KNMI-Publication; 227

HISKLIM 15

# Automatic Curve Extraction (ACE) Framework Documentation

*Hans van Piggelen, Jeroen Lichtenauer and Theo Brandsma*

De Bilt, 2010

## KNMI-Publication 227

## HISKLIM-15

De Bilt, 2010

PO Box 201
3730 AE De Bilt
Wilhelminalaan 10
De Bilt
The Netherlands
http://www.knmi.nl
Telephone +31(0)30-220 69 11
Telefax +31(0)30-221 04 07

Authors: H.E. van Piggelen, J.F. Lichtenauer, T. Brandsma

The KNMI-program HISKLIM aims at making historical land and sea climate data from Dutch sources physically accessible, with the highest possible time resolution and quality. The program started in 2000 and will run 5 to 10 years.

| | |
|---|---|
| HISKLIM-1 | Het KNMI-programma HISKLIM (HIStorisch KLIMaat) / T. Brandsma, F.B. Koek, H. Wallbrink en G.P. Können. (also KNMI-publication 191) |
| HISKLIM-2 | Gang van zaken 1940-48 rond de 20.000 zoekgeraakte scheepsjournalen / H. Wallbrink en F.B. Koek. (also KNMI-publication 192) |
| HISKLIM-3 | Historische maritieme windschalen tot 1947 / H. Wallbrink en F.B. Koek. (Memorandum) |
| HISKLIM-4 | Onbekende weersymbolen in oude Extract-Journalen (1826-1865). / H. Wallbrink en F.B. Koek. (Memorandum) |
| HISKLIM-5 | CLIWOC, Multilingual Meteorological Dictionary; an English-Spanish-Dutch-French dictionary of wind force terms used by mariners from 1750-1850 (also KNMI-publication 205) |
| HISKLIM-6 | DIGISTAD (DIGitaliseren STADswaterkantoor). H.W. Riepma. (Memorandum) |
| HISKLIM-7 | Parallel air temperature measurements at the KNMI-terrain in De Bilt (the Netherlands) May 2003–April 2005, Interim report. / T. Brandsma. (also KNMI-publication 207) |
| HISKLIM-8 | Hisklim COADS, Final report. / H. Wallbrink and F.B. Koek. (also KNMI-publication 210) |
| HISKLIM-9 | DIGISTAD, Disclosure of the hourly meteorological observations of the Amsterdam City Water Office 1784-1963, Final report. / H. Wallbrink and T. Brandsma. (also KNMI-publication 220) |
| HISKLIM-10 | Report on meteorological observations at Willemstad, Curaçao, during the period 1910-1946. / P.V.J. Girigori. (Memorandum) |
| HISKLIM-11 | The US Maury collection metadata 1796-1861. / H. Wallbrink, F.B. Koek and T. Brandsma (also KNMI-publication 225) |
| HISKLIM-12 | Historical maritime wind scales until 1947. / H. Wallbrink and F.B. Koek. (translation and update of Hisklim 3; Memorandum) |
| HISKLIM-13 | Historical wind speed equivalents of the Beaufort scale, 1850-1950. / H. Wallbrink and F.B. Koek. (Memorandum) |
| HISKLIM-14 | Data acquisition and keypunching codes for marine meteorological observations at the Royal Netherlands Meteorological Institute,1854 – 1968. / H. Wallbrink and F.B. Koek. (also KNMI-publication 226) |



● ● ● ●

# Automatic Curve Extraction (ACE) Framework Documentation

Hans van Piggelen, Jeroen Lichtenauer and Theo Brandsma

Royal Netherlands Meteorological Institute (KNMI)
De Bilt, The Netherlands

February, 2010

# Contents

# Preface

Digitization of the information on meteorological strip charts is a time consuming activity. This is especially true when high-resolution output is needed like 5-minute rainfall sums. In the scope of the Dutch research program 'Climate changes Spatial Planning', KNMI undertook the task of, among others, digitizing hundreds of station years of rainfall strip charts and paper rolls from Dutch stations. The end result is a dataset with long time series of 5-minute resolution rainfall data. To facilitate the digitization, a framework was developped that allows for Automatic Curve Extraction (ACE) from scanned images of strip charts and paper rolls. This report contains the documentation of the ACE framework.

# Chapter 1

# Introduction

In the archives of meteorological services, huge amounts of strip charts and paper rolls are stored. These charts and rolls may contain information not yet available in digital form but of interest for climate research and applications.

Strip charts and paper rolls have been used for continuous recording of elements like air temperature, humidity, cumulative rainfall depth, wind speed and direction and air pressure. In many parts of the world the recorders have now been replaced by automatic measuring devices whose output is digitally stored in databases. The time resolution of the latter is often an order of magnitude larger than that of the data that have been manually extracted from the strip charts and paper rolls. For instance, in the Netherlands temperature and rainfall are now operationally stored at 10 minutes resolution and for some stations even at 1 minute resolution. In contrast, the strip charts have been used in the past mostly for extracting hourly values. Nevertheless, the charts and rolls can be still be used to extract information with a time resolution of about 5 to 10 minutes. To do this manually is too labor-intensive and therefore often not feasible.

Here we present the Automatic Curve Extraction (ACE) framework that automates the labor-intensive extraction work for rainfall strip charts and paper rolls. Although the framework is developed for rainfall it can be suited for other elements as well. The framework consists of four basic steps: (1) scanning of the strip charts and paper rolls to digital images with a fast document scanner, (2) applying the curve extraction software in a batch process to determine the coordinates of the lines on the images, (3) visually inspecting the results of the curve extraction software, (4) using post processing software to correct the curves that were not correctly determined in step (3). This document describes all of these steps in detail.

Qualitative data is essential for good historical climate analysis. Now that historical weather data can increasingly contribute to modern climate models,

it is important to have understanding in historical weather behaviour. One of these behaviours is the amount of rainfall each day and the rain differential in time. From around 1890 until 1980, meteorologists have collected precipitation data using special rain measurement devices, so-called pluviographs, at several locations in The Netherlands. The data was recorded by automatically drawing a curve on graphical strips (containing a rectilinear grid) each day, resulting in a graph describing the total amount of rainfall versus time (see Figure 1). In later years, paper rolls where used having up to 30 days with similar consecutive grids on them.

The goal of this project and program is to extract from each digitally scanned image the curve values, representing the amount of rainfall at a certain time. Ultimately, these values are transformed into rainfall intensity values with a target time resolution of 5 minutes. The rainfall intensity resolution is somewhere around 0,1 mm (millimetres) depending on the image vertical resolution.

# Chapter 2

# Data

## 2.1 Image dataset

The complete image dataset consists of over 110,000 strip charts and more than 5000 rolls. These recordings were made from the end of the 19th century until halfway the 90s last century at 9 different geographical locations throughout the Netherlands. An example strip is shown in Figure 2.1 and an example roll is shown in Figure 2.2.

## 2.2 Metadata

In addition to the image database, two other sources of information are used. First, series information was gathered from all scanned image and stored in a small table. In this table the grid start time and length, the presence of a curved vertical axis and the grid and curve colors are stored per location and period having images with similar features. A separate table contains the average RGB values and colour component comparison thresholds for each color. This latter table is not used anymore, but this present for compatibility reasons.

A large online Microsoft Access database was constructed which contains information for each separate scanned image. Each entry contains an image ID, corresponding date and file code, the primary and processing state of the image, some notes, the grid start time and the last edit time stamp together with the responsible user and location.

While the before mentioned information sources are present before any processing is performed, in the end for each image additional data will be stored. In the next chapter, this will be extensively described.

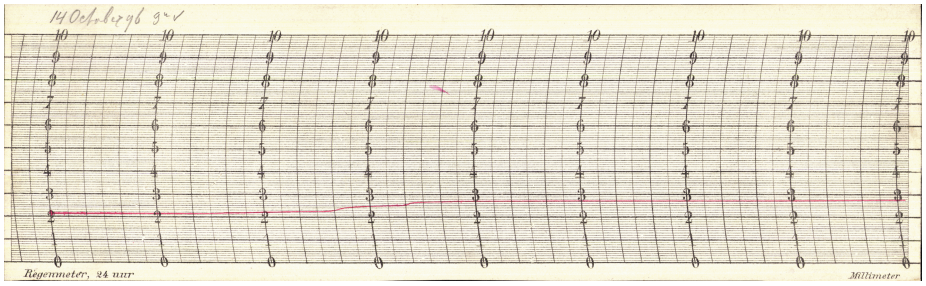


Figure 2.1: Example strip recorded in The Bilt at the KNMI on the 14th of October 1896. As is clear, the vertical axis is curved and the image is slightly tilted. The grid and curve colours are black and red respectively.

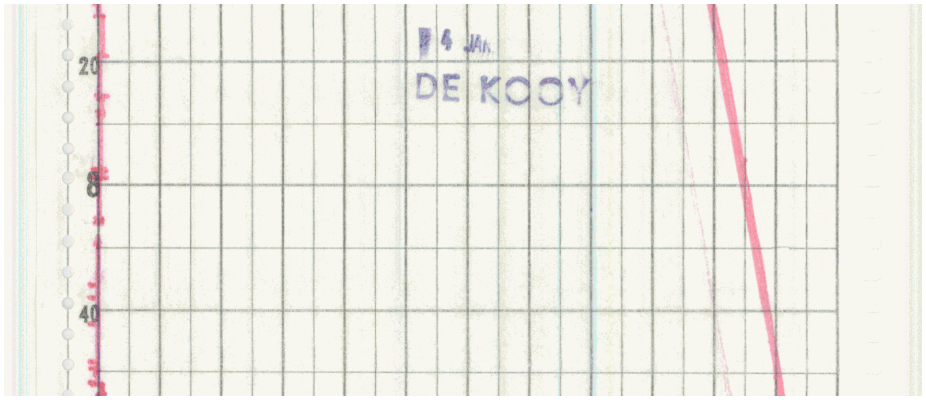


Figure 2.2: Example part of a roll image. The complete image is almost 250 times as large in vertical direction. As is clear, the image orientation is different, as is the resolution in both directions.

# Chapter 3

# Framework

The Automatic Curve Extraction (ACE) framework was built specifically for ease of use, speed, portability, precision and most importantly, data output. To achieve a successful detection and conversion of the image data into usable rainfall tables, a certain workflow is necessary requiring specifically designed tools. Typically, images can directly be processed using the curve extraction program, but in case of roll images, prior pre-processing (digital cutting) and marking is required. After processing, the result can be visually inspected and if necessary corrected. Ultimately, the resulting data is assembled into rainfall intensity tables. Figure 2 schematically shows this workflow in far more detail. These details will be discussed in Chapter 4.

## 3.1 Applications

Several applications have been developed to ensure successful detection and processing of the images. The initial detection and processing is done by a command-line interface (CLI) application, as is the parsing of the data output into usable tables. The pre-processing of the roll images and post-processing of the curve detection results are done in graphical user interface (GUI) applications. Below a table is shown which enumerates all programs and will give the relevant section in which an extensive description is given about the program.

| Name | Description | Interface | Section |
|---|---|---|---|
| Roller | Pre-processing of roll images | Graphical user | 6.2 |
| CurveExtractor | Processing of images | Command-line | 6.3 |
| PostACE | Post-processing of detection results | Graphical user | 6.4 |
| ParseTracks | Detection result parsing | Command-line | 6.5 |

Table 3.1: The main applications used throughout the ACE framework.

Some smaller tools have been developed to analyse and transform data, but these are not part of the main ACE framework.

## 3.2 Application design

The framework is entirely written in C++ using the Standard Template Library (STL) classes and some MMX code. For the graphical user interface programs, the MFC library version 6.0 was used. While these library classes require a license in order to be used, they assured rapid development and a broad range of functional possibilities for the applications. To able to load JPEG images, the free IJG JPEG library[1] was implemented which has a special license (Section 3.5). The principal design tool was Microsoft Visual C++ Studio version 6.0 SP6.

As far as possible and suitable for the design, the applications were written using an object-oriented design approach. This holds for the storage and call of image and database data, the core algorithms are implemented using a faster, direct access approach. This ensures a fast and precise execution of the programs.

While most imaging programs are designed to completely load an image at once, due to the size of the bitmap images and slow access over networks, a streamed bitmap class was implemented. In short this means that only the currently used (or visible) section of the images is loaded into memory, thereby significantly reducing the load times and required memory resources. All programs involving image loading and saving have this functionality implemented. If the workstation holds sufficient memory and if the operating system supports it, the image data will be cached into memory leading to quick image display if the requested section has been displayed before.

## 3.3 Institute specific implementation

The ACE framework is a set of tools standing on their own, but in its design it has been adapted to the infrastructure and database structure of the Royal Netherlands Meteorological Institute (KNMI). Therefore, if external implementation is wished for, thorough internal adaptation will be necessary. This can be easily achieved because of the straightforward design of the applications.

Example institute specific implementation design choices are:

- Processing is focused on pluviograph recordings
- The database structure and name, table column names and internal connections
- File names of scanned image and data files contain specific information (see Chapter 7 and 8

[1] `http://www.ijg.org`

- Some system environment variables are used

- Many batch files and commands are adapted to the data storage structure

## 3.4 Additional documentation

Besides this general documentation about the framework and project, two additional supplements exist. First, the so-called code documentation describes the structure and functions of the code which form the programs and applications after compilation. Second, the manuals of the graphical user interface programs provide information on how to use the various functionalities and how to maintain an effective workflow.

The code documentation is automatically generated from the code source itself. Almost all functions and methods have descriptions and parameter explanations defined directly above them in the function declaration header files. The compiled code documentation can be found in the source directories in HTML webpage form. In addition, the code within the functions themselves have additional comments to help the programmer understand the structure and workings of that particular function.

For the autogeneration of the code to work, a document markup language is used called Doxygen[2]. By parsing the source code with the Doxyfiles provided with the code using Doxywizard, the HTML webpages and images will be automatically created.

## 3.5 Licensing

The ACE framework and its internal programs and applications are licensed under the GNU General Public License. It is provided 'as is' but with no warranty.

ACE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version. A copy of the GNU GPL is provided in the source archive.

For the IJG JPEG library the following legalese holds: The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

The IJG JPEG software is copyright (C) 1991-2009, Thomas G. Lane, Guido Vollbeding. All Rights Reserved except as specified below.

[2] `http://www.doxygen.org`

# Chapter 4

# Methodology

To successfully process an image and detect its features, several steps are involved which require careful attention. In this chapter these steps are discussed in detail, and possible problems, solutions and alternatives are discussed. For ease of exposition, most of the time, strip charts are used as reference. However, the methodology is equally applicable for the roll images with the exception of the grid detection step.

## 4.1 Scanning strip charts and rolls

Strip charts are scanned using the Canon DR5010C transit scanner, capable of scanning at a speed of 20 charts per minute at a maximum resolution of 600 dots per inch (dpi). Since physically cutting the rolls was not allowed, the Contex Chameleon G600 scanner, designed for large image sizes, was used to scan the rolls with a speed of 150 centimeter per minute at 300 dpi resolution. Using these resolutions, a number of 10 to 15 pixels for the curve and grid line thickness is achieved, which is considered sufficient. The curves can now easily be distinguished from the background and the center of the line can be accurately estimated.

Due to the large number of strip charts and rolls, file storage requirements are a concern. Together with the output data resulting from the framework, the necessary amount of data to be stored has become rather large (over 3TB). Considering this in advance, the strip images are stored in high-quality JPEG format to save storage space. Since the JPEG image image format is a lossy compression codec, it can be assumed that some details will be lost. Fortunately, this loss is minimal and all features in the images can still be successfully detected. The roll images are stored in Windows Bitmap (BMP) or similar TIFF format using a 256-color palette to save storage space. In addition, this allows fast and partial loading of these images.

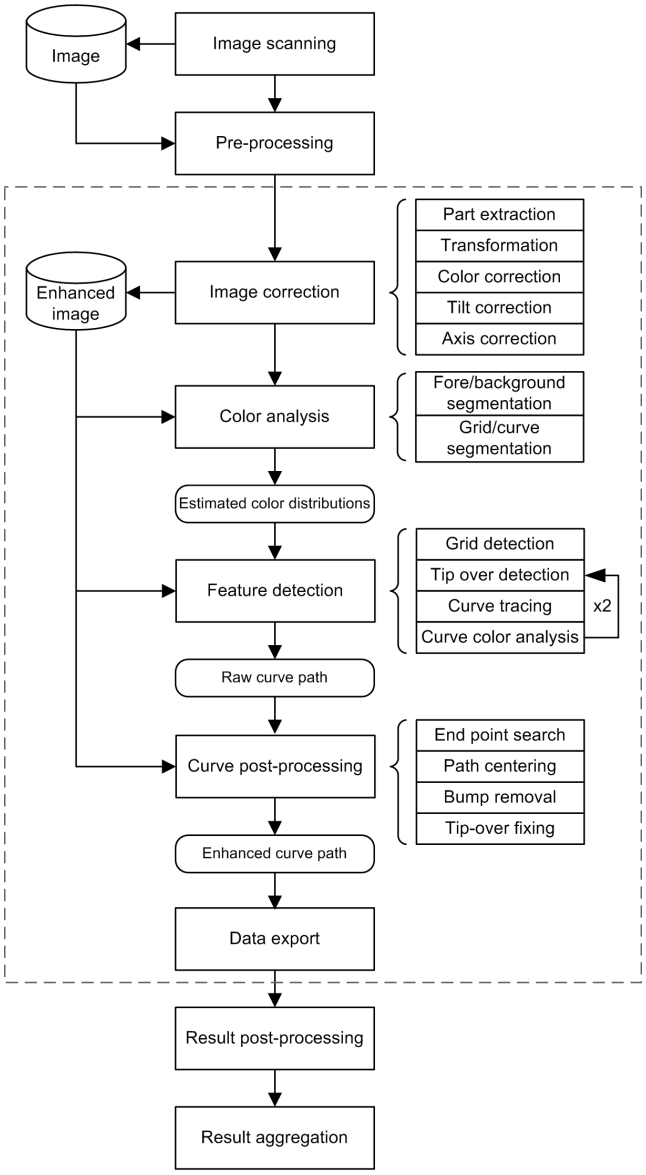


Figure 4.1: Schematic overview of the processing of an image in the framework. The sub-steps of the image correction step are optional or applied as-needed. Steps within the dashed box are fully automatic, and therefore can be applied as a batch job. See text for details.

## 4.2 Pre-processing

For each measurement location, periods with identical features (e.g. the start and end times of the graphs and the presence of a curved axis) are described in a database and used later on in the processing of the images.

A property of the rolls is that they mostly contain multiple day recordings in a sequence. For a correct detection of the rainfall values on the rolls, the exact start and end date and time of the roll recording are needed, as well as the exact locations of the day transitions and the upper and lower grid boundaries. Sometimes, only arbitrary time markings made by the observer are available. Before the application of ACE, the roll images were digitally cut to make sure that each day on these images is processed correctly. To avoid the error-prone automatic detection of these lines and markings, the day transitions and grid boundary positions were identified manually. For this specific task, the GUI application Roller was developed (see Table 3.1). Roller is capable of handling the large bitmap images, digitally cutting them and to provide the necessary tools to accomplish the marking of the marker and grid positions.

## 4.3 Processing and detection of image features

The main part of the ACE framework is the detection of image features and the conversion of pixel coordinates into usable rainfall intensity values. This is an extensive and sophisticated process and is described in Chapter 5.

## 4.4 Semi-automatic post-processing

The results of ACE are examined using PostACE. We developed this software to visually inspect the results of the automatic extraction and to adjust the found features in the images if needed. If necessary, the extraction with ACE can be repeated within PostACE using user-defined image features, such as altered tip-over moment positions.

In case the trace still doesn't match the exact position of the curve found on the image, correction lines can be drawn to force the curve point's vertical positions across a longer interval. In addition, polygons can be drawn to exclude the enclosed pixels during the curve tracing phase by setting the corresponding probability values to zero (background).

## 4.5 Result aggregation

Finally, the extraction results of the individual images are combined into a rainfall intensity table with arbitrary time resolution and time range. In our case, the time resolution was set to 5 minutes, corresponding to a width of roughly 30 pixels for the strip charts and 60 pixels for rolls. Using the start and

end points of each curve with corresponding times, the exact pixel positions of the 5 minute intervals are determined so that each interval exactly matches one of the 12 intervals in an hour. The actual difference in pixel height within each interval corresponds to the amount of rainfall during that period and thus the rainfall intensity.

# Chapter 5

# Detection and processing of image features

Here we present a detailed description of several procedures involved in the development of ACE (dashed box in Figure 4.1). Most of the following text is directly taken from [3].

## 5.1 Image correction

Image correction for rolls starts with the extraction of the correct part of the scanned image, since only a single day is processed at a time. Images are transformed automatically to the common strip orientation as shown in Figure 2.1. Optionally, color correction can be applied if images do not have enough contrast to successfully complete the detection process.

Tilt correction utilizes the repetitiveness of the recurring horizontal lines of the grid in vertical direction to measure the necessary local shift. First, a suitable reference column is located in the central part of the image. All other pixel columns will be vertically aligned with this reference column. By cross-correlating target columns every 200 pixels in horizontal direction with this fixed reference column and finding the optimal vertical pixel shift leading to the best overlay match, the whole image can be straightened, eliminating any tilt if present. For every chosen column, neighboring columns within a specified interval of 150 columns are checked to select the most suitable column for comparison. This is accomplished by autocorrelating these columns with themselves. If correlation is high compared to neighboring columns, the column is probably located at a vertical grid line in the image. If correlation is low, the column is suitable for comparison. The column with the lowest autocorrelation is selected as the column to be used for shift measurement. Once the shifts have been found, the set of shifts is filtered so that any erroneous peak shifts are removed and a smooth correction is realized. Ultimately, the shifts are

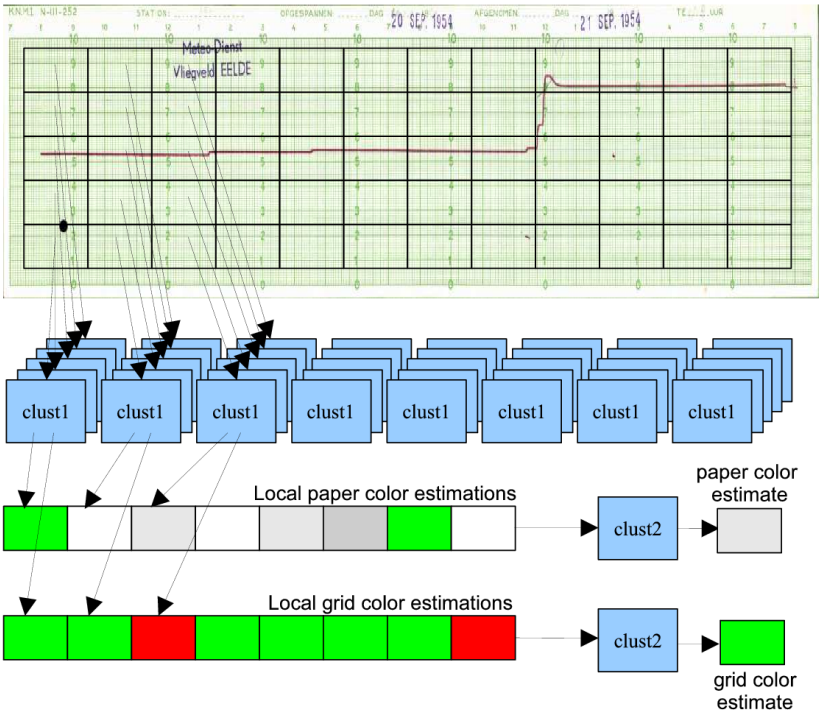


Figure 5.1: Schematic overview of the automatic color modeling process. The strip chart is divided into blocks placed centrally on the image. Of each block the grid and paper pixels are clustered into two segmentation groups (clust1) of which the mean colors are calculated and grouped into the local color estimations (clust2). The global grid and paper colors are derived from these groups. See text for a detailed explanation of each step.

interpolated and applied to each pixel column of the image.

Vertical axis correction involves similar calculations. Instead of columns, all pixel rows are cross-correlated with a suitable reference row located centrally in the image. The curve of the vertical axis can be described by a second degree polynomial. We fitted shifts to this function resulting in a smooth correction. Note that this step is only performed if the metadatabase indicted that the specific chart displays curvature of the vertical axes.

## 5.2 Foreground/background segmentation

The color distributions of grid and paper are robustly estimated in a two-step clustering process, guided by four assumptions about relative coverage and intensity of blank paper and grid lines:

1. Blank paper covers about 90% of the surface between grid lines.
2. Blank paper is brighter than anything else in the image.
3. Grid lines cover about 10% of the surface within a grid area.
4. Grid lines are darker than anything else in the image.

Assumptions 1 and 3 are not strict. ACE should be able to handle a reasonable amount of deviation from the expected coverage. Assumption 2 holds for all of our examples, because adding ink to paper usually leads to a reduction of brightness. However, assumption 4 only holds when an image contains nothing else but grid and paper. It is often violated when a piece of curve, handwritten text or markers are present as well. This is why we have split the color estimation process into a cascade of two clustering procedures. We first estimate grid and paper color in Small Grid Blocks (SGBs) of the image, and, in the second step, apply clustering to the outcomes of all SGBs. This is based on the fifth (and crucial) assumption:

5. The grid area of an image can be broken up into square blocks, of which more than 50% only contain grid and paper, without a curve, markings or handwriting.

Assumption 5 means that assumptions 1 to 4 hold for at least 50% of SGBs. For these blocks, paper and grid color estimation will be reliable. For the remaining blocks, successful color estimation cannot be guaranteed. By subsequently clustering the estimated mean paper and grid colors of all SGBs, the failed SGB outcomes are discarded and successful results can be combined to increase estimation accuracy. Figure 5.1 shows a schematic overview of the color modeling process. In the following paragraphs the whole process is described in detail.

**Clust1: SGB color modeling** The size of SGBs is chosen at least as large as one period of the grid pattern (both horizontally and vertically), so that the thicker grid lines are always within a block. Larger blocks would not be beneficial, as they would result in a higher percentage of blocks containing a piece of the curve or other ink from stamps or handwriting. The SGBs are only extracted from the middle region of the image, to ensure that they actually contain grid lines. It is possible to extract overlapping blocks from an image in order to maximize the percentage of blocks that do not contain anything but blank paper and grid. However, this results in more blocks and thus higher computational load. This did not seem necessary for our data. Here we only used adjoining blocks, which was sufficient for our data.

The first step of clustering is applied to every individual SGB. The clustering procedure is a modification of Expectation Maximization (EM). In a number of iterations (5 iterations turned out to be enough to converge to a stable result), the estimates of the mean and covariance of grid and paper color are updated by re-computing the mean color of each cluster. A good initialization of paper and grid color is crucial to a successful clustering. The initialization of paper

color $c_P^{(0)}$ is done by computing the mean color $\bar{c}(B)$ of the brightest 90% of pixels $B$ in the SGB. Similarly, the initial grid color $c_G^{(0)}$ is found by the mean color $\bar{c}(D)$ of the darkest 10% of pixels $D$ in the SGB. Brightness $I$ of a pixel $x$ is defined as

$$I(x) \quad = \quad \frac{r(x) + 2g(x) + b(x)}{4} \tag{5.1}$$

where r(x), g(x) and b(x) are the red, green and blue components of $x$, respectively. This is because multiplications and divisions by two are executed much faster in digital hardware, and most of current imaging devices use a ratio of two green sensors for every pair of blue and red sensors. This means that green is more reliably measured than red or blue. The mean color $\bar{c}(S) = [\bar{r}(x_i), \bar{g}(x_i), \bar{b}(x_i)]^T$ of the colors $c(x_i) = [r(x_i), g(x_i), b(x_i)]^T$, $i \in S$ of the pixels in set $S$ is defined as

$$\bar{c}(S) \quad = \quad \frac{1}{N} \sum_{i \in S} c(x_i) \tag{5.2}$$

Each iteration $k$ starts with re-grouping SGB pixels to the paper $P^{(k)}$ and the grid $G^{(k)}$ clusters. This is based on the Euclidean distance to the respective estimates of mean colors in RGB space. To be robust against deviations from the expected coverages, only the closest half of the respective expected coverage is used. This means that the paper cluster $P^{(k)}$ is formed by the 45% closest SGB pixels to $c_P^{(k-1)}$, while the grid cluster $G^{(k)}$ is formed by the 5% closest SGB pixels to $c_G^{(k-1)}$. Finally, the colors are re-estimated by $c_P^{(k)} = \bar{c}(P^{(k)})$ and $c_G^{(k)} = \bar{c}(G^{(k)})$.

$c_P^{(k)}$ and $c_G^{(k)}$ of the last iteration are used as the estimated paper $c_P(m)$ and grid $c_G(m)$ color, respectively, for the SGB with index $m$. The 3x3 covariance matrix of grid color in RGB space $\Sigma_G$ is estimated over the last grid cluster $G^{(k)}$. In our scanned images, the background was often saturated. This makes it impossible to calculate the covariance of paper color $\Sigma_P$. However, most of the variation in paper, as well as grid color, was caused by gradual transition between grid and paper. These smooth transitions are caused by variations in the amount of grid ink on the chart/rolls, but also by interpolation around edges during the scanning process. This implies that the covariances of paper and grid are quite similar. Therefore, we could assume that paper color has the same covariance as grid color.

**Clust2: Combining SGB outcomes** From the procedure described above, we now have two sets of color values $c_P(m)$ and $c_G(m)$, $m \in \{1, \ldots, N_{SGB}\}$, where $N_{SGB}$ is the total number of SGBs. A single-color clustering is applied to both sets, which differs from the above procedure for the individual SGBs only in the computation of the initial color and the percentage of closest values assigned to a cluster. The initial color for each set (paper and grid procedures are now identical) is estimated as the mean color of the entire set, and the 50%

closest values are now used for assigning values to a cluster. The SGB grid covariance matrices $\Sigma_G(m), m \in G(k)$ which correspond to the grid color values that are assigned to the grid cluster after the final iteration, are averaged per element, to obtain an overall estimate of the grid color covariance.

**Grid likelihood** For estimation of grid borders, it is important to have an estimate of the grid pattern. The likelihood of a pixel $x$ belonging to a grid line is estimated by:

$$L_G(x) = 255\left(1 - \min(1, \frac{||c(x) - c_G||}{||c_P - c_G||})\right) \tag{5.3}$$

where $||\cdot||$ denotes the L2 norm of a vector (Euclidean distance in RGB space) and $\min(a, b)$ equals $a$ for $a < b$ and $b$ otherwise. The normalization with the distance between paper and grid color ensures that paper color is given a likelihood around 0, while the multiplication with 255 makes the results of a whole image suitable to be saved as a typical 8-bit monochrome image. Since this measure does not correspond to actual statistics, it has to be seen rather as a fuzzy classification than as a likelihood.

**Background likelihood** Now that we have estimated statistics of paper and grid color, we can use this information to recognize anything that looks like neither of them (the foreground). Since the curve is drawn with different ink, it will be part of this foreground. Therefore, foreground likelihood will help to locate curve pixels.

The Mahalanobis distance is used to take into account the directional dependency of the variance of background color in RGB space. Simply combining the Mahalanobis distances to mean paper $c_P$ and grid color $c_G$ will lead to classifying the smooth transitions between grid and paper as foreground. This is because pixels on the smooth edges are unlike grid color, but also unlike paper color. Instead, we use the knowledge that the colors on these soft edges are a linear combination of grid and paper color (assuming the smoothing is the same in all color channels). Instead of computing the Mahalanobis distance to $c_P$ and $c_G$, a reference background color $c_{GP}(x)$ is defined as the projection of a pixel onto the vector between $c_G$ and $c_P$:

$$c_{GP}(x) = c_G + \hat{v}_{PG} \cdot \max(0, (c(x) - c_G)^T \hat{v}_{PG}) \tag{5.4}$$

$$\hat{v}_{GP} = \frac{c_P - c_G}{||c_P - c_G||} \tag{5.5}$$

where $\max(a, b)$ is $a$ for $a > b$ and $b$ otherwise. The max function ensures that when the projection falls below $c_G$ (darker than grid color), $c_G$ is used directly as the background color. Now, the unnormalized foreground likelihood $L^*_{FG}$ is calculated as:

$$L^*_{FG}(x) = \frac{\sqrt{(c(x) - c_{GP}(x))^T \Sigma_G^{-1} (c(x) - c_{GP}(x))}}{I(x)} \tag{5.6}$$

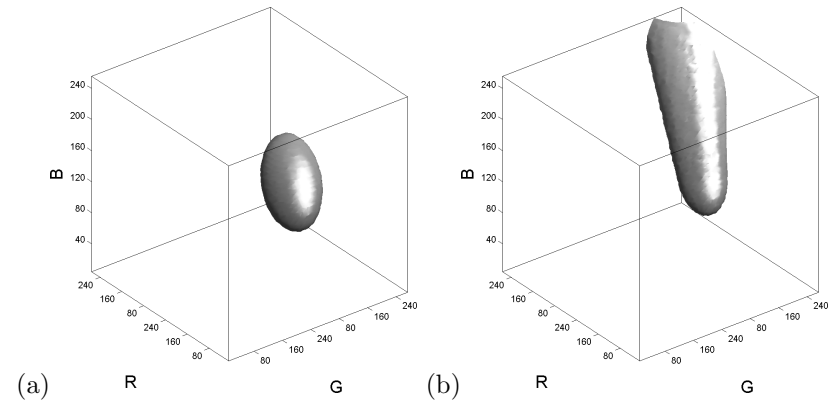


Figure 5.2: Isosurface of likelihood in RGB space, in (a) for the Mahalanobis distance to mean grid color and in (b) for the foreground likelihood model $L_{FG}(x)$

The division by brightness is done to allow more variation of brighter background colors. This assumes that color variance scales with brightness. The difference between this model and a straightforward Mahalanobis distance is shown in Figure 5.2. In (a) an elliptical surface in RGB space is plotted that has an equal Mahalanobis distance to the mean grid color. (b) shows a surface in the same space that has an equal foreground likelihood as calculated by $L^*_{FG}(x)$.

To effectively use this background likelihood model, it is crucial to scale it in a way that results in comparable likelihood values for all images, independent of color variance or the estimation thereof. This is done in a robust empirical estimation of the boundary value $L_{FG}^{(maxBG)}$ of $L^*_{FG}(x)$ over the SGBs that were selected for the grid color cluster $G(k)$ in the last step of the clust2 procedure for grid color. For each of these SGBs, the 99th percentile of $L^*_{FG}(x)$ is determined over all pixels in the block. A clustering of these percentiles is done that is identical to the clust2 procedure, resulting in $L_{FG}^{(maxBG)}$ as the mean of the final cluster.

Besides being scaled, the foreground likelihood is also mapped by a Gaussian function centered at 0. This is because differences between colors with a large distance from the background colors are relatively less important. A Gaussian mapping reduces likelihood differences between these different foreground colors, while still keeping nuances that may be important to track the curve path through stains or other clutter or artifacts that are different from the estimated background colors. Because the resulting likelihood $L_{FG}(x)$ does not correspond to an actual probability, it has to be regarded as a fuzzy foreground classification:

$$L_{FG}(x) = 255\left(1 - \exp\left\{-\frac{1}{2}\left(\frac{\alpha_{BG} L^*_{FG}(x)}{L_{FG}^{(maxBG)}}\right)^2\right\}\right) \tag{5.7}$$

The scaling with 255 is again to be able to save the result of the likelihood computations as an 8-bit monochrome image. $\alpha_{BG}$ is a factor that scales the maximum background likelihood to a certain factor of the Gaussian standard deviation. $\alpha_{BG} = 0.4$ was chosen as an educated guess. This value should keep most of the background likelihood values on the high part of the Gaussian, preventing large contrasts in the likelihoods between different background pixels, while achieving a high contrast between foreground and background likelihoods.

## 5.3 Grid detection

In order to transform the curve pixel coordinates into usable rainfall and time data, it is necessary to know the exact locations of the grid boundaries which represent the known start and end times on the horizontal axis and the range of rainfall on the vertical axis. Using the grid likelihood image, the locations are found by estimating the mean distance between the grid lines. For each row or column the sum of likelihoods of all pixels is calculated. To make sure that slightly skewed grid lines are still incorporated in the calculation, for each row or column the sum of an imaginary skewed line with a maximum skew of 20 pixels centered on that same row or column is regarded as well. From these sum values the local maxima are extracted that exceed a sum value threshold of $L_g$ times the row or column length, respectively. Initially, $L_g$ is set to 50. All maxima arising from small successive row or column variations are discarded. Now, the mean mutual distance between two successive maxima and its standard deviation indicate how well the initial distance estimation fits the image. If the standard deviation is larger than 2 pixels, the estimation is probably wrong and most likely too many maxima are taken into account. Therefore, with a raised threshold $L_{\mathrm{g,new}} = L_g + 1$, a lower number of maxima is taken into account. With this new set of maxima the distance estimation is repeated. As soon as the estimation is accepted, the locations of the outermost maxima having a distance comparable with the mean are marked as the outer boundaries of the grid. If no good estimation can be found, the first maximum is used as the boundary location.

## 5.4 Tip-over detection

Before the actual curve is traced, the tip-over moments need to be detected, since they cause the curve to be discontinuous. The detection of these tip-overs involves similar methodology as in the grid detection step. In the curve likelihood image, a tip-over manifests itself as an almost vertical dark line. The pixels of each column in an image are summed, and if any sudden negative peaks in total column sum are found, they most likely correspond to one or more tip-overs. To make sure that the more skewed tip-overs are taken into account as well, an empirically estimated skew of 24 pixels regarding the whole line length is allowed. A drawback of this method is that sometimes vertical time markings

made with the curve pen color are mistaken for tip-overs. During the initial tracing in the next step they will be treated as such, but during the secondary post-processing step they will be filtered out automatically and only the actual tip-overs should remain.

Another drawback is that there is a probability that less visible tip-overs are missed during detection. These are not taken into account, preventing a correct curve trace. Manual addition of these tip-overs during post-processing solves this issue.

In rare cases, tip-overs are retrograde (i.e. their skew is negative). The strip chart image and curve likelihood image are corrected for this artifact by moving all pixels prior to the tip-over horizontally to the left. This way, the tip-over skew is corrected and the curve tracing will not skip any parts positioned on the chart at the same time. Since this alteration is only small, the shift in time is not an issue. Prograde tip-overs (the normal situation) are not corrected since these cause no problems during tracing and skews are usually small (depending on the time needed for emptying the reservoir) and therefore have negligible influence on the outcome.

## 5.5 Curve tracing

A dynamic programming algorithm is applied to the curve likelihood image to trace the curve on the image. The algorithm is capable of tracing a continuous curve in right-to-left direction, using the principle of optimality ([1]). It solves a globally defined problem by finding the optimal solution of the individual smaller subproblems which together compromise this global problem. In horizontal direction, the image is divided into sections separated by the tip-over moments found in the previous step. If no tip-overs were found, the whole image is treated as one section. For each section, the minimal cost path is determined, using the pixel likelihood values as cost factors. In a mathematical sense, the minimal cost path is determined as the sum of all the minimum cost sub-steps between consecutive pixel columns:

$$\min\left[C(x_1, x_2, \ldots, x_M)\right] = \min_{j=1,\ldots,N}\left[C(x_M^j)\right] \tag{5.8}$$

with $C(x_i^j)$ the total accumulated cost at pixel $(i, j)$, $M$ the total number of columns and $N$ the total number of rows in the image ([2]). $C(x_M^j)$ are thus the pixels in the last (furthest right) column containing the total sum from left to right. Starting from left, between each pair of consecutive columns the minimal cost step from a certain pixel of the right column, the node, to a pixel on the left column is determined (Figure 5.3). A user-definable change in the y-coordinate is allowed with this step, in our case a search width of 25 pixels, centered on the node height. Note that this implicitly limits the detectable steepness of the curve and the ultimate maximum rainfall intensity that can be determined automatically. The accumulated cost of the pixel on the left is added to the

value of the pixel on the right and the corresponding optimal step (change in height) is remembered. This is repeated for every column pair. In the end, each pixel in the outermost right column has the sum value of the minimal path from the left to that particular pixel on the right. Since the coordinate of the best neighboring pixel is known for each pixel, the minimum cost path can be traced back from right to left, starting from the last column pixel with the lowest total sum. Since the curve image might be noisy, the traced curve path most likely does not represent the actual path of the curve. It also runs accross the entire image instead of starting and stopping at the actual curve end points. This will be dealt with later on during post-processing.

### 5.5.1 Dynamic programming

Dynamic programming is an optimization method based on the principle of optimality. It searches for optima of functions in which not all variables are simultaneously interrelated. In this case the aim is to find the best path (minimum cost) between the starting point A (one of the pixels in the most left column) and end point C (pixel in the most right column). The cost is in this case represented by the colour component value of each pixel. The principle of optimalitys main point is that when the optimal path between A and C goes through B, then its parts A-B and B-C are also optimal (for illustration see Figure 5.3).

To translate this into an algorithm, the image is traversed column by column, starting with the second column from left. For every pixel in the considered column, the minimum cost path to a pixel in the previous column is searched, using a specified search radius. To make sure that a continuous path is found, the value of the pixel having the minimum value in the search range is added to the pixel considered. When the next column is processed, then the influence of the previous column is still apparent. Also the difference in pixel coordinate height is stored to later be used as a pointer.

If all columns are processed, the minimum total cost can be found at a certain location in the last column. Since for each location the path direction to its best predecessor is known, the minimum path can be traced back. For an illustration, see Figure 4.

## 5.6 Initial curve post-processing

Because the curve now follows the exact minimal cost path, which is not necessarily the best solution, the result must be improved by repositioning the curve points to the nearby local vertical line center of the curve on the image. This line center is found by measuring the weighted sum of the intensities of the background pixels found in the row part centered to the current pixel position in the curve likelihood image. The row with the minimum background pixel intensity is most likely the vertical line center. We derived a mathematical formula which

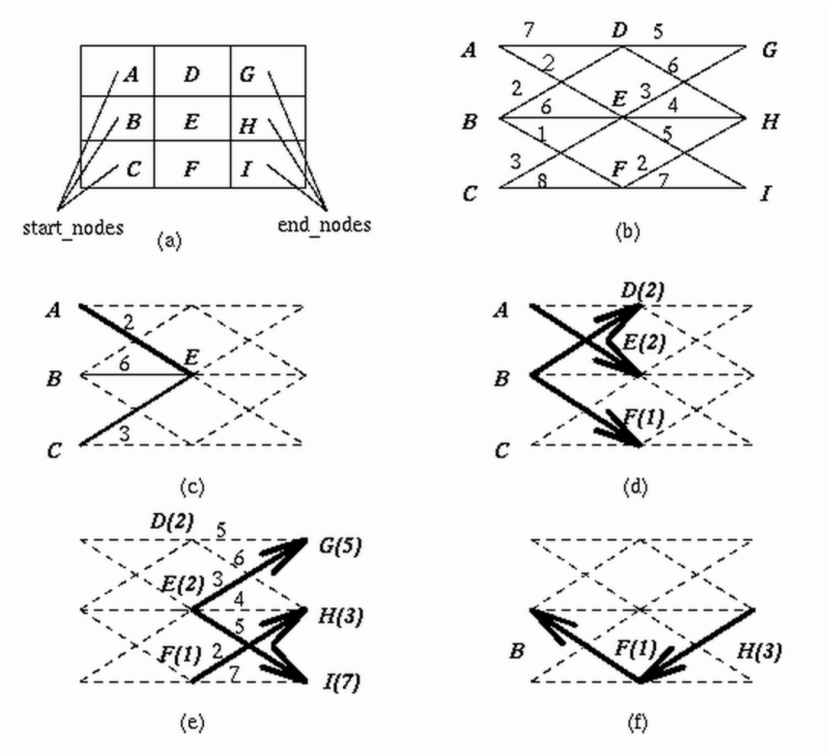


Figure 5.3: Imagine a 3x3 image with coordinates or nodes A-I (a). To find a path from left to right each node has a value or cost (b). To find the minimal cost path, the best route from each consecutive column pixel to its predecessor must be found, here for example from E (c). After this has been determined, the optimality (in value and direction) is known for each column pixel (d). Repeat this for the last column and add each found cost to this column, remember its directions (e). Now the most efficient route is calculated by tracing from the minimum pixel in the last column back to the starting column (f).

expresses the centering of the curve point at $(x_0, y_0)$:

$$y' = \min_{y=-N,\ldots,N} \left[ \sum_{x=-M}^{M} p(x_0 + x, y_0 + y) w(x) \right] \tag{5.9}$$

with $y'$ the adjusted vertical line center, $p(x, y)$ the intensity of the pixel at coordinate $(x, y)$, $w(x)$ a ramped weight factor for a pixel at position $x$:

$$w(x) = 1 - |x/M|. \tag{5.10}$$

$M$ is the number of pixels regarded horizontally in each direction and $N$ the number of rows regarded vertically in both directions.

The path of the curve still traverses the whole image from left to right. For strip charts the curve itself almost always starts and ends at a different time on the grid, while for rolls it starts and ends at the left and right image borders. Therefore, the current path of the curve on the strip charts needs to be trimmed from all pixels not belonging to the curve, which are located outside of the curve begin and end points. In this initial step, an initial estimate of the positions of these points is derived from the series database (Chapter 2) together with the known coordinates of the grid.

## 5.7 Curve color remodeling

The current state of the path is used to adjust the color model for the curve pixels. From all pixels belonging to the curve path a new mean color is calculated. The Euclidean distance is used to remove pixels located far from the mean. This improves the mean resulting in a refined estimate of the curve color. Using the corresponding histogram, the curve likelihood image is reconstructed. The tip-over detection and curve tracing are repeated using the steps outlined in the previous paragraphs. In many cases this results in an improved estimate of the curve path.

## 5.8 Secondary curve post-processing

Secondary curve post-processing is now applied to the newly traced curve. Again, the start and end point of the curve need to be detected for the strip charts. This time, morphological operators are applied to the path to detect the current start and end points of the curve. Since the segmented path may contain gaps of undetected curve, the first path segment from the left that is longer than 150 pixels, is assumed to be part of the curve line and its outmost left point is marked as the curve start point. The same is repeated for the outermost right part. Here, the outmost right point of this segment is marked as the curve end point. Again, any path points not belonging to the curve are excluded in further calculations. This step is followed by the alteration of the path to the center of curve line, as described in the initial post-processing step.

A common artifact is the presence of bumps in the path. These emerge because of the changes in color of the curve on the image (e.g. when running through a grid line) or because of the presence of time markings. The removal of these bumps is accomplished by convolving the traced path by a continuous morphological opening and closing of a local part of the curve. Events, such as the ends of a line and tip-over moments, must be treated as if the line on one side of the event continues horizontally over the event. This means virtually replicating the last value during filtering near the events. Otherwise, these desired extremes will also be cut-off. Figure 5.4 illustrates this idea. The only parame-

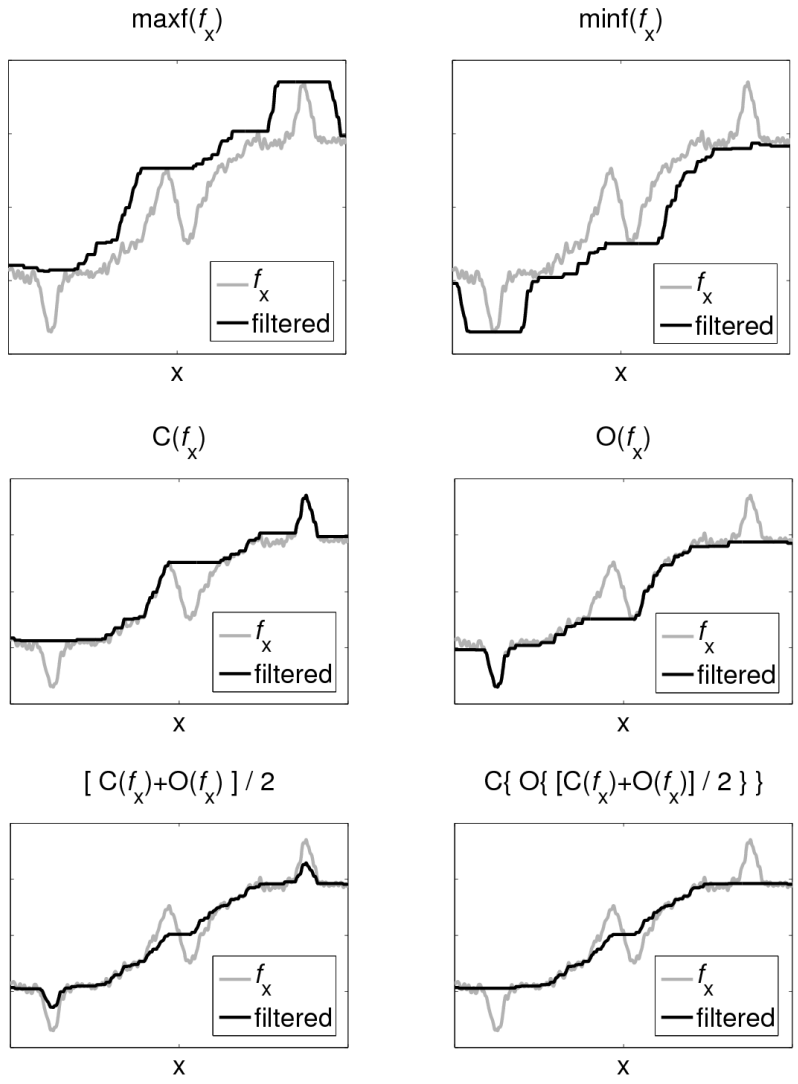


Figure 5.4: Mathematical display of the equivalent of an opening and closing applied to the curve initially found on the image. On the left is the original path, on the right the outcome after application. Two separate processing path ways are displayed: the first row applies a maximum operator followed by a minimum operator. The second row applies the same but in reverse order. The outcome curve is the average of the results of the two pathways.

ter to choose here is the convolution window size, which must be larger than the largest bump present in the curve. The bump removal operation preserves the

exact shape of monotonically increasing or decreasing signals and is therefore a safe way to improve the quality of the trace. Furthermore, a relatively large window size can be chosen for cumulative measurements (e.g. rainfall) since the change in height of the path is not altered.

If there are any erroneous steps (changes in vertical height) present in the path, a correction is applied to ensure a continuous path. By calculating the derivative of the path, any sudden changes in path height are detected by extracting the coordinates corresponding to peak values in the derivative. A second degree polynomial is fitted to the neighboring 40 pixels of the corresponding curve point $(x_0, y_0)$ to get a good estimate of the direction of the path around that peak. The result is applied to the original path:

$$y(x) = y(x)w(x) + y'(x)(1 - w(x)) \quad \text{for } x \in [x_0 - M, x_0 + M] \tag{5.11}$$

where $y(x)$ is the original path, $y'(x)$ is the fitted path and $w(x)$ is a weight factor according to Eq. 5.10.

The pieces of the curve close to the tip-over moments obtain special treatment: a predefined number of curve points left or right next to the tip-over moment will be positioned according to the average increase in vertical position of the curve points before or after these points (respectively). Any tip-overs whose left and right path have a small difference in height are probably not tip-overs and are therefore removed.

# Chapter 6

# Program execution

In this chapter the running and operation of the command-line interface programs are briefly discussed. For an in-depth explanation of the usage of the graphical user interface (GUI) programs, please refer to their respective manuals.

All programs are designed to run in Microsoft Windows XP, and are expected to run in newer versions of this operating system. Other operating systems are currently not supported, but the command-line interface programs are expected to run without much code adjustment —though recompilation is necessary— in other operating systems as well. The GUI applications need a complete rewrite regarding the database connections and graphical elements. The latter could have been avoided by using a different interface framework. The currently used framework (Microsoft Foundation Classes (MFC)) might be replaced in future versions of the ACE framework.

## 6.1 Installation

All programs are provided without installation tools or similar. All programs are able to run from any directory provided that the paths in the configuration files (Section 7.4) are properly set up. For the graphical user interface applications an ODBC connection to the named database in required.

## 6.2 Roller

To digitally cut and mark the roll image files prior to curve extraction, Roller provides the tools needed to accomplish these tasks. It is a graphical user interface application as is shown in Figure 6.1. At any vertical position horizontal markers can be added, which have two vertical grid lines running to neighboring markers. Special commands make adding and positioning markers fairly easy. Roller needs a configuration file in its directory (`config.ini`) to run properly

(see Section 7.4) and a connection with the series database.

Once the roll image has been properly marked, the markings can be saved to a small Roller file which contains the coordinates of the markers and grid lines and the corresponding times (see Section 8.3).

## 6.3 CurveExtractor

The main program of the framework is the automatic curve extraction program called CurveExtractor. This program will analyze the colors and features of the input image. Most of the work and necessary steps are determined automatically by the program, either by applying prior-defined information or during the detection run. If the user demands manually set parameters or wants the program to skip certain detection steps, it is possible to enforce this through the command-line. In Table 6.1 a complete list of parameter options is given together with the default values and brief explanation. Table 6.2 enlists the available flags. CurveExtractor does need the data tables (see Section 2.2 and 7.5) located in subdirectory `data`. CurveExtractor will only run if a valid file name is argumented.

The main syntax for the CurveExtractor is:

```
extcurve filename [options] [flags]
```

where *filename* is any image file (JPEG, TIFF, BMP supported), *options* are the optional parameters (syntax: `-option value`) and *flags* are the optional flags. CurveExtractor should run without any problems if no options or flags are given. In addition, `extcurve --version` will display version information, while `extcurve --help` will display similar help information about running the program.

Two particularly important flags are the `useace` and `useacefull` arguments. These will force the program to use previously detected color, tilt correction and feature detection data. For example, `extcurve BE197101_01.jpg useace` will process the image `BE197101_01.jpg` using the file `BE197101_01.ace` as detection data input. This will cause the program to finish much faster as it will skip the color detection and tilt correction steps. Instead, the binary data in the ACE-file will be used instead. If the `useacefull` argument is used, the grid, tip-over and tracing detection steps will be skipped as well. Consequently, the program will finish even quicker.

CurveExtractor will produce several files after processing has completed successfully. Lets consider the example program execution:

```
extcurve BE197101_01.jpg
```

| Option | Description | Default value |
|---|---|---|
| sf | strip data file | `stroken.csv` |
| cf | colour data file | `colors2.csv` |
| af | forced ace file data file | *autodetect* |
| start | roll start day | 1 |
| lim | roll day count | *all* |
| width | tracing search width | 25 |
| smoothwidth | path smoothing width | *disabled* |
| contrast | image contrast | *no adjust* |
| brightness | image brightness | *no adjust* |
| gamma | image gamma | *no adjust* |
| sideext | roll side extension size | 100 |
| colrep | color analysis repeat | *disabled* |
| dbl | show debug information level | *disabled* |

Table 6.1: CurveExtractor main options. Default values are only given if applicable.

| Flag | Description |
|---|---|
| loadonly | load images only (for testing purposes) |
| extract | load and save images only (for extraction) |
| curve | save curve likelihood image |
| grid | save grid likelihood image |
| curveonly | only save curve likelihood image (no processing) |
| gridonly | only save grid likelihood image (no processing) |
| traceonly | only trace the path (skip grid, tilt, curve) |
| skiptilt | skip tilt correction |
| skipcurve | skip curve correction |
| skiptrace | skip path tracing |
| skippath | skip path post-processing |
| skipfixing | skip path tip over post-processing |
| skipstep | skip path destepping |
| skipexport | skip tracked data export |
| skipsave | don't save result image |
| skiptipover | skip tip over detection |
| skipdip | skip path dip detection |
| tiltsimple | correct tilt only, not image curvature |
| gridsimple | apply simple grid detection |
| binary | output data to binary format (deprecated) |
| afdonly | load Ace File Data only |
| metaonly | load metadata only |
| useace | use AFD data of previous run |
| useacefull | use AFD data of previous run completely |
| roll | force treatment as roll |
| halfwidth | save with half horizontal size |
| debug | show debug information |
| tables | output debug tables |
| quiet | suppress most output messages |

Table 6.2: CurveExtractor main flags.

The following description will explain the output file naming conventions used based on the input file name. First, a new image `BE197101_01.tracked.jpg` will be drawn, which has the tilt and axis corrected original image with the detected image features plotted on top it (see Section 8.1). This provides an easy way of visually checking the produced results. All internal tables are written to the ACE File Data (AFD) file `BE197101_01.ace` (Section 8.4) and the detection results are output to a Tracked Data file `BE197101_01.tracked` (Section 8.2). To be able to later find out what the program exactly output, a log file `BE197101_01.log` is created (Section 8.5). If argumented, the curve and grid likelihood will be plotted to images `BE197101_01.curve.jpg` and `BE197101_01.grid.jpg` as well (Section 8.1).

## 6.4 PostACE

As soon as the image is processed by CurveExtractor, the results can be visually inspected and adjusted in the post-processing program PostACE. This graphical user interface application provides the necessary tools to do virtually anything considered relevant with the detection results. Figure 6.2 gives an impression how the application looks like.

PostACE requires a connection with the series database (Section 7.8), the data tables (Section 7.5) and a configuration file located in its application directory (Section 7.4). PostACE will output the same files as CurveExtractor.

## 6.5 ParseTracks

The final step in using the framework is assembling the separate curve detection results into usable rainfall intensity tables. This can automatically be accomplished with the program ParseTracks. If necessary, the output resolution and other parameters can be adjusted through the command-line. In general, the output table can only be constructed for a certain time range in years, but it is also possible to generate a single date table (see below). The parameter options and flags are listed in Tables 6.3 and 6.4, respectively. To run properly, ParseTracks needs a configuration file (`config.ini`, see Section 7.4), a precipitation table (Section 7.6) and an hourly rainfall table (Section 7.7) in its directory. ParseTracks will only run if a valid location is argumented.

The main syntax for ParseTracks is as follows:

```
ptc location [options] [flags]
```

where *location* is any available location, *options* are the optional parameters (syntax: `-option value`) and *flags* are the optional flags.

ParseTracks can run in two special modes. First, if `ptc` *location* `checkexist` is executed, the program will only check which files can be located it expects to

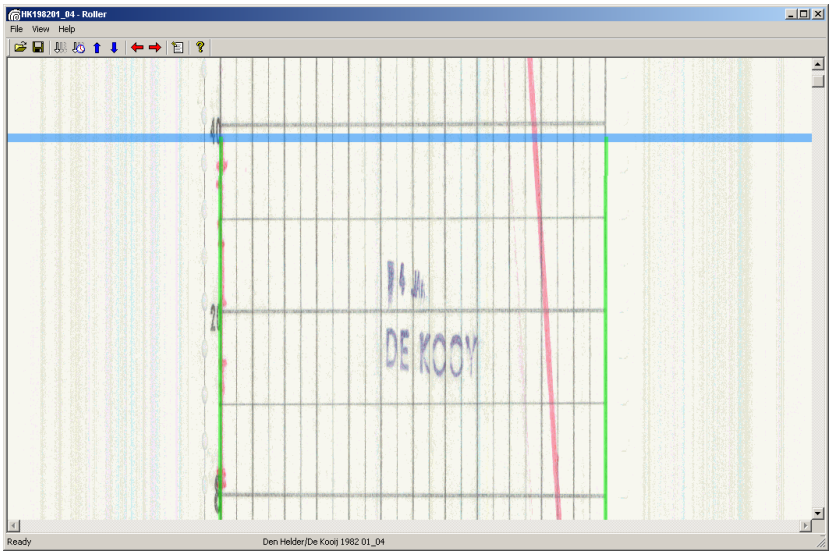


Figure 6.1: Example usage of Roller roll image pre-processing application.

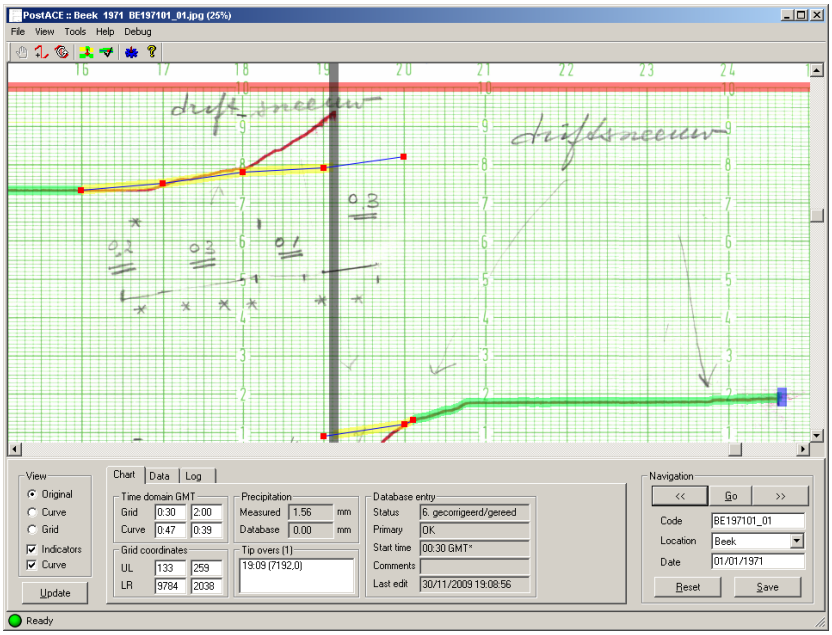


Figure 6.2: Example use of PostACE post-processing application.

| Option | Description | Default value |
|---|---|---|
| start | start year | *first available* |
| end | end year | *last available* |
| res | output resolution (minutes) | 5 |
| lim | total day count limit | *all* |
| dirlim | total directory count limit | *all* |
| hfile | hourly precipitation file | *autodetect* |
| wpl | warning precipitation level | 1 mm |
| dbl | show debug information level | *disabled* |

Table 6.3: ParseTracks main options.

| Flag | Description |
|---|---|
| metadata | load metadata only |
| report | report tracked correction |
| keepdup | keep duplicate tracked files |
| skipproc | skip actual processing |
| skipsec | skip secondary strips |
| debug | show debug messages |
| quiet | suppress most output messages |

Table 6.4: ParseTracks main flags.

find according to the series database for that particular location. Any problems are comprehensively reported. Second, `ptc` *location* `-date` *date* will extract a single date for the given location.

ParseTracks will output a rainfall intensity table in accordance with the KNMI table entry conventions (Section 8.6). In addition, a log file is created providing information about the corresponding program run.

# Chapter 7

# Input data

In this chapter the input data and files needed to run the programs and applications properly are described.

## 7.1 Nomenclature

Many possibly unfamiliar terms are used throughout this and the next chapter. The following table (Table 7.1) provids a short description about each term.

| Term | Description |
|---|---|
| Strip code | The code of a strip corresponding to the file name of the image. See Section 7.2 |

Table 7.1: Nomenclature table.

## 7.2 Strip code

Each image's file name consists of a strip code and an extension. The extension identifies the image format, while the strip code holds information about the source of the image and the time the meteorological recording was made. In general, for each recording day, a seperate strip code exists, but, as shown below, this is not always the case. A strip code follows strict convention which is applied throughout the framework. Examples, the image `DB196506_04.jpg` has strip code `DB196506_04`, while `HK198206_10.bmp` has strip code `HK198206_10`. In the following the above examples will be decomposed into their various pieces of information.

In general, a strip code is constructed as follows:

```
LLYYYYMM_SS_RR
```

where LL holds the location abbreviation code, YYYY and MM are the corresponding year and month, SS is the sequence number and RR stands for the roll image index number. An example location abbreviation code is DB, the code for 'De Bilt'. By convention, the sequence number and roll image index number run from 1 until the last number in the sequence.

Note that the sequence number does not correspond with the actual date. The reason for this is that in many cases multiple days have been recorded on one strip and for rolls, the image contains multiple days in sequence. In case of the first, the data files are distinguished by adding a single alphabetical character directly after the strip code, e.g. DB190101_01a. No particular order is expected from these characters, the series database is used to lookup the corresponding date.

In case of roll images, the addendum identifies the day image index number of all consecutive days on the roll. Obviously, for strips this number is omitted together with the underscore.

## 7.3 Input images

The main source of information for the programs are the scanned images. These images are expected to have a valid strip code, and to have the image features necessary to successfully detect the rainfall curve on the image. These features are the grid and the curve. It is expected that these grids and curves have darker pixel intensities compared to the background (paper) pixel colors. Also, the horizontal (time) axis increases from left to right, as does the vertical (rainfall) axis from bottom to top.

Supported image formats are JPEG, BMP and TIFF. Other formats can easily be implemented by creating new classes derived from the main image class. There are no physical limitations to the image formats other than the format's limitations itself, with an exception for TIFF images. Here, the data is expected to be in normal order, i.e. the pixel data is running from top-left to bottom-right. The TIFF specification does not strictly require this, and therefore this should be taken into account while loading these images. Because this complicates the streamed loading process, this was omitted and therefore the data is expected to be in normal order.

While loading the images, the image size determines whether the image will be treated as a normal strip or a roll. If the image width is larger than or equal to the image height, it is expected to be a normal strip, any other case will be treated as a roll.

## 7.4 The configuration file `config.ini`

As mentioned in Chapter 6, the graphical user interface applications require a configuration file providing them the information to successfully load and write image and tracked data, and to load the current settings regarding navigation

and display. Also, user specific settings are stored in these files. The configuration file's file name is `config.ini` by default. This cannot be changed. All settings are identified with a keyword, followed by the delimiter and its value. The delimiter is always the equal-sign '='. Each text line contains a single setting.

A configuration file for the PostACE application might look like:

```
ImageRoot=Z:\stroken_scans
ResultRoot=Z:\stroken_tracked
DataRoot=Z:\ACE\data
Database=metadata
Location=AM,Amsterdam,1
Location=VL,Vlissingen,310
ColorTable=0,255,64,0,255,0,0,0,0,0,0,0,0,0,255,0,255,128,0,0,255,255,0,0
LineThickness=7
OpacityLevel=50
UserLastStrip=testuser,1,AM190101_01
```

Table 7.2 provides all accepted keywords within a configuration file and its description. Any other keywords are ignored.

### 7.4.1 Directory structures

Special attention is required for the directory structures of the image and result image paths. Each source and result image are found in the respective location and year maps. Regarding the above example configuration, the source image `DB195106_20.jpg` is expected to be in `Z:\stroken_scans\De Bilt\1951`, while the resulting image and tracked data should be found in `Z:\stroken_tracked\De Bilt\1951`.

## 7.5 Data tables

Throughout the framework data tables are used which provide information about consecutive recording periods for each location. Two tables are being used, the color definition table and the location recording periode table. The first is no longer used and the latter is only being used to retrieve the strip's grid time length. For compatibility reasons, the complete description of each table is still included.

The color table defines color names for either the grid and curve. Per text line, one entry is defined. An entry consists of a color combination for the grid and curve and their accompanying color definitions and comparison functions. An entry might look like:

```
"Black";"Red";200;200;200;180;150;190;170;150;160;-1;-1;-1;1;-1;-1;1;-1;-1
```

The first two names correspond to the grid and curve color respectively. The nine numbers thereafter are reference color component values ranging from 0 (dark) to 255 (bright), defining RGB colors. In particular order, the first trio represents the grid color, followed by the curve color and the tip-over color. The final nine numbers correspond to the color component comparison functions formerly used in the detection program. Three possible values exist: -1, 0 and 1. While comparing, each pixel color is compared with all RGB combinations using the corresponding function. -1 requires the component value to be lower than the reference, a 0 indicates the comparison is omitted and a 1 requires the value to be larger than the reference component color value. Since these tables are no longer used, further explanation is currently not included in this documentation.

## 7.6 Precipitation table

The table assembly program ParseTracks requires a database input table. This table is directly derived from the series database (Section 7.8), but to avoid a ODBC-connection this table is constructed to allow direct reading from disk. The precipitation table's file name is arbitrary, but currently hardcoded into the program code.

The table consists of text entries, one per row, containing information about each strip. Without further explanation (please refer to the Series Database explanation for that), an entry holds holds columns defined in Table 7.3, all having strict column widths. The columns do not have a delimiter. Example entries are:

```
380 17-09-1954  BE195409_01                    6  g  2,3  07:00gm
380 01-03-1991  BE199103_01_02;BE199103_10_01 2  -  0,8  00:00gm
380 30-05-1991                                 1  o  0    00:00gm
```

As is visible, in the second row two strip codes are present separated by a semicolon (;). Entries do not require a strip code (row 3). Note that the primary code and status columns only contain the first character of their respective codes. Refer to Section 7.8 for an indepth explanation.

## 7.7 Hourly precipitation table

Similar to the precipitation table are the hourly precipitation tables. These table are used by ParseTracks to compare the found intensity rainfall values with the known hourly precipitation values. In general, per location a separate file exists. The file name can be arbitrary, but currently a convention is used, e.g. `Beek380_rh.txt`. It consists of the location name, KNMI code and the `_rh.txt` addendum to indicate it is a hourly precipitation table. By definition all hourly value can be put in one file, but this leads to long loading times and unhandy

| Keyword | Description | Example |
|---|---|---|
| ImageRoot | Source image path | `Z:\stroken_scans` |
| ResultRoot | Result image and trace data path | `Z:\stroken_tracked` |
| DataRoot | Data tables path | `Z:\ACE\data` |
| Database | ODBC Data service name (DSN) | `metadata` |
| CycleMode | Navigation cycle mode | `index` |
| Location | Recording location abbreviation, name and KNMI code | `AH,Amsterdam HB,2` |
| ColorTable | Color table (R,G,B,A) multitude | 0,255,64,0,255,0,0,0 |
| LineThickness | Line drawing thickness | 7 |
| OpacityLevel | Overall drawing opacity level (%) | 50 |
| AllTransparent | Object transparency (1 or 0) | 1 |
| AutoNext | Automatically navigate to next image on save | 0 |
| SaveJPEG | Save JPEGs on save | 1 |
| UserLastStrip | User last edited strip code (name, do load, strip code) | `testuser,1,AM190101_01` |

Table 7.2: Accepted configuration keywords.

| Offset | Width | Description | Example |
|---|---|---|---|
| 0 | 4 | Location KNMI code | 380 |
| 4 | 12 | Date | 17-09-1954 |
| 16 | 30 | Strip code | `BE195409_01` |
| 46 | 3 | Status code | 6 |
| 49 | 3 | Primary code | `g` |
| 52 | 5 | Precipitation amount ($10^{-2}$ mm) | 2,3 |
| 57 | 7 | Start time | 07:00gm |

Table 7.3: Column definitions for the precipation table.

| Offset | Width | Description | Example |
|---|---|---|---|
| 0 | 5 | Location KNMI code | 380 |
| 6 | 8 | Date code | 19700101 |
| 15 | 4 | Time (HHMM) | 100 |
| 20 | 6 | Precipitation amount ($10^{-1}$ mm) | 6 |

Table 7.4: Column definitions for the hourly precipation table.

large files. Furthermore, these tables are retrieved through the KNMI website, where the files are set out in this manner.

The table consists of text entries, one per row, containing information about each hour of rain. An entry holds columns defined in Table 7.4, all having strict column widths and separated by a comma (,). An example is given below.

```
380,19700101, 100,     0
```

## 7.8 Series database

One of the most important sources of information is the series database. This online database is required while post-processing the strips and rolls as it contains the corresponding date, edit status and grid start time. For each date there is a separate entry in the table, and because multiple images might describe one recording session date, multiple strip codes can be present in the code field of an entry. Table 7.5 gives a summary about the fields in the database table.

Not all fields are used during post-processing, currently only the station name, recording date, primary code, status code, strip code and comments are loaded at startup. The status code, last edit user and date fields are updated on save, and possibly the strip code as well. The possible primary and status codes and their descriptions are enlisted in Tables 7.6 and 7.7.

The strip code field requires careful attention. Normally, this field contains a single strip code, but in some special cases two might be present. In case a recording for a single day was made on two strips, the corresponding strip codes are present in a single database entry separated by a semi-colon (;). Similar, in case of rolls, the recording paper was changed during the day and therefore a single day has multiple sources. While this table is being loaded, these special entries are carefully processed and treated as two separate entries each containing one of the codes. It is important to keep the codes in chronological order, as they are processed during assembly in the same manner.

| Name | Type | Description | Example |
|---|---|---|---|
| Id | AutoNumber | Auto-generated identification number | 10661 |
| station | Text | Station name | Amsterdam |
| datum | Date/Time | Recording date | 10/10/1956 |
| primair | Text | State of scanned image | OK |
| status | Text | State of digitizing process | 6. gecorrigeerd/gereed |
| Opmerkingen | Memo | Comments present on image | Mist |
| bestandsnaam | Text | Strip code | `AM193107_08d` |
| datum_lamu | Date/Time | Last edit time stamp | 28-10-2009 10:42:12 |
| starttijd | Text | Strip grid start time | 7.00 uur NT |
| digitaliseerbaar | Text | ?? | |
| gebr_lamu | Text | Last edit user | testuser |
| strook | Yes/No | Strip or roll? | Yes |

Table 7.5: Field descriptions of the series database.

| Entry | Description |
|---|---|
| geen neerslag | No rainfall was recorded on strip |
| krom gescand | Strip was erroneously scanned |
| OK | Strip can be processed |
| ontbreekt | Strip is not present in archive |

Table 7.6: Possible primary codes in the corresponding series entry field.

| Entry | Description |
|---|---|
| 1. niet gescand | Strip has not been scanned |
| 2. gescand | Strip has been scanned, requires processing |
| 3. opgeslagen in MOS | Strip has been archived on the MOS system |
| 4. verbeterd/gecontroleerd | Strip is ready for processing after scan correction |
| 5. gedigitaliseerd | Strip has been processed and requires verification |
| 6. gecorrigeerd/gereed | Strip processing results have been verified and approved |

Table 7.7: Possible status codes in the corresponding series entry field.

# Chapter 8

# Output files

In this chapter the output files produced by running the various programs are described in detail.

## 8.1 Output images

By running CurveExtractor with a valid input image, the processing will produce a similar image with the detected image features plotted on top of it. This image is tilt and curved vertical axis corrected and serves as a visual indication how well the detection worked without using the post-processing program PostACE. By convention, the output file name will be equal to the input image's file name with `tracked.` placed just before the extension. The output image format is currently programmed to be in JPEG format. Example: input image `VL197401_08.jpg` will have output image `VL197401_08.tracked.jpg`. For roll images, input roll image `HK199204_10.bmp` will result into `HK199204_10_01.tracked.jpg`, `HK199204_10_02.tracked.jpg`, etc. up till the number of recording days present on the roll.

### 8.1.1 Curve and grid images

Provided that their output has been argumented on the command-line, CurveExtractor will save the grid and curve likelihood images to disk. These grayscale JPEG images represent the likelihood for each corresponding pixel it belongs to either the background or the grid or curve respectively. These images will have similar file names as the input image, with either `grid.` or `curve.` inserted before the file extension. For example, input image `AM190101_01.jpg` will produce `AM190101_01.grid.jpg` and `AM190101_01.curve.jpg`.

## 8.2 Tracked data file

The most important file produced by tracing the input image is the tracked data file. These files will contain all the information for the post-processing program PostACE to rebuild the output image from the original input image without re-detecting its features. The only exception to this is the tilt and curved vertical axis correction. These steps can be identically reproduced by just performing these steps again, or by using the binary data in the AFD file (Section 8.4). The tracked data files are located in the result directory and have extension `.tracked`.

Two versions of the tracked data file exist. The first version, which is no longer used and deprecated, was defined in early stages during the creation of the framework and turned out to be too inflexible for adding new features, requires strict ordering and did not allow too many variation in the number of found features, later applied changes and so on. This version is no longer produced, although some older detection results are still formatted in this revision. A new definition was created (revision 2.0) which allows addition of new features without breaking older files and is fully flexible in the amount of data, permits the inclusion of later changes made in PostACE. Revision 2.0 of the tracked file data consists of a preamble of keywords determining the type of data and its contents followed by a long list of all path data. The postamble contains later changes made during post-processing. All keywords are followed by an equal-sign, even if there is no value defined for that keyword.

A tracked data file always starts with the tracked identifier and its version on line 1 of the file: `KATR20`. After this line, the preamble keywords follow in any particular order. The accepted keywords and their descriptions are listed in Table 8.2. The end of the preamble is marked by the `start` keyword after which all the path data is listed. The preamble will never change after the initial detection. For each pixel a separate line is reserved having 5 different columns. Example consecutive pixel rows are

```
59919 1922 1922 0 22
59933 1920 1920 0 23
59946 1919 1919 0 24
59959 1918 1917 0 25
```

The first column is the corresponding time in seconds on the image, followed by the originally found vertical pixel coordinate, and current vertical pixel coordinate. The fourth column represents the KNMI quality code and the latter the total amount of rainfall measured from the beginning of the curve. Because of errors in the early (but still used) versions of CurveExtractor, the seconds and total amount of rainfall numbers might be incorrect and should be neglected generally. The end of this section is marked by the `end` keyword. The number of pixel rows is expected to be equal to the width of the input image.

| Keyword | Description | Example value |
|---|---|---|
| line | Correction line points (x-coordinate, y-coordinate; pixels) | 6017,736,6397,705 |
| plane | Erase plane points (x-coordinate, y-coordinate; pixels) | 3266,100,3477,210,... |
| tipovers | Set of replacing tip-overs (x-coordinate, skew; pixels) | 3050,-12,6142,-6 |
| changegrid | Adjust detected grid (left, top, right, bottom; pixels) | 349,220,7016,1971 |
| addindicator | Add an path indicator (index, x-coordinate; pixels) | 1,1029 |
| changeindicator | Change indicator position (index, x-coordinate; pixels) | 2,1567 |
| forced | Forced point (x-coordinate, y-coordinate; pixels) | 4456,910 |
| Deprecated keywords | | |
| deltipover[1] | Delete detected tip-over (index) | 0 |
| addtipover[1] | Add tip-over (index, pixel, skew) | 1,3010,-6 |

[1] Replaced by keyword 'tipovers'.

Table 8.1: Accepted tracked data file correction type keywords. In between parentheses the values are explained together with the unit after the semi-colon (;). The latter two rows are deprecated keywords, i.e. they are still accepted but replaced by other keywords.

The postamble contains corrections made during post-processing using the PostACE application. A correction is build out of five keywords in a row in the following order: `correction`, the correction type, `time`, `date` and `end`. If the file is loaded again in PostACE, the corrections will automatically be applied and overrule any information provided in the preamble. An example correction is

```
correction=
changegrid=72,220,7026,1991
time=16:43:29
date=14/07/2009
end=
```

In this example, the grid is later adjusted to fit the image more exactly or possibly the grid detection step went wrong during processing in CurveExtractor. Either way, the new coordinates are presented here together with the time and date of correction. Possible correction type keywords are listed in Table 8.1.

| Keyword | Description | Example value |
|---|---|---|
| delim | Entry line delimiter | , |
| author | Creator of original | testuser |
| location | Computer used during creation | COMP001 |
| time | Time of creation | 21:39:43 |
| date | Date of creation | 24/04/2009 |
| program | CurveExtractor version used during creation | ACE 2.0.17 |
| source | Source image type (roll or strip) | strip |
| postprogram | Post-processing program used | PostACE 2.0.8.40 |
| input | Original input image | `DB193901_01.jpg` |
| output | Output image | `DB193901_01.tracked.jpg` |
| strip | Data table strip file | `stroken.csv` |
| color | Data table color file | `colors2.csv` |
| grid | Detected grid coordinates | 349,220,7026,1991 |
| verticaloffset | Applied vertical offset (roll images only; pixels) | 219 |
| pathrange | Start and end point of curve (pixels) | 349,6778 |
| timerange | Time range of strip grid (seconds) | 25200,117600 |
| precrange | Precipitation range of strip grid ($10^{-2}$ mm) | 0,1000 |
| sideext | Vertical image extension used on both sides of grid (roll images only, pixels) | 100 |
| xunit | The physical unit of the x-axis | s |
| yunit | The physical unit of the y-axis | cmm |
| tipovers | Detected tip-overs (x-coordinate in pixels, skew; pixels) | 2012,-12,4968,2 |
| tipskew | Detected tip-overs after skew correction | 2012,-12,4968,2 |
| dip | Detected dip (x-coordinate, y-coordinate, vertical change, window start, window end; pixels) | 9748,2165,12,9740,9761 |
| columncount | Number of columns in pixel row entry | 5 |
| start | Indicates start of path data | (no value) |
| end | Indicates start of path data | (no value) |

Table 8.2: Accepted tracked data file keywords. In between parentheses the values are explained together with the unit after the semi-colon (;).

| Keyword | Description | Example value |
|---|---|---|
| program | Roller version used for creation | Roller 1.12.0.30 |
| author | Last edit user | testuser |
| location | Last edit location | COMP001 |
| date | Last edit date | 08/10/2009 |
| time | Last edit time | 17:07:53 |
| invertedroll | Is the roll inverted? (1 or 0) | 0 |
| invertedaxis | Is the axis inverted? | 0 |
| marker | Day transition marker (y-coordinate, grid-left, grid-right; pixels) | 18055,307,2679 |
| timemarker | Time marker (y-coordinate, grid-left, grid-right; pixels) | 9055,317,2579 |
| endmarker | Roll end marker (y-coordinate, grid-left, grid-right; pixels) | 455,321,2659 |
| startdate | Roll start date (day, month, year) | 6,1,1982 |
| starttime | Roll start time (hour, minutes, seconds) | 0,0,0 |
| enddate | Roll end date (day, month, year) | 6,1,1982 |
| endtime | Roll end time (hour, minutes, seconds) | 0,50,0 |
| comments | Additional comments | |

Table 8.3: Accepted roller data file keywords. In between parentheses the values are explained together with the unit after the semi-colon (;).

## 8.3 Roller data file

To be able to digitally cut the roll images into separate consecutive days, roller data files are created using the roll image pre-processing application Roller. They provide the exact pixel coordinates of day transitions and grid boundary locations. Special end markers mark the end of the roll recording line, while time markers indicate time markings made by the observer. The start and end time and date indicate the exact times the roll recording started and ended. Roller data files have file extension `.roller` and are located in the input image directories. They have the same file name as the corresponding image, e.g. `VL199201_01.bmp` has Roller data file `VL199201_01.roller`.

All Roller data files all start with the roller identifier followed by a binary version number, e.g. 'RLRD' plus the binary version number. The newest version supported so far is version 3, and therefore the binary version will have the character corresponding with ASCII character 0x03. Hereafter, text lines follow providing Roller and other programs with information about creation, the properties of the roll and the defined markers. All text lines start with a keyword, followed by an equal-sign '=' and its corresponding value. Table 8.3 enlists all accepted keywords.

## 8.4 ACE file data file

In addition to the tracked data files resulting from the detection processing done by CurveExtractor, this program also creates a binary ACE file data (AFD) file containing several histograms and tables providing the necessary information to recreate the grid and curve likelihood images or to apply the tilt and vertical curved axis correction. It also contains the path and tip-over data. The AFD file is located in the result image directory and has file name extension `.ace`.

An AFD file offsets with three zeros and a binary version number. The newest version supported is version 4. After this version the AFD header is located, of which the fields are explained in Table 8.4. Directly after this header all above mentioned histograms and arrays are placed. A histogram is defined as an three-dimensional array containing conversion colors for each RGB bin possible. Since a full range (256 x 256 x 256) would take up far too much storage space, the RGB has only the defined number of bins (usually 64) in each color component, making its size 64 x 64 x 64 = 262144 bins. Each bin is stored in a byte, making the size thus 262144 bytes (in case of 64 bins per color component). All arrays are defined as integer-arrays, thus their byte lengths equal four times the array size. The complete AFD file structure is presented in Table 8.5.

## 8.5 Log files

Log files are created during program runs of all programs. They contain information about the various steps taken and other debug information, depending on the program or application. Log files are meant for user reference and can not be interpreted by any program.

## 8.6 Rainfall intensity table

The final step in the framework is the assemblation of the various tracked data files into a formatted table, usable by any institute. The tables are produced by the program ParseTracks by extracting the values in a specified resolution from consecutive intervals. The exact output format depends on the formats used in the corresponding institute, in this section the format used at the KNMI is briefly explained. The table consists of text rows containing the physical entity type and value. In this case the physical entity is precipitation or rainfall and the value is expressed in $10^{-1}$ mm. The text rows' columns are strictly spaced and separated by a comma (,).

Example rows are

```
A380a, 19691231, 0005, 5Min, pr, 0.0, 0.0, 0,
A380a, 19691231, 0010, 5Min, pr, 0.0, 0.0, 0,
```

| Offset | Length | Data type | Description |
|---|---|---|---|
| 0 | 4 | integer | Image width (pixels) |
| 4 | 4 | integer | Image height (pixels) |
| 8 | 4 | integer | Histogram size |
| 12 | 4 | integer | Number of path points |
| 16 | 4 | integer | Number of tip overs |
| 20 | 4 | integer | Tilt correction array size |
| 24 | 4 | integer | Curve start point (pixels) |
| 28 | 4 | integer | Estimated line width (pixels) |
| 32 | 20 | Grid[1] | Grid |
| 52 | 3 | RGB[2] | Grid color (RGB) |
| 55 | 9 | Matrix[3] | Covariance matrix |
| 64 | 3 | RGB | Paper color (RGB) |
| 67 | 3 | RGB | Curve color (RGB) |
| 70 | 3 | RGB | Curve color normalized |
| 73 | 4 | Boolean[4] | Curve color used |
| 77 | 4 | Boolean | Background color used |

[1] Grid: (left, top, right, bottom, error), 5 times integer
[2] RGB: (red, green, blue), 3 times byte
[3] Matrix: 3x3 byte matrix, 9 times byte
[4] Boolean: (true or false), equals integer

Table 8.4: Field descriptions of the ACE file data header structure. The total length of the structure is 80 bytes.

| Length | Type | Description |
|---|---|---|
| 4 | AFD Identifier | 3 times zero and binary version |
| 80 | Header | ACE file data header |
| 262144[1] | Histogram | Background color histogram |
| 262144 | Histogram | Grid color histogram |
| 262144 | Histogram | Grid vs Paper color histogram |
| 262144 | Histogram | Paper color histogram |
| 262144 | Histogram | Curve color histogram |
| various | Point array[2] | Tilt correction points (x,y) |
| various | Point array | Path points (x,y) |
| various | Point array | Tip-over points (x,skew) |
| various | Point array | Curved vertical axis correction (x,y) |

[1] In case of a 64 per color component histogram size (defined in the AFD header)
[2] A point is a combination of two integers and therefore 8 bytes in size

Table 8.5: Description of the ACE file data file structure.

| Offset | Width | Description | Example value |
|---|---|---|---|
| 0 | 12 | KNMI location code | A380a |
| 13 | 9 | Date | 19691231 |
| 23 | 5 | Time | 0005 |
| 30 | 13 | Interval resolution | 5Min |
| 44 | 13 | Physical entity abbreviation | pr |
| 57 | 14 | Measurement entity value | 0.0 |
| 73 | 14 | Corrected entity value | 0.0 |
| 87 | 6 | KNMI quality code | 0 |

Table 8.6: Rainfall table entry column descriptions.

```
A380a, 19691231, 0015, 5Min, pr, 0.0, 0.0, 0,
```

The first column indicates the recording location, followed by the date, time, resolution and entity type. The two numbers following are the directly extracted rainfall intensity and its hourly precipitation corrected equivalent. The latter column is the KNMI quality code. See also Table 8.6. Note that the extra spaces inbetween columns have been removed for clarity.

In case the program ParseTracks is run in debug mode sevaral columns are added after the above mentioned columns. Given the following example,

```
A380a, 19691231, 0005, 5Min, pr, 0.0, 0.0, 0, 6536, 1301, 1.00, 1.00
```

the extra columns are the x-coordinate of the interval, the initial y-coordinate, the fraction of the interval used and the hourly precipitation correction factor. The interval fraction can be either lower or higher than 1, since some intervals might be incomplete or overlap others. The hourly precipitation correction is calculated by summing all intervals in a particular hour and weighing this with the given hourly value. Again the extra spaces inbetween columns have been removed for clarity.

# List of Tables

# Bibliography

[1] R.E. Bellman. *Dynamic programming*. Princeton University Press, Princeton, New Jersey, 1957.

[2] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. CL-Engineering, 3 edition, 1998.

[3] H.E. Van Piggelen, T. Brandsma, H. Manders, and J. Lichtenauer. Automatic curve extraction for digitizing rainfall strip charts. *In preparation*, 2010.