Image analysis,
an approach to measure grass roots from images

(HS-IDA-MD-01-302)

Jonas Hansson
Department of Computer Science
Högskolan i Skövde, PO Box 408
SE-54128 Skövde, SWEDEN

Final year project
on the study programme in computer science 2001
Supervisor: Dan Lundh
Image analysis, 
an approach to measure grass roots from images

Submitted by Jonas Hansson to Högskolan Skövde as a dissertation for the degree of M.Sc., in the Department of Computer Science.

4/9 2001

I certify that all material in this dissertation which is not my own work has been identified and that no material is included for which a degree has previously been conferred on me.

Signed: _______________________________________________
Image analysis,  

an approach to measure grass roots from images  

Jonas Hansson (a97jonha@ida.his.se)

Abstract

In this project a method to analyse images is presented. The images document the development of grassroots in a tilled field in order to study the movement of nitrate in the field. The final aim of the image analysis is to estimate the volume of dead and living roots in the soil. Since the roots and the soil have a broad and overlapping range of colours the fundamental problem is to find the roots in the images. Earlier methods for analysis of root images have used methods based on thresholds to extract the roots. To use a threshold the pixels of the object must have a unique range of colours separating them from the colour of the background, this is not the case for the images in this project. Instead the method uses a neural network to classify the individual pixels. In this paper a complete method to analyse images is presented and although the results are far from perfect, the method gives interesting results.

Keywords: image analysis, neural networks, backpropagation, roots, soil.
# Table of contents

1 Introduction ........................................................................................ 1

2 Background ......................................................................................... 2

2.1 Data .................................................................................................................2
2.2 Other approaches to analyse root images ..........................................................3
2.3 Image analysis ..........................................................4
2.4 Analysis of the problem ..........................................................4

3 Thesis statement .................................................................................. 7

3.1 Motivation .......................................................................................................7
3.2 Objectives ........................................................................................................7
3.2.1 Preparing data ...............................................................................................8
3.2.2 Adapting a method ........................................................................................8
3.2.3 Implementation of method ............................................................................8
3.2.4 Test of performance ......................................................................................8
3.2.5 Generalisation ...............................................................................................9
3.2.6 Human interface ............................................................................................9

4 Method............................................................................................... 10

4.1 Using a neural network to identify pixels of dead and living roots...............10
4.2 Applying a threshold ......................................................................................11
4.3 Measuring the identified objects .....................................................................11
4.4 Filtering out non-root objects .........................................................................12

5 Implementation ................................................................................. 13

5.1 Network input ................................................................................................13
5.2 Network set-up ...............................................................................................14
5.3 Extracting and evaluating objects ...................................................................15

6 Results................................................................................................ 17

6.1 Network performance .....................................................................................17
6.2 Extracting and evaluating object .....................................................................19

7 Analysis and conclusion .................................................................... 21

8 Discussion and future work .............................................................. 23

References ............................................................................................ 24

Appendix A Images from different steps of the algorithm. .......... 25
Appendix B Histograms ........................................................................... 29
Appendix C C++ Code................................................................. 30
main7.cpp for training, evaluating and using the neural network ..................30
tva.cpp used to apply the threshold.................................................................40
tre.cpp used for extracting and measuring the thresholded images...............46
image.h a header for handling the images.........................................................49
jonas.h a general utility header........................................................................60
vector.h containing a class for storing data about roots..................................65
BPbox.h containing a class that handles the neural the network......................67
1 Introduction

In many fields, both within research and industry, there is a need for image analysis. This need is often fulfilled by human analysis. There are computerised methods but they often lack in quality of analysis and, in particular, in flexibility. Many image analysis tasks intend to measure the shape of objects in the image. In order to do this the objects must first be identified and extracted from the images. The task of finding objects in an image is easy if it is possible to use a threshold, for instance if all pixels belonging to objects are lighter than every other pixel. If it is not possible to find the object with a threshold the task of finding the objects gets far more complicated.

The images in this project come from a study conducted by the Swedish University of Agriculture Science. In order to follow the development of grass roots the roots were filmed from plastic tubes in the soil. The image analysis task is to find the roots in the soil, measure their size and estimate the volume of the roots. Since the roots as well as the soil vary in colour the key problem is to identify the roots in the soil; it is not possible to use a threshold. Instead the images are pre-processed using a neural network before applying a threshold. From the thresholded images the identified object is measured. Finally the objects are filtered based on shape and the objects identified as roots are printed out.

This paper first describes the nature of the images to be analysed and gives a general background to some aspects of image analysis. Next the intentions of this project are described. Then the method used to analyse the images is presented and the implementation of the method is described. The reached results are described and analysed. Finally the method is discussed and ideas for future work are presented.
2 Background

Modern agriculture including modern fertilisers is vital for supplying a growing population with food. Still, the large amount of fertilisers constitutes a threat to the environment. Fertilisers leaching from soil to water threaten to turn lakes and bays into sterile wastelands. One method to minimise this leaching is to grow a catch crop together with the main crop. In an experiment conducted by the Department of soil sciences at the Swedish University of Agriculture Science the main crop was spring wheat and the catch crop was perennial rye grass (Lolium perenne L.). The focus of the experiment was to investigate how the catch crop affected the movement of nitrate in the soil. In order to investigate the development of the root systems of the plants, 24 plastic tubes were pushed down the soil and the growth and decay of the roots were documented using video camera. This documentation was conducted at six occasions from July 1996 to March 1997. During the documentation the camera could cover one sixth of the tube’s diameter and stopped to film 24 times while moving upwards. Thus, the recorded video film, in a sense, consists of stills of the soil and roots. When these numbers are combined the total number of stills becomes 20 736.

From each of the stills the goal was to analyse the individual roots in the image and measure its length, diameter and state, where state here simply refers to whether the root is living or dead. The purpose was to estimate the amount of roots in the soil. Similar analysis has been conducted manually for similar images from forestry where the roots are large and few. The images in this case contain many more roots, making the task too laborious to be done manually. At this stage the department of computer science at University of Skövde was contacted with the question whether is it possible to perform this analysis with a computerised mechanism.

2.1 Data

Using the software Media100 the videotapes were converted into RGB images, each with a height of 576 pixels and a width of 768 pixels. In many of these images the roots are fairly easy for a human inspector to see as they appear as differently coloured, usually lighter, ribbons. These images will constitute the input for an analysing system. The desired output from the system can be thought of as a set of tables, one for each image, where the rows are the roots and the columns are length, diameter and state, respectively. From these tables the estimated amount of roots in the soil can easily be calculated.

Even though many of the images are relatively easy to understand for a layman they differ substantially. The roots differ in width from about 5 pixels for the smaller to about 30 pixels for the larger. There are differences in the colour of the root and the shape of the roots; some of them are straight while others have a highly irregular shape. The soil surrounding the roots differ in colour depending on the soil’s inherit colour and how wet it was at the time of filming. The texture of the soil also differs with areas of different colour. Finally, the sharpness of the images differs, see Figure 1.

Hence, the high level task of this final year project is to devise a system that from these subterranean images derives a set of tables comprising data about the roots.
2.2 Other approaches to analyse root images

One method for computerised measurement of root images it to take up the plants from the soil, wash them and scan them against a high contrast background. In an experiment concerning nitrogen absorption by turfgrass, the roots were scanned against a white background except adventitious roots that, due to their white colour, were scanned against a black background. The resulting images were then analysed using a Delta-T SCAN image analysis software (Sullivan et al., 2000). Handling roots in this manner makes the image analysing task comparatively easy, all the background has one distinct colour and every other colour is root. The drawback, apart from the work required to create the images, is that the method is destructive. Since the plants are torn out of the soil it is impossible to follow the development of individual roots.

The main method for analysing images of roots in soil has relied on humans to mark the roots. This has been done with and without the use of computers, where the first method involves tracing the roots with the mouse and the latter uses transparent sheets (Hendrick and Pregitzer 1993; Cheng et al. 1991). One method for computerised analysis of images of roots in soil is presented by Vamerali et al. (1999). Their method uses the blue band since that band has the biggest difference in colour between roots and background. The technique applied to the blue band images in their experiment was first to enhance the contrast of the images. Then they applied a threshold to the images to produce binary images. From these binary images the root skeletons were extracted by repeated boundary erosion, basically to remove the outmost pixels on the major axis of the objects, thus producing one-pixel wide lines representing the roots. The lines shorter than a defined minimum root length were filtered out resulting in a set of lines representing the roots. This approach is highly interesting regarding the task in this paper, however there are differences. In the experiment conducted by Vamerali et al. the plant, sugarbeets, were planted in lysimeters filled with sandy, clay, silty, loam or organic soil. As noted by Vamerali et al. the quality of the analysis is influenced by the background’s colour and homogeneity, they have particular problem with the light sandy soil. The minimum root length filter improves the analysis, but depending on the definition of the minimum root
length either short roots are filtered out or long non-root-object are classified as roots (Vamerali et al., 1999). The images in the task presented in this paper differ from the images used by Vamerali et al. in that they are much more diverse. The soil is from an actual field with a large amount of object such as stones and old, ploughed in, straws. Also the crop differs, instead of beets it is species of grass. The impact of these differences is that the Vamerali et al. method would not give good results for this data, at least not without significant modifications.

2.3 Image analysis

Any task of extracting data about objects from images can be divided into two subtasks, first to extract the individual objects and second to measure these objects. (Castleman, 2000). The first of these subtasks can be thought of as from the original image create a set of images, each of them representing one object and nothing else. The second subtask is then to measure these images.

The object locator algorithm uses one of three different segmentation approaches. The first method is the region approach where each pixel in the original image is classified as belonging or not belonging to an object. The second approach is the boundary approach where the aim is to directly identify the boundaries between the regions. The third method, edge detection, is to first identify the edge pixels and then to link them together to form the boundaries between objects. This third approach must in its second step be able to distinguish edges that are boundaries and those that are within objects (Castleman, 2000).

The simplest way to implement the region approach is to use a threshold. If all object pixels differ in brightness from all background pixels it is possible to apply a threshold to extract the objects. The correct value of this threshold can be determined using a histogram (Rosandich 1997). A threshold can also work on other values than brightness, such as saturation or degree of blue in the pixel. Another way to apply the region approach is to use a neural network. The input to the network is data from the pixel and the desired output is a classification into a set of classes. By using the network to classify every individual pixel the whole image can be classified. One interesting advantage of this approach is that is not necessary to define exact threshold-values, instead the network can be trained on examples (Civco, 1993). To reduce the impact of noise in the images, Jensen et al. used a 3*3 moving window to extract data to be fed to the network (Jensen et al., 2001).

2.4 Analysis of the problem

The easiest approach to image segmentation is methods that are based on the region approach by using a threshold. This approach is widely used in image analysis applications. An example is the MR_RIPL 2.0 program that analyses root images, this program requires that the roots are lighter than the background. (The MR-RIPL 2.0 user’s guide). For the images in this task there is no range of colours that separate a root-pixel from a soil-pixel. The roots tend to be lighter than the surrounding soil but both the roots and the soil vary highly in brightness. In the left image in Figure 2 a threshold is found by testing various values. The threshold value 103 is efficient for extracting the root in the image but large areas in the upper part of the image also becomes white. The left image in Figure 2 is only a small fraction of
the original image; the problem of different soil colour would increase if the whole image were thresholded. The threshold value 103 that works well around the root in the left image is not able to extract the root in the right image in Figure 2. Hence, to use a threshold the best threshold value must be calculated not only for individual images but the threshold value must be calculated individual for different parts of the images. The threshold used in figure are applied on the greyscale level, perhaps better threshold values can be found by if the threshold values is calculated on the individual colours. But this approach will make the task of finding the best threshold value much harder. For any RGB encoded pixel there are 16 581 373 \((255^2-2)\) possible threshold to consider.

If there is no distinct set of colours that root pixels have and soil pixels do not have, how is it possible to identify roots? It is impossible to directly identify a root-pixel but it is possible to identify an area that is on the border between root and soil. In the context here, an edge is a potential root-edge. An edge is defined as a line along spatial discontinuity (Rosandich 1997) or less formal; a border area between two colours. Any root is surrounded with edges but the edges can be surrounding stones or representing the border between wet and dry soil. So edge is a start but it is not enough to identify the roots.

Even though the roots in the images vary in shape there are some traits of the roots shape that are shared by the roots and can be used to classify an object as a root. In terms of image analysis a root can be seen as two long edges stretching over the image. In order to detect these edges there must be a difference between the colour of the root and the surrounding image, if there is no difference it is impossible for any system to identify the root. Furthermore, the edges are lying at a relatively short distance, and this distance is relatively constant – if the edges do not have these properties they probably surround a non-root object.
The roots in the images can be classified into two classes, dead and living. Compared to the task of finding the roots it is relatively hard for a layman to classify a root as dead or living. Generally, living roots has straighter edges and a smoother texture, see Figure 1. Another feature to help the classification is that dead roots tends to shrink leaving a hole around it, this can be seen as a dark border around the root. (This can be seen in image 960813-15 in appendix A).
3 Thesis statement

The background of this thesis is an experiment conducted to analyse root development of spring wheat and perennial rye grass. This experiment resulted in a large set of images of roots in soil and the focus of this paper is to investigate computerised methods to analyse these images. The desired final output from a suitable method is a measurement of the amount of living and dead root in the soil, i.e. the proportion of the dead and living root mass in the soil volume. In order to accomplish this, the key problem is to identify the root objects in the images and to distinguish these objects from irrelevant objects. At the centre of the investigated method is the use of a neural network to classify individual pixels as belonging to one of the three classes; living root, dead root or soil.

3.1 Motivation

Many experiments concerning plants produce images of roots in soil. This method to document natural processes gives good data on how root systems develop. The development of individual roots over time can be investigated since the documentation does not effect the root. The basic problem with this approach to document roots is the task of analysing the images. This task has usually been conducted manually, requiring many and tardy hours.

Methods for computerised analysis of these subterranean images have been proposed. However these methods usually require that the images have some specific properties, such as that the roots have a lighter colour than the surrounding soil. Thus, these methods have problem when confronted with images from field studies. The images in this task come from a real field so the soil is highly heterogeneous and the roots vary in colour. In addition there are non-root objects in the soil making the analysis more difficult.

3.2 Objectives

The goal of this master thesis can be decomposed into a number of objectives of varying importance. The first four in the list presented below are the ones of most importance. In addition they are, to some degree, in chronological order – it is necessary to choose method before implementing the method.

- Preparing data
- Adapting a method
- Implementation of method
- Test of performance
- Generalisation
- Human interface
3.2.1 Preparing data

The original data from the department of soil sciences at the Swedish university of agriculture science was recorded on six videotapes in common VHS-format. These videotapes can be seen as consisting of a set of stills and these stills must be extracted and stored in a computerised way. The format of these images should be a format that does not use lossy compression. Thus formats like TIFF or BMP are appropriate, which to use is a mere practical matter.

In addition to this video to pixel-image conversion it might be necessary to pre-process the data to ease the task for the system. Various pre-processing methods could be considered and these are likely to fall within two categories. One category concerns pre-processing of the images as a whole, the methods here are likely to be the same as those that would ease any human inspection of the images. These include methods found in any image-processing program, like edge enhancement and unsharp mask. The other category concerns transformations of the individual input sets, for instance, transforming the pixel values from the standard 0-255 range to a real value ranging between 0 and 1. For each input set these values will perhaps need to be normalised with respect to brightness. Which pre-processing methods that will be used depend on the method the system uses and the guidelines found in the literature concerning this method. The final goal of this step should be to convert the original video data into data sets suitable for the analysing system.

3.2.2 Adapting a method

Before implementing any image analysing system, it is necessary to choose what method the system will use. This choice should be based on the literature available about similar problems. First and foremost this concerns which methods that has been successfully implemented and, equally important, which that has unsuccessfully been implemented. In addition, from these successful implementations it is possible to get important clues. These clues include various hints on how to fine-tune the system, as well as hints on pitfalls to avoid. The desired output from this step is one method, or a set of methods, likely to be able to solve the task of analysing the images.

3.2.3 Implementation of method

In order to investigate the adequacy of a potential method it should be implemented. At this stage the hints and pitfalls identified in the literature plays a vital role. The aim at this stage is to device a system complete enough to test whether the method chosen can solve the task. This implies that other features of the system such as its human interface and the speed at which it process the data are given a low priority.

3.2.4 Test of performance

Having an implemented system it has to be tested with respect to how well it solves the task. This is likely to be an iterative process where various setting in the system are tested in order to find the good parameters. Then this fine-tuning is completed, the performance of the system should be evaluated. Two measurements can be identified a priori. The first is whether
the system gives any relevant results, or more precise, if it performs better than pure random. The second is if the system performs good enough. The threshold for this measurement has to be identified in co-operation with biological expertise.

3.2.5 Generalisation

Having a system that performs well on the set of root images from this field experiment it is interesting to investigate if the system performs well given another set of images. These other images could stem from experiments conducted in a different kind of soil and with different kinds of plants. Is the system able to analyse these images, perhaps with minor modifications? However, this is not a prioritised objective in this project. Still, the data set presented above is highly variable with regard to colour and shape of the roots and the colour of the soil and requires any system to generalise, at least to some degree, in order to solve the task.

3.2.6 Human interface

Having a system that is able to analyse the images, one additional step in this project can be identified. To enable easy use of the system, the system should be encapsulated with a user-friendly interface. Even if a user-friendly interface is a desirable property this is not a prioritised property of this project, nor are other software properties such as speed and memory requirements.
4 Method

As described above the method used by Vamerali et al. (1999) is based on three main steps. First a threshold is applied to the images so that potential roots become black and the surrounding soil become white. Next these black objects are skeletonised, reducing them to 1-pixel wide lines. The third step is to filter out root skeletons shorter than a defined minimum root length (Vamerali et al., 1999).

The method proposed here will be based on the method used by Vamerali et al., but modified in two ways. First, the roots in this task are not necessarily lighter than the surrounding soil and consequently, the images must be pre-processed in some manner to produce the binary black and white images. As Vamerali et al. the segmentation will be based on the region approach but in a manner that evaluate the individual pixel together with surrounding pixels. Second the method used by Vamerali et al. filter out non-roots by considering the length of the skeletons. This is not enough; indeed Vamerali et al. reports problem when there are larger light objects in the soil.

The method presented here will work in four steps:

1 Using a neural network to identify pixels of dead and living roots.
2 Applying a threshold.
3 Measuring the identified objects.
4 Filtering out non-root objects.

4.1 Using a neural network to identify pixels of dead and living roots

As described above it is not possible to identify a root pixel by just inspecting that individual pixel; it is necessary to consider a larger part of the image. It would be very hard to exactly specify what relation a pixel should have to other pixels to be classified as a root pixel. The method presented here will try to solve this problem by using a neural network. The idea is to present a backpropagation network with parts of the images and let the network decide if the pixel in the centre of the input image belongs to a living root, to a dead root or to the surrounding (Rumelhart et al., 1986). With this approach it is not necessary to manually define what separates a class of pixel from other classes of pixels, instead the network can be trained on examples (Civco, 2000).

The size of the input to the neural network should be large enough to give the network the data needed to make this decision. By presenting the network with every possible image part of the decided size, every pixel in the image is classified. The only pixels that do not get classified are those along the edges of the original images. This is because the network only classifies the pixel in the middle of the input parts. Given that the image parts will be much smaller than the original images the share that does not get classified is small.

If dead and living roots are assigned one colour each and the output from the network is fed to an image of the same size as the original image, the described process will produce an image
where the roots will appear in their assigned colours. Thus this first step can be seen as a method to transform the image analysis problem to a problem suitable for applying a threshold.

4.2 Applying a threshold

The output from the first step is images with colours reflecting the network’s confidence in its classification of the pixels as dead or living roots. In these images the colour-values of the individual pixels will range from the minimum to the maximum, i.e. from 0 to 255. In order to extract, measure and evaluate the object the next step is to extract the potential root objects to distinct objects. All pixel with colour-value above the threshold-value will be set to have the maximum colour-value, all other pixels will be set to the minimum.

The output from the second step can be seen as two images from each of the original image. One image where potential living roots appear as black objects and one with potential dead roots as black objects.

4.3 Measuring the identified objects

The third step intends to measure the shape of the potential roots produced in the earlier steps. Both the length and the width of the objects should be measured. The length of the objects is, as in the Vamerali et al. (1999) method, the length of the object’s skeleton. The width of the objects can be measured by computing how long a line can be drawn within the object if the line cross the skeleton by 90°. Since the width of the object vary it is necessary to sample the width of the object at several points, see Figure 3.

![Figure 3 Measuring objects. Grey colour represents a root. The black lines are the measurements of the object’s width. The length of the object is derived from the number of lines. Notice that a root with a branch is measured as two objects.](image-url)
4.4 Filtering out non-root objects

The fourth and final step is to use the measurements from step three to filter out irrelevant objects. As in the method used by Vamerali et al. (1999), objects that are too short should be filtered out. In addition the intention is to use the measurements of width. Too thick objects should be filtered out. It is also possible to filter out objects that vary too much in width. Thus, the final output is a set of measurements of objects classified as root. In this particular project the desired final output is an estimation of the amount of dead and living root mass in the soil. This is easily calculated from the length and average width of the objects classified as roots.
5 Implementation

In order to have an intuitive view of the progress of the system the decision was taken that all input to the system, and consequently all output from intermediate steps, should be in the form of images. All parts of the system was coded in C++, see Appendix C. The code was compiled with a SUN sparc compiler using the command “CC” and executed on a sparc sun4u.

5.1 Network input

For training and testing the neural network system uses two images in parallel. One of these images is the unmanipulated image from which the input to the network is extracted. The second image is a copy of the first on which the roots has been manually marked with red for living roots and green for dead. From this second image the desired output for the network is extracted. Thus, the correct classification for the pixel that the network shall classify is found by looking at the same coordinates in the second image.

From the set of all images, 9 images were selected as train and test images on the basis that they contained objects of different shape. Copies of the selected images were made. On these copies the root objects were marked with green for dead and red for living roots. From each of these marked images a small part containing roots typical for that image was cut out and saved as a different image, and the same part in the unmarked images was treated in the same manner. This produced 36 images, two sets with 9 images each for training and two sets with 9 images each for testing. Since the network is intended to classify individual pixels a large number of train- and test-cases can be extracted from these images.

To further increase the number of input-cases a mechanism to vertically flip half of the inputs was implemented. The result of this is a huge number of input-cases, too large to process within reasonable time. In addition, the distribution of the different classes in these input-cases is highly biased since there are far more soil pixels than there are root pixels. Thus a method to filter out the bulk of the input cases was implemented. This method counts the number of pixels in each class that has been fed to the network. Since pixels of dead roots is least common every such pixel in the training set is fed to the network. Pixels of living roots is only feed to the network if the number of living pixels already trained on do not exceed the number of dead pixels already trained on. Pixels of soil are only trained on if the combined number of dead and living pixels already trained on do not exceed the number of soil pixels trained on. The consequence of this filtering is that all dead root pixels and most living root pixels in the training set is used as input to the network. For soil pixels, only a small fraction of the available pixels are trained on. Since the system processes the image sequentially by going from left to right, bottom to top, the soil pixels that are trained on lies just to the right of the roots. To get a more representative distribution of soil pixels to train on the filter for soil pixels also uses a random function refusing 9 of 10 inputs. This spreads the soil inputs over an area about ten times as wide as the root. For testing every root pixel is tested, while a filtering mechanism ensures that soil pixels are only tested if the number of tested soil pixels does not exceed the combined number of root pixels. Using this methods the network in each epoch is trained on 60 667 pixels, 15 166 dead, 15 167 living and 30 334 soil and tested on 22 791 pixels, 3 714 dead, 7 681 living and 11 396 soil. Due to the vertical flip of the inputs the total
number of possible inputs for root pixels is twice the number presented above. For soil pixels
the random selection of pixels to train makes the total number of possible training inputs 20
times as big.

The input to the network contains the colour data for red, green and blue of a 29*29 pixel
window. This size is chosen since it enables a whole root to be contained in the window. The
data is converted to a real number ranging from 0 to 1 and these numbers are normalised. In
addition the maximum and minimum value, before normalisation, is fed to the network. This
combined gives 2 525 input nodes.

5.2 Network set-up

The network in this project is based on a backpropagation network originally written by
Karsten Kutza and designed to predict sunspots, see appendix C. This code was changed in
many ways and a large number of settings were tested. As described above the input layer has
2 525 nodes while the output layer has two. The first node in the output layer intends to
classify the pixel as root or soil, giving the output 1 for roots and the output 0 for soil. The
second node in the output layer intends to classify roots into dead or living, giving the output
1 for dead roots and 0 for living, if the pixels is a soil pixels the target for the second node is
0.5.

Since the network’s task is to classify the inputs into discrete classes the exact output is not
important as long as it falls into the correct class, i.e. above 0.5 if the target is 1. Thus a
specialised training was implemented that only backpropagates the error if the absolute error
is more than 0.25 in any of the output-nodes. The benefit of this method is not only to speed
up the process but also to avoid the network’s tendency to favour one specific class.

In every epoch the network is trained on 60 667 pixels and then tested on 22 790 pixels. The
number of epochs the network is trained is not defined in advance, instead the training is
terminated when the network stops improving. To implement this the network-error for the
test cases is measured after every epoch. The measurement selected for this task is the number
of misclassifications in node one, the node that classifies into root and soil. After every epoch
this misclassification ratio is measured. If the error has decreased the weights are saved and a
new epoch is initiated, if the error has not decreased the training is terminated without saving
the weights. Thus, after training the best weight settings found are always the last-but-one and
it is this setting that is saved to be used for classification.

After the training is terminated the network is used to analyse the nine test images. The output
of this usage is fed to a new image of the same size. On this new image the degree of red
shows the network’s confidence in classifying the individual pixels as living root and green as
dead root. In addition, the blue channel is used for the network’s classification as any root.
Thus, the value for each pixel is calculated as

\[
\text{red} = 255 \times \text{output}[1] \times (1 - \text{output}[2])
\]
\[
\text{green} = 255 \times \text{output}[1] \times \text{output}[2]
\]
\[
\text{blue} = 255 \times \text{output}[1]
\]
5.3 Extracting and evaluating objects

The output from the previous step is a set of images where the network has been used to classify the test-images. On these images the networks confidence in its classification is reflected in the amount of colour in the individual pixels. The darker a pixel is the more confident the network is in classifying that pixel as soil, the lighter the pixel is the more confident the network is in classifying the pixel as a root object. To ease the measurement of the object the images should be thresholded to create solid objects. Since a pixel can have any value from 0 to 255 the middlemost value, 127, was used as threshold. The output images from the previous step are in colour, consisting of red, green and blue. The thresholding is done individually for each of the three colours. Thus for each of the images the thresholding can be seen as transforming three greyscale images into three bitmap images where the pixels either are black or white.

The output from the thresholding can be seen as a set of bitmap images where potential roots are white and everything else is black. The next step is to measure these objects. The intention is to measure the width of the objects at several positions. From this the maximum and average width can be calculated and the length of the objects can be derived from the number of width measurements.

The first step to measure the objects is to find them. If the images are seen as bitmap images where pixels belonging to an object are white and all other pixels are black, objects can be found by traversing the image bottom up, left to right until a white pixel is found. Having found an object the next step is to find a suitable starting point for the measurements. From the found white pixel the longest line possible to draw without leaving the white area is found. Measuring how long it is possible to move in both directions before reaching a black pixel, repeating this 180 times in every angle and then choosing the longest do this. On this longest line the middlemost pixel is chosen as starting point for the measurements.

The width of an object at any position is defined to be the shortest line possible to draw that cross the position and has both its endpoints next to a black pixel, see Figure 3. To stabilise the process it is necessary to restrict the algorithm to only search for lines that do not differ from the previous with more than 45°, thus the algorithm examines the lines in the angles previous - 45 to previous + 45. For the first width measurement the angle of the previous is defined to be the angle of the longest possible line (see above) + 90. Having found the first line, i.e. width measurement, the position where to take the next width measurement has to be found. This is done in two steps. First the centre point on the found line is calculated and then the algorithm takes a three pixel step in a straight angle from the found line, i.e. angle of the found line + 90. By repeating this process the algorithm measure the width of the object at every third pixel along the object’s length. This is repeated until the end of the object, i.e. when the next position to make a measurement from is a black pixel. Having reached one end of the object the process is repeated in the other direction. The starting point for this is second part is calculated by taking the original starting point and going three pixels in the opposite direction, i.e. the angle of the longest line + 270. The process of finding the width at every third pixel is repeated until the second end of the object, by then the whole object is measured.
To avoid measuring the same object over and over again the object has to be filled with another colour than white. To do this a semicircle is painted behind every width measurement, the semicircle has a diameter equal to the width and consequently goes from one edge of the object to the other. The first semicircles lies above the area that the algorithm measures when the first end is reached and the algorithm starts measuring the second part of the object. This would mean that this area is not white, and therefore not considered to belonging to an object. To avoid this the very first semicircle is repainted with white when the first end of the object is reached. This painting of the object is not complete, small areas along the edge of the object remains white. But since these areas are short and thin they are removed in the subsequent filtering. This whole process, finding white pixels, finding starting points and measuring objects are repeated until the whole image is examined, i.e. when the mechanism for finding white pixels reaches the upper right corner.

The objects measured in the step above are all white objects in the thresholded images, some of these objects are proper roots, and some are not. Since proper roots tend to share some properties regarding their shape it is possible to use these properties to filter out non-roots. The properties used here are that roots tend to be long and thin both in absolute and relative values. To enable filtering, three values is calculated for each of the objects: the maximum width, the average width and the length (length=number of width measurements * 3). Based on these values four criteria for filtering was used to filter out non-root objects. The exact settings for these filters were decided experimentally. To be classified as a root the object has to comply with all of the following criteria:

- length >= 15 pixels
- average width <= 30 pixels
- maximum width <= 50 pixels
- length >= 5*average width

The final output after this last step is printed out giving the measurements of the filtered objects, their length, width sampling, maximum width and average width. Since the specific aspect that is of most importance to the biologist intended to use the system is the proportion of root mass in the soil the volume of every root is estimated. This root volume is calculated based on the simplification that the roots are of cylinder shape so that the volume can be estimated from the length and average width.
6 Results

In this chapter the results of using the implemented system is described. The main focus in this chapter is the results of the neural network and of the system as a whole.

6.1 Network performance

In this project a large number of network settings was evaluated. The network’s performance was evaluated based on the misclassification ratio in node one; i.e. the share of pixels in the test set that is misclassified into the classes soil and root. This ratio was calculated separately for the three classes: soil, dead roots and living roots and the average of these ratios was used to compare different network settings. This metric was also used as to decide when to terminate the training as described in section 5.2.

<table>
<thead>
<tr>
<th>Hidden layer 2</th>
<th>Hidden layer 3</th>
<th>Learning rate</th>
<th>Momentum</th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>50</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>600</td>
<td>50</td>
<td>0.02</td>
<td>0.0</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>0.05</td>
<td>0.0</td>
</tr>
<tr>
<td>400</td>
<td>50</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>400</td>
<td>50</td>
<td>0.05</td>
<td>0.0</td>
</tr>
<tr>
<td>300</td>
<td>50</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>200</td>
<td>50</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>400</td>
<td>50</td>
<td>0.1</td>
<td>0.5</td>
</tr>
<tr>
<td>500</td>
<td>-</td>
<td>0.05</td>
<td>0.0</td>
</tr>
<tr>
<td>400</td>
<td>50</td>
<td>0.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 1. Tested network settings.
For all tested networks the input layer consisted of 2,525 nodes and the output layer of 2 nodes. The settings in the lowest row of the table gave the best results and was consequently used to analyse the images.

The network that performs best of the tested consists of four layers, see Table 1. As described above the number of nodes in the input layer is 2,525 and in the output layer 2. The first hidden layer has 400 nodes and the second hidden layer has 50. The network was trained with a learning rate of 0.1. The use of momentum decreased the performance so momentum was set to 0. The performance of this network peaked already in the first epoch. The misclassification ratio in the first node for this network is 0.2165 for dead roots, 0.2106 for living and 0.2102 for soil. This means that nearly 80% of the pixels are correctly classified into soil and roots. Compared to the classification in root and non-root pixels the classification of roots into dead and living give less good results. The misclassification ratio in the second output node is 0.3735 for dead roots and 0.5024 for living, see Table 2.
<table>
<thead>
<tr>
<th>Pixel class</th>
<th>Misclassification node one</th>
<th>Misclassification node two</th>
<th>Average error node one</th>
<th>Average error node two</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead roots</td>
<td>0.2165</td>
<td>0.3735</td>
<td>0.2567</td>
<td>0.4650</td>
</tr>
<tr>
<td>Living roots</td>
<td>0.2106</td>
<td>0.5024</td>
<td>0.2205</td>
<td>0.4793</td>
</tr>
<tr>
<td>Soil</td>
<td>0.2102</td>
<td>-</td>
<td>0.3527</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2. Network performance.
Node one intends to classify pixels into the classes root and soil. Node two intends to classify root pixels into dead and living roots.
The misclassification is the share of the test pixels that is incorrectly classified by the node. Average error is the average absolute difference between desired and actual output.

After training the network is put to use on the test images. The output of the network for each pixel is transformed into colour data in the new images. The amount of green in each pixel represents the networks confidence in classifying the pixels as a dead root, the amount of red in the pixel represent the classification as a living root and the amount of blue as any root, see Appendix B. The images created from the network’s output clearly shows the network’s inability to distinguish between dead and living roots. Especially the green channel, representing dead roots give strange results with strong output for the edge of roots but low output at the core of objects, see Figure 4. Thus, the decision was taken to concentrate on the blue channel, representing roots of any kind. Consequently, the later steps of the method; thresholding, measurement and filtering, are only conducted on the blue channel and do not discriminate between dead and living roots.

Figure 4. Network output.
The first part of the image is the input image containing a living root stretching diagonally over the image and a round non-root object above. Next the network output for living roots, dead roots and any roots is shown.
6.2 Extracting and evaluating object

Due to the decision to not distinguish between dead and living roots the output from the network-part of the system can be seen as nine greyscale images. The pixel-value of the individual pixel, i.e. their brightness, represents the network’s confidence in classifying that pixel as belonging to a root. The next step of the method is to apply a threshold to the images. The perfect output for node one is 0 when classifying a soil pixel and 1 when classifying a root pixel, represented as 0 and 255, respectively, on the output images. Thus, the middlemost value, 127, was chosen as threshold value. Any pixel with a lower pixel value is set to black, every other pixel is set to white.

The result of applying the threshold can be seen in Figure 5 as the difference between the second and third section of the images. All white objects in the third sections of the images are potential root objects, some are proper roots and some are misclassifications. The next step is to measure these objects as described in section 5.3. When the objects are measured the system gets a large numbers of arrays describing the object as a number of width measurements where the length of the objects can be derived from the number of widths. The last step of the method presented here is to filter these arrays to remove objects whose shape does not resemble that of a root. The arrays that remains after filtering are presented to the screen. The data presented are x and y giving the starting point of the object, where the co-ordinates 0, 0 is the lower right corner. Next a number of measurements of the object is presented: length, maximum width, average width and an estimation of the volume of the object. And finally the width measurements are printed. The units in these measurements are number of pixels or cubic pixels. For the image 960905-15 the printout is:

Vector 74 14 62 12 8.225806 3294.871973
4 5 7 8 10 10 9 10 10 9 10 11 12 12 9 8 8 11 11 9 7 5 4 4 6 8 10 10 7 1
Vector 127 41 56 14 9.750000 4181.067123
12 12 12 13 14 14 13 13 11 10 10 10 10 8 7 6 7 5 4 4 3 12 12 11 12 13 9 6

As described in section 5.3 the algorithm that measures the object fills the object it measures by painting semicircles behind every width measurement. The implementation of the algorithm changes the colour of the semicircles for every new object giving every object its unique colour. With the help of these unique colours and the co-ordinates printed to the screen it is possible to manually edit the output images from the system so that only the objects that
the system consider to be roots remains. This editing is not a part of the method; it is only done in order to evaluate the results. Since the filling of the object only is intended to prevent the same object from being measured over and over again the shape of the filled object is not exactly the same as that of the original object. This difference in shape can be seen when comparing the third and fourth sections of the image in Figure 5. The third section is the output after threshold and the fourth is the final, manually edited, image representing the object the system consider to be roots.

The imperfection in the filling of the objects must be kept in mind when analysing the final output of the system. The actual results, presented as the screen-output above, are somewhat better than what the images in Figure 5 and Appendix A imply. The images of the final output can also be used to measure the results on a pixel-by-pixel level. By comparing each individual pixel of the images used as target for the network and the final output images the share of pixels that is correctly classified can be estimated, these results are presented in Table 3.

<table>
<thead>
<tr>
<th>Kind of pixel</th>
<th>Number of pixels</th>
<th>Ratio of correctly classified</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>244 256</td>
<td>0.9083</td>
</tr>
<tr>
<td>Soil</td>
<td>233 069</td>
<td>0.9197</td>
</tr>
<tr>
<td>Root</td>
<td>11 187</td>
<td>0.6691</td>
</tr>
<tr>
<td>Dead root</td>
<td>3 673</td>
<td>0.5889</td>
</tr>
<tr>
<td>Living root</td>
<td>7 514</td>
<td>0.7084</td>
</tr>
</tbody>
</table>

Table 3 final results. 
Pixel-by-pixel comparison of the target classification and the images representing the final output.
7 Analysis and conclusion

Given the nature of the images in this task, no system can perfectly analyse them. The high resolution and the imperfect sharpness of the images make it impossible to exactly pinpoint the border of root objects. And even if the resolution would be lower and the sharpness perfect there would be borderline cases with pixels lying just over the border between root and soil. Also the distinction between living and dead roots have borderline cases. As a root dies it slowly change its appearance from that of a living root to that of a dead. Thus, no system can exactly classify a root as dead or living. This lack of perfection must be kept in mind when evaluating the system designed in this project. The results presented above compare the system output to a target output. This target output is derived from a set of images where the root objects are marked manually. These images can not be considered the absolute truth, the exact border of the marked objects is an approximation and there might be misclassification of entire objects. However, the target images are created in co-operation with biological expertise and are exact enough for the purpose here.

The results can be divided into two categories, the system’s ability to find roots of any kind and secondly the system’s ability to classify these roots into dead and living. The results for the first category are relatively good. The network’s ability to distinguish root pixels from soil pixels reaches almost 80% correct classification. This is far beyond pure random and in the images of network output the roots are generally visible as lighter objects, see Figure 5. The later filtering step of the method can not identify root objects not found by the network but it can remove non-root object. Also this step is relatively accurate. The filtering is done perfectly in image 960905-15 and 961007-22. For other images, such as image 960813-15, both root objects and non-root objects are removed.

While the system performs relatively well on the task of identifying the root object, the performance concerning classification of roots into dead and living works less well. The network is able to correctly classify 63% of the dead pixels and 49% of the living. Of all root pixels 54% are correctly classified into dead and living so the network seems to have some ability to distinguish between dead and living. But the margin over pure random is just 4 percentage points and this is far below what necessary for a useful system. Consequently, the later steps of the method concentrated on root objects of any kind, rather than to separate them into dead and living.

These differences in results between finding and classifying roots are not surprising. A human inspector has relatively small problems finding the roots but to tell if they are dead or living is much harder. Even experts have problems making the distinction.

In Chapter 3.2.4, Test of performance, two measurements were defined. The first was whether the system performed above pure random and the second was if the system performed good enough to be useful. For the task of classifying the roots into dead and living the system can be considered to perform just, but only just, above random. Given the mediocre results the second measurement of usability must be considered not fulfilled. For the task of identifying roots of any kind the results are better. The system performs far better than pure random so the first measurement is passed with good margin. The measurement of usefulness must be
considered with respect to alternative methods, in this case manual analysis of the images. This manual inspection is not only very expensive but the accuracy of the analysis is not perfect. This imperfection comes from problem associated with analysis of the images, described in the first paragraph of this chapter. In addition, the field study that the images in this task stems from produced 20,736 images. Human analysis of such a huge number of images is bound to encounter human errors. Keeping this in mind the task of finding and measuring roots can be considered to be fulfilled since the department of soil sciences at the Swedish university of agriculture science intends to use the system designed in this project to analyse the images.
8 Discussion and future work

The system implemented in this project is intended to test the principles of the method. It is likely that far better performance can be reached if the system is properly fine-tuned. In particular, the neural network step can probably be further improved. The architecture and settings for the network can be further improved. Perhaps backpropagation is not the best kind of network paradigm for this task. Since the network is intended to classify high dimensional data it would be interesting to use a self-organising map (Kohonen, 1988). The performance of the network can perhaps be improved by pre-processing the input. The current system uses RGB-colours as input. Another encoding of colour images is HSB that represent colour using its hue, saturation and brightness. Compared to RGB, HSB-encoding is closer to how humans perceive colours and perhaps the network will find such encoding easier to interpret (Russ, 1992). Also the choice of training inputs can be improved. Since there are far more soil pixels than root pixels the system selects a subset of the soil pixels randomly. By inspecting the output image from the network it might be possible to find types of soil pixels that the system tends to misclassify. With this information it would be possible to make a more informed selection of soil pixels to train.

Since the current system only intends to test the method some issues has to be addressed in order to transform it into a proper application. It must be equipped with a user interface allowing easy use of the system. In addition, the current system would take approximately one hour to analyse one complete image; this time should be decreased. As any program this can be achieved by optimising the code. Another way to speed up the system is to decrease the resolution of the images. If the images are transformed such that each square of four pixels are combined into one pixel the network input is decreased with a factor four, assuming that the input cover the same area in the image. The impact on time is obvious, the impact on the quality of classification is harder to predict. The quality might decrease since the network simply gets less data. But the quality might increase since the network gets a less noisy, less chaotic data.

The system implemented in this project is intended to measure roots. However, the high level method is not restricted to just roots, it might be possible to use the method for other image analysis projects. The results for classification of roots into dead and living imply that the method has its limitations but there might still be other domains there the method will produce usable results, possibly better than for the roots in this project.
References


Jensen R. J., Qiu F., Patterson K., A neural network image interpretation system to extract rural and urban land use and land cover information from remote sensor data. Geocarto international vol. 16 No. 1. March 2001.


Appendix A
Images from different steps of the algorithm.
All the images shows first the test image (originally in full RGB-colours), then the networks output for the blue channel, then the result of applying the threshold value 127 and finally the result after filtering. The number refers to the date when the image was taken and the depth (the date in Swedish format: YYMMDD).

960703-27, living root.
The system gives relatively good results in this case. Only a small non-root object remains after the final step.

960813-15, dead root
The input image contains one root going vertical in the right side of the image. The final image shows that the system only recognise parts of the proper image together with some non-root objects.

960905-15, dead root
This is the same root as in the image above but the picture is take almost one month later. The system performs very well in this case.
960905-26, living root
The root goes diagonal across over the image, the round object above is not a root. Apart from a small object at the bottom of the image the system is able to find the correct object.

961007-11, dead root
In the input a small root is going horizontal over the image. The system is not able to identify the root.

970321-09, living root
In this case a root is going diagonal in the right part of the input. The system identifies that root together with some none-root objects.

970321-27, living roots
The input shows two thin roots in the right part. The system works relatively porly on this image, identifying parts of the roots together with several non-root objects.
961007-07, living root

The input contains one young root with root-hairs going vertical over the image and a black stone. The systems performance can only be considered as a complete failure on all accounts.
961007-22, dead roots

The input contains one horizontal root in the upper part of the image and one vertical in the lower part. The output from the network recognises these roots together with a large number of non-root objects. After filtering only the correct objects remain.
Appendix B
Histograms

Histograms showing the colour-distribution in the images produced by the network. X-axis shows the colour-value, y-axis the proportion of pixels. The histograms to the left is calculated on the pixels that should have the colour, the histograms to the right is calculated on the pixels that should not.
Appendix C
C++ Code

main7.cpp for training, evaluating and using the neural network

```cpp
#include "jonas.h"
#include "image.h"
#include "BPbox.h"

FILE *logg;
double alpha=0.0;
double eta=0.1;
double gain=1;

void loggFlush()
{
  fclose(logg);

  if ((logg=fopen("logg.txt","a"))==NULL)
  {
    printf("\nloggFlush the logg file is impossible to open \n");
    exit (1);
  }
}

void trainDriver(BPbox &net, Arr<imageHandler> &trainFa, Arr<imageHandler> &trainIn, ulong ix, ulong iy)
{
ulong i, xe, ye, fdead=0, fliving=0, fsoil=0, x, y, turns=0;
imageHandler ipart;
double dead, living;
double dr=0, dk=0, lr=0, lk=0, sr=0, sk=0; //Diffs, exept the (irrelevant) sk that is average
Arr<double> input;
Arr<double> output;
Arr<double> facit(2);
int flipp;

for (i=0; i<trainFa.gelangd(); i++)
{
  if (! (( (trainFa[i].getWidth()==trainIn[i].getWidth()) &&
            (trainFa[i].getHeigth()==trainIn[i].getHeigth()) )))
    {printf("bonusDriver()
the two images in set %d was of different size", i); exit(1);};
  //The outer limits of the input-taking
  xe=trainFa[i].getWidth()-ix;
  ye=trainFa[i].getHeigth()-iy;
  x=0;
  y=0;
  printf("\n");

  while (y<ye+1)
  {
    flipp=rand() % 2;

    if (flipp)
    {
      if (trainFa[i].isColour(x+flippPos, y+iy/2, 0, 255, 0)) dead=1;
      else dead=0;
      if (trainFa[i].isColour(x+flippPos, y+iy/2, 255, 0, 0)) living=1;
      else living=0;
    }
```
else
{
  if (trainFa[i].isColour(x+ix/2, y+iy/2, 0, 255, 0)) dead=1;
  else dead=0;
  if (trainFa[i].isColour(x+ix/2, y+iy/2, 255, 0, 0)) living=1;
  else living=0;
}

/*This condition is to ensure that, in the long run, there's equal amount
of the different cases.
The assumption is that soil is more common than living which is more
common than dead.
I've changed it since it got too few soils from living images. */
if ((dead==1) || ((living==1) && (fdead >= fliving)) || (((dead==0) &&
(living==0)) && (fdead+fliving >= fsoil)))
{
  turns=turns+1;
  if (dead==1) fdead=fdead+1;
  else if (living==1) fliving=fliving+1;
  else fsoil=fsoil+1;

  trainIn[i].imagePart(ipart, x, y, ix, iy, 0);
  if (flipp) trainIn[i].VFlipp();
  ipart.toDouble(input);

  //Node one - root or not. Node two what kind of root. Thus: r-node and k-
  node
  if (dead > 0.5) //Deads
  {
    facit[0]=1;
    facit[1]=1;
  }
  else if (living > 0.5) //Living
  {
    facit[0]=1;
    facit[1]=0;
  }
  else //Soils
  {
    facit[0]=0;
    facit[1]=0.5;
  }

  net.train(output, 1, 1, input, facit);

  if (dead > 0.5) {dr=dr + abs(1-output[0]);
  dk=dk + abs(1-output[1]);}
  else if (living > 0.5) {lr=lr + abs(1-output[0]);
  lk=lk + output[1];}
  else {sr=sr + output[0];
  sk=sk + output[1];}

  imageHandler       fbild;
  double             d,l;
  trainFa[i].imagePart(fbild, x, y, ix, iy, 0);
  if (flipp) fbild.VFlipp();
  if (fbild.isColour(ix/2, iy/2, 0, 255, 0)) d=1;
  else d=0;
  if (fbild.isColour(ix/2, iy/2, 255, 0, 0)) l=1;
  else l=0;
  if ((d != dead) || (l != living))
  {
  }
printf("\nNu sket det sig!");

printf("\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b%12lu%5lu%5lu",turns,x,y);
}

x=x+1;
if (x>xe)
{x=0;
y=y+1;
}
}

if (fdead>0) printf("%.5f %.5f ", (double) dr/fdead,(double) dk/fdead);
if (fliving>0)printf("%.5f %.5f ", (double) lr/fliving,(double) lk/fliving);
if (fsoil>0) printf("%.5f %.5f",(double) sr/fsoil,(double) sk/fsoil);

printf("\nIncluding this train-image I traversed %lu dead, %lu living and %lu soils",fdead, fliving, fsoil);
}

printf("\nIn this Trainrun I traversed %lu images with %lu dead, %lu living and %lu soils",trainFa.gelangd(),fdead, fliving, fsoil);
fprintf(logg,"\nIn this Trainrun I traversed %lu images with %lu dead, %lu living and %lu soils",trainFa.gelangd(),fdead, fliving, fsoil);

if (fdead>0)
{
printf("\nThe diffs for deads %.5f %.5f.", (double) dr/fdead,(double) dk/fdead);
fprintf(logg,"\nThe diffs for deads %.5f %.5f.", (double) dr/fdead,(double) dk/fdead);
}
if (fliving>0)
{
printf("\nThe diffs for living %.5f %.5f.", (double) lr/fliving,(double) lk/fliving);
fprintf(logg,"\nThe diffs for living %.5f %.5f.", (double) lr/fliving,(double) lk/fliving);
}
if (fsoil>0)
{
printf("\nThe diffs for soil %.5f %.5f.", (double) sr/fsoil,(double) sk/fsoil);
fprintf(logg,"\nThe diffs for soil %.5f %.5f.", (double) sr/fsoil,(double) sk/fsoil);
}
loggFlush();
}

void testDriver(BPbox &net, Arr<imageHandler> &trainFa, Arr<imageHandler> &trainIn, ulong ix, ulong iy, double &diff)
{ulong i, xe, ye, fdead=0, fliving=0, fsoil=0, x, y, turns=0;
imageHandler ipart;
double dead, living;
double dr=0, dk=0, lr=0, lk=0, sr=0, sk=0; //Diffs, except the 
(irrelevant) sk that is average
Arr<double> input;
Arr<double> output;

for (i=0; i<trainFa.gelangd(); i++)
{
    if (! (( (trainFa[i].getWidth()==trainIn[i].getWidth()) &&
            (trainFa[i].getHeigth()==trainIn[i].getHeigth()) ))) {printf("bonusDriver()
the two images in set %d was of different size", i); exit(1);};
    //The outer limits of the input-taking
    xe=trainFa[i].getWidth()-ix;
    ye=trainFa[i].getHeigth()-iy;
    x=0;
    y=0;
    printf("\n");
    while (y<ye+1)
    {
        if (trainFa[i].isColour(x+ix/2, y+iy/2, 0, 255, 0)) dead=1;
        else dead=0;
        if (trainFa[i].isColour(x+ix/2, y+iy/2, 255, 0, 0)) living=1;
        else living=0;

        //To timeconsuming to test every soil
        if ((living+dead > 0) ||  (fliving+fdead >= fsoil))
        {
            trainIn[i].imagePart(ipart, x, y, ix, iy, 0);
            turns=turns+1;
            if (dead==1) fdead=fdead+1;
            else if (living==1) fliving=fliving+1;
            else fsoil=fsoil+1;

            ipart.toDouble(input);
            net.use(output, 1, input);

            if (dead > 0.5)            {dr=dr + abs(1-output[0]); dk=dk + abs(1-
                                            output[1]);}
            else if (living > 0.5)     {lr=lr + abs(1-output[0]); lk=lk + output[1];}
            else             {sr=sr + output[0];        sk=sk + output[1];}

            printf("%12lu%5lu%5lu",turns, x, y);
        }
        x=x+1;
        if (x>xe)
        {
            x=0;
            y=y+1;
        }
    }

    if (fdead >0 ) printf("   %.5f %.5f  ", (double) dr/fdead,(double)
            dk/fdead);
    if (fliving >0 )printf("%.5f %.5f   ", (double) lr/fliving,(double)
                lk/fliving);
    if (fsoil >0 )  printf("%.5f %.5f",(double) sr/fsoil,(double) sk/fsoil);

    printf("Including this test-image I traversed %lu dead, %lu living and
            %lu soils",fdead, fliving, fsoil);
}

33
printf("\nIn this testrun I traversed %lu images with %lu dead, %lu living and %lu soils", trainFa.gelangd(), fdead, fliving, fsoil);
    fprintf(logg, "\nIn this testrun I traversed %lu images with %lu dead, %lu living and %lu soils", trainFa.gelangd(), fdead, fliving, fsoil);

    if (fdead >0 )
    {
        printf("\nThe diffs for deads %.5f.%.5f.", (double) dr/fdead, (double) dk/fdead);
    
        fprintf(logg,"\nThe diffs for deads %.5f.%.5f.",(double) dr/fdead,(double) dk/fdead);
    }

    if (fliving >0 )
    {
        printf("\nThe diffs for living %.5f.%.5f.",(double) lr/fliving, (double) lk/fliving);

        fprintf(logg,"\nThe diffs for living %.5f.%.5f.",(double) lr/fliving,(double) lk/fliving);
    }

    if (fsoil >0 )
    {
        printf("\nThe diffs for soil %.5f.%.5f.",(double) sr/fsoil,(double) sk/fsoil);

        fprintf(logg,"\nThe diffs for soil %.5f.%.5f.",(double) sr/fsoil,(double) sk/fsoil);
    }

    diff=((double)sr/fsoil+(double)lr/fliving+(double)lk/fliving+(double)dr/fdead+(double)dk/fdead) / (double) 5;

    loggFlush();
}

//This one is highly specific
void imageOpen(int train, Arr<imageHandler> &iSet, Arr<imageHandler> &fSet)
{
    /*
    fe960703-27.bmp  ft960703-27.bmp  ie960703-27.bmp  it960703-27.bmp
    fe961007-07.bmp  ft961007-07.bmp  ie961007-07.bmp  it961007-07.bmp
    fe961007-11.bmp  ft961007-11.bmp  ie961007-11.bmp  it961007-11.bmp
    fe961007-22.bmp  ft961007-22.bmp  ie961007-22.bmp  it961007-22.bmp
    fe970321-09.bmp  ft970321-09.bmp  ie970321-09.bmp  it970321-09.bmp
    fe970321-27.bmp  ft970321-27.bmp  ie970321-27.bmp  it970321-27.bmp
    */

    if (train) //It's train-time
    {
        //The new approach here is to train the smallest last, to avoid
        specialization on living.
        iSet.nylangd(9);
        fSet.nylangd(9);

        iSet[5].reader("pics/it960703-27.bmp");
        iSet[4].reader("pics/it960813-15.bmp"); //Pure size indicate a different
        order between 2 and 4 but, 2 has more roots
        iSet[2].reader("pics/it960905-15.bmp");
    }
void readAndUse()
{
ulong              i, xe, ye, x, y, images=8, ix=30, iy=30;
imageHandler       ipart, inputImage, outputImage;
Arr<double>        input;
Arr<double>        output;
char               *fname;
Arr<int>           nodes;
BPbox        net;
net.read("net.net");
printf("\nStarting to create images");
net.giveNodes(nodes);
printf("\nNet:");
for(ulong j=0; j<nodes.gelangd(); j++)
{
}
printf( " %d", nodes[j]);
}

for (i=0; i<images; i++)
{
    //The filenames are written in stone
    /*
        iSet[5].reader("pics/ie960703-27.bmp");
        iSet[4].reader("pics/ie960813-15.bmp");
        iSet[2].reader("pics/ie960905-15.bmp");
        iSet[8].reader("pics/ie960905-26.bmp");
        iSet[7].reader("pics/ie961007-07.bmp");
        iSet[0].reader("pics/ie961007-11.bmp");
        iSet[6].reader("pics/ie961007-22.bmp");
        iSet[1].reader("pics/ie970321-09.bmp");
        iSet[3].reader("pics/ie970321-27.bmp");
    */
    if (i==0)
    {
        inputImage.reader("pics/ie960703-27.bmp");
        fname="ou960703-27.bmp"
    }
    if (i==1)
    {
        inputImage.reader("pics/ie960813-15.bmp");
        fname="ou960813-15.bmp"
    }
    if (i==2)
    {
        inputImage.reader("pics/ie960905-15.bmp");
        fname="ou960905-15.bmp"
    }
    if (i==3)
    {
        inputImage.reader("pics/ie960905-26.bmp");
        fname="ou960905-26.bmp"
    }
    if (i==4)
    {
        inputImage.reader("pics/ie961007-07.bmp");
        fname="ou961007-07.bmp"
    }
    if (i==5)
    {
        inputImage.reader("pics/ie961007-11.bmp");
        fname="ou961007-11.bmp"
    }
    if (i==6)
    {
        inputImage.reader("pics/ie961007-22.bmp");
        fname="ou961007-22.bmp"
    }
    if (i==7)
    {
        inputImage.reader("pics/ie970321-09.bmp");
        fname="ou970321-09.bmp"
    }
    if (i==8)
    {
        inputImage.reader("pics/ie970321-27.bmp");
    }
fname="ou970321-27.bmp"
}

outputImage.create(inputImage.getWidth(), inputImage.getHeight(), 24);
outputImage.blacken();

// The outer limits of the input-taking
xe=inputImage.getWidth()-ix;
ye=inputImage.getHeight()-iy;
x=0;
y=0;
printf("\nCreating no %lu of %lu \n", i+1, images+1);

while (y<ye+1)
{
    inputImage.imagePart(ipart, x, y, ix, iy, 0);
    ipart.toDouble(input);
    net.use(output, 1, input);

    outputImage.setColour(x+ix/2, y+iy/2, (uchar) (output[1]*255),(uchar)
    (output[0]*255), 0);
    printf("%5lu%5lu", x, y);
    x=x+1;
    if (x>xe)
    {x=0;
y=y+1;
    }
}
outputImage.BMPRGBWriter(fname);
}
}

void TTdriver(BPbox &net)
{
double              diff=10000, diffSave;
Arr<imageHandler>   trainFa, trainIn, testFa, testIn;
ulong               ix=30, iy=30; // Size of the image windows

imageOpen(1, trainIn, trainFa);
imageOpen(0, testIn, testFa);

do
{
diffSave=diff;
trainDriver(net, trainFa, trainIn, ix, iy);
net.save("tilf.net"); // If its terminated during testing, at least this
gets saved

testDriver(net, testFa, testIn, ix, iy, diff);
printf(" The average diff is %.5f.", diff);
fprintf(logg," The average diff is %.5f.", diff);
if (diff <= diff)
{ /* This is what to do if the nets are better than ever:
net.save("net.net"); */
}
while ( (diff <= diffSave) && (diff > 0)); // Shall stop when diff stopps
decreasing

if (! (diff <= diffSave))
{...}
void createAndTrain()
{
printf("\ncreateAndRun() started\n");
Arr<int>            nodes(4);
ulong               ix=30, iy=30; //Size of the image windows

    nodes[0]=ix*iy*3;
nodes[1]=400;
nodes[2]=50;
nodes[3]=2; //First output is for dead roots, second for living

    BPbox        net(nodes.gelangd(),nodes);
    net.set(alpha, eta, gain);

    fprintf(logg,"\n\n*** New Run *** starting from scratch\n");

    fprintf(logg,"\n\nNet:\n");
    printf("\nNet:\n");
    for(ulong j=0; j<nodes.gelangd(); j++)
    {
        fprintf(logg," %d", nodes[j]);
        printf(" %d", nodes[j]);
    }
    loggFlush();

    TTdriver(net);
}

void readAndTrain()
{
printf("\nreadAndTrain() started\n");
Arr<int>            nodes;
Arr<imageHandler>   trainFa, trainIn, testFa, testIn;
ulong               ix=30, iy=30; //Size of the image windows

    BPbox        net;
    net.read("net.net");
    net.set(alpha, eta, gain);

    fprintf(logg,"\n\n*** New Run *** starting from file\n");

    net.giveNodes(nodes);
    fprintf(logg,"\n\nNet:\n");
    printf("\n\nNet:\n");
    for(ulong j=0; j<nodes.gelangd(); j++)
    {
        fprintf(logg," %d", nodes[j]);
        printf(" %d", nodes[j]);
    }
    loggFlush();

    TTdriver(net);
}
void main()
{
printf("\nStartt 2");

if ((logg=fopen("logg.txt","a"))==NULL)
{
printf("\nmain(): the logg file is impossible to open \n");
exit (1);
}

createAndTrain();
// readAndTrain();
// readAndUse();

printf("\nSlutt\n");
}
#include "jonas.h"
#include "image.h"

void compTresh(uchar &greenT, uchar &redT, uchar &blueT)
{
    imageHandler fImage, oImage;
    ulong images=9, x, y, i, j, sum, pixels=0;
    Arr<ulong> dh(256), lh(256), gh(256), rh(256), oh(256), bh(256); //Histograms. Absolute
    Arr<double> ddh(256), dlh(256), dgh(256), drh(256), doh(256), dbh(256); //Histograms. Proportional
    double or, ow, sr, sw, tor, tow, tsr, tsw;
    //Confusion matrix. The one with t's are temporary
    FILE *fil;

    redT=0;    //Tresholds, if they remain 0; something is wrong
    greenT=0;

    dh.nolla();
    lh.nolla();
    gh.nolla();
    rh.nolla();
    oh.nolla();
    bh.nolla();

    for(i=0; i<images; i++)
    {
        if (i==0)
        {
            fImage.reader("pics/fe960703-27.bmp");
            oImage.reader("ou960703-27.bmp");
        }
        if (i==1)
        {
            fImage.reader("pics/fe960813-15.bmp");
            oImage.reader("ou960813-15.bmp");
        }
        if (i==2)
        {
            fImage.reader("pics/fe960905-15.bmp");
            oImage.reader("ou960905-15.bmp");
        }
        if (i==3)
        {
            fImage.reader("pics/fe960905-26.bmp");
            oImage.reader("ou960905-26.bmp");
        }
        if (i==4)
        {
            fImage.reader("pics/fe961007-07.bmp");
            oImage.reader("ou961007-07.bmp");
        }
        if (i==5)
        {
            fImage.reader("pics/fe961007-11.bmp");
            oImage.reader("ou961007-11.bmp");
        }
        if (i==6)
fImage.reader("pics/fe961007-22.bmp");
oImage.reader("ou961007-22.bmp");
}
if (i==7)
{
    fImage.reader("pics/fe970321-09.bmp");
oImage.reader("ou970321-09.bmp");
}
if (i==8)
{
    fImage.reader(" pics/fe970321-27.bmp");
oImage.reader("ou970321-27.bmp");
}
if (!((fImage.getWidth()==oImage.getWidth())&&
(fImage.getHeigth()==oImage.getHeigth())))
    {printf("bonusDriver() the two images in set %d was of different size", i); exit(1);};
printf("\nMeasuring no %lu of %lu ", i+1, images);
x=15;//Skall kanske va 14, se även nere
y=15;
while (y < fImage.getHeigth()-15)
{
    if (fImage.isColour(x, y, 255, 0, 0)) //Red-Living
    {
        lh[oImage.getColour(x, y, 'r')]++;
        gh[oImage.getColour(x, y, 'g')]++;
        oh[oImage.getColour(x, y, 'b')]++;
    } else if (fImage.isColour(x, y, 0, 255, 0)) //Green-Dead
    {
        dh[oImage.getColour(x, y, 'g')]++;
        rh[oImage.getColour(x, y, 'r')]++;
        oh[oImage.getColour(x, y, 'b')]++;
    } else  //Soil
    {
        gh[oImage.getColour(x, y, 'g')]++;
        rh[oImage.getColour(x, y, 'r')]++;
        bh[oImage.getColour(x, y, 'b')]++;
    }
    pixels++; x=x+1;
    if (x > fImage.getWidth())
    {x=0;
        y=y+1;
    }
}
printf("\nI have seen %lu pixels",pixels);
sum=dh.uSum();
for(j=0; j<256; j++) ddh[j]=(double)dh[j]/sum;
sum=1h.uSum();
for(j=0; j<256; j++) dlh[j]=(double)1h[j]/sum;
sum=gh.uSum();
for(j=0; j<256; j++) dgh[j]=(double)gh[j]/sum;
sum=rh.uSum();
for(j=0; j<256; j++) drh[j]=(double)rh[j]/sum;
sum=oh.uSum();
for(j=0; j<256; j++) doh[j]=(double)oh[j]/sum;
sum=bh.uSum();
for(j=0; j<256; j++) dbh[j]=(double)bh[j]/sum;

if ((fil=fopen("histograms.txt","w"))==NULL)
{printf("%s%s%s
","ncomputeTresh: histogram.txt impossible to open");
exit (1);}
for(j=0; j<256; j++) fprintf(fil, "%f ",dbh[j]); fprintf(fil, 
);
for(j=0; j<256; j++) fprintf(fil, "%f ",dhh[j]); fprintf(fil, 
);
for(j=0; j<256; j++) fprintf(fil, "%f ",dgh[j]); fprintf(fil, 
);
for(j=0; j<256; j++) fprintf(fil, "%f ",drh[j]); fprintf(fil, 
);
for(j=0; j<256; j++) fprintf(fil, "%f ",doh[j]); fprintf(fil, 
);
fclose(fil);

//Red
or=0;
sw=1;
sr=0;
ow=1;
for (i=1; i<256; i++)
{
 tor=0;
tow=0;
tsor=0;
tsw=0;
for(j=0; j<i; j++)
{
 tow=tow+dlh[j];
tsor=tsor+drh[j];
}
for(j=j; j<256; j++)
{
 tor=tor+dlh[j];
tsw=tsw+drh[j];
}
if (tor+tsor-tow-tsw > or+sr-ow-sw)
{
 or=tor;
 ow=tow;
 sr=tsor;
 sw=tsw;
 redT=(uchar)i;
}
printf("%f %f %f %f %f %d", or, sr, ow, sw, (or+sr-ow-sw)/(double)2, redT);

//Green
or=0;
sw=1;
sr=0;
ow=1;
for (i=1; i<256; i++)
{
 tor=0;
tow=0;
tsor=0;
tsw=0;
for(j=0; j<i; j++)
```c
{ 
tow=tow+ddh[j];
  tsr=tsr+dgh[j];
} for(j=j; j<256; j++)
{ 
  tor=tor+ddh[j];
  tsw=tsw+dgh[j];
} if (tor+tsr-tow-tsw > or+sr-ow-sw)
{
  or=tor;
  ow=tow;
  sr=tsr;
  sw=tsw;
  greenT=(uchar)i;
}
printf("\nGreen %.5f %.5f %.5f %.5f %.5f %d", or, sr, ow, sw, (or+sr-ow-sw)/(double)2, greenT);

//Blue
or=0;
ow=1;
sr=0;
sw=1;
for (i=1; i<256; i++)
{
  tor=0;
  tow=0;
  tsr=0;
  tsw=0;
  for(j=0; j<i; j++)
  { 
    tow=tow+doh[j];
    tsr=tsr+dbh[j];
  } for(j=j; j<256; j++)
  { 
    tor=tor+doh[j];
    tsw=tsw+dbh[j];
  } if (tor+tsr-tow-tsw > or+sr-ow-sw)
  {
    or=tor;
    ow=tow;
    sr=tsr;
    sw=tsw;
    blueT=(uchar)i;
  }
}
printf("\nBlue %.5f %.5f %.5f %.5f %.5f %d", or, sr, ow, sw, (or+sr-ow-sw)/(double)2, blueT);

if(greenT * redT *blueT == 0) {printf("\ngreenT or redT was 0, and that's wrong."); exit(1);}
}
void applyTresh(uchar greenT, uchar redT, uchar blueT)
{ulong i,x, y, images=9;
  imageHandler inputImage, outputImage;
  char *fname;
  uchar red, green, blue;
```
for (i=0; i<images; i++)
{
  if (i==0)
  {
    inputImage.reader("ou960703-27.bmp");
    fname="tr960703-27.bmp";
  }
  if (i==1)
  {
    inputImage.reader("ou960813-15.bmp");
    fname="tr960813-15.bmp";
  }
  if (i==2)
  {
    inputImage.reader("ou960905-15.bmp");
    fname="tr960905-15.bmp";
  }
  if (i==3)
  {
    inputImage.reader("ou960905-26.bmp");
    fname="tr960905-26.bmp";
  }
  if (i==4)
  {
    inputImage.reader("ou961007-07.bmp");
    fname="tr961007-07.bmp";
  }
  if (i==5)
  {
    inputImage.reader("ou961007-11.bmp");
    fname="tr961007-11.bmp";
  }
  if (i==6)
  {
    inputImage.reader("ou961007-22.bmp");
    fname="tr961007-22.bmp";
  }
  if (i==7)
  {
    inputImage.reader("ou970321-09.bmp");
    fname="tr970321-09.bmp";
  }
  if (i==8)
  {
    inputImage.reader("ou970321-27.bmp");
    fname="tr970321-27.bmp";
  }
}

outputImage.create(inputImage.getWidth(), inputImage.getHeight(), 24);

x=0;
y=0;
printf("\nTrashing no %lu of %lu ", i+1, images+1);

while (y<inputImage.getHeight()-1)
{
  if (inputImage.getColour(x, y, 'r') >= redT) red=255;
    else red=0;

  if (inputImage.getColour(x, y, 'g') >= greenT) green=255;

  x=x+1;
  if (x==inputImage.getWidth()) x=0; y=y+1;

  if (x==outputImage.getWidth())}
}
else green=0;

if (inputImage.getColour(x, y, 'b') >= blueT) blue=255;
else blue=0;

outputImage.setColour(x, y, red, green, blue);

x=x+1;
if (x>inputImage.getWidth())
{x=0;
 y=y+1;
}
outputImage.BMPRGBWriter(fname);
}

void main()
{printf("nStartt");

applyTresh(255, 255, 127);

printf("nSlutt\n");
}

*/
void main()
{printf("nStartt");
 uchar            greenT, redT, blueT;

compTresh(greenT, redT, blueT);
printf("n%d %d %d", greenT, redT, blueT);

applyTresh(255, 255, 127);
applyTresh(greenT, redT, blueT);

printf("nSlutt\n");
}*/
tre.cpp used for extracting and measuring the thresholded images

#include "jonas.h"
#include "image.h"
#include "vector.h"

FILE *fil;

void colInc(uchar &r, uchar &g, uchar &b)
{
  if ((r==255) && (g==255) && (b==255)) {r=0; g=0; b=0;}
  else if ((g==255) && (b==255)) {r=r+1; g=0; b=0;}
  else if (b==255) {g=g+1; b=0;}
  else b=b+1;
}

void findStart(imageHandler &image, int &angle, ulong &length, ulong &uX, ulong &uY, ulong iX, ulong iY, uchar r, uchar g, uchar b)
{
  double dx=(double) iX, dy=(double) iY;

  angle=image.longestRadie(iX, iY, r, g, b);
  length=image.goWhile(angle, (double) iX, (double) iY, r, g, b, r, g, b);
  //image.line(angle, iX, iY, length, 255, 0, 0);
  image.nextCoor(dx, dy, angle, length/2);
  uX=(ulong) dx;
  uY=(ulong) dy;
  angle=(angle+90) % 360;
  //image.setColour( uX, uY, 0, 255, 0);
}

void traceObject(imageHandler &image, ulong iX, ulong iY, int angle, int &eAngle, ulong &eLength, ulong &eX, ulong &eY, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2, Vector &vec)
{
  ulong length=10;
  double x=(double) iX, y=(double) iY, sX, sY;

  eLength=0;

  while ((length>0) && (x>0) && (ulong)x<image.getWidth()) && (y>0) && ((ulong)y<image.getHeigth()) && (image.isColour(round(x), round(y), r1, g1, b1))
  {
    sX=x;
    sY=y;
    image.shortestLine(angle, length, x, y, sX, sY, r1, g1, b1, r2, g2, b2);
    if (length > 0) vec.setCoor((ulong) x,(ulong) y);
    vec.addWidth(length);
    if (eLength==0)
    {
      eLength=length; //These remember, and return, the first finding
      eAngle=angle;
      eX=(ulong) x;
      eY=(ulong) y;
    }
    image.nextCoor(x, y, (angle+270) % 360, 2);
  }
}

void findObjects(imageHandler &image, uchar r1, uchar g1, uchar b1)
{  ulong x=0, y=0, oX, oY, eLength, eX, eY;  int angle, eAngle;  uchar r2=255, g2=0, b2=0;  double dX, dY;
   Vector vec;  Arr<Vector> root;

   image.findColour(x, y, r1, g1, b1);

   while (x+y > 0)
   {
     findStart(image, angle, eLength, oX, oY, x, y, r1, g1, b1);
     if (eLength > 10)
     {
       traceObject(image, oX, oY, angle, eAngle, eLength, eX, eY, r1, g1, b1, r2, g2, b2, vec);
       image.pieIf(eAngle, 180, eX, eY, eLength, r2, g2, b2, r1, g1, b1);
       dX=(double) eX; dY=(double) eY;
       image.nextCoor(dX, dY, (eAngle+90) % 360, 3);
       eX=(ulong) dX; eY=(ulong) dY;
       traceObject(image, eX, eY, (eAngle+180) % 360, eAngle, eLength, eX, eY, r1, g1, b1, r2, g2, b2, vec);
       if (vec.areYouRoot())
       {
         root[vec.gelangd()]=vec;
         root[vec.gelangd()-1]=vec;
       }

       vec.clear();
     }
   }

   for(x=0; x<root.gelangd(); x++)
   {
     root[x].write();
     root[x].fwrite(fil);
   }

   void vectorize(imageHandler &image, uchar r1, uchar g1, uchar b1)
   {
     findObjects(image, r1, g1, b1);
   }

   void driver(char *fname)
   {
     imageHandler image;
     printf("\n");
     printf(fname);
     fprintf(fil, "\n");
     fprintf(fil, fname);
     image.reader(fname);
     findObjects(image, 0, 0, 255);
     fname[0]='u';
     fname[1]='t';
     image.BMPRGBWriter(fname);
   }

   47
void main()
{printf("\nStartt");
    char                  *fname;

    if ((fil=fopen("utfile.txt", "w"))==NULL) {printf("\nImpossible to open
the bloody file."); exit (1); }

    fname="tr970321-27.bmp";
driver(fname);
    fname="tr970321-09.bmp";
driver(fname);
    fname="tr961007-22.bmp";
driver(fname);
    fname="tr961007-11.bmp";
driver(fname);
    fname="tr961007-07.bmp";
driver(fname);
    fname="tr960905-26.bmp";
driver(fname);
    fname="tr960905-15.bmp";
driver(fname);
    fname="tr960813-15.bmp";
driver(fname);
    fname="tr960703-27.bmp";
driver(fname);

    fclose(fil);

    printf("\nSlutt");
}
image.h a header for handling the images

#ifndef IMAGE_H
#define IMAGE_H

#include  "jonas.h"
#include  <math.h>

//These two since fread and fwrite dont work properly on UNIX
ulong ulongRead(FILE *f)
{
    return fgetc(f)+fgetc(f)*256+fgetc(f)*65536L+fgetc(f)*16777216L;
}

void ulongWrite(FILE *f, ulong in)
{
    fputc(in % 256, f);
    fputc((in % 65536L) / 256, f);
    fputc((in % 16777216L) / 65536L, f);
    fputc(in / 16777216L, f);
}

class Pixel
{
public:
    Pixel();
    Pixel(uchar pr, uchar pg, uchar pb);
    Pixel(const Pixel& obj);
    ~Pixel() {};
    void set(uchar pr, uchar pg, uchar pb);
    void setOne(int c, uchar v);
    uchar getOne(int c);
    Pixel& operator=(Pixel& inObj);
    void RGBtoHSI();

protected:
    uchar  r,b,g;
};

class imageHandler
{public:
    imageHandler();
    imageHandler(ulong w, ulong h, int bytes);
    imageHandler(char *fname);
    virtual ~imageHandler() {};

    //These are almost-constructors
    void create(ulong w, ulong h, int bytes);
    void reader(char *fname);

    void BMPRGBWriter(char *fname);
    void RGBtoHSI();
    ulong pixelPos(ulong x, ulong y);
    void imagePart(imageHandler &res, ulong sX, ulong sY, ulong w, ulong h,
    int angle);
    int isColour(ulong x, ulong y, uchar r,  uchar g,  uchar b);
    //Tells if position has the colour
    void nextCoor(double &x, double &y, int angle, ulong step);
    //Move step steps in the angle. Node the double-coordinates
    double getFacit(uchar r,  uchar g,  uchar b);
    //Returns 1 if the middle pixel has the parameter value, else 0
}
void toDouble(Arr<double> &res);
//returns the pixel-value as a Arr of doubles from 0 to 1
ulong getWidth() {return width;};
ulong getHeigth() {return heigth;};
void setColour(ulong x, ulong y, uchar r, uchar g, uchar b);
void blacken();
void VFlipp();
uchar getColour(ulong x, ulong y, char c);
ulong goWhile(int angle, double sX, double sY, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2); //Shall return the euclidian distance from start to the point there the parameter colour stops
void shortestLine(int &angle, ulong &distance, double &x, double &y, double sX, double sY, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2); //Shall return the shortest line to the edge of the two parameter colour.
int longestRadie(ulong x, ulong y, uchar r, uchar g, uchar b); //Shall return the angle of longest radie to the edge of the two parameter colour.
void findColour(ulong &x, ulong &y, uchar r, uchar g, uchar b); //Moves left to right, low to high until the parameter colour
void pie(int startA, int steps, ulong x, ulong y, ulong l, uchar r, uchar g, uchar b); 
void pieIf(int startA, int steps, ulong x, ulong y, ulong l, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2); 
void line(int angle, ulong sX, ulong sY, ulong l, uchar r, uchar g, uchar b);
void lineIf(int angle, ulong sX, ulong sY, ulong l, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2);
int within(ulong x, ulong y);
// void nextEdge(ulong &x, ulong &y, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2, uchar r3, uchar g3, uchar b3);
void test(int ant);

protected:
void BMPRGBReader(FILE *fil);

Arr<Pixel> cData;
ulong width, heigth, offset;
};

Pixel :: Pixel()
{set(0,0,0);}

Pixel :: Pixel(uchar pr, uchar pg,uchar pb)
{set(pr, pg, pb);}

Pixel :: Pixel(const Pixel& obj)
{set(obj.r, obj.g, obj.b);}

void Pixel :: set(uchar pr, uchar pg, uchar pb)
{r=pr;
g=pg;
b=pb;
}

void Pixel :: setOne(int c, uchar v)
{ if (c==0) r=v;
  else if (c==1) g=v;
}
else if (c==2) b=v;
else {printf("\nP\Pixel.setOne was called with wrong value on c %d",c);
exit(1);}  
}

uchar Pixel :: getOne(int c)
{
   if (c==0) return r;
   else if (c==1) return g;
   else if (c==2) return b;
   else {printf("\nP\Pixel.getOne was called with wrong value on c %d",c);
   exit(1);}  
   return 0; //Never reached  
}

Pixel& Pixel :: operator=(Pixel &inObj)
{
   r=inObj.r;
   g=inObj.g;
   b=inObj.b;
   return *this;
}

void Pixel :: RGBtoHSI()//Does not bloody work
{
   static double hmax=-100, hmin=100;
   double h, s, i, dr=(double)r/255, dg=(double)g/255, db=(double)b/255;

   h=(double)acos(nollDiv(((double)0.5*(dr-dg+dr-db)), sqrt(sqr(dr-dg)+(dr-db)*(dg-db))));
   s=(double)1-(((double)3/(dr+dg+db))*minAvTre(dr, dg, db));
   i=((double)1/(double)3)*(dr+dg+db);

   if (h>hmax) {hmax=h; printf("\nmax h %f %d %d %d %f ",hmax, r, g, b,
   nollDiv(((double)0.5*(dr-dg+dr-db)), sqrt(sqr(dr-dg)+(dr-db)*(dg-db)))) ;}
   if (h<hmin) {hmin=h; printf("\nmin h %f %d %d %d %f ",hmin, r, g, b,
   nollDiv(((double)0.5*(dr-dg+dr-db)), sqrt(sqr(dr-dg)+(dr-db)*(dg-db)))) ;}

   r=(uchar)((h*510)-255);
   g=(uchar)(s*255);
   b=(uchar)(i*255);
}

imageHandler :: imageHandler()
{
   offset=54;
}

imageHandler :: imageHandler(ulong w, ulong h, int bits)
{
   create(w, h, bits);
}

imageHandler :: imageHandler(char *fname)
{
   reader(fname);
}

void imageHandler :: create(ulong w, ulong h, int bits)
{
   if (bits != 24) {printf("%s\n","imageHandler :: imageHandler(): at this
   moment in time I can only handle RGB-images"); exit (1);}
width=w;
height=h;
offset=54;
cData.nylangd(width * heigth);

for(ulong i=0; i<width * heigth; i++)
cData[i].set(1,100,2); //gotta fill with something
}

void imageHandler :: reader(char *fname)
{
   FILE   *rfile;

   if ((rfile=fopen(fname,"rb"))==NULL)
   {
      printf("%s%s%s
","imageHandler :: imageHandler():",fname," impossible to
open");
      exit (1);
   }

   /*In some distant future this function migth handle more kinds of files
and, in addition, not simply die then problem occure*/

   BMPRGBReader(rfile);
   fclose(rfile);
}

void imageHandler :: BMPRGBReader(FILE *rfile)
{char           ch;
ulong          ultemp, p=0;
unsigned       intemp;

   ch=fgetc(rfile);
   if (ch !='B')
   {
      printf("%s%s%s
","imageHandler :: BMPRGBReader()",rfile," not a BMP-
file");
      exit (1);
   }

   ch=fgetc(rfile);
   if (ch !='M')
   {
      printf("%s%s%s
","imageHandler :: BMPRGBReader()",rfile," not a BMP-
file");
      exit (1);
   }

   fseek(rfile,10,SEEK_SET); //Junk
   offset=ulongRead(rfile);
   fseek(rfile,14+4,SEEK_SET); //Junk
   width=ulongRead(rfile);
   //printf("\nThe files width: %lu", width);
   heigth=ulongRead(rfile);
   //printf("\nThe files heigth: %lu", heigth);
   fseek(rfile,14+14,SEEK_SET); //Junk
intemp=fgetc(rfile);
fgetc(rfile);
if ((intemp !=24) && (intemp !=3435921432)) //ICA-client cant handle math
{printf("%s%s\n","imageHandler :: BMPRGBReader()",rfile," Bugger, not a
24-bit RGB");
exit (1);
}
ultemp=ulongRead(rfile);
if (ultemp !=0)
{printf("%s%s\n","imageHandler :: BMPRGBReader()",rfile," Bugger, this
file is compressed");
exit (1);
}

fseek(rfile,offset,SEEK_SET); //Junk

cData.nyaml(width * heigth);

//The colours are in BGR-order (why make it easy)

for(ulong h=0; h<heigth; h++)
{
for(ulong w=0; w<width; w++)
{for(unsigned i=0; i<3; i++)
{
if (i==0) cData[p].setOne(2, fgetc(rfile));
if (i==1) cData[p].setOne(1, fgetc(rfile));
if (i==2) cData[p].setOne(0, fgetc(rfile));
if (feof (rfile)) {printf("\n imageHandler.BMPRGBReader() got an
unexpexted end-of-file during pixel-reading"); exit(1);}
}
p=p+1;
}
for(unsigned i=0; i<width % 4; i++)
fgetc(rfile);
}

fclose(rfile);
}

void imageHandler :: BMPRGBWriter(char *fname)
{ulong p=0, i;
FILE   *wFile;
ulong   loTemp;

if ((wFile=fopen(fname,"wb"))==NULL)
{printf("%s%s\n","imageHandler :: BMPRGBWriter()":",fname," impossible to
open");
exit (1);
}

//Fill up ofsett
for(i=0; i<offset; i++)
fputc(0, wFile);

fseek(wFile,0,SEEK_SET);

fputc('B', wFile);
fputc('M', wFile);
fseek(wFile,10,SEEK_SET); //Junk
ulongWrite(wFile, offset);
ulongWrite(wFile, 40); //"Number of bytes in header - currently 40"
fseek(wFile,14+4,SEEK_SET); //Junk
ulongWrite(wFile, width);
ulongWrite(wFile, heigth);
ftypec(1 , wFile); //"Number of colour planes - must be set to 0" dont know why i set it to 1
ftypec(0 , wFile);
for(ulong h=0; h<heigth; h++)
{
    for(ulong w=0; w<width; w++)
    {
        fputc(cData[p].getOne(2), wFile);
        fputc(cData[p].getOne(1), wFile);
        fputc(cData[p].getOne(0), wFile);
        p=p+1;
    }
    for(unsigned i=0; i<width % 4; i++)
    {fputc(0, wFile);}
}
//To write the filesize
loTemp=ftell(wFile);
fseek(wFile,2,SEEK_SET);
ulongWrite(wFile, loTemp);
fclose(wFile);
void imageHandler :: RGBtoHSI()
{
    for(ulong i=0; i<cData.gelangd(); i++)
        cData[i].RGBtoHSI();
}
ulong imageHandler :: pixelPos(ulong x, ulong y)
{
    if (x > width)
        {printf("\nimageHandler.pixelPos got too big x: %lu ", x); exit(1);} //Junk
    if (y > heigth)
        {printf("\nimageHandler.pixelPos got too big y: %lu ", y); exit(1);} //Junk
    return y*width + x;
}
void imageHandler :: test(int ant)
{
    printf("\n");
    for(int i=0; i< ant; i++)
    {
        printf("%d ", cData[i].getOne(2));
        printf("%d ", cData[i].getOne(1));
        printf("%d  ", cData[i].getOne(0));
    }
}

void imageHandler :: imagePart(imageHandler &res, ulong sX, ulong sY, ulong w, ulong h, int angle)
{double          dx, dy, dsx, dsy;

    angle=(angle-90) % 360; //So that the angle should be the left edge, and 
    sX and sY should lower left.
    sX=sX+w;

    res.create(w, h, 24);

    for(ulong j=0; j<h; j++)
    {
        dsy=j*(mySin((double) (angle+90) % 360)));
        dsx=j*(myCos((double) (angle+90) % 360)));

        for(ulong i=0; i<w; i++)
        {
            dy=i*(mySin((double) angle));
            dx=i*(myCos((double) angle));

            res.cData[res.pixelPos(w-i-1, j)] = cData[pixelPos(round(dx+dsx+sX),
                                                  round(dy+dsy+sY))];
        }
    }
}

int  imageHandler   :: isColour(ulong x, ulong y, uchar r, uchar g, 
                              uchar b)
{
    if ((cData[pixelPos(x, y)].getOne(0)==r) &&
        (cData[pixelPos(x, y)].getOne(1)==g) &&
        (cData[pixelPos(x, y)].getOne(2)==b)) return 1;
    else return 0;
}

ulong euclides(ulong ix1, ulong iy1, ulong ix2, ulong iy2)
{long        x1, y1, x2, y2;
    x1=(long) ix1;
    y1=(long) iy1;
    x2=(long) ix2;
    y2=(long) iy2;

    return (ulong) sqrt( (double) (sqr(x1-x2) + sqr(y1-y2)) );
}

double imageHandler :: getFacit(uchar r, uchar g, uchar b)
{    if (isColour(width/2, heigth/2, r, g, b)) return 1;
    else return 0;
}
void imageHandler :: toDouble(Arr<double> &res)
{
    res.nylangd(cData.gelangd() * 3);
    for(ulong i=0; i<cData.gelangd(); i++)
    {
        for(ulong j=0; j<3; j++)
            res[i*3 + j] = (double)cData[i].getOne(j) / (double)255;
    }
}

void imageHandler :: setColour(ulong x, ulong y, uchar r, uchar g, uchar b)
{
    cData[pixelPos(x, y)].set(r, g, b);
}

void imageHandler :: blacken()
{
    for(ulong i=0; i<cData.gelangd(); i++)
        cData[i].set(0, 0, 0);
}

void imageHandler :: VFlipp()
{
    ulong h, w;
    Pixel temp;
    for(h=0; h<heigth; h++)
    {for(w=0; w < width / 2; w++)
    {
        temp = cData[pixelPos(w, h)];
        cData[pixelPos(w, h)] = cData[pixelPos(width-1-w, h)];
        cData[pixelPos(width-1-w, h)] = temp;
    }}
}

uchar imageHandler :: getColour(ulong x, ulong y, char c)
{
    int par;
    if (c=='r') par=0;
    else if (c=='g') par=1;
    else if (c=='b') par=2;
    else {printf("\n imageHandler :: getColour(): got wrong c parameter\n");
        exit (1);
    }
    return cData[pixelPos(x, y)].getOne(par);
}

void imageHandler :: line(int angle, ulong sX, ulong sY, ulong l, uchar r, uchar g, uchar b)
{double dx, dy, dl;
    dl=(double) l;
    for(ulong i=0; i<l; i++)
    {
        dy=i*(mySin((double) angle));
        dx=i*(myCos((double) angle));
        cData[pixelPos(round(dx+sX), round(dy+sY))].set(r, g, b);
    }
}
void imageHandler :: lineIf(int angle, ulong sX, ulong sY, ulong l, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2)
{
    double dx, dy, dl;

    dl=(double) l;

    for(ulong i=0; i<l; i++)
    {
        dy=i*(mySin((double) angle));
        dx=i*(myCos((double) angle));
        if ((dx+(double)sX>=0) && (dy+(double)sY>=0) && (within(round(dx+sX), round(dy+sY))) && (isColour(round(dx+sX), round(dy+sY), r1, g1, b1))
            cData[pixelPos(round(dx+sX), round(dy+sY))].set(r2, g2, b2);
    }
}

void imageHandler :: pie(int startA, int steps, ulong x, ulong y, ulong l, uchar r, uchar g, uchar b)
{
    int i;

    for (i=startA; i<startA+steps; i++)
        line(i % 360, x, y, l, r, g, b);
}

void imageHandler :: pieIf(int startA, int steps, ulong x, ulong y, ulong l, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2)
{
    int i;

    for (i=startA; i<startA+steps; i++)
        lineIf(i % 360, x, y, l, r1, g1, b1, r2, g2, b2);
}

ulong imageHandler :: goWhile(int angle, double sX, double sY, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2)
{
    double dx=0, dy=0;
    ulong i=0;

    //Its bloody impossible to use ">= 0"
    while ((dx+sX >= 0) && (dy+sY >= 0) && (round(dx+sX) < width) &&
        (round(dy+sY) < height) && ((isColour(round(dx+sX), round(dy+sY), r1, g1, b1)) || (isColour(round(dx+sX), round(dy+sY), r2, g2, b2)))
    {
        dy=i*(mySin((double) angle));
        dx=i*(myCos((double) angle));
        i=i+1;
    }
    return (ulong) sqrt(dx*dx+dy*dy);
}

int imageHandler  :: longestRadie(ulong x, ulong y, uchar r, uchar g, uchar b) //Shall return the longest radie to the edge of the two parameter colour.
{
    int i, res;
    ulong maxDist=0, tDist;

    for(i=0; i<360; i++)
    {
        tDist=goWhile(i, x, y, r, g, b, r, g, b);
        if (tDist > maxDist)
            {res=i; maxDist=tDist;}
    }
    return res;
void imageHandler :: shortestLine(int &angle, ulong &distance, double &x, double &y, double sX, double sY, uchar r1, uchar g1, uchar b1, uchar r2, uchar g2, uchar b2)
{
    ulong tDist;
    int i, tAngle, inAngle=angle;
    double x1, x2, y1, y2;

    distance=65000;

    for(i=0; i<90; i++)
    {tAngle=(inAngle+315+i) % 360;
        tDist=goWhile(tAngle, sX, sY, r1, g1, b1, r2, g2, b2) +
        goWhile(tAngle+180, sX, sY, r1, g1, b1, r2, g2, b2);
        if (tDist < distance)
        {
            distance=tDist;
            angle=tAngle;
            x1=sX;
            y1=sY;
            nextCoor(x1, y1, angle, goWhile(tAngle, sX, sY, r1, g1, b1, r2, g2, b2));
            x2=sX;
            y2=sY;
            nextCoor(x2, y2, angle+180, goWhile((tAngle+180) % 360, sX, sY, r1, g1, b1, r2, g2, b2));
            x=x1+((x2-x1) / 2);
            y=y1+((y2-y1) / 2);
        }
    }

    if (distance > 0) pie(angle, 180, round(x), round(y), (distance/2) + 1, r1, g1, b1, r2, g2, b2);
    // line(angle,(ulong) x,(ulong) y, distance/2, r2, g2, b2);
    // line(angle+180,(ulong) x,(ulong) y, distance/2, r2, g2, b2);

    void imageHandler :: nextCoor(double &x, double &y, int angle, ulong step)
    {
        double dx, dy;

        dy=step*(mySin((double) angle));
        dx=step*(myCos((double) angle));

        x=x+(double)x;
        y=y+(double)y;
    }

    void imageHandler :: findColour(ulong &x, ulong &y, uchar r, uchar g, uchar b)
    {
        while (! ((x==width-1) && (y==heigth-1))) && (! (isColour(x, y, r, g, b))))
        {
            x=x+1;
            if (x==width)
            {x=0;
             y=y+1;
            }
if ((x==width-1) && (y==heigth-1)) //If the end of the image is found 0 0 is returned
{x=0;
y=0;
}

void imageHandler :: oneStep(ulong &x, ulong &y) //Moves one step to right, wrap around
{
x=x+1;
if (x>=width)
{x=0;
y=y+1;
}

int imageHandler :: within(ulong x, ulong y)
{
return ((x<width) && (y<heigth));
}

#endif
#include <stdio.h> 
#include <stdlib.h> 
#include <math.h> 

typedef unsigned long int ulong; 
typedef unsigned char uchar; 

template <class Typ> 
class Arr 
{
    public:
        Arr();  
        Arr(ulong plangd); 
        Arr(const Arr<Typ>& obj); 
        ~Arr(); 
        virtual void nylangd(ulong nlangd); 
        virtual ulong gelangd(); 
        virtual Typ& operator[] (ulong i); 
    
    protected:
        Typ *rad;  
        ulong langd; 
}; 

template <class Typ> 
class NumArr : public Arr<Typ> 
{
    public:
        NumArr();  
        NumArr(ulong plangd); 
        NumArr(const Arr<Typ>& obj); 
        void nolla();  
        ulong uSum();  
        long lSum();  
        Typ Sum(); 
        double dSnitt(); 
        Typ max(); 
}; 

template <class Typ> 
//Default-konstruktor 
Arr<Typ> :: Arr() 
{ 
    langd=0;  
    rad=0; 
} 

template <class Typ> 
Arr<Typ> :: Arr(ulong plangd) 
{ 
    if (plangd < 0) 
    {printf("Arr.Arr fick negativ storlek %d\n",plangd);  
        exit(1); 
    } 
    rad=new Typ[plangd];
langd=plangd;
}

```cpp
template <class Typ>
Arr<Typ> :: Arr(const Arr<Typ>& obj)
{
    langd=obj.langd;
    rad=new Typ[langd];
    for(ulong i=0; i<langd; i++)
        rad[i]=obj.rad[i];
}
```

```cpp
template <class Typ>
Arr<Typ> :: ~Arr()
{
    delete [] rad;
}
```

```cpp
template <class Typ>
Typ& Arr<Typ> :: operator[](ulong i)
{
    if ((i>=langd) || (i<0))
    {if (i<0) printf("Arr[] fick negativt index %d\n",i); 
     if (i>=langd) printf("Arr[] fick för högt index %d\n",i); 
     exit(1);
    }
    /*Varning: denna koll borttagen av tidsskäl */
    return rad[i];
}
```

```cpp
template <class Typ>
void Arr<Typ> :: nylangd(ulong nlangd)
{ulong i;
    Typ            *tilf;
    if (nlangd < 0)
    {printf("Arr.nylangd fick negativ storlek %d\n",nlangd);
     exit(1);
    }
    tilf=new Typ[nlangd];
    for (i=0; (i<nlangd) && (i<langd); i++)
        tilf[i]=rad[i];
    delete [] rad;
    rad=tilf;
    langd=nlangd;
}
```

```cpp
template <class Typ>
ulong Arr<Typ> :: gelangd()
{
    return langd;
}
```

```cpp
template <class Typ>
NumArr<Typ> :: NumArr()
:Arr<Typ>()
```
template <class Typ>
   NumArr<Typ>   :: NumArr(ulong plangd)
      :Arr<Typ>(plangd)
{}

template <class Typ>
   NumArr<Typ>   :: NumArr(const Arr<Typ>& obj)
      :Arr<Typ>(obj)
{}

template <class Typ>
   void NumArr<Typ>   :: nolla()
   {
      for(ulong i=0; i<langd; i++)
         rad[i]=0;
   }

template <class Typ>
   ulong NumArr<Typ> :: uSum()
   {
      ulong            res=0;
      for(ulong i=0; i<langd; i++)
         res=res+rad[i];
      return res;
   }

template <class Typ>
   long NumArr<Typ> :: lSum()
   {
      long            res=0;
      for(ulong i=0; i<langd; i++)
         res=res+rad[i];
      return res;
   }

template <class Typ>
   Typ NumArr<Typ> :: Sum()
   {
      Typ            res=0;
      for(ulong i=0; i<langd; i++)
         res=res+rad[i];
      return res;
   }

template <class Typ>
   double NumArr<Typ> :: dSnitt()
   {
      double            sum=0;
      for(ulong i=0; i<langd; i++)
         sum=sum+(double)rad[i];
      return nollDiv(sum,(double) langd);
   }

template <class Typ>
   Typ NumArr<Typ> :: max()
for(ulong i=0; i<langd; i++)
    if (rad[i]>max) max=rad[i];

return max;
}

//Här är lösa funktioner

//Fanimej helt otroligt att jag ej kan hitta denna inbyggt
ulong round(double intal)
{
    double tilf;
    tilf=floor(intal);
    if (intal - tilf >0.5) return (ulong) tilf+1;
    else return (ulong) tilf;
}

//Fanimej helt otroligt att jag ej kan hitta denna inbyggt II
const double Pi=3.1415926535897932;

double angleToRadiant(double angle)
{ return (((int) (360-angle)+450) % 360) *(Pi/180); }

double radiantToAngle(double radiant)
{ return (360 - (int)(radiant*(180/Pi))+450) % 360; }

double mySin(double angle)
{ return sin(angleToRadiant(angle)); }

double myCos(double angle)
{ return cos(angleToRadiant(angle)); }

template <class Typ2>
Typ2 abs(Typ2 tal)
{
    if (tal>0) return tal;
    return tal*(-1);
}

template <class Typ2>
Typ2 sqr(Typ2 tal)
{ return tal*tal;
}

template <class Typ2>
Typ2 minAvTre(Typ2 ett, Typ2 tva, Typ2 tre)
{ if ((ett<tva) && (ett<tre)) return ett;
  if (tva<tre) return tva;
  return tre;
}

template <class Typ2>
Typ2 nollDiv(Typ2 talj, Typ2 namn)
{ if (namn==0) return 0;
  return talj/namn;
}
vector.h containing a class for storing data about roots

```c
#ifndef VECTOR_H
#define VECTOR_H

#include "jonas.h"

const ulong STEPSIZE=2;  //The length of each width-sampling

class Vector
{
public:
    Vector() {};
    Vector(Vector& obj);
    ~Vector() {};
    void clear();
    void setCoor(ulong pX, ulong pY);
    void addWidth(ulong w);
    Vector& operator=(Vector& bj);
    void write();
    void fwrite(FILE *fil);
    ulong giveLength() {return width.getLangd();};
    ulong giveWidth(ulong i) {return width[i];};
    int areYouRoot();

protected:
    NumArr<ulong>       width;
    ulong      x, y;
};

Vector :: Vector(Vector& obj)
{
    x=obj.x;
    y=obj.y;

    width.nyLangd(obj.giveLength());

    for(ulong i=0; i<obj.giveLength(); i++)
        width[i]=obj.giveWidth(i);
}

void  Vector :: clear()
{
    x=0;
    y=0;
    width.nyLangd(0);
}

void  Vector :: setCoor(ulong pX, ulong pY)
{
    x=pX;
    y=pY;
}

void  Vector :: addWidth(ulong w)
{
    width.nyLangd(width.getLangd()+1);
    width[width.getLangd()-1]=w;
}
```

65
Vector& Vector :: operator=(Vector &obj)
{
    x=obj.x;
    y=obj.y;

    width.nylangd(obj.giveLength());
    for(ulong i=0; i<obj.giveLength(); i++)
        width[i]=obj.giveWidth(i);

    return *this;
}

void Vector :: write()
{
    printf("\nVector %lu %lu %lu %lu %f %f\n",x , y, width.gelangd()*STEPSIZE, width.max(), width.dSnitt(),
    Pi*sqr(width.dSnitt()/2)*width.gelangd()*STEPSIZE);
    // printf("\nVector %lu %lu %lu %lu \n",x , y, width.gelangd()*3,
    width.max());
    for(ulong i=0; i<width.gelangd(); i++)
        printf("%lu ",width[i]);
}

void Vector :: fwrite(FILE *fil)
{
    fprintf(fil, "\nVector %lu %lu %lu %lu %f %f\n",x , y,
    width.gelangd()*STEPSIZE, width.max(), width.dSnitt(),
    Pi*sqr(width.dSnitt()/2)*width.gelangd()*STEPSIZE);
    // printf("\nVector %lu %lu %lu %lu \n",x , y, width.gelangd()*3,
    width.max());
    for(ulong i=0; i<width.gelangd(); i++)
        fprintf(fil, "%lu ",width[i]);
}

int Vector :: areYouRoot()
{
    if (width.gelangd()*STEPSIZE < 15) return 0;
    if (width.dSnitt() > 30) return 0;
    if (width.max() > 50) return 0;
    if ((double)width.gelangd()*STEPSIZE < (double)5*width.dSnitt()) return 0;

    return 1;
}
#endif
BPbox.h containing a class that handles the neural network

#include "jonas.h"

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

typedef int BOOL;
typedef int INT;
typedef double REAL;

#define FALSE 0
#define TRUE 1
#define BIAS 1
#define sqr(x) ((x)*(x))

REAL RandomEqualREAL(REAL Low, REAL High)
{
    return ((REAL) rand() / RAND_MAX) * (High-Low) + Low;
}

//Konstanter omgjorda till variable:
int NUM_LAYERS; //Antalet lager
int *Units; //Skall ange antal noder i varje lager. 0 är inputlagret. Orginalversion: INT
Units[NUM_LAYERS] = {N, 10, M};

//Detta e la själva nät-strukturen
typedef struct { /* A LAYER OF A NET: */
    int Units; /* - number of units in this layer */
    REAL* Output; /* - output of ith unit */
    REAL* Error; /* - error term of ith unit */
    REAL** Weight; /* - connection weights to ith unit */
    REAL** WeightSave; /* - saved weights for stopped training */
    REAL** dWeight; /* - last weight deltas for momentum */
} LAYER;

typedef struct { /* A NET: */
    LAYER** Layer; /* - layers of this net */
    LAYER* InputLayer; /* - input layer */
    LAYER* OutputLayer; /* - output layer */
    REAL Alpha; /* - momentum factor */
    REAL Eta; /* - learning rate */
    REAL Gain; /* - gain of sigmoid function */
};
REAL          Error;         /* - total net error */
} NET;

**************************************************************************
****
INITIALIZATION
**************************************************************************

void GenerateNetwork(NET* Net)
{
  INT l,i;
  Net->Layer = (LAYER**) calloc(NUM_LAYERS, sizeof(LAYER*));
  for (l=0; l<NUM_LAYERS; l++) {
    Net->Layer[l] = (LAYER*) malloc(sizeof(LAYER));
    Net->Layer[l]->Units      = Units[l];
    Net->Layer[l]->Output     = (REAL*) calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Error      = (REAL*) calloc(Units[l]+1, sizeof(REAL));
    Net->Layer[l]->Weight     = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->WeightSave = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->dWeight    = (REAL**) calloc(Units[l]+1, sizeof(REAL*));
    Net->Layer[l]->Output[0]  = BIAS;
    if (l != 0) {
      for (i=1; i<=Units[l]; i++) {
        Net->Layer[l]->Weight[i]     = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
        Net->Layer[l]->WeightSave[i] = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
        Net->Layer[l]->dWeight[i]    = (REAL*) calloc(Units[l-1]+1, sizeof(REAL));
      }
    }
    Net->InputLayer  = Net->Layer[0];
    Net->OutputLayer = Net->Layer[NUM_LAYERS - 1];
    Net->Alpha       = 0.9;
    Net->Eta         = 0.25;
    Net->Gain        = 1;
  }

void RandomWeights(NET* Net)
{
  INT l,i,j;
  for (l=1; l<NUM_LAYERS; l++) {
    for (i=1; i<=Net->Layer[l]->Units; i++) {
      for (j=0; j<Net->Layer[l-1]->Units; j++) {
        // Net->Layer[l]->Weight[i][j] = RandomEqualREAL(-0.5, 0.5); Det
        Net->Layer[l]->Weight[i][j] = RandomEqualREAL(-0.01, 0.01);
      }
    }
  }
}
void SetInput(NET* Net, REAL* Input)
{
    INT i;

    for (i=1; i<=Net->InputLayer->Units; i++) {
        Net->InputLayer->Output[i] = Input[i-1];
    }
}

void GetOutput(NET* Net, REAL* Output)
{
    INT i;

    for (i=1; i<=Net->OutputLayer->Units; i++) {
        Output[i-1] = Net->OutputLayer->Output[i];
    }
}

/**************************************************************************
****
P R O P A G A T I N G   S I G N A L S
**************************************************************************/

PROPAGATING SIGNALS

void PropagateLayer(NET* Net, LAYER* Lower, LAYER* Upper)
{
    INT i,j;
    REAL Sum;

    for (i=1; i<=Upper->Units; i++) {
        Sum = 0;
        for (j=0; j<=Lower->Units; j++) {
            Sum += Upper->Weight[i][j] * Lower->Output[j];
        }
        Upper->Output[i] = 1 / (1 + exp(-Net->Gain * Sum));
    }
}

void PropagateNet(NET* Net)
{
    INT l;

    for (l=0; l<NUM_LAYERS-1; l++) {
        PropagateLayer(Net, Net->Layer[l], Net->Layer[l+1]);
    }
}

/**************************************************************************
****
B A C K P R O P A G A T I N G   E R R O R S
**************************************************************************/

BACKPROPAGATING ERRORS

void ComputeOutputError(NET* Net, REAL* Target)
{  
  INT i;
  REAL Out, Err;

  Net->Error = 0;
  for (i=1; i<=Net->OutputLayer->Units; i++) {
    Out = Net->OutputLayer->Output[i];
    Err = Target[i-1]-Out;
    Net->OutputLayer->Error[i] = Net->Gain * Out * (1-Out) * Err;
    Net->Error += 0.5 * sqr(Err);
  }
}

void BackpropagateLayer(NET* Net, LAYER* Upper, LAYER* Lower)  
{  
  INT i,j;
  REAL Out, Err;

  for (i=1; i<=Lower->Units; i++)  
  {  
    Out = Lower->Output[i];
    Err = 0;
    for (j=1; j<=Upper->Units; j++)  
      Err += Upper->Weight[j][i] * Upper->Error[j];
    Lower->Error[i] = Net->Gain * Out * (1-Out) * Err;
  }
}

void BackpropagateNet(NET* Net)  
{  
  INT l;

  for (l=NUM_LAYERS-1; l>1; l--)  
  {  
    BackpropagateLayer(Net, Net->Layer[l], Net->Layer[l-1]);
  }
}

void AdjustWeights(NET* Net)  
{  
  INT l,i,j;
  REAL Out, Err, dWeight;

  for (l=1; l<NUM_LAYERS; l++)  
  {  
    for (i=1; i<=Net->Layer[l]->Units; i++)  
    {  
      for (j=0; j<=Net->Layer[l-1]->Units; j++)  
      {  
        Out = Net->Layer[l-1]->Output[j];
        Err = Net->Layer[l]->Error[i];
        dWeight = Net->Layer[l]->dWeight[i][j];
        dWeight = Net->Layer[l]->dWeight[i][j] + Net->Eta * Err * Out + Net->Alpha * dWeight;
        Net->Layer[l]->Weight[i][j] += Net->Eta * Err * Out + Net->Alpha * dWeight;
        Net->Layer[l]->dWeight[i][j] = Net->Eta * Err * Out;
      }
    }
  }
}
void SimulateNet(NET* Net, REAL* Input, REAL* Output, REAL* Target, BOOL Training)
{
    SetInput(Net, Input);
    PropagateNet(Net);
    GetOutput(Net, Output);

    if (Training) {
        ComputeOutputError(Net, Target); // Denna rad har jag flytta innanför
                                            // vilkoret.
        BackpropagateNet(Net);
        AdjustWeights(Net);
    }
}

// Under rubriken "SIMULATING THE NET" finns drivers till
// ovanstående, de är dock domänspecifika.

class BPbox
{
    public:
        BPbox() {started=FALSE;};
        BPbox(int layers, Arr<int> nodes); //nodes determines the number of nodes
              in each layer, starting with inputlayer
        virtual void starter(int layers, Arr<int> nodes);
        virtual void read(char *fname);
        -BPbox() {};
        virtual void train(Arr<double> &res, int epochs, int no, Arr<double>
              pinput, Arr<double> pfacit); // number is number of training set. Pos in
              put 0-dimension actual output, next average mean square error
        virtual int  boolTrain(Arr<double> &res, int epochs, int no, Arr<double>
              pinput, Arr<double> pfacit); // Perform training only if needed, if err >=
              0.5. Facit==0.5 is considered as irrelevant.
        virtual void use(Arr<double> &res, int no, Arr<double> pinput);
        virtual void set(double alpha, double eta, double gain);
        virtual void save(char *fname);
        virtual void giveNodes(Arr<int> &nodes);

        ulong     rounds;

    protected:
        NET       net;
        BOOL      started;
        int       ins,outs;
    }

    BPbox :: BPbox(int layers, Arr<int> nodes)
    {
        started=FALSE;
        starter(layers, nodes);
    }

    void BPbox :: starter(int layers, Arr<int> nodes)
    {
        if (started==TRUE) {printf("\nBPbox.starter: You cant start me twice");
                        exit (1);}
    }
if ((unsigned) layers!=nodes.gelangd()) {printf("\nBPbox.starter: Number of layers and number of nodes-assignment disagree"); exit (1);} 

started=TRUE; 
rounds=0; 
ins=nodes[0]; 
outs=nodes[layers-1];

NUM_LAYERS=layers; 
Units=new int[layers]; 
for(int i=0; i<layers; i++) 
Units[i]=nodes[i]; 

GenerateNetwork(&net); 
RandomWeights(&net); 

//Stulet från InitializeApplication(), verkar viktigt...

    net.Alpha = 0.5;                         /* - momentum factor */
    net.Eta = 0.05;                         /* - learning rate */
    net.Gain = 1;                           /* - gain of sigmoid function */
}

void  BPbox :: train(Arr<double> &res, int epochs, int no, Arr<double> pinput, Arr<double> pfacit) //number is number of training set. Pos in putput 0-dimension actual output, next average mean square error 
{int           e, n, i;
    double        *input=new double[ins];
    double        *output=new double[outs];
    double        *target=new double[outs];

    if (pinput.gelangd() != (unsigned) no*ins) {printf("\nBPbox.train: I was expecting %d",no*ins," but got %d",pinput.gelangd()," inputs. \n"); exit (1);} 
    if (pfacit.gelangd() != (unsigned) no*outs) {printf("\nBPbox.train: I was expecting %d",no*outs," but got %d",pfacit.gelangd()," facits. \n"); exit (1);} 

    res.nylangd(pfacit.gelangd() * epochs);

    for(e=0; e<epochs; e++) 
    {
        for(n=0; n<no; n++)
        {
            for(i=0; i<ins; i++)
                input[i]=pinput[n*ins + i];

            for(i=0; i<outs; i++)
                target[i]=pfacit[n*outs + i];

            SimulateNet(&net, input, output, target, TRUE);
            rounds=rounds+1;

            for(i=0; i<outs; i++)
                res[e*no + n*outs + i]=output[i];
        }
    }
delete [] input;
delete [] output;
delete [] target;

72
void SimulateNet(NET* Net, REAL* Input, REAL* Output, REAL* Target, BOOL Training)
{
    SetInput(Net, Input);
    PropagateNet(Net);
    GetOutput(Net, Output);

    if (Training) {
        ComputeOutputError(Net, Target); //Denna rad har jag flytta innanför vilkoret.
        BackpropagateNet(Net);
        AdjustWeights(Net);
    }
}

int BPbox :: boolTrain(Arr<double> &res, int epochs, int no, Arr<double> pinput, Arr<double> pfacit) //Perform training only if needed, if err >= 0.5. Facit==0.5 is considered as irrelevant.
{
    int           e, n, i, train, retur=0;
    double        *input=new double[ins];
    double        *output=new double[outs];
    double        *target=new double[outs];

    if (pinput.gelangd() != (unsigned) no*ins) {printf("BPbox.train: I was expecting %d",no*ins," but got %d",pinput.gelangd()," inputs. \n"); exit (1);}
    if (pfacit.gelangd() != (unsigned) no*outs) {printf("BPbox.train: I was expecting %d",no*outs," but got %d",pfacit.gelangd()," facits. \n"); exit (1);}

    res.nylangd(pfacit.gelangd() * epochs);

    for(e=0; e<epochs; e++)
    {
        for(n=0; n<no; n++)
        {
            for(i=0; i<ins; i++)
                input[i]=pinput[n*ins + i];

            for(i=0; i<outs; i++)
                target[i]=pfacit[n*outs + i];

            SetInput(&net, input);
            PropagateNet(&net);
            GetOutput(&net, output);

            for(i=0; i<outs; i++)
                res[e*no + n*outs + i]=output[i];

            train=0;
            for(i=0; i<outs; i++)
            {
                if((abs(0.5-target[i]) > 0.1) && (abs(output[i]-target[i]) >= 0.25))
                    train=1;}//Should be trained if 0/1 target and error is more than 0.25

                if (train)
                {
                    printf("\n");
                    for(i=0; i<outs; i++)
                    { /*
printf("%.3f %.3f ", target[i], output[i]);
*/
  retur=retur+1;
  ComputeOutputError(&net, target);
  BackpropagateNet(&net);
  AdjustWeights(&net);
}

rounds=rounds+1;
}
}
delete [] input;
delete [] output;
delete [] target;
return retur;
}

void  BPbox :: use(Arr<double> &res, int no, Arr<double> pinput)
{
  int           n, i;
  double        *input=new double[ins];
  double        *output=new double[outs];
  double        *target=new double[outs];  //Bara med för att kunna skicka nått
  if (pinput.gelangd() != (unsigned) no*ins) {printf("BPbox.use: I was expecting %d",no*ins," but got %d",pinput.gelangd()," inputs. 
"); exit (1);}
  res.nylangd(no*outs);
  for(n=0; n<no; n++)
  {
    for(i=0; i<ins; i++)
      input[i]=pinput[n*ins + i];
    SimulateNet(&net, input, output, target, FALSE);
    for(i=0; i<outs; i++)
      res[n*outs + i]=output[i];
  }
  delete [] input;
  delete [] output;
  delete [] target;
}

void  BPbox :: set(double alpha, double eta, double gain)
{
  net.Alpha = alpha;  /* - momentum factor */
  net.Eta   = eta;    /* - learning rate */
  net.Gain  = gain;   /* - gain of sigmoid function */
}

void  BPbox :: save(char *fname)
{
  int         i,j,l;
  FILE        *fil;
if ((fil=fopen(fname,"wb"))==-NULL)
{printf("%s%s\n","BPbox.save() the file ",fname," is impossible to open");
 exit (1);
}
fwrite(&NUM_LAYERS,sizeof(NUM_LAYERS),1,fil);
for(i=0; i<NUM_LAYERS; i++)
{ fwrite(&Units[i],sizeof(Units[i]),1,fil);
}
fwrite(&net.Alpha,sizeof(net.Alpha),1,fil);
fwrite(&net.Eta,sizeof(net.Eta),1,fil);
fwrite(&net.Gain,sizeof(net.Gain),1,fil);
   for (l=1; l<NUM_LAYERS; l++) {
      for (i=1; i<=net.Layer[l]->Units; i++) {
         for (j=0; j<=net.Layer[l-1]->Units; j++) {
            fwrite(&net.Layer[l]->Weight[i][j],sizeof(net.Layer[l]-
>Weight[i][j]),1,fil);
         }
      }
   }
fclose(fil);
}

void BPbox :: read(char *fname)
{
int         i,j,l;
FILE        *fil;
if (started==TRUE) {printf("BPbox.read(): You cant start me twice");
 exit (1);}
 started=TRUE;
 rounds=0;                //Does not remember old trainings
if ((fil=fopen(fname,"rb"))==-NULL)
{printf("%s%s\n","BPbox.read() the file ",fname," is impossible to open");
 exit (1);
}
fread(&NUM_LAYERS,sizeof(NUM_LAYERS),1,fil);
Units=new int[NUM_LAYERS];
for(i=0; i<NUM_LAYERS; i++)
{ fread(&Units[i],sizeof(Units[i]),1,fil);
}
GenerateNetwork(&net);
ins=Units[0];
outs=Units[NUM_LAYERS-1];
fread(&net.Alpha,sizeof(net.Alpha),1,fil);
fread(&net.Eta,sizeof(net.Eta),1,fil);
fread(&net.Gain,sizeof(net.Gain),1,fil);
   for (l=1; l<NUM_LAYERS; l++) {
      for (i=1; i<=net.Layer[l]->Units; i++) {
         for (j=0; j<=net.Layer[l-1]->Units; j++) {
            fread(&net.Layer[l]->Weight[i][j],sizeof(net.Layer[l]-
>Weight[i][j]),1,fil);
         }
      }
   }
fread(&net.Layer[l]->Weight[i][j],sizeof(net.Layer[l]->Weight[i][j]),1,fil);
    if (feof (fil)) {printf("\n BPbox.read got an unexpected end-of-file during weight-reading"); exit(1);}
}
}
fclose(fil);
}

void BPbox :: giveNodes(Arr<int> &nodes)
{
    nodes.nylangd(NUM_LAYERS);
    for(int i=0; i<NUM_LAYERS; i++)
        nodes[i]=Units[i];
}

/*
Första output är XOR
andra är bool(input[0]) om output[0]==1, annars 0.5
andra är bool(input[2]) om output[0]==0, annars 0.5
*/

void fixadata(int sets, Arr<double> &input, Arr<double> &output)
{
    for(int i=0; i<sets; i++)
    {
        for(int j=0; j<3; j++)
            input[i*3 + j]=RandomEqualREAL( 0, 1);

        if(((input[i*3 + 0] > 0.5) && (input[i*3 + 2] > 0.5)) || ((input[i*3 + 0] < 0.5) && (input[i*3 + 2] < 0.5))) output[i*3]=0;
        else output[i*3]=1;

        if(output[i*3]==1)
        {
            output[i*3+1]=round(input[i*3 + 0]);
            output[i*3+2]=0.5;
        }
        else
        {
            output[i*3+2]=round(input[i*3 + 2]);
            output[i*3+1]=0.5;
        }
    }
}

void printData(int sets, Arr<double> &input, Arr<double> &output)
{
    for(int i=0; i<sets; i++)
    {
        for(int j=0; j<3; j++)
            printf("%0.2f ",input[i*3 + j]);

        printf("\n");

        for(j=0; j<3; j++)
            printf("%0.2f ",output[i*3 + j]);

        printf("\n");
    }
}

void crateandtrain()
{
ulong i, j, outs=3, trained=0;
unsigned sets=10000;
Arr<int> noder(3);
Arr<double> input(sets*outs), output(sets*outs), retur;
NumArr<ulong> err(outs);

printf("%d\n",sets);
noder[0]=3;
noder[1]=10;
noder[2]=outs;

BPbox natet(outs, noder);

for(i=0; i<500; i++)
{
    printf("%u ",i);
    fixadata(sets, input, output);
    trained=trained+(ulong)natet.boolTrain(retur, 1, sets, input, output);
    // natet.train(retur, 1, sets, input, output);
}

printf("\n%lu",trained);

fixadata(sets, input, output);
natet.use(retur, sets, input);

err.nolla();

for(i=0; i<sets; i++)
{
    for(j=0; j<outs; j++)
        if (abs(output[i*outs+j]-retur[i*outs+j]) > 0.5) err[j]=err[j]+1;
}

printf("\nTrain: ");
for(j=0; j<outs; j++)
    printf("%.6f ", (double)err[j]/(double)sets);

natet.save("pelle.net");

}

void readanduse()
{
    ulong i, j, outs=3;
    unsigned sets=10000;
    Arr<double> input(sets*3), output(sets*3), retur;
    NumArr<ulong> err(outs);
    BPbox natet;
    natet.read("pelle.net");

    fixadata(sets, input, output);
    natet.use(retur, sets, input);

    err.nolla();

    for(i=0; i<sets; i++)
    {
        for(j=0; j<outs; j++)
            if (abs(output[i*outs+j]-retur[i*outs+j]) > 0.5) err[j]=err[j]+1;
printf("\nTest: ");
for(j=0; j<outs; j++)
    printf("%.6f ",(double)err[j]/(double)sets);
}

void main()
{
    unsigned sets=100;
    Arr<double> input(sets*3), output(sets*3);
    crateandtrain();
    readanduse();
}
*/