUERY_NR);
    }
  }

  // Get image formats
  struct v4l2_format format;
  memset(&format, 0, sizeof(format));

  uint32_t width[5] = {640, 800, 1280, 1600, 2048};
  uint32_t height[6] = {480, 600, 960, 1200, 1536};

  uint32_t ResolutionID = 0;
```

```
      // YUYV
      for (uint32_t i = 0; i < 5; i++) {
        format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        format.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
        format.fmt.pix.width = width[i];
        format.fmt.pix.height = height[i];
        int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
        if (ret != -1 && format.fmt.pix.height == height[i] &&
            format.fmt.pix.width == width[i]) {
          Resolution_t res;
          res.Width = format.fmt.pix.width;
          res.Height = height[i];
          res.ID = ResolutionID++;
          res.format = PixelFormat::YUYV;
          currentCam.Resolutions.push_back(res);
        }
      }

      // MJPEG
      for (uint32_t i = 0; i < 5; i++) {
        format.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        format.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
        format.fmt.pix.width = width[i];
        format.fmt.pix.height = height[i];
        int ret = ioctl(currentCam.fd, VIDIOC_S_FMT, &format);
        if (ret != -1 && format.fmt.pix.height == height[i] &&
            format.fmt.pix.width == width[i]) {
          Resolution_t res;
          res.Width = format.fmt.pix.width;
          res.Height = format.fmt.pix.height;
          res.ID = ResolutionID++;
          res.format = PixelFormat::MJPG;
          currentCam.Resolutions.push_back(res);
        }
      }

      close(currentCam.fd);
      retVal.push_back(currentCam);
    }
  }

  for (uint32_t i = 0; i < retVal.size(); i++) {
    if (retVal[i].Resolutions.size() == 0) {
      retVal.erase(retVal.begin() + i);
      i--;
    }
  }

  return retVal;
}

bool Microscope::IsOpened() {
  if (cap == nullptr) {
    return false;
  } else {
    return cap->isOpened();
```

```
  }
}

bool Microscope::openCam(Cam_t *cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (AvailableCams[i] == *cam) {
      closeCam(SelectedCam);
      SelectedCam = cam;
      cap = new cv::VideoCapture(SelectedCam->ID);
      if (!cap->isOpened()) {
        throw Exception::MicroscopeException(EXCEPTION_NOCAMS,
                                             EXCEPTION_NOCAMS_NR);
      }
      cap->set(CV_CAP_PROP_FRAME_WIDTH, SelectedCam->SelectedResolution->Width);
      cap->set(CV_CAP_PROP_FRAME_HEIGHT,
               SelectedCam->SelectedResolution->Height);
      for (Controls_t::iterator it = SelectedCam->Controls.begin();
           it != SelectedCam->Controls.end(); ++it) {
        SetControl(&*it);
      }
      return true;
    }
  }
  return false;
}

bool Microscope::openCam(std::string &cam) { return openCam(FindCam(cam)); }

bool Microscope::openCam(int &cam) { return openCam(FindCam(cam)); }

Microscope::Cam_t *Microscope::FindCam(int cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (cam == AvailableCams[i].ID) {
      return &AvailableCams[i];
    }
  }
  return nullptr;
}

Microscope::Cam_t *Microscope::FindCam(string cam) {
  for (uint32_t i = 0; i < AvailableCams.size(); i++) {
    if (cam.compare(AvailableCams[i].Name) == 0) {
      return &AvailableCams[i];
    }
  }
  return nullptr;
}

bool Microscope::closeCam(Cam_t *cam) {
  if (cap != nullptr) {
    if (cap->isOpened()) {
      cap->release();
    }
    delete cap;
    cap = nullptr;
  }
```

```
}

void Microscope::GetFrame(cv::Mat &dst) {
  openCam(SelectedCam);
  sleep(SelectedCam->delaytrigger);
  if (RunEnv == Arch::ARM) {
    for (uint32_t i = 0; i < 2; i++) {
      if (!cap->grab()) {
        throw Exception::CouldNotGrabImageException();
      }
      sleep(SelectedCam->delaytrigger);
    }
    cap->retrieve(dst);
  } else {
    for (uint32_t i = 0; i < 2; i++) {
      if (!cap->read(dst)) {
        throw Exception::CouldNotGrabImageException();
      }
    }
  }
}

void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes) {
  // create the brightness steps
  Control_t *brightness = GetControl("Brightness");
  Control_t *contrast = GetControl("Contrast");

  uint32_t brightnessStep =
      (brightness->maximum - brightness->minimum) / noframes;
  int8_t currentBrightness = brightness->current_value;
  int8_t currentContrast = contrast->current_value;
  contrast->current_value = contrast->maximum;

  cv::Mat currentImg;
  // take the shots at different brightness levels
  for (uint32_t i = 1; i <= noframes; i++) {
    brightness->current_value = brightness->minimum + (i * brightnessStep);
    GetFrame(currentImg);
    HDRframes.push_back(currentImg);
  }

  // Set the brightness and back to the previous used level
  brightness->current_value = currentBrightness;
  contrast->current_value = currentContrast;

  // Perform the exposure fusion
  cv::Mat fusion;
  cv::Ptr<cv::MergeMertens> merge_mertens = cv::createMergeMertens();
  merge_mertens->process(HDRframes, fusion);
  fusion *= 255;
  fusion.convertTo(dst, CV_8UC1);
}

Microscope::Control_t *Microscope::GetControl(const string name) {
  for (Controls_t::iterator it = SelectedCam->Controls.begin();
       it != SelectedCam->Controls.end(); ++it) {
```

```
    if (name.compare(it->name) == 0) {
      return &*it;
    }
  }
  return nullptr;
}

void Microscope::SetControl(Control_t *control) {
  if ((SelectedCam->fd = open(SelectedCam->devString.c_str(), O_RDWR)) == -1) {
    throw Exception::MicroscopeException(EXCEPTION_NOCAMS, EXCEPTION_NOCAMS_NR);
  }

  struct v4l2_queryctrl queryctrl;
  struct v4l2_control controlctrl;

  memset(&queryctrl, 0, sizeof(queryctrl));
  queryctrl.id = control->ID;
  if (ioctl(SelectedCam->fd, VIDIOC_QUERYCTRL, &queryctrl) == -1) {
    if (errno != EINVAL) {
      close(SelectedCam->fd);
      throw Exception::MicroscopeException(EXCEPTION_QUERY, EXCEPTION_QUERY_NR);
    } else {
      close(SelectedCam->fd);
      throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
                                           EXCEPTION_CTRL_NOT_FOUND_NR);
    }
  } else if (queryctrl.flags & V4L2_CTRL_FLAG_DISABLED) {
    close(SelectedCam->fd);
    throw Exception::MicroscopeException(EXCEPTION_CTRL_NOT_FOUND,
                                         EXCEPTION_CTRL_NOT_FOUND_NR);
  } else {
    memset(&controlctrl, 0, sizeof(controlctrl));
    controlctrl.id = control->ID;
    controlctrl.value = control->current_value;

    if (ioctl(SelectedCam->fd, VIDIOC_S_CTRL, &controlctrl) == -1) {
      // Fails on auto white balance
      // throw Exception::MicroscopeException(EXCEPTION_CTRL_VALUE,
      //                                      EXCEPTION_CTRL_VALUE_NR);
    }
  }
  close(SelectedCam->fd);
}
}
```

B.0.2 Beaglebone Black Class

```
/* 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
 */

/*! \class BBB
The core BeagleBone Black class used for all hardware related classes.
Consisting of universal used method, functions and variables. File operatio
polling and threading
*/

#pragma once

#define SLOTS
\
  "/sys/devices/bone_capemgr.9/slots" /*!< Beaglebone capemanager slots fil

#include <fstream>
#include <sstream>
#include <string>
#include <sys/stat.h>
#include <pthread.h>
#include <unistd.h>
#include <sys/epoll.h>
#include <fcntl.h>
#include <regex>
#include <stdexcept>

#include "GPIOReadException.h"
#include "FailedToCreateGPIOPollingThreadException.h"
#include "ValueOutOfBoundsException.h"

using namespace std;

namespace Hardware {
typedef int (*CallbackType)(
    int); /*!< CallbackType used to pass a function to a thread*/

class BBB {
public:
  int debounceTime; /*!< debounce time for a button in milliseconds*/

  BBB();
  ~BBB();

protected:
  bool threadRunning;             /*!< used to stop the thread*/
  pthread_t thread;               /*!< The thread*/
  CallbackType callbackFunction; /*!< the callbakcfunction*/

  bool DirectoryExist(const string &path);
  bool CapeLoaded(const string &shield);
```

```
  string Read(const string &path);
  void Write(const string &path, const string &value);

  /*! Converts a number to a string
  \param Number as typename
  \returns the number as a string
  */
  template <typename T> string NumberToString(T Number) {
    ostringstream ss;
    ss << Number;
    return ss.str();
  };

  /*! Converts a string to a number
  \param Text the string that needs to be converted
  \return the number as typename
  */
  template <typename T> T StringToNumber(string Text) {
    stringstream ss(Text);
    T result;
    return ss >> result ? result : 0;
  };
};
}
```

```
/* 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 "BBB.h"

namespace Hardware {
/*! Constructor*/
BBB::BBB() {
  threadRunning = false;
  callbackFunction = NULL;
  debounceTime = 0;
  thread = (pthread_t)NULL;
}

/*! De-constructor*/
BBB::~BBB() {}

/*! Reads the first line from a file
\param path constant string pointing towards the file
\returns this first line
*/
string BBB::Read(const string &path) {
  ifstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " + path).c_str());
  }
  string input;
  getline(fs, input);
  fs.close();
  return input;
}

/*! Writes a value to a file
\param path a constant string pointing towards the file
\param value a constant string which should be written in the file
*/
void BBB::Write(const string &path, const string &value) {
  ofstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " + path).c_str());
  }
  fs << value;
  fs.close();
}

/*! Checks if a directory exist
\returns true if the directory exists and false if not
*/
bool BBB::DirectoryExist(const string &path) {
```

```
  struct stat st;
  if (stat((char *)path.c_str(), &st) != 0) {
    return false;
  }
  return true;
}

/*! Checks if a cape is loaded in the file /sys/devices/bone_capemgr.9/slots
\param shield a const search string which is a (part) of the shield name
\return true if the search string is found otherwise false
*/
bool BBB::CapeLoaded(const string &shield) {
  bool shieldFound = false;

  ifstream fs;
  fs.open(SLOTS);
  if (!fs.is_open()) {
    throw Exception::GPIOReadException("Can't open SLOTS");
  }

  string line;
  while (getline(fs, line)) {
    if (line.find(shield) != string::npos) {
      shieldFound = true;
      break;
    }
  }
  fs.close();
  return shieldFound;
}
}
```

B.0.3 GPIO Class

```
/* 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
 * This code is based upon:
 * Derek Molloy, "Exploring BeagleBone: Tools and Techniques for Building
 * with Embedded Linux", Wiley, 2014, ISBN:9781118935125.
 * See: www.exploringbeaglebone.com
 */

#pragma once
#include "BBB.h"

#define EXPORT_PIN "/sys/class/gpio/export"
#define UNEXPORT_PIN "/sys/class/gpio/unexport"
#define GPIOS "/sys/class/gpio/gpio"
#define DIRECTION "/direction"
#define VALUE "/value"
#define EDGE "/edge"

using namespace std;

namespace Hardware {
class GPIO : public BBB {
public:
  enum Direction { Input, Output };
  enum Value { Low = 0, High = 1 };
  enum Edge { None, Rising, Falling, Both };

  int number; // Number of the pin

  int WaitForEdge();
  int WaitForEdge(CallbackType callback);
  void WaitForEdgeCancel() { this->threadRunning = false; }

  Value GetValue();
  void SetValue(Value value);

  Direction GetDirection();
  void SetDirection(Direction direction);

  Edge GetEdge();
  void SetEdge(Edge edge);

  GPIO(int number);
  ~GPIO();

private:
  string gpiopath;
  Direction direction;
  Edge edge;
  friend void *threadedPollGPIO(void *value);
```

```
  bool isExported(int number, Direction &dir, Edge &edge);
  bool ExportPin(int number);
  bool UnexportPin(int number);

  Direction ReadsDirection(const string &gpiopath);
  void WritesDirection(const string &gpiopath, Direction direction);

  Edge ReadsEdge(const string &gpiopath);
  void WritesEdge(const string &gpiopath, Edge edge);

  Value ReadsValue(const string &gpiopath);
  void WritesValue(const string &gpiopath, Value value);
};

void *threadedPollGPIO(void *value);
}
```

```
/* 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 "GPIO.h"

namespace Hardware {
GPIO::GPIO(int number) {

  this->number = number;
  gpiopath = GPIOS + NumberToString<int>(number);

  if (!isExported(number, direction, edge)) {
    ExportPin(number);
    direction = ReadsDirection(gpiopath);
    edge = ReadsEdge(gpiopath);
  }
  usleep(250000);
}

GPIO::~GPIO() { UnexportPin(number); }

int GPIO::WaitForEdge(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&this->thread, NULL, &threadedPollGPIO,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::FailedToCreateGPIOPollingThreadException();
  }
  return 0;
}

int GPIO::WaitForEdge() {
  if (direction == Output) {
    SetDirection(Input);
  }
  int fd, i, epollfd, count = 0;
  struct epoll_event ev;
  epollfd = epoll_create(1);
  if (epollfd == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "GPIO: Failed to create epollfd!");
  }
  if ((fd = open((gpiopath + VALUE).c_str(), O_RDONLY | O_NONBLOCK)) == -1)
    throw Exception::GPIOReadException();
  }

  // read operation | edge triggered | urgent data
  ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
  ev.data.fd = fd;
```

```
  if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "GPIO: Failed to add control interface!");
  }

  while (count <= 1) {
    i = epoll_wait(epollfd, &ev, 1, -1);
    if (i == -1) {
      close(fd);
      return -1;
    } else {
      count++;
    }
  }
  close(fd);
  return 0;
}

GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath); }
void GPIO::SetValue(GPIO::Value value) { WritesValue(gpiopath, value); }

GPIO::Direction GPIO::GetDirection() { return direction; }
void GPIO::SetDirection(Direction direction) {
  this->direction = direction;
  WritesDirection(gpiopath, direction);
}

GPIO::Edge GPIO::GetEdge() { return edge; }
void GPIO::SetEdge(Edge edge) {
  this->edge = edge;
  WritesEdge(gpiopath, edge);
}

bool GPIO::isExported(int number __attribute__((unused)), Direction &dir, Edge &ed
  // Checks if directory exist and therefore is exported
  if (!DirectoryExist(gpiopath)) {
    return false;
  }

  // Reads the data associated with the pin
  dir = ReadsDirection(gpiopath);
  edge = ReadsEdge(gpiopath);
  return true;
}

bool GPIO::ExportPin(int number) {
  Write(EXPORT_PIN, NumberToString<int>(number));
  usleep(250000);
}

bool GPIO::UnexportPin(int number) {
  Write(UNEXPORT_PIN, NumberToString<int>(number));
}

GPIO::Direction GPIO::ReadsDirection(const string &gpiopath) {
  if (Read(gpiopath + DIRECTION) == "in") {
```

```
    return Input;
  } else {
    return Output;
  }
}

void GPIO::WritesDirection(const string &gpiopath, Direction direction) {
  switch (direction) {
  case Hardware::GPIO::Input:
    Write((gpiopath + DIRECTION), "in");
    break;
  case Hardware::GPIO::Output:
    Write((gpiopath + DIRECTION), "out");
    break;
  }
}

GPIO::Edge GPIO::ReadsEdge(const string &gpiopath) {
  string reader = Read(gpiopath + EDGE);
  if (reader == "none") {
    return None;
  } else if (reader == "rising") {
    return Rising;
  } else if (reader == "falling") {
    return Falling;
  } else {
    return Both;
  }
}

void GPIO::WritesEdge(const string &gpiopath, Edge edge) {
  switch (edge) {
  case Hardware::GPIO::None:
    Write((gpiopath + EDGE), "none");
    break;
  case Hardware::GPIO::Rising:
    Write((gpiopath + EDGE), "rising");
    break;
  case Hardware::GPIO::Falling:
    Write((gpiopath + EDGE), "falling");
    break;
  case Hardware::GPIO::Both:
    Write((gpiopath + EDGE), "both");
    break;
  default:
    break;
  }
}

GPIO::Value GPIO::ReadsValue(const string &gpiopath) {
  string path(gpiopath + VALUE);
  int res = StringToNumber<int>(Read(path));
  return (Value)res;
}

void GPIO::WritesValue(const string &gpiopath, Value value) {
```

```
  Write(gpiopath + VALUE, NumberToString<int>(value));
}

void *threadedPollGPIO(void *value) {
  GPIO *gpio = static_cast<GPIO *>(value);
  while (gpio->threadRunning) {
    gpio->callbackFunction(gpio->WaitForEdge());
    usleep(gpio->debounceTime * 1000);
  }
  return 0;
}
}
```

B.0.4 PWM Class

```
/* 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 "BBB.h"
#include <dirent.h>

#define OCP_PATH "/sys/devices/ocp.3/"
#define P8_13_FIND "bs_pwm_test_P8_13"
#define P8_19_FIND "bs_pwm_test_P8_19"
#define P9_14_FIND "bs_pwm_test_P9_14"
#define P9_16_FIND "bs_pwm_test_P9_16"

#define PWM_CAPE "am33xx_pwm"
#define P8_13_CAPE "bspwm_P8_13" //_14
#define P8_19_CAPE "bspwm_P8_13" //_14
#define P9_14_CAPE "bspwm_P9_14" //_16
#define P9_16_CAPE "bspwm_P9_16" //_16

#define P8_13_CAPE_LOAD "bspwm_P8_13_14"
#define P8_19_CAPE_LOAD "bspwm_P8_13_14"
#define P9_14_CAPE_LOAD "bspwm_P9_14_16"
#define P9_16_CAPE_LOAD "bspwm_P9_16_16"

namespace Hardware {
class PWM : public BBB {
public:
  enum Pin // Four possible PWM pins
  { P8_13,
    P8_19,
    P9_14,
    P9_16 };
  enum Run // Signal generating
  { On = 1,
    Off = 0 };
  enum Polarity // Inverse duty polarity
  { Normal = 1,
    Inverted = 0 };

  Pin pin; // Current pin

  uint8_t GetPixelValue() { return pixelvalue; }
  void SetPixelValue(uint8_t value);

  float GetIntensity() { return intensity; };
  void SetIntensity(float value);

  int GetPeriod() { return period; };
  void SetPeriod(int value);
```

```
  int GetDuty() { return duty; };
  void SetDuty(int value);
  void SetIntensity();

  Run GetRun() { return run; };
  void SetRun(Run value);

  Polarity GetPolarity() { return polarity; };
  void SetPolarity(Polarity value);

  PWM(Pin pin);
  ~PWM();

private:
  int period;          // current period
  int duty;            // current duty
  float intensity;     // current intensity
  uint8_t pixelvalue; // current pixelvalue
  Run run;             // current run state
  Polarity polarity;  // current polaity

  string basepath;     // the basepath ocp.3
  string dutypath;     // base + duty path
  string periodpath;   // base + period path
  string runpath;      // base + run path
  string polaritypath; // base + polarity path

  void calcIntensity();
  string FindPath(string value);
};
}
```

```
/* 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 "PWM.h"

namespace Hardware {
/// <summary>
/// Constructeur
/// </summary>
/// <param name="pin">Pin</param>
PWM::PWM(Pin pin) {
  this->pin = pin;

  // Check if PWM cape is loaded, if not load it
  if (!CapeLoaded(PWM_CAPE)) {
    Write(SLOTS, PWM_CAPE);
  }

  // Init the pin
  basepath = OCP_PATH;
  switch (pin) {
  case Hardware::PWM::P8_13:
    if (!CapeLoaded(P8_13_CAPE)) {
      Write(SLOTS, P8_13_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_13_FIND));
    break;
  case Hardware::PWM::P8_19:
    if (!CapeLoaded(P8_19_CAPE)) {
      Write(SLOTS, P8_19_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_19_FIND));
    break;
  case Hardware::PWM::P9_14:
    if (!CapeLoaded(P9_14_CAPE)) {
      Write(SLOTS, P9_14_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_14_FIND));
    break;
  case Hardware::PWM::P9_16:
    if (!CapeLoaded(P9_16_CAPE)) {
      Write(SLOTS, P9_16_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_16_FIND));
    break;
  }

  // Get the working paths
  dutypath = basepath + "/duty";
  periodpath = basepath + "/period";
  runpath = basepath + "/run";
```

```
  polaritypath = basepath + "/polarity";

  // Give Linux time to setup directory structure;
  usleep(250000);

  // Read current values
  period = StringToNumber<int>(Read(periodpath));
  duty = StringToNumber<int>(Read(dutypath));
  run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
  polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));

  // calculate the current intensity
  calcIntensity();
}

PWM::~PWM() {}

/// <summary>
/// Calculate the current intensity
/// </summary>
void PWM::calcIntensity() {
  if (polarity == Normal) {
    if (duty == 0) {
      intensity = 0.0f;
    } else {
      intensity = (float)period / (float)duty;
    }
  } else {
    if (period == 0) {
      intensity = 0.0f;
    } else {
      intensity = (float)duty / (float)period;
    }
  }
}

/// <summary>
/// Set the intensity level as percentage
/// </summary>
/// <param name="value">floating value multipication factor</param>
void PWM::SetIntensity(float value) {
  if (polarity == Normal) {
    SetDuty(static_cast<int>((value * duty) + 0.5));
  } else {
    SetPeriod(static_cast<int>((value * period) + 0.5));
  }
}

/// <summary>
/// Set the output as a corresponding uint8_t value
/// </summary>
/// <param name="value">pixel value 0-255</param>
void PWM::SetPixelValue(uint8_t value) {
  if (period != 255) {
    SetPeriod(255);
  }
```

```
  SetDuty(255 - value);
  pixelvalue = value;
}

/// <summary>
/// Set the period of the signal
/// </summary>
/// <param name="value">period : int</param>
void PWM::SetPeriod(int value) {
  string valstr = NumberToString<int>(value);
  Write(periodpath, valstr);
  period = value;

  calcIntensity();
}

/// <summary>
/// Set the duty of the signal
/// </summary>
/// <param name="value">duty : int</param>
void PWM::SetDuty(int value) {
  string valstr = NumberToString<int>(value);
  Write(dutypath, valstr);
  duty = value;

  calcIntensity();
}

/// <summary>
/// Run the signal
/// </summary>
/// <param name="value">On or Off</param>
void PWM::SetRun(Run value) {
  int valInt = static_cast<int>(value);
  string valstr = NumberToString<int>(valInt);
  Write(runpath, valstr);
  run = value;
}

/// <summary>
/// Set the polarity
/// </summary>
/// <param name="value">Normal or Inverted signal</param>
void PWM::SetPolarity(Polarity value) {
  int valInt = static_cast<int>(value);
  string valstr = NumberToString<int>(valInt);
  Write(runpath, valstr);
  polarity = value;
}

/// <summary>
/// Find the current PWM path in the OCP.3 directory
/// </summary>
/// <param name="value">part a the path name</param>
/// <returns>Returns the first found value</returns>
string PWM::FindPath(string value) {
```

```
  auto dir = opendir(OCP_PATH);
  auto entity = readdir(dir);
  while (entity != NULL) {
    if (entity->d_type == DT_DIR) {
      string str = static_cast<string>(entity->d_name);
      if (str.find(value) != string::npos) {
        return str;
      }
    }
    entity = readdir(dir);
  }
  return "";
}
}
```

B.0.5 General project file

```
/* 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 "ADC.h"
#include "EC12P.h"
#include "eqep.h"
#include "GPIO.h"
#include "PWM.h"
#include "SoilCape.h"
#include "Microscope.h"
#include "CouldNotGrabImageException.h"
#include "ADCReadException.h"
#include "FailedToCreateGPIOPollingThreadException.h"
#include "FailedToCreateThreadException.h"
#include "GPIOReadException.h"
#include "MicroscopeNotFoundException.h"
#include "ValueOutOfBoundsException.h"
```

```
/* 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 <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class ValueOutOfBoundsException : public std::exception {
public:
  ValueOutOfBoundsException(string m = "Value out of bounds!") : msg(m){};
  ~ValueOutOfBoundsException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* 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 <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class ADCReadException : public std::exception {
public:
  ADCReadException(string m = "Can't read ADC data!") : msg(m){};
  ~ADCReadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* 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 <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class FailedToCreateGPIOPollingThreadException : public std::exception {
public:
  FailedToCreateGPIOPollingThreadException(
      string m = "Failed to create GPIO polling thread!")
      : msg(m){};
  ~FailedToCreateGPIOPollingThreadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* 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 <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class FailedToCreateThreadException : public std::exception {
public:
  FailedToCreateThreadException(string m = "Couldn't create the thread!")
      : msg(m){};
  ~FailedToCreateThreadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* 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
 */
#define EXCEPTION_OPENCAM "Exception could not open cam!"
#define EXCEPTION_OPENCAM_NR 0
#define EXCEPTION_NOCAMS "Exception no cam available!"
#define EXCEPTION_NOCAMS_NR 1
#define EXCEPTION_QUERY "Exception could not query device!"
#define EXCEPTION_QUERY_NR 3
#define EXCEPTION_FORMAT_RESOLUTION "Exception No supported formats and resolutior
#define EXCEPTION_FORMAT_RESOLUTION_NR 4
#define EXCEPTION_CTRL_NOT_FOUND "Control not found!"
#define EXCEPTION_CTRL_NOT_FOUND_NR 5
#define EXCEPTION_CTRL_VALUE "Control value not set!"
#define EXCEPTION_CTRL_VALUE_NR 5


#pragma once
#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class MicroscopeException : public std::exception {
public:
  MicroscopeException(string m = EXCEPTION_OPENCAM,
                      int n = EXCEPTION_OPENCAM_NR) : msg{m}, nr{n} {
}
  ~MicroscopeException() _GLIBCXX_USE_NOEXCEPT {}
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); }
  const int *id() const _GLIBCXX_USE_NOEXCEPT { return &nr; }

private:
  string msg;
  int nr;
};
}
}
```

```
/* 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 <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class CouldNotGrabImageException : public std::exception {
public:
  CouldNotGrabImageException(string m = "Unable to grab the next image!")
      : msg(m){};
  ~CouldNotGrabImageException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```