Helsinki University of Technology
Department of Computer Science and Engineering
Telecommunications Software and Multimedia Laboratory

IMAGE-BASED DETECTION OF DEFECTIVE LOGS

Seppo Enarvi

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology in Espoo, on August 16, 2006.

Supervisor: Professor Tapio Takala
Instructor: M.Sc. Kimmo Koskenohi

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS
Department of Computer Science and Engineering
http://www.tkk.fi/

Author
Seppo Enarvi

Date
August 14, 2006

Pages
11 + 84

Title of thesis
IMAGE-BASED DETECTION OF DEFECTIVE LOGS

Professorship
Interactive Digital Media

Professorship Code
T-111

Supervisor
Professor Tapio Takala

Instructor
M.Sc. Kimmo Koskenohi

This thesis describes the development of a computer vision system that was installed at the Stora Enso wood handling terminal in Uimaharju. A measurement station is responsible for scaling the logs that the terminal receives, but until now grading has been entirely manual. The computer vision system substantially reduces the work load of the human grader by automatically detecting defects from log end images. The human grader will only grade the logs that the software suspects as being defective.

A comprehensive survey of basic image segmentation techniques is given. In particular their application for the segmentation of color images is discussed. An explanation of issues related to selecting a color space for a particular purpose and a review of the most common color spaces is included. The development of the computer vision system that comprises image acquisition, segmentation, object recognition, and feature classification is described.

The major merit of the thesis is the development of algorithms that localize the end of a log from a camera image, and detect if there are visible defects on the surface of the log end. Localization of the log end is based on three-dimensional tables that represent typical wood colors, and the circular shape of the log end. Defects are detected using statistical features of the log end pixel colors.

Keywords: wood quality, image processing, color image segmentation, object recognition

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto http://www.tkk.fi/	DIPLOMITYÖN TIIVISTELMÄ
Tekijä Seppo Enarvi	Päiväys 14.8.2006
	Sivumäärä 11 + 84
Työn nimi IMAGE-BASED DETECTION OF DEFECTIVE LOGS KUVALÄHTÖINEN VAJAALAATUISTEN TUKKIEN TUNNISTAMINEN	
Professuuri Vuorovaikutteinen digitaalinen media	Koodi T-111
Työn valvoja Professori Tapio Takala	
Työn ohjaaja Diplomi-insinööri Kimmo Koskenoja	

Tässä diplomityössä on kuvattu Stora Enson Uimaharjun puuterminaalille rakennetun konenäköjärjestelmän kehitystyö. Puuterminaalilla on mittalaitos joka vastaa terminaalin vastaanottamien tukkien kuutioinnista, mutta tähän asti laadutus on tehty täysin manuaalisesti. Konenäköjärjestelmä vähentää huomattavasti ihmisen laadutukseen käyttämää työmäärää tunnistamalla automaattisesti laatuvirheitä tukinpääkuvista. Laaduttajan tarvitsee käydä läpi vain tukit joita ohjelma epäilee vajaalaatuisiksi.

Diplomityö sisältää katselmuksen keskeisiin kuvien segmentointialgoritmeihin. Erityisesti käsitellään niiden soveltamista värikuvien segmentointiin. Käyttötarkoitukseen soveltuvan väriavaruuden valintaan liittyviä kysymyksiä selitetään ja mukana on katsaus yleisimpiin väriavaruuksiin. Konenäköjärjestelmän, joka sisältää kuvankaappauksen, segmentoinnin, hahmontunnistuksen ja ominaisuuksien erottamisen, kehitys selostetaan.

Työn suurin ansio on algoritmien kehittäminen, jotka paikallistavat tukin pään kamerakuvasta ja tunnistavat onko siinä näkyviä laatuvirheitä. Tukin pään paikallistaminen perustuu kolmiulotteisiin taulukoihin, jotka edustavat tyypillisiä puun värejä, sekä tukin pään pyöreään muotoon. Laatuvirheet havaitaan tukinpääpikselien värien tilastollisten ominaisuuksien perusteella.

Avainsanat: puun laatu, kuvankäsittely, värikuvan segmentointi, hahmontunnistus

ACKNOWLEDGEMENTS

This thesis is based on my work in a project ordered by Genera Oy, as a subcontractor of Stora Enso Oyj. The project was part of the PUULA research and development project coordinated by Metsäteho Oy. The objective of PUULA was to develop automatic image-based scaling and grading of wood raw material.

The supervisor of the thesis at Helsinki University of Technology was Prof. Tapio Takala, and my instructor was M.Sc. Kimmo Koskenohi from Genera Oy. I am grateful to Kimmo Koskenohi for providing me the opportunity to work with him and learn from his practical and creative way of thinking.

Especially I would like to thank my mother, who kindly hosted me the time I was writing this thesis, and gave me financial support throughout my studies, and my father, who sparked my interest in computer science, taught me how to write computer programs, and advised me to consider the Master of Science program in Helsinki University of Technology.

Espoo, August 16, 2006
Seppo Enarvi

CONTENTS

ABBREVIATIONS

ACRM: Approximate Color-Reflectance Model

API: Application Programming Interface

CCD: Charge-Coupled Device; digital camera sensor technology

CIE: Commission Internationale de l'Eclairage; International Commission on Illumination

CMY: Cyan, Magenta, Yellow; color space

CRT: Cathode Ray Tube; monitor technology

CT: Computed Tomography

DIAPM: Department of Aerospace Engineering, Politecnico di Milano

FSMLabs: Finite State Machine Labs, Inc.

HSV: Hue, Saturation, Value; color space

HTTP: Hypertext Transfer Protocol

IEEE: Institute of Electrical and Electronics Engineers, Inc.

I/O: Input / Output

MLP: Multi-Layer Perceptron; feedforward neural network with three or more layers

MSE: Mean Square Error

MVE: Minimum Volume Ellipsoid; statistical cluster estimator

PAL: Phase-Alternating Line; analog color television standard

PDF: Probability Density Function

POSIX: Portable Operating System Interface; standard API for system calls

RGB: Red, Green, Blue; color space

RTAI: RealTime Application Interface for Linux

RTOS: Real-Time Operating System

SPD: Spectral Power Distribution

sRGB: Standard RGB; color space

VMEbus: VERSAmodule Eurocard; computer architecture

WLAN: Wireless Local Area Network

1 INTRODUCTION

1.1 The purpose of the thesis

Stora Enso Group's Uimaharju Sawmill and Enocell Pulp Mill are located on the same industrial site in the village of Uimaharju in eastern Finland. The wood handling terminal at the site is responsible for receiving, scaling, and grading the raw material. It receives some 200 batches of logs each month, each of which contains a couple of hundreds of logs.

Scaling is done in the measurement system of the wood handling terminal, called Mitla (see Figure 1). The weight and dimensions of each log are determined automatically. The measurement system compares the dimensions against predefined standards and sorts for example too long and too thick logs into separate bins. The measurement system also recognizes and separates out logs that contain iron objects since they would be harmful if ended up in a sawmill. Further sorting will be done manually after the measurement system has processed the batch.

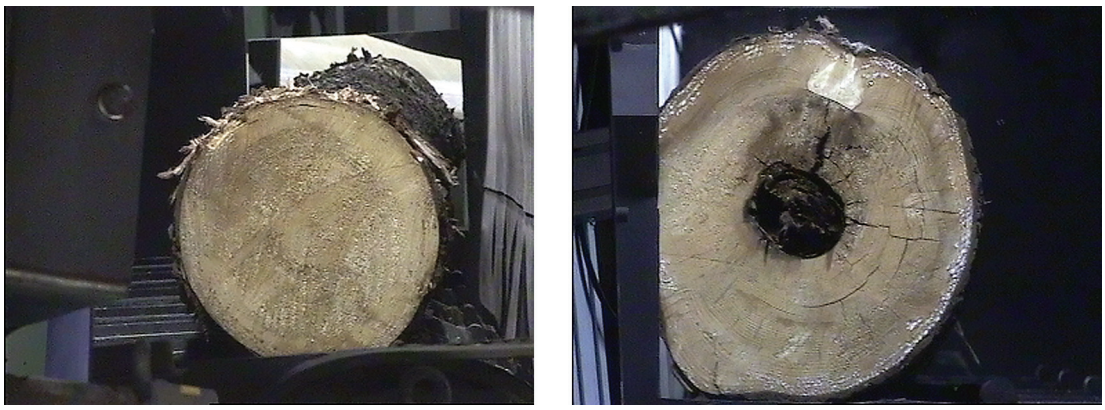
Figure 1. The Mitla measurement system for timber.

Along with the measurement data, several images are taken of each log and archived on a computer. An employed person inspects the measurement data and images in a separate office building and grades the batch for pricing. Most of the logs do not contain defects and would not have to be graded individually. Our aim was to develop software that reduces the manual work by automatically detecting most of the logs that do not contain visible defects.

The grade of a log that contains defects, such as rot, blue stain, shake, or a fork, has to be reduced, or the log has to be rejected. A system that recognizes the type of the defect and classifies the defective logs automatically would be extremely demanding to implement in practice. Thus our starting point was that the logs that the software considers potentially defective still have to be graded by a human. Also, the software will detect the logs whose dimensions do not meet the specified standards using the measurement data. Those logs will be shown to the grader in any case.

The motivation of the project was to reduce work load, and consequently reduce labor cost at the wood handling terminal. The objective was that on average 90 % of the logs could be graded automatically. This would reduce the work load by half a man-year. In the process the quality of the grading images was improved by enhancing the photographing conditions.

The system required minimal hardware investments. Three cameras had already been taking images of the logs for the purposes of manual grading, and the software was designed to use the same images to recognize defective logs. Two cameras take images of both ends of the log (see Figure 2), and one camera gives an overall image of the log.



a b

Figure 2. Both ends of a log. (a) This end of the log does not contain defects. (b) Rot is clearly visible in this end of the log.

Implementation of the software did not require much more than well-known image processing techniques either. The main contribution of this thesis is the application of those techniques in practice. Defects in wood quality can often be seen on the surface of the log ends. The log end can be located from an image on the grounds of its typical color and shape. Defects can then be detected as variations in the log end color. Figure 2 shows both log

end images of a rotten log. The rot does not extend to the end in Figure 2(a). In Figure 2(b) the rot is clearly visible as a black area in the center of the log end.

1.2 The scope and requirements for the project

The grading was not designed to be foolproof. There are situations when it is not possible to determine the internal quality of the wood simply by looking at images—not even by a human. Obvious examples include logs whose ends are covered by snow or clay, and logs that have defects inside that do not extend to the surface of the log end. In such situations our software is unable to determine the quality of the log. However, a person inspecting the images could not do any better.

In the first phase the software is only required to recognize if there are any kinds of defects in the wood or not. Afterwards the accuracy of the software may be improved and the algorithm may be enhanced to identify the type of the defect and grade every log automatically. That is, however, out of the scope of this thesis. Some of the logs will without doubt erroneously pass the detection even though they contain defects, and vice a versa. A small percentage of such errors will be tolerated.

The new software that attempts to detect defects runs without user interaction. On the other hand, because of its limitations, a human grader is still needed. All the logs that look suspicious to the software will still be rejected or graded by a human grader.

Requirements for the automatic grading are summarized here:

1. The software runs without user interaction.
2. Every log will be marked as either passed or failed. Passed logs are those where no internal defects were noticed. Failed logs are those that the software suspects as being defected.
3. Only an average of 10 % of the logs (called false negatives) will be marked as failed, but do not actually have any defects. Failed logs will be shown on the computer screen in the Mitla office to a human grader. In addition, logs, whose measured dimensions do not meet the standards selected for the batch, will be manually graded.
4. The software recognizes at least an average of 90 % of the logs that do not contain internal defects and marks them as passed. Passed logs

whose dimensions meet the quality agreement, will not be shown to the person in the Mitla office. Those logs will be automatically qualified.

5. It is not specified how large percentage of the logs marked as passed may actually contain defects (false positives). The number of false positives would be difficult to monitor, since logs that pass the automatic grading are not verified by anyone. As an extreme example, software that marked all the logs as passed would not conflict with any of the above specifications. However, it is acknowledged that the number of false positives will have to be kept very low.
6. Computation is adequately fast. Since the software is not used for sorting the logs but only grading for pricing, the defect detection does not have to work in real time. However, the wood handling terminal may receive as many as 20 batches of logs during one day, and the software will have to process them before the next day.
7. The old system will run in parallel with the new system, and will be used as it has been used so far, until the new system is proven to be fully operational.

1.3 Automatic grading of wood

1.3.1 Unsawn logs

Inspection of quality is necessary in various stages of the production chain of wood from a forest to a finished product. In the first place, accurate sorting and pricing of raw material based on its quality makes better realization of the value potential in each log possible. For example, the quality requirements of sawmills are different from those of pulp mills [1].

Our software operates on log end images taken in the Mitla measurement station, described in more detail in Chapter 2. In such environment each log has to be processed fast and the image quality may be low. Grading is difficult because of the roughness of the log ends, and sometimes even impossible because of dirt or snow. Perhaps those reasons explain why computer vision systems in such environments are often limited to size and shape measurements, and grading is done manually. On the other hand the installation costs of a computer vision system, such as the one developed in this project, are minimal. Recent development in consumer electronics has brought the computer and camera prices down and it seems that the development will continue while the image quality is improving.

Österberg et al. have developed algorithms for grading based on log end images [1]. They use the Fourier transform of local neighborhoods [2] to determine annual ring density and orientation maps. By analyzing the maps it is possible to detect rot and extract several features of the log that measure the quality of the wood. Their methods were tested on images taken in a controlled environment from logs that were cut using a sharp chain saw. The methods were reported to work well with logs that contained a moderate amount of defects.

In contrast to their analysis, our system is not able to extract features other than the information whether the log contains defects or not. On the other hand our software processes images from untreated log ends cut with a chain saw or a harvester. Some of the log ends may be damaged or badly rotten, and annual rings are not necessarily visible at all. The images have been taken in the Mitla measurement station using consumer grade computer and video hardware.

While we were able to increase the image quality, the resolution of standard PAL video cameras is limited. To avoid interlaced images, we use only one field whose resolution is 384 × 288 pixels. The log end does not fill the entire image, so the resolution of the log end is even lower. Also, a practical system needs a method for outlining the log end from the image.

1.3.2 Sawmills

When a log is taken to a sawmill, someone decides whether the log is more valuable as lumber, veneer, or chips. If the log is made into lumber, a grader or an automatic grading system inspects the board to find any defects prior to cut-up. Finally, the board is graded in order to determine its value. [3]

The purpose of the defect detection prior to cut-up is to enable the operator to generate the optimal way to cut the board to give the highest quantity or value. There is typically very little time to make the decision. With the stress involved, inspection by a human operator becomes unreliable. [4]

Some automatic grading systems are able to process boards up to 9 times as fast as a human grader. Lycken compared one automatic grading system with manual grading and noticed that the automatic grading is also more accurate resulting in better value yield and quality yield. [5]

Different technologies have been employed for scanning internal features of logs, including gamma rays, x-rays, nuclear magnetic resonance, microwaves, ultrasound, vibration, and longitudinal stress waves. These technologies vary in degree of penetration and scanning resolution, but a greater resolution comes with a greater price. [6]

Knowledge of the internal structure of unsawn logs makes optimal sawpattern placement and increase in value yield possible [7]. However, technologies that measure wood density cannot detect color defects, such as stains [3]. Good results have been attained also with cameras that sense visible light, perhaps because the environment is not as demanding as in pulp mills [1].

A typical color image-based automatic grading system at a sawmill scans the four sides of a board, performs image segmentation, and detects or extracts some features of the regions that potentially contain defects. The features are then used to detect and classify defects. At their simplest the features can be statistical properties such as the mean and standard deviation of the pixel intensities. [4]

The vast majority of the segmentation algorithms that these systems use are based on thresholding [8]. For example Conners et al. [9] first used thresholding to separate the board from the background. Then they segment the board image according to color clusters that can be found from its histogram. Those colors that occur most frequently are considered as clear wood. Regions that contain less frequently occurring colors are potential defects. We will give a detailed explanation of the most common segmentation techniques in Chapter 4.

The features that are used to classify the defects can be tonal or textural features of the pixels in the region, or based on the size and shape of the region. Tonal features measure the statistical properties of the pixel colors, and textural features are related to the spatial organization of the colors.

The structure of these computer vision systems is similar to that of our system, but they work in a less demanding environment. Simple segmentation algorithms can be used if the color of the background can be chosen and the board images are clear. With higher quality images defects can be classified more easily as well.

1.4 The organization of the thesis

The next chapter describes the Mitla measurement system, and the components of the new defect detection system, in more detail.

The conditions where the images are taken were improved slightly to obtain higher quality images. Chapter 3 describes the changes that we made to the environment, and the software that we developed for acquiring images.

A computer vision task typically starts with partitioning the image into regions that correspond to object in the scene. The algorithm either needs to understand the entire scene, or localize a single object from the image. The process where the image is partitioned into meaningful regions is called segmentation. Chapter 4 gives a review of the most common segmentation techniques, and their applicability in different color spaces.

The image processing algorithms that we developed are presented starting from Chapter 5, where a lightweight algorithm is developed for localization of the log from an image. The algorithm is used in the real-time image acquisition component to select the image where the log lies as close to the center of the image as possible.

In Chapter 6 we develop a more accurate pixel-based classifier for segmentation of the log end images. The pixel-based classifier labels each pixel as being part of the log end or not. In practice some of the pixels will be misclassified. An algorithm that recognizes the location of the log end using the labels given by the pixel classifier is developed in Chapter 7. After the log end pixels have been found, evidence of defects will be calculated from their statistical characteristics. The decision whether the log contains defects or not is a function of these characteristics. The features that we extract from the log end region and the decision function is described in Chapter 8.

In order to be of practical use the system furthermore needs to recognize situations where two or more logs enter the measurement system simultaneously. We developed an algorithm for double feed detection that is described in Chapter 9.

The realization of the objectives of the project is evaluated in Chapter 10. Finally, we conclude the thesis in Chapter 11.

2 THE ENVIRONMENT

2.1 Existing Mitla measurement system

Mitla measurement system consists of a track through which logs are carried. During that time the logs go through the following measurements:

1. The measurement of a batch starts when a crane lifts the logs onto a weighting table, where four scales weight the logs.
2. Before the logs are photographed and their dimensions are measured, they have to be separated from each other. There is a step feeder built for this purpose that feeds the transfer conveyor with one log in approximately two seconds—just enough time for a log to be photographed before the next log is fed.
3. The logs are photographed from both sides. One overhead camera gives a panoramic view of the transfer conveyor. These images are archived, and shown along with the measurement data at the Mitla office.
4. The logs are dropped into a free fall, during which several cameras take images of the log from various angles against a bright surface. These images are used to calculate the dimensions of the log.
5. Every now and then the system selects an arbitrary log for control measurement, as required by the official wood measurement regulations.

Computing is distributed among several computers. Figure 3 gives a general picture of the interactions between these computers.

1. The system can be controlled from the cranes using a touch screen. The cranes send commands to a server, and receive data and surveillance images via IEEE 802.11 Wireless LAN connection.
2. A Windows-based server communicates with the cranes, and controls the operation of the step feeder, the transfer conveyor, and the control track through a VMEbus computer.
3. A Motorola VMEbus single-board computer mediates I/O between the Windows server and the physical machinery. In addition, it receives data from optical sensors and sends a message to a serial port when a log passes the sensors.
4. A Windows-based server receives messages from a serial port and takes images using a frame grabber board, when a log passes the optical sensors. In addition it takes surveillance images all the time, and sends them to any cranes that are listening using a HTTP connection.

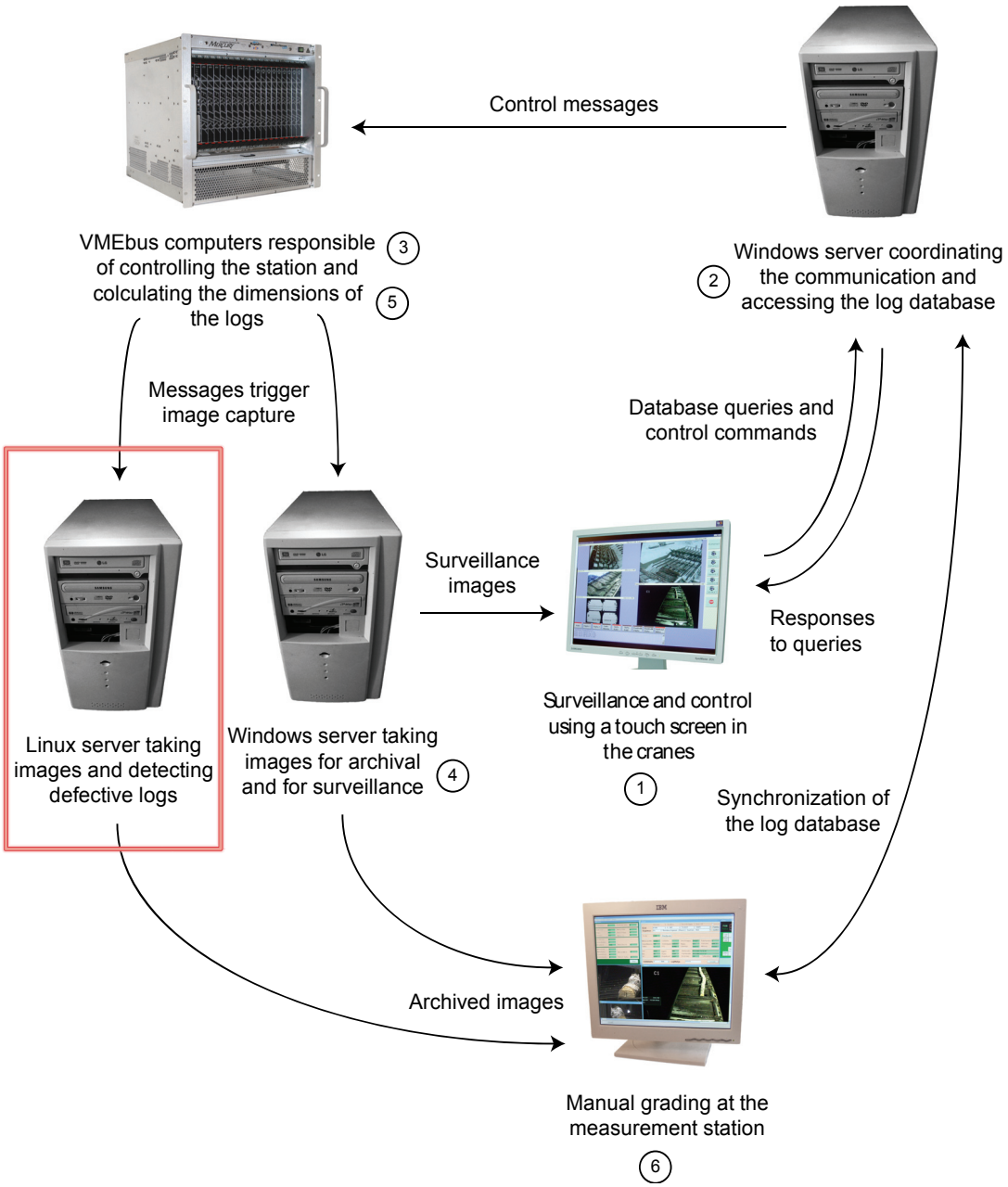


Figure 3. Simplified communication diagram of the computers involved in the Mitla system. The new computer performing automatic grading is outlined with a red rectangle.

5. A VMEbus single-board computer captures silhouette images from logs when they are dropped into the free fall, and calculates their volume and dimensions from the images.
6. Measurement data, as well as the archival images are transferred to a computer in the Mitla office using Ethernet local area network. After the entire batch has been processed, a person at the Mitla office grades the logs.

2.2 The structure of the new automatic grading system

The most important artifact of this project is software that inspects images taken from both ends of a log and detects defects in the wood. The new system was developed in a computer that runs independently of the old system, so that the logs could be graded manually until the defect detection was verified to operate correctly. Eventually the operation of the old computer that has been taking archival and surveillance images will be discontinued, so the new server has to be able to capture images and send surveillance images to the cranes as well.

The new software runs on Linux operating system. We use the standard Linux kernel, version 2.6.13.4, and a consumer grade computer. A frame grabber board with multiple inputs is required. The step feeder passes 35 logs per minute through. The computer takes one general image above the transfer conveyor and log end images from both sides. It receives a signal from a serial port triggered by optical sensors when a log passes the cameras. Because of an existing flaw in the optical sensors, the signal may sometimes arrive to the computer too early. To cope with this problem we have to take several images from the log ends and choose the best ones.

It takes approximately 700 ms to capture all the images. It became evident that after taking all the images and choosing the best ones, there is not enough time to grade the log before the next log arrives. Fortunately it is not necessary to detect the defects in real time, since the operator at the Mitla office does not have to grade the batch right away. Thus defect detection was separated into a real-time component and a non-real-time component.

The real-time component captures constantly surveillance images, and when it receives a signal from the serial port reader, it captures the archival images. The camera configuration, and some issues related to the software capturing images, is discussed in Chapter 3. The algorithm we developed for selecting the best frame is described in Chapter 4.

The non-real-time component periodically checks the disk for images of logs that it has not processed yet. After processing a log, it writes the results to another file. The file also indicates that the log has already been processed and should be bypassed in subsequent cycles. The development of the software that detects defects in logs is described in Chapters 6 through 8.

A third process monitors changes in the surveillance images. Each time the image capture server writes new surveillance images, the file monitor notifies each of the HTTP servers that send images to cranes.

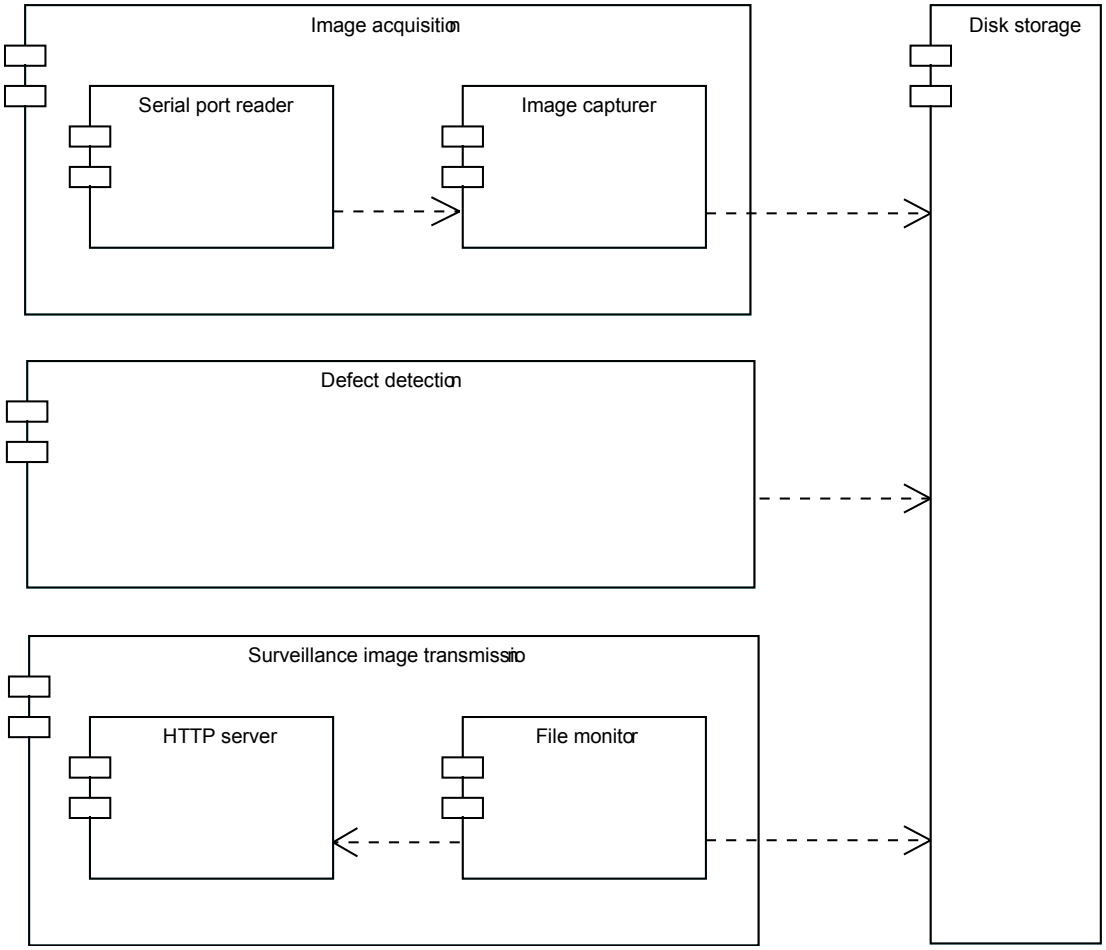


Figure 4. Components of the new server.

The component diagram in Figure 4 shows the dependencies between the components in the new server. Coupling between the three processes is low because they do not need any inter-process communication other than the images stored on a hard disk, making debugging easier.

3 IMAGE ACQUISITION

3.1 Lighting

Sometimes the easiest way to improve the accuracy of a computer vision system is to enhance the image quality. We made some modifications to the photographing conditions to improve the quality of the log end images.

Logs may pass the cameras from various distances, but the distance should not affect the brightness, or the scale, of the log, as seen to the camera. We replaced the few lights that were already installed with several more intensive lights, scattered evenly above the transfer conveyor. In addition to the more even illumination, the intensity enables us to use smaller aperture that provides greater depth of focus.

3.2 Field of view

Earlier the cameras that take the log end images had been placed almost immediately next to the transfer conveyor. As a result, field of view[1] had to be wide, and consequently a short log on one end of the transfer conveyor appeared a lot smaller to the camera in the other end. The problem had been solved by placing two cameras on each side of the transfer conveyor with different focal lengths. Each time a log passed the cameras, its placement was determined using optical sensors. If the log was close to the cameras on either side, the camera with shorter focal length was selected on that side, and vice versa.

We took the cameras further away from the transfer conveyor and increased their focal length to compress perspective. They are photographed through mirrors; otherwise the cameras would not fit inside the building. Now the logs appear approximately same sized to the cameras, regardless of their position on the transfer conveyor. Unfortunately it was physically impossible to position the mirrors on level with the transfer conveyor. Instead, the mirrors have to be somewhat higher. As a result the entire trunk is usually seen to the camera, not just the log end.

We use CCD cameras that have 1⁄3” (4.8 mm x 3.6 mm) sensors. The logs passed the cameras from only 1–4 meter distance. The cameras had zoom lenses adjusted to different focal lengths. The maximum height of an object can be calculated given its distance to the lens, sensor height, and focal

[1] The part of the world that is visible to the camera at any given time.

length, as derived in Appendix A using the pinhole camera model. Assuming sensor height is s and focal length is F, the maximum height of an object at distance D is

$$S = \frac{s}{F} D . \tag{1}$$

For example, a single 16 mm lens would have provided a field of view that is 23 cm high at one meter distance (3.6 mm ⁄ 16 mm · 100 cm), and 90 cm high at four meter distance. Logs close to the camera might not have fitted in the picture, and logs further away would have appeared too small. We moved the cameras to 6–9 meter distance and installed lenses with fixed 50 mm focal length. Now the field of view at six meter distance is 43 cm high, and at nine meter distance it is 65 cm high. Because the perspective is fairly constant over the breadth of the transfer conveyor, the old system no longer needed two cameras on each side. We took two of the cameras from the old system, and connected them to the new server.

3.3 Depth of focus

Another thing that might become a problem, when we only use one camera on each side, is *depth of focus*[1]. The placement of the logs varies several meters, and they should still appear sharp enough regardless of their position. A formula for depth of focus was derived in Appendix A:

$$DOF = \frac{DH}{H - D} - \frac{DH}{H + D} = \frac{2D^2 H}{H^2 - D^2} \tag{2}$$

Here D is the distance to which the lens is focused. H is the *hyperfocal distance*[2] of the lens that depends among other things on the aperture number. Unfortunately we do not know what the aperture numbers were before and after the change. To give a clue of what happens when we increase distance and focal length, depth of focus is calculated in Table 1 for some common aperture sizes.

---

[1] The distance from the closest point that appears to be in focus to the furthest point that appears to be in focus.

[2] The nearest distance, such that a lens focused at that distance has a depth of focus that stretches to infinity.

Table 1. Depth of focus for some camera configurations.

	F = 16 mm D = 2.5 m	F = 50 mm D = 7.5 m
n = 2.8	1.1 m	0.9 m
n = 5.6	2.5 m	1.9 m
n = 8	4.5 m	2.8 m

3.4 Capture signal

The measurement system uses optical sensors to trigger the capture of log images. A serial cable that delivers this signal was tapped off and connected to the new computer. The message that triggers the image capture uses ASCII character set. The format is shown in Table 2. It contains the following fields:

- STX (ASCII Start of Text) is a transmission control character that indicates the start of a message.
- Mode is either ASCII “a”, “b”, “c”, or “d”. Earlier, when there were two cameras on either side, Mode field was used to indicate which cameras should be used. Now that the depth of focus is greater, Mode field is redundant.
- Batch ID and Log ID give a unique identifier for each log. They are both 6-digit decimal numbers encoded using ASCII characters “0” through “9”.
- CR (ASCII Carriage Return) is a transmission control character that indicates the end of a message.

Table 2. The format of the message that triggers image capture.

0	1	2 3 4 5 6 7	8 9 10 11 12 13	14
STX	Mode	Batch ID	Log ID	CR

3.5 Serial port latency

Image acquisition proved to be one of the most challenging tasks in this project because of various problems with image acquisition under Linux environment. For reasons that are still unclear to us there was occasionally significant latency in the capture signal. As a result the images were taken too late. Real-time problems like this are hard to trace because they can be related to any of the software and hardware components in the system. The

program logic was correct; the only problem was that the system could not meet the time demand.

Sometimes the only option to trace real-time problems is to increase logging. We wrote the millisecond precision system time to a log between function calls in the serial port reader. The serial port reader uses standard UNIX select and read system calls to wait for and read data from a serial port. We noticed that occasionally, either the select or the read system call has been delayed, for even more than a second, before the program has received data fed to the serial port. The log that was supposed to be photographed had passed the cameras before the signal had arrived.

Unfortunately the standard Linux kernel gives no guarantees for response time. Several patches have been written that attempt to decrease its scheduling latency. The problem they address is that the standard kernel code is not pre-emptible—once a system call is made, the kernel will complete the task, before control is returned to user space. One such patch that renders much of the kernel code pre-emptible was migrated into the stock kernel in version 2.5. We enabled this patch with the CONFIG_PREEMPT option in our installation.

A more sophisticated, more complex, and sometimes more expensive approach would be to use a hard real-time operating system, such as RTLinux from FSMLabs, or the RTAI from DIAPM [10]. These systems contain a small kernel that provides a limited set of real-time services. The real-time kernel runs the standard Linux kernel as the lowest priority task. Real-time tasks are implemented as kernel modules, have direct access to memory and hardware, and get executed whenever they need, while all the functionality of the standard Linux kernel is available for non-real-time tasks.

We were reluctant to installing a real-time kernel, however, since our company is located far away from Uimaharju and maintenance that requires physical access to the computer is difficult. Requesting real-time scheduling and higher priority with sched_setscheduler POSIX function did not solve the problem either. Fortunately, when we tried to isolate the problem, we noticed that it only occurs when the defect detection is running. As a workaround we decided to put the defect detection to sleep for the time images are captured.

4 COLOR IMAGE SEGMENTATION

4.1 Definition of image segmentation

Image segmentation is an important step in most computer vision applications. Pham and Alcock concluded their review on automated grading and defect detection stating that a major weakness in such systems is segmentation [4]. It was the most challenging part of our project as well. Hence we have devoted an entire chapter for segmentation techniques.

Segmentation can be defined as the process of partitioning an image into regions so that the following is true [11]:

- Pixels inside each region are connected.
- Each region is homogeneous according to some criterion.
- The union of any two adjacent regions is not homogeneous according to the criterion.

The homogeneity of a region can be defined using some property of the pixels it contains, such as gray level, color, or texture. Segmentation techniques have still been an active area of research, and various techniques have been proposed in the literature. Many of them can be divided into one of the following classes: [12]

- Region-based methods detect similarity. Their essence is in finding homogeneous regions.
- Boundary-based methods detect discontinuity. Their essence is in finding boundaries for homogeneous regions.
- A large number of new algorithms combine information obtained using both region-based and boundary-based techniques.

Gonzalez and Woods note that segmentation of complex images is one of the most difficult tasks in image processing, and it should be considered if the task can be made any easier by adjusting the photographing environment [2]. We had some control over the environment; changes that we made were discussed in Chapter 3.

In this chapter we give a review of elementary image segmentation techniques. In particular we have looked into their application for the segmentation of color images. Therefore the properties of color and their representation are first discussed. Segmentation algorithms have often been designed to imitate the way human vision discriminates objects, but to our project rele-

vant is only that the segmentation algorithm discriminates the desired objects from the background as successfully as possible. However, we process images taken using standard PAL video cameras. They represent color in a way that is designed after the human perception; any information that a human cannot perceive would be redundant in most video applications. For that reason it is necessary to understand the basics of how the human eye senses color in order to understand what information color images contain.

4.2 Limitations of the human perception

There are four types of receptors in the retina of the human eye: three types of cones, and rods. Of these only the cones are involved in color vision. The cones are sensitive to a wide band of wavelengths, but each of the cone types is more sensitive in a particular region of the spectrum. The cone types are named red, green, and blue—roughly after the colors they are most sensitive to. [2]

The energy absorbed by a cone is the integral of the spectral power distribution of the incident light weighted by a cone response function:

$$R = \int_{\lambda} E(\lambda) S_R(\lambda) d\lambda \tag{3}$$

$$G = \int_{\lambda} E(\lambda) S_G(\lambda) d\lambda \tag{4}$$

$$B = \int_{\lambda} E(\lambda) S_B(\lambda) d\lambda \tag{5}$$

$S_R(\lambda)$, $S_G(\lambda)$, and $S_B(\lambda)$ are the cone response functions that define the spectral sensitivity of the red, green, and blue cones respectively. R, G, and B are the absorbed energies. $E(\lambda)$ is the SPD of the incident light.

The mixture of R, G, and B determines the color that the person sees. The visual system is able to perceive light whose wavelength is anything from around 400 to 700 nm. However, the SPD of the light that generates a particular color sensation is not unambiguous—R, G, and B can have same values with different spectral power distributions. Correspondingly any color sensation can be represented by three variables, but a display system that uses only three *primaries* is not able to produce all the colors perceivable by a human. [2]

Typically color is encoded as a three-dimensional vector. The same information can be represented in different ways using different coordinate systems.

A coordinate system along with a subspace within that system defines a *color space* [2]. Practically all general purpose video hardware is designed for human perception, and operates in a three-dimensional color space, or on grayscale images.

Color CCD cameras operating at visible light spectrum contain a grid of light sensitive sensors. There are three types of filters over the sensors that selectively attenuate some wavelengths of incident light. The spectral response of the filters should closely match the cone response functions $S_R(\lambda)$, $S_G(\lambda)$, and $S_B(\lambda)$. The cone responses have been empirically measured. In 1931 CIE standardized the cone response data of the "Standard Observer" [13], depicted in Figure 5.

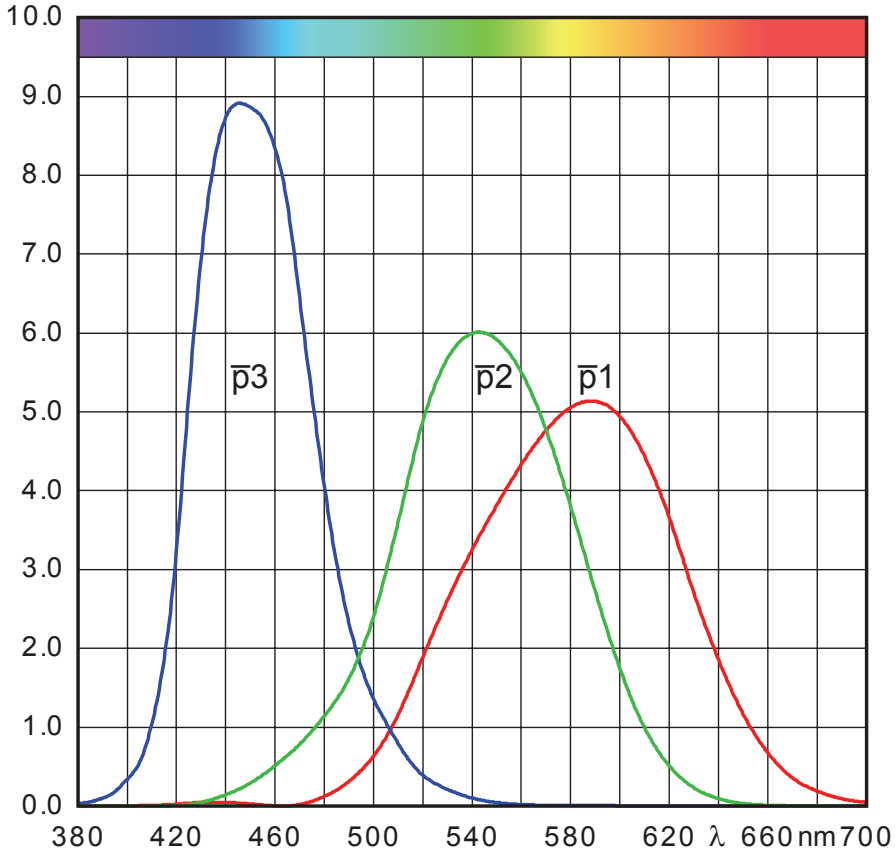


Figure 5. The cone response diagram of the Standard Observer. (Image courtesy of Gernot Hoffmann).

The video signal that our frame grabber receives from the CCD cameras is transmitted as analog PAL signal. PAL signal contains frames in YUV color space, where color is encoded as one luminance and two chrominance components [13]. Frame grabber boards typically store the acquired images in RGB color space, or in some modification of YUV color space that saves

bandwidth. We use the Video for Linux API [14], for reading the frames. Video for Linux provides color space conversion back to RGB.

Since our images are in RGB color space that is based on human perception, we tend to evaluate the solvability of an image processing task by visually inspecting the images. It should be noted that humans can discern thousands of shades of color, but only approximately 20 shades of gray [2]. The images of an 8-bit RGB camera can resolve at most 256 shades of gray and more than 16 million shades of color. Indeed color can be a significant clue in image segmentation, but still the literature on color image segmentation is not as extensively present as that on monochrome image segmentation [11].

Even more information could be obtained with special hardware that is not bound to the limitations of the human perception. Multispectral cameras measure more than three independent spectral bands, providing a better spectral resolution. The range of wavelengths can also be extended beyond the visible range. For example, the SmartSpectra system provides six bands that are configurable in the range 400–1000 nm. [15]

4.3 Color spaces

Selection of the coordinate system is important for the effectiveness of the segmentation, and the optimal color space depends on the contents of the image [16]. We will first give an overview of different color spaces.

4.3.1 RGB

RGB is a Cartesian coordinate system, where red, green, and blue components form an orthogonal basis. The major problem of RGB color space with regard to segmentation is that it is difficult to combine the color information inherent in each component. RGB image is a special case of multispectral image, where the spectral information is captured in three overlapping components. While the human eye has a very similar mechanism of capturing color, our brain uses a different representation for interpreting the image. In RGB color space all the color components depend on both intensity and chromaticity: if the intensity of a pixel changes, all the components will change accordingly, and the color information that humans perceive is scattered over all the three components as well. [11]

Numerous coordinate systems have been proposed that can be obtained by linear or non-linear transformation from RGB color space. Shih presents the classification of color spaces into objective and subjective color spaces [17].

An objective color space, such as RGB or CMY, is hardware-oriented, based on physical measurements. Subjective color spaces are more descriptively called perceptual color spaces. They are developed from the premises of our visual sensation, and based on perceptual variables like hue, saturation, and brightness.

4.3.2 HSV

Perceptual color spaces are by no means a new invention. The Munsell Color System, which is still often referenced, specified color as a combination of chroma, hue, and value—components that are closely related to saturation, hue, and brightness respectively—in as early as 1905 [18]. HSV (hue, saturation, value) color space is a similar but simpler concept designed for computer graphics. Geometrically it is a cylindrical coordinate system tilted so that the vertical axis (the z-axis) runs from black towards white through the colors whose red, green, and blue components are all equal to each other (see Figure 6). In other words, the vertical axis runs through the shades of grey. We call that axis the value axis. Saturation is the distance of a color from the value axis. Hue is the azimuthal axis (the θ-axis), spanning through the wavelengths of visible light.

HSV was first used to ease color selection in interactive painting programs [19]. Smith draws an analogy to a painter who first chooses a pigment, and then lightens it by adding white, or pales it by adding grey. Some authors have justifiably questioned the benefits of perceptual color spaces for color selection [20]. Nevertheless, they may prove to be useful in various image processing tasks. Some tasks may be easier to perform in one color space than another because of the organization of color in the coordinate space. Joblove and Greenberg bring out examples from three-dimensional graphics where color undergoes a transformation that follows a complex curve in RGB color space, but might be computed by simply

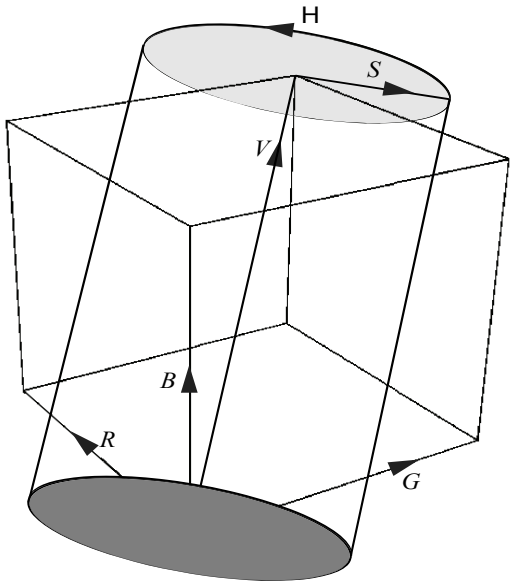


Figure 6. The Cartesian RGB coordinate system and the cylindrical HSV coordinate systems.

performing linear interpolation in some other color space [21].

The major problem with HSV color space regarding image segmentation is that the hue component has a singularity at the value axis. When saturation is low, a small change in RGB values may cause a large jump in hue. On the other hand, HSV coordinate system provides a good premise for segmenting images in accordance with human perception, since our vision can easily discriminate different hues, while disregarding changes in intensity and saturation. [11]

4.3.3 CIE XYZ

Color spaces that we have discussed so far actually depend on the hardware that produces the colors. In the case of an RGB camera, the filters determine the values for R, G, and B. When the image is displayed on a CRT monitor, the phosphors determine the primary colors, of which every color mixture is composed. Device-independent color spaces are needed, if color has to be preserved identically, when images are interchanged between different systems.

Device-independent color spaces are defined in terms of reference primaries. From the cone response data, depicted in Figure 5, color matching functions can be calculated that transform captured color into the color space specified by particular primaries. CIE standardized in 1931 color matching functions for a color space that spans the entire gamut of human vision. The color space is based on three imaginary primaries, X, Y, and Z—as stated before, any combination of three real primaries is not able to produce all the perceivable colors. The system is designed so that the Y component is a measure of luminance of a color. Chromaticity coordinates are obtained by normalizing the quantities: [13]

$$x = \frac{X}{X + Y + Z} \tag{6}$$

$$y = \frac{Y}{X + Y + Z} \tag{7}$$

$$z = \frac{Z}{X + Y + Z} \tag{8}$$

Because the sum of chromaticity coordinates, $x + y + z$, always equals to 1, a function of chromaticity can be depicted using a two-dimensional diagram—

the third chromaticity coordinate is a function of the other two. In general, color spaces are depicted in a diagram where x is the horizontal axis and y is the vertical axis. In that case $z = 1 - (x + y)$, and z has the highest value in the origin. Figure 7 shows the gamut of human vision in CIE XYZ coordinate system. The gamut of colors that graphics hardware can achieve is usually well inside the gamut of human vision, and depends on the primaries of the specific hardware. Shown in the figure is sRGB color space that is a standard designed to approximate the color spaces of typical desktop hardware [22].

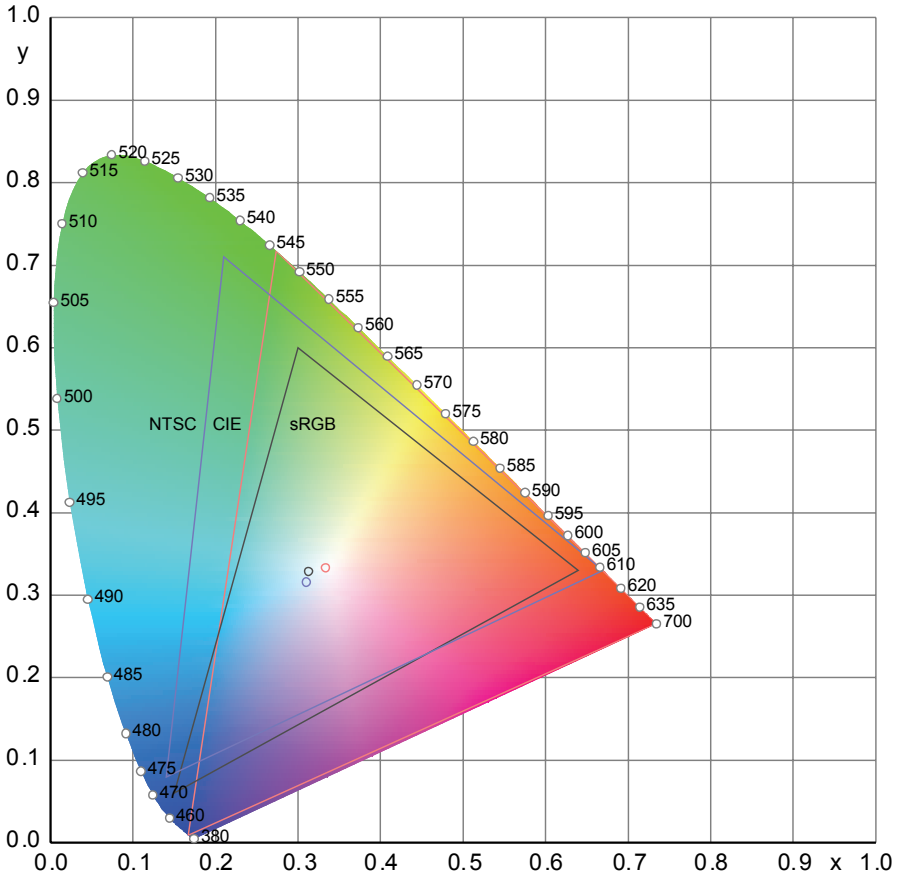


Figure 7. The gamut of human vision in normalized CIE XYZ coordinate system. (Image courtesy of Gernot Hoffmann.)

Colors can be transformed to a device-independent coordinate system with a simple matrix multiplication, but this requires knowledge of the primaries of the source color space. Since we only process images coming from a specific hardware, we would not benefit from using a standard color space. If the program was supposed to work in different environments with the same parameters, it might be judicious to convert the frames to a standard color space before processing them, but then the lighting conditions should also be similar in each system.

4.3.4 Perceptually uniform color spaces

Sometimes a color space that furthermore maps color in a perceptually uniform way is useful. For example, a change in hue in two different places of the *spectral locus* might seem equally large to an observer, but the distance moved in the CIE XYZ coordinate system might be unequal. In fact the Munsell Color System is a perceptually uniform color space [23], but it was not designed to be computationally simple. In 1976 CIE standardized two perceptually uniform color spaces, (L*u*v*) and (L*a*b). The errors they suffer in uniformity are about 6:1 over the visible gamut [24].

Perceptually uniform color spaces have the advantage when storing digital images that the visual effect of quantization and round-off errors is more or less constant over the visible gamut [24]. Limb et al. discuss the benefits of a color space, in which the sensitivity to errors is constant at all points, when encoding video signals [13].

4.4 Histogram thresholding

Most color image segmentation methods found from the literature are generalizations of monochrome image segmentation into a three-dimensional color space [11]. Even trivial monochrome image segmentation algorithms may become complicated when color information should be employed. Understanding how the algorithms work in one dimension helps to understand the problem in three dimensions.

The idea of gray scale thresholding is to divide the intensity scale into ranges that correspond to different classes of objects. The homogeneity criterion simply says that a region is homogeneous if the intensities of all the pixels inside the region fall into exactly one of the intensity ranges. Segmentation according to such criterion would be done by scanning the image pixel by pixel, and labeling the pixels according to their intensity [2].

4.4.1 Selecting an optimal threshold

The intensity ranges can be defined in terms of threshold values. A fixed threshold value often fails to give adequate results, because there is no universal value that would serve as a good threshold for all images, or even in all areas of a single image. That is why many algorithms for finding the optimal threshold for a particular image have been proposed. Some algorithms divide the image into sub-images and select a separate threshold for each

sub-image. The key issues in these algorithms are how to divide an image into sub-images, and how to select an optimal threshold [2].

The optimal value for the threshold in a particular image can be defined as the number of pixels that will be assigned to a wrong class. If an object in an image is entirely brighter than any part of the background, the histogram of the image will consist of at least two distinct modes: one is the histogram of the background, and the other is the histogram of the object. In this idealistic situation, selecting a threshold from somewhere between the modes would result in perfect separation of the object and the background.

In practice parts of the histograms of the different objects in the image overlap. The optimal threshold value that results on average in the least amount of wrong classifications can be calculated, if those histograms are known a priori. One way of phrasing the problem is to consider the pixel intensities of the different objects as random variables and their histograms as estimates of their probability density functions. The optimal threshold can then be calculated by estimating the shapes of the PDFs and the relative sizes of the objects. [2]

For example, assuming that an object in an image and the image background have Gaussian PDFs with equal variances, and the object fills exactly half of the image, the optimal threshold is the average of the means of the PDFs [2]. Unfortunately there is no theoretical evidence that individual PDFs obey Gaussian distribution [25].

Another approach would be to estimate a continuous PDF for the entire image from its histogram, and search for dominant modes from the PDF. Two dominant modes typically indicate the presence of an edge, and provide a candidate for threshold value in between the peaks. A continuous PDF can be derived from a histogram by minimizing the resulting mean square error, but finding an analytical solution is not a simple matter. [2]

4.4.2 Color images

Thresholding is a one-dimensional operation. An analogous color image operation would be dividing the color space into subspaces and classifying pixels according to which subspace contains the pixel's color. The problem with color image segmentation is how to define the subspaces. It is not possible to divide a color space into arbitrary subspaces using threshold values.

Each color component can be separately thresholded, but this fails to utilize the color information as a whole for each pixel [11]. If the components are thresholded independently, relations between the components cannot be noticed. For example, if we wanted to select all the yellow pixels from an image, we could set a criterion that the red and the green component have to be above some threshold, and the blue component has to be below another threshold. The criterion would specify a color subspace that contains yellow colors, but is not as compact as possible—red and green colors would become selected as well.

Sometimes it is possible to transform color into a suitable coordinate system, where one axis represents a feature whose values are different in each class. Thresholding can then be done using only that color component. For example, all the yellow pixels can be selected in the HSV color space by thresholding hue. Still thresholding is only one-dimensional operation, and it is often difficult—or impossible—to find an adequate threshold.

4.4.3 Feature space clustering

When moving from monochrome images to color, the problem of automatically selecting a threshold changes to automatically dividing the color space into subspaces that correspond to different objects in the image. In general the pixels can be classified using any local image features. Those features are arranged in feature space vectors. The feature space can have only one dimension, such as the pixel intensity, or it can have arbitrarily many dimensions.

Each object in an image is likely to form a cluster of adjacent feature vectors, just as dominant modes in a histogram often correspond to different objects in the image. The procedure of finding the centers of the significant clusters in the feature space is called feature space analysis, or, in statistics, the multivariate location problem [25]. The problem was formulated by Jolion et al. as follows: given a set of points in a p-dimensional space, find the best partition of the space into significant clusters [26].

Some methods that estimate cluster centers operate on a discrete feature space. If the feature vector components are discrete variables, as in the case of color components, there is no loss of information [26]. Other methods use a continuous coordinate system to avoid problems caused by quantization artifacts [25]. We already noted that the intensity histogram of a monochrome image can be used to estimate the continuous PDF of the pixel intensities,

from which peaks produced by different regions can be searched. Similarly the feature space can be modeled as a sample from a multivariate probability distribution [25].

Clustering techniques often assume that the clusters follow Gaussian probability distribution, or depend on parameters, such as the number of clusters in the feature space, that have to be known a priori. Guessing the correct settings may prove to be difficult. The technique should also be robust, meaning that it will perform reasonably well even if some data points are not following the distribution of the cluster. A widely used estimator from statistics that has been applied in computer vision is the minimum volume ellipsoid estimator. [25]

Jolion et al. propose a robust clustering algorithm that is based on the MVE estimator. The MVE estimator finds the minimum volume ellipsoid that encloses h = 0.5 percent of the data points in the feature space. The algorithm proposed by Jolion et al. applies the MVE estimator with different values for the h parameter. The compactness of the found clusters is measured by comparing the distribution of the data points inside each minimum volume ellipsoid with a multivariate Gaussian distribution. The cluster that best fits the Gaussian distribution is labeled as a significant cluster and removed from the data set. The algorithm is then repeated over the rest of the data points to find further clusters. The required amount of computation is reduced by random sampling. [26]

4.5 Other region-based methods

Region growing is an approach where a cluster of pixels is expanded by adding adjacent pixels. The pixels are chosen so that the region remains homogeneous according to the selected criterion. When there are no such pixels left, the algorithm stops. This method requires an initial set of “seed” points. [2]

Region splitting and merging is in a way a contrary approach. It starts from the state where the entire image is a single region. Unless the entire image satisfies the homogeneity criterion, the region is divided into, for example, four quadrants. Each of these subregions is divided into smaller and smaller regions, until every region satisfies the homogeneity criterion. Finally any adjacent regions whose union is homogeneous have to be merged. [2]

These two methods are not limited to monochrome images in any way—they are equally suitable for homogeneity criteria that concern any kinds of image features.

4.6 Boundary-based methods

Segmentation methods that are based on discontinuity of some image feature include point detection, line detection and edge detection. If the feature is a scalar value, such as the pixel intensity, the algorithms can often be implemented using linear spatial filters.

These algorithms rarely produce perfect boundaries for regions. Edges may be more than one pixel wide, or they may have discontinuities. In practice the points detected as edge points have to be linked in order to get connected boundaries for regions. [2]

4.6.1 Point and line detection

Detecting an isolated pixel that differs in intensity with its background is straightforward. A filter mask that accentuates such pixels, but mutes areas with constant intensity, is shown in Figure 8(a). When all the image pixels below the mask have an identical intensity, the response of the filter will be zero, because the sum of the coefficients equals to zero. If the pixel below the center of the mask has greater or lower intensity compared to its neighbors, the response of the filter will be greater or lower than zero respectively. Those pixels that have intensity over a positive threshold or below a negative threshold in the filtered image will then be labeled as isolated points. [2]

Detecting a line that runs in horizontal, vertical, or diagonal direction is similarly straightforward. The filter mask that is shown in Figure 8(b) accentuates lines that run in horizontal direction, and mute areas that have a constant intensity in vertical direction. The filter mask that is shown in Figure 8(c) accentuates lines that slant to the right at an angle of 45° degrees, and mutes areas that have a constant intensity in the parallel direction. Similar masks can be constructed that detect vertical lines and lines that slant to the left at an angle of 45° degrees. [2]

-1	-1	-1
-1	8	-1
-1	-1	-1

-1	-1	-1
2	2	2
-1	-1	-1

2	-1	-1
-1	2	-1
-1	-1	2

a b c

Figure 8. Filter masks that detect (a) isolated points, (b) horizontal lines, and (c) diagonal lines.

Lines that run in an arbitrary direction are not easily detected using a 3 × 3 mask, but if the direction of the line is known a priori, the image can be rotated so that lines oriented in the known direction will become for example horizontal.

4.6.2 Edge detection

Edge detection is the most common approach for detecting discontinuity of pixel intensities [2]. Many edge detection techniques are said to be *parallel*, meaning that whether or not a set of points is on an edge does not depend on previously found edge points [11]. For example, a common method for edge detection is by modeling the image as a continuous function $f(x, y)$, and approximating its gradient $\nabla f(x, y)$ [2]. The gradient can be approximated using linear spatial filters that could in principle be applied to each pixel in parallel.

Edge detection techniques that are not parallel are said to be *sequential*. A shortcoming of sequential algorithms is that they require an initial point for the detection process [11]. Some straightforward parallel techniques are presented below.

The gradient of scalar function f, denoted by ∇f, is a vector function that points at the direction where f increases most rapidly. The rate at which f increases at that direction is given by the magnitude of the gradient, $|\nabla f|$. The gradient of a continuous two-dimensional function $f(x, y)$ is defined as [27]

$$\nabla f(x,y) = \left[\frac{\partial f(x,y)}{\partial x} \quad \frac{\partial f(x,y)}{\partial y} \right]. \tag{9}$$

The magnitude of gradient is simply

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2} \,. \qquad (10)$$

The rate at which some image feature changes can be approximated by linear spatial filters that attempt to calculate quantities from the discrete pixels that are closely related to the partial derivatives of the continuous function. These quantities are then combined similarly to Equation (10) to find the pixels where the feature changes most rapidly.

There are different approximations; particularly popular ones are the so-called Sobel operators, shown in Figure 9. The difference between the first and the third column in the 3 × 3 image region approximates the partial derivative with respect to x, and the difference between the first and the third row approximates the partial derivative with respect to y. The coefficients are greater at the center to achieve some smoothing and suppress noise. [2]

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

a b

Figure 9. Sobel operators that approximate the first partial derivative (a) with respect to x, and (b) with respect to y.

When the transition at the edge is gradual, filters that approximate second-order differential operators, such as the Laplacian, are advantageous over the gradient approximations [28]. The Laplacian of scalar function f, denoted by $\nabla^2 f$ or Δf, is a scalar function defined as the sum of the unmixed second partial derivatives [27]:

$$\Delta f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2} \qquad (11)$$

If the value of function $f(x)$ increases over some interval of x, the second derivative of $f(x)$ has a peak where the transition begins, and a negative peak where the transition ends. An imaginary straight line from the positive peak to

the negative peak crosses zero near the midpoint of the transition. This *zero-crossing* property can be used to find edges from an image. However, typically the Laplacian is too sensitive to noise, and it is used in combination with a smoothing filter. [2]

4.6.3 Color edge detection

All the edge detectors we have discussed so far operate only on scalar values. In color images the value stored at each pixel, $\mathbf{c}(x, y)$, is a—typically three-dimensional—vector. Any color space may be used, but for simplicity we denote the components with R, G, and B.

Another way of looking at it is that the image is composed of three component images, I_R(x, y), I_G(x, y), and I_B(x, y), each of which contains scalar pixel values. Edges can be detected individually from each component image, but combining the results is not trivial, because of localization inaccuracies [28].

In a suitable coordinate system it may be sufficient to detect edges using only one color component. For example, a very simple edge detector operates on the intensity image, defined as the sum of the component images:

$$I(x,y) = I_R(x,y) + I_G(x,y) + I_B(x,y) \tag{12}$$

Taking the vector sum of the gradients of the color components is in fact equal to taking the gradient of the intensity image that is the sum of the component images, because gradient is a linear operator [29]:

$$\nabla I_R(x,y) + \nabla I_G(x,y) + \nabla I_B(x,y) = \nabla(I_R(x,y) + I_G(x,y) + I_B(x,y)) \tag{13}$$

The major problem with the sum of the gradients is that if two of the gradients have equal magnitude but opposite direction, their sum would provide a null vector [30]. This problem is avoided if instead of taking the vector sum of the gradients the sum of the magnitudes of the gradients is taken [28]:

$$\left|\nabla I_R(x,y)\right| + \left|\nabla I_G(x,y)\right| + \left|\nabla I_B(x,y)\right| \tag{14}$$

However, optimal results cannot be achieved with any of the above approaches, because the relationships between the color components are ignored [28].

Edge detectors that are based on the vector differences of the colors of adjacent pixels have been developed [28]. Tao and Huang make use of feature

space analysis [29]. For each pixel the vector that contains the magnitudes of the gradients, $\mathbf{g}(x, y)$, is projected onto various direction vectors, $\mathbf{d}_i$:

$$\mathbf{g}(x,y)\cdot\mathbf{d}_i = \begin{bmatrix} |\nabla I_R(x,y)| & |\nabla I_G(x,y)| & |\nabla I_B(x,y)| \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \end{bmatrix} \tag{15}$$

Equation (15) gets the highest value when $\mathbf{g}(x, y)$ and $\mathbf{d}_i$ are parallel. The algorithm finds the direction vectors from the center of each cluster in the color space to the centers of all the other clusters. Near an edge $\mathbf{g}(x, y)$ should be parallel to one of these direction vectors. Edge magnitude is defined as the maximum of the projections of $\mathbf{g}(x, y)$ onto the direction vectors.

Color variants of the Canny operator are more advanced color edge detectors. They are based on the 2 × 3 Jacobian matrix of the mapping from the image plane to the color space: [28]

$$\mathbf{J} = \begin{bmatrix} \nabla I_R(x,y) \\ \nabla I_G(x,y) \\ \nabla I_B(x,y) \end{bmatrix} \tag{16}$$

The eigenvector of the 2 × 2 matrix $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ that corresponds to the largest eigenvalue gives the direction in the image at which the largest change of color occurs. [28]

Some experiments suggest that nearly all edges can be detected using only the intensity image [31]. Thus the benefits of color edge detectors should be weighed against the additional complexity. Even monochrome edge detectors can provide useful information for more sophisticated color image segmentation approaches.

4.6.4 Edge linking

Filtering an image with an edge detection operator and thresholding the resulting is not enough to produce closed boundaries for image regions. Noise, nonuniform illumination, and other effects cause inaccuracies in the edge detector. Consequently the detected edge points still have to be linked into meaningful edges. [2]

Methods that link detected edge points within a small neighborhood according to some similarity criteria are simple to implement. Similarity may be based, for example, on the direction and magnitude of the gradient vectors at

the edge points. The neighborhood of each edge point is scanned for other edge points and the points are linked if the angle and the vector difference between their gradient vectors are small enough. [2]

Hough Transform is a convenient and popular technique that can be used for edge linking, presented for instance in [2]. Given a set of points Hough Transform is used to find parameters for curves so that the points lie as close as possible to a small number of curves.

As an example we consider the case where the points should be fitted onto lines. Each point (x_i, y_i) that fits onto the line $y = ax + b$ satisfies

$$y_i = ax_i + b\,. \tag{17}$$

The problem is to find such line, i.e. the parameters (a, b) when the points (x_i, y_i) are known. Such parameters satisfy Equation (17), which may also be written as

$$b = -x_i a + y_i\,. \tag{18}$$

In the two-dimensional space that spans the possible values for parameters (a, b), the values that satisfy Equation (18) are found from the intersection of the lines $b = -x_i a + y_i$. In practice all the lines do not intersect at a single location in the parameter space, because the points in the image plane rarely lie on a single line. Locations near which several lines intersect can be found by subdividing the parameter space into disjoint cells and counting the number of lines that pass through each cell.

Edge points that lie approximately on a single line can be found using Hough Transform. Those points should be linked if they are close enough to their closest neighbor.

4.7 Physics-based approaches

Some segmentation approaches use a model of the physical process underlying image formation to better understand the image [31]. The idea is to ignore discontinuities in color that are caused by optical effects, such as shading, shadows, and highlights, and partition the image into regions that more closely resemble the real objects in the scene [11]. These approaches make use of conventional segmentation algorithms, but are essentially more complex than techniques with simple homogeneity criteria.

Among the most popular image formation models are the dichromatic reflection model [32], and the approximate color-reflectance model (ACRM) [33]. The dichromatic reflection model is a reasonable approximation for a large class of inhomogeneous dielectrics, and ACRM combines it with a unichromatic reflection model for metals [33].

These models are made computationally feasible, and do not follow perfectly the surface reflectance laws. ACRM states that when a surface is illuminated with a single spectral power distribution $L(\lambda)$, the power of the reflected light, $E(g, \lambda)$, can be accurately approximated by a function of the form

$$E(g,\lambda) = L(\lambda)M_S(g)C_S(\lambda) \tag{19}$$

for metals, and

$$E(g,\lambda) = L(\lambda)M_S(g)C_S(\lambda) + L(\lambda)M_B(g)C_B(\lambda) \tag{20}$$

for inhomogeneous dielectrics. Parameter g indicates dependency on the geometry of reflection (the direction of the incident light and the viewing direction), and λ indicates dependency on the wavelength of light. Subscript S indicates *surface reflection*, and subscript B indicates *body reflection*[1].

Functions $M_S(g)$, $C_S(\lambda)$, $M_B(g)$, and $C_B(\lambda)$, are not given but depend on the material. The significance of the equations comes from the fact that because the hue of the reflected light does not depend on geometry, the spectral cluster of the colors reflected by a single material always takes a certain simple shape. Thus images may be segmented with the homogeneity criterion that the histogram of each region has to match the shape predicted by the equations. [34]

Healey uses ACRM to segment color images into regions whose boundaries follow the material boundaries in the image [31]. He employs both boundary-based and region-based segmentation techniques. A segmentation algorithm similar to region splitting and merging is used to find regions whose histogram is likely to have been produced by a single material. However, edges are first detected from the intensity image to simplify the region splitting—

[1] The process when light enters the body of an object and is reflected by particles inside the object is called body reflection. Surface reflection takes place on the surface of an object. Typically body reflection tints the light with the object color while surface reflection produces highlights that have the same color as the illumination.

regions that do not contain edge pixels are assumed to correspond to a single material.

5 ADAPTING TO EARLY TRIGGERING OF IMAGE CAPTURE

5.1 Localizing the log in real time

Before the logs enter the transfer conveyor, where they are photographed, a step feeder separates them from each other. The step feeder passes 35 logs per minute through, so there is approximately two seconds time to photograph each log and write the images to disk before the next log is fed.

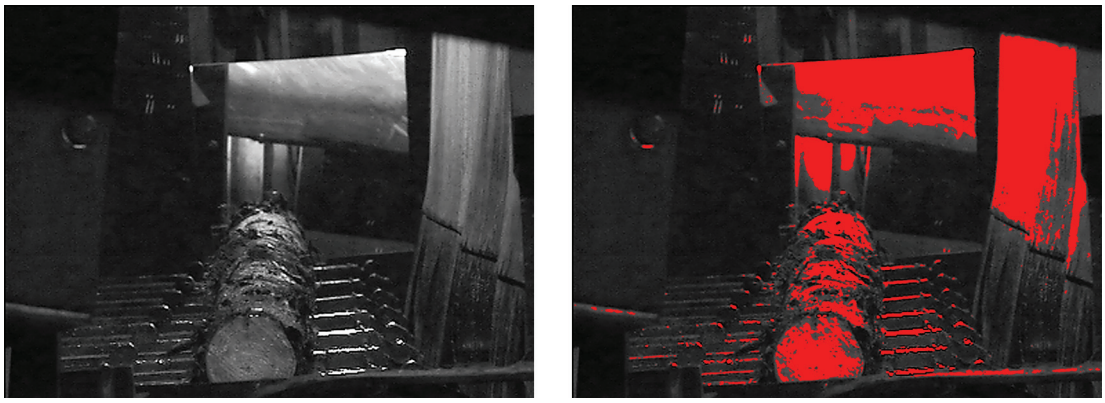
Optical sensors trigger the image capture when a log leaves the step feeder. A log that has an unusual shape may trigger the sensor before its ends are in the camera line. Also, a log occasionally leaves the step feeder in such an orientation, that the image capture gets triggered before one or both of the log ends are in the camera line. This flaw has always been in the system, but it was due to be fixed along with the development of the computer vision system.

Fortunately the signal is never transmitted too late. It is sufficient to continue capturing frames for a short period of time, and choose a frame where the log end is visible. The only challenging task is finding the correct frame in the limited amount of time that is available before the next log arrives. It takes approximately 700 ms to capture all the images we need, which leaves not much more than a second of time to select the best frames and write them to disk. In this chapter we present the selection algorithm we developed.

Because of the time constraints we did not want to use the same image segmentation algorithm that is described in Chapter 6. After all, we do not have to find the exact region of pixels that covers the log end—it is sufficient to get an estimate of the horizontal position of the log, if there is a log in the image. We apply an algorithm that localizes the log on all the frames taken using the two cameras. The frame that we choose to save is the one where the log is closest to the horizontal center of the frame.

We started with an algorithm that did not use color information, but only brightness of each pixel. It simply calculated the sum of the brightness values in every pixel-column. The column that had the greatest sum was an approximation for the horizontal position of the log. There are bright areas in the background too, but because the log usually appears in the image as a vertical bar, usually a column somewhere near the center of the log is the brightest column.

This algorithm was too sensitive to changes in the environment. The mirrors through which the log ends are photographed reflect light to the camera on the opposite side, confusing the algorithm. In addition, depending on its orientation, a carpet on the side of the images may reflect light, also forming bright pixel-columns. Figure 10(a) shows an image containing bright areas in the background, and Figure 10(b) shows the same image with bright areas tinted red. Even if a column at the location of the log would be the brightest column, it is difficult to make the decision whether there is a log in the image or not, if the brightness of the background changes from time to time.



a b

Figure 10. (a) Grayscale image of a log. (b) The same image with the pixels whose brightness is over a certain threshold tinted red.

Early computer vision systems that detected defects from lumber, processed gray scale images because color cameras were expensive and color image processing was slow. A fast and simple alternative to segmentation that was often used was to divide the image into regularly shaped sub-images. A natural choice is to use rectangular regions. Defects were detected by extracting statistical features such as the mean and standard deviation of the pixels inside each region. [4]

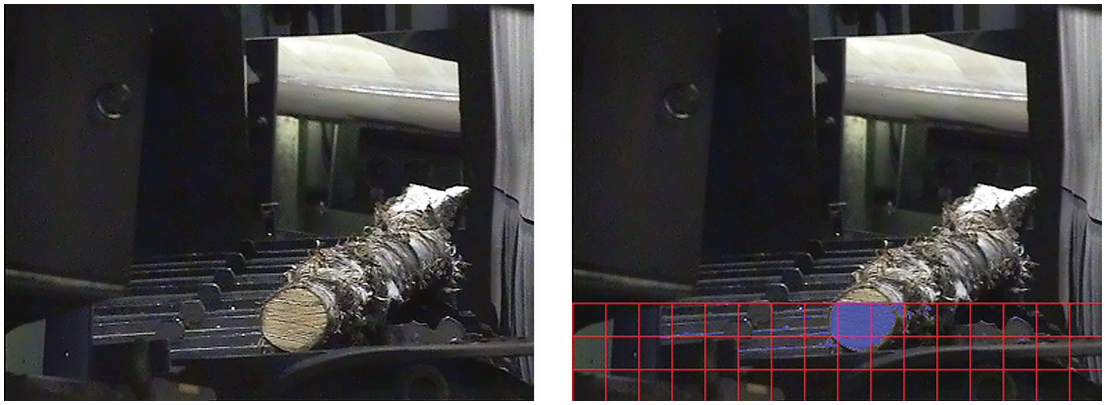
The problem with the subdivision approach is that to accurately locate the objects the region size has to be small. However, to accurately calculate the statistical features a large region is needed. Another benefit of segmentation over the subdivision approach is that it makes measurement of features that are based on the size and shape of the region possible.

The subdivision approach can be used for finding an approximate location for the log. The accuracy of the location is sufficient for us even with a large region size. The region size just needs to be small enough so that at least one

region always contains mostly log end pixels. In principle the only difference to our first approach is that the regions are no longer pixel-columns, but square cells.

The algorithm is described in detail below:

1. The image is divided into square 24 x 24 pixel disjoint cells. The optimal cell size was found by experimenting with different sizes.
2. Starting from the bottom row, and moving up one row at a time, the number of bright pixels in each cell in the current row is counted.
3. The cell with the largest number of bright pixels is a candidate for the location of the log end. If at least 60 % of the pixels in that cell are bright, the algorithm stops, and the log end has been found. Otherwise the algorithm continues on the next row.
4. Only a given number of rows will be searched. If the number of bright pixels in the candidate cell in the last row is not above the threshold, no log was found from the image.



a b

Figure 11. (a) Image of a log that passes the cameras slightly askew. (b) Illustration of the frame selection algorithm operating on the image.

Figure 11 shows an image of a log that passes the cameras slightly askew. The red grid in the right hand-image shows the cells that the algorithm has analyzed. Pixels whose RGB values were within the lower and upper bounds have been tinted blue in the right-hand image. In the second lowest row none of the cells had enough blue pixels, but the algorithm has stopped in the third row. The approximate location for the log end is the ninth cell in the third row.

The essential enhancement to our first approach comes from the fact the log end is usually below other large areas of bright pixels. Once the log end is found, the search stops before examining other bright areas above.

The end of the log, which is of most interest with regard to grading, will be at the center of the chosen image. If the log end is too dark, for example because of clay, the log is usually found from the next row, because the top of the log reflects bright light to the camera. We set an upper limit for the number of rows that will be searched, so that if there is no log in the image, the algorithm will not continue to the bright areas in the surroundings.

Finally, we made a minor improvement by changing the algorithm to operate on RGB color, instead of brightness, of each pixel. We set lower and upper limits for the red, green, and blue components. The upper limits were set to exclude pure white in order to distinguish wood colors from snow and the shiny metallic surface of the transfer conveyor. Using lower limits for each of the RGB components makes some level of distinction between dark wood and equally bright but different colored background possible.

5.2 Discussion

This algorithm works very accurately. If a better precision was needed, the algorithm could be enhanced to detect logs based on changes between adjacent frames. That would make the algorithm only slightly more complex and slower. If a frame was subtracted from the next frame and there was a log in both the frames, most likely the position of the log in the first frame would contain negative pixels, and the position of the log in the second frame would contain positive pixels, while other areas would be close to zero.

6 LOG IMAGE SEGMENTATION

6.1 Selecting a segmentation method

Our grading algorithm determines the internal quality of the wood from the cross section of the log. Thus the only region it needs to examine is the circular region that outlines the end of the log that is closer to the camera. The objective of the segmentation algorithm that we will develop in this chapter, and the log end recognition algorithm that will be developed in the next chapter, is to localize the log end region inside the log images.

Strategies for image segmentation were reviewed in Chapter 4. Basically our segmentation algorithm is required to classify each pixel into one of two classes: those lying inside the log end region and those lying in the background. The former class is referred to as object class in our discussion. The easiest starting point for log end localization is a pixel-based segmentation technique that classifies each pixel individually based on its color. Since the background of our log images is generally dark, it is reasonable to suspect that color could discriminate between log end and background.

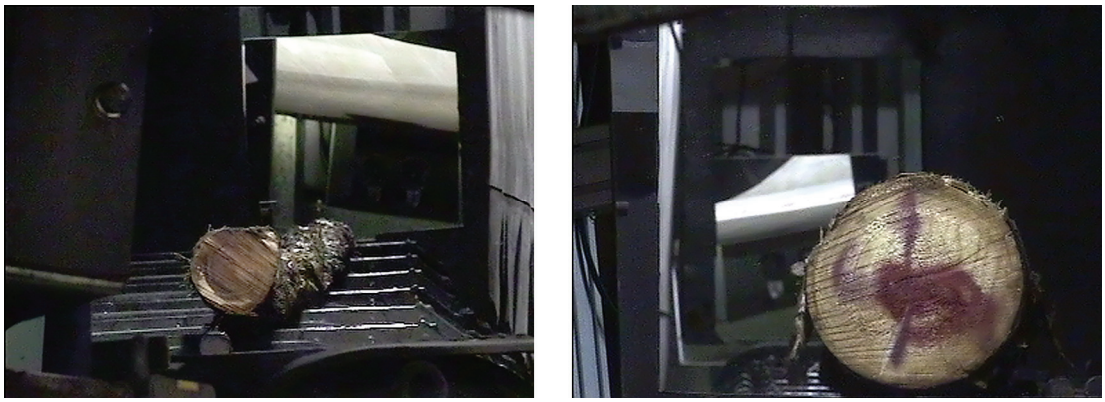
Another option would have been to use region splitting and merging. It is only slightly more demanding to implement, but enables to use a homogeneity criterion that is not limited to the color of individual pixels. The problem with region growing, as well as with sequential edge detection methods, is that they require an initial seed point, which might be given by a user in interactive applications. Our segmentation algorithm is required to operate unsupervised, so other means for initializing the region growing would have to be developed.

Using sequential or parallel edge detection methods for segmentation tends to be rather complicated. They are generally not able to segment images by themselves, but have to be combined with other segmentation approaches [11]. Edge detectors give useful information for image understanding, but we are not interested in analyzing every part of the image. An edge detector applied on a log image would produce a vast amount of edges that would not as such help us to localize the log end.

Physics-based approaches are substantially more complicated than region-based or boundary-based segmentation algorithms that simply operate on color. The simple physical models can be used to detect material changes,

but they regard discontinuity in color as two different materials [35]. This is unacceptable, since logs are far from single-colored objects.

Maxwell and Shafer have developed a complex framework for segmentation of images into regions that correspond to coherent, possibly multicolored, surfaces in the scene [35]. Still the objects in the scene are required to be piecewise uniform dielectrics. Log material is often not uniform (see Figure 12), but contains complex texture, and possibly off-color, rot, snow, or clay. Thus physics-based approaches are not expected to perform best in this case.



a b

Figure 12. Log end images. In practice the log ends are often of rough and inhomogeneous material.

In Chapter 4 a pixel-based classifier was developed for finding an approximate horizontal location of the log. The algorithm searched for pixels whose color is inside the cubic subspace of RGB color space defined by lower and upper limits for the red, green, and blue coordinates. To be able to grade the log, the log end has to be found precisely. Thus we started looking for a color space where the log end could be more precisely discriminated from the background.

6.2 HSV wood color model

We have to specify the color subspace that contains as much wood color, and as little background color as possible. The algorithm then scans through the image and assigns the pixels that have color in the specified subspace to object class. A color subspace that accurately discriminates the log end from the background may be easier to define in some coordinate systems than others.

We analyzed existing log images and observed that in HSV coordinate system the hue component remains fairly constant in log end pixels, but value component varies considerably from log to log. In such case color normalization [36] is often used so that chromaticity can be compared independently of brightness:

$$r = \frac{R}{R + G + B} \tag{21}$$

$$g = \frac{G}{R + G + B} \tag{22}$$

For example, because the skin colors of different people vary mostly in brightness rather than in chromaticity, some face tracking systems use normalized RGB color [37]. Essentially same effect is achieved if RGB color is transformed to a perceptual color space and the brightness component is ignored [38]. While computation is slightly more efficient in a two-dimensional color space, some information is obviously lost—it is no longer possible to use brightness to classify pixels. Normalized RGB color space has the additional flaw that low intensity colors are very noisy [11].

We did not want to ignore brightness completely, so we started experimenting with a model where the log end color is defined by giving lower and upper limits to each of the three HSV components. The ranges were chosen by analyzing existing log images; the range was very narrow for hue, and wider for saturation and value. The conversion from RGB to HSV color space is discussed in Appendix C.

With a large, heterogeneous set of logs, a color subspace defined by mere lower and upper limits for each component grows too large, and background pixels will be incorrectly assigned to object class. We tried to increase the accuracy by creating a list of reference colors and detecting pixels whose color is close to some of the reference colors. We selected manually the reference colors from log images. At first the Euclidean distance was used as a measure of similarity, and later we set three parameters that limit the distance of each color component from the closest reference color. The program then compared each pixel to each reference color.

6.3 The final wood color model

6.3.1 Wood species

With different wood species included, the list of reference colors became too large to maintain, and the segmentation algorithm still was not accurate enough. Especially bark has often color that is similar to the color of a dark log end. Using a single color subspace to identify logs of every species resulted in too many background pixels incorrectly assigned to object class.

We needed a priori information about the wood species. Fortunately such information exists. We are able to read the species of the logs in each batch from a database. The final algorithm uses different color subspaces to represent the color of spruce, pine, birch, and aspen, and combined data in case the detection of the wood species fails.

6.3.2 Adaptive membership tables

The final segmentation algorithm still uses pixel-based classification. However, reference colors are not manually written to a configuration file anymore. A separate application is used to train the classifier. The application examines training patterns and calculates three-dimensional membership tables for the classifier.

The training patterns have been manually cropped from hundreds of existing log end images and organized into training sets. For each sort of wood, and for each of the two cameras, two training sets were collected: object set, which contains log end patterns, and background set, which contains mostly bark. We collected separate training sets for the two cameras because lighting and geometry might slightly differ between the cameras. However, we came to the conclusion that a single membership table can be used for both cameras.

The membership tables are indexed by three color component values. In mathematical terms they define a mapping from the color space to a set of classes. In this case there are only two classes: object and background. The classifier decides the class to which each pixel should be assigned based on the table that has been created for the particular wood species. Because the mapping can be precisely defined with discrete color values, the conversion to HSV color space becomes pointless. Therefore we keep the image data in RGB color space, and use the RGB color vectors as indices into the membership tables.

The membership tables are built from the three-dimensional histograms of the training sets. It should be noted that because the histograms of log end and bark overlap, it is not possible to perfectly discriminate the log end from the background by comparing individual pixel colors. The best that we can do is to find out which colors appear more frequently in log end pixels than in background pixels.

The color resolution of the membership tables is determined by the quantization of the indices. 24-bit RGB vector would provide more than enough color resolution—one table would occupy a large amount of memory (16 million elements), and a large amount of training data would be needed to get reliable values for every element. To store the color table more efficiently we use a different encoding for the indices that takes only 19 bits. Thus the table size is reduced to half a million elements.

We noticed that in general the R, G, B components of wood colors are close to each other (saturation is low), and $R > G > B$. A table index contains G as a measure of the overall brightness. Chromaticity is encoded as the differences $G - R$ and $G - B$. Because brightness is not important for detecting wood color, we use the 5 most significant bits of G, and 7 most significant bits of each of the differences. Conners et al. developed a system for detecting defects in hardwood lumber [9]. They decided to use only the red and blue channels of color images for segmentation. Although it was not justified in their paper, the decision was probably based on the same observations that the difference is greatest between the red and blue components.

6.3.3 Bayes classifier

The classifier that minimizes the average loss incurred in assigning a feature vector to a wrong class is called the Bayes classifier [2]. In our classifier the feature vector is a pixel color and the class is either object or background. We will assume that the loss of classifying a pixel incorrectly always equals to 1, so that the total loss equals to the number of misclassifications. Because we have only two classes, the average loss incurred in assigning a pixel incorrectly to one class reduces to the probability that the pixel actually comes from the other class.

The application that builds the membership tables counts the occurrences of each color in object and background training sets. The number of occurrences of color (R, G, B) in object and background training images are denoted by $n_o(R, G, B)$ and $n_b(R, G, B)$ respectively. The probability that a ran-

dom pixel of which only its color is known belongs to object or background class can then be estimated from those numbers:

$$p_o(R,G,B) = \frac{n_o(R,G,B)}{n_o(R,G,B) + n_b(R,G,B)} \quad (23)$$

$$p_b(R,G,B) = \frac{n_b(R,G,B)}{n_o(R,G,B) + n_b(R,G,B)} \quad (24)$$

The Bayes classifier would assign a pixel to object class if $p_o(R, G, B) > p_b(R, G, B)$ and vice versa. From Equations (23) and (24) can be seen that $p_o(R, G, B) > p_b(R, G, B)$ if and only if $n_o(R, G, B) > n_b(R, G, B)$.

6.3.4 Fuzzy regions

There is often uncertainty associated with the regions that a segmentation algorithm produces, but the algorithm simply partitions the image to the best of its ability. Segmentation is a low-level task that is followed by other processing, so the decisions will affect all the higher-level activities. If the higher-level algorithms are not able to use the information about the uncertainty of the regions, the final output of the system will be biased by the low-level decisions. [11]

The idea behind fuzzy segmentation techniques is that the membership of each pixel in each object is vague, and the information of its uncertainty is retained to higher levels. Often the edges in an image are not crisp, and the higher-level algorithms have more flexibility, when the region boundaries are not fixed in the first step. [11]

To facilitate fuzzy decisions to some extent, we wanted to store in the color tables the degree to which we are confident that a pixel of color R, G, B is part of the log end, instead of just the Boolean value that indicates whether $n_o(R, G, B) > n_b(R, G, B)$ is true or not. The confidence is higher if there are more samples of that color in the training data, and if there is a greater difference between $n_o(R, G, B)$ and $n_b(R, G, B)$.

The class to which a pixel belongs can be decided based on the difference of the numbers, and the magnitude of the difference reflects the confidence of the membership. Because the total number of samples in the training data would otherwise affect the magnitude of all the decision table values, we normalize the differences so that they are not greater than 1. The following membership function assigns each pixel to one of the two classes:

$$m_o(R,G,B) = \frac{n_o(R,G,B) - n_b(R,G,B)}{\max_{r,g,b}\left[n_o(R,G,B) - n_b(R,G,B)\right]} \quad (25)$$

The membership tables we obtained for each of the wood species using the membership function are visualized in Appendix D. Our segmentation starts by scanning the image and classifying pixels according to the membership values. A positive membership value suggests that the color appears more often in log end than in bark. Such pixels will be classified as belonging to object class. The fact that the membership function employs color information as a whole makes this method essentially different from the approaches that operate on one component at a time.

6.4 Discussion

Assuming that the loss of misclassification always equals to 1, the algorithm is optimal pixel-based classifier in the sense that it minimizes the average loss in misclassifications. An assumption that we have made in Equations (23) and (24) is that the training data is representative of all the pixels in an image. We have added more training data when we have noticed that some colors are not classified correctly. The result is that there is relatively more log end pixels in the training data than in the actual images, but the classification works in practice. Limitations of the algorithm include inflexibility to changes in the environment, for example in lighting conditions.

6.4.1 Region-based segmentation

In a constant environment the segmentation algorithm performs as well as classification based on individual pixel colors can perform. The next improvement could be a region-based segmentation algorithm with homogeneity criteria that consider a larger neighborhood than a single pixel.

As we mentioned earlier, physics-based approaches are not designed for objects that are not uniformly colored. An interesting subject for future research would be to study the statistical properties of the log end regions in order to find similar regularities from the histograms of individual logs that physics-based approaches use to detect regions that correspond to a single material. We have already collected image data of log ends, but so far we have only combined the data of different logs into a single histogram. Log end might be located with a region-based approach, such as region splitting and merging, using a homogeneity criterion that sets constraints on the shape of the histogram of each region.

Textural features could be used for region-based segmentation as well. However, they are computationally expensive and might be feasible only for classification after potential defect areas have been found [4]. We give examples of textural features that could be measured when we discuss classifiers in Section 8.2.2.

6.4.2 Combining different segmentation algorithms

Region growing and sequential edge detection methods require an initial seed point. In an unsupervised application the seed points either have to be chosen randomly or by another algorithm. Some approaches use the information obtained from an edge detector to initialize region growing [12]. Because our segmentation algorithm is only required to outline one region in the image, edge detectors do not lend themselves as the basis of our segmentation algorithm. A better approach could be to use our pixel-based classifier to seed region growing.

Another way to combine information from region-based and boundary-based algorithms is by using the result of one algorithm to guide the homogeneity decision of another. Region splitting and merging, or region growing, can use homogeneity criteria that in addition to the other criteria constraints that a region is not homogeneous if it contains edge points. Edges and regions can also be detected independently and boundaries that were not detected by both methods can be removed in post processing. [12]

Those approaches essentially combine a region-based and a boundary-based algorithm that would result in over-segmentation, i.e. produce too many boundaries, if either of them was used alone. Because a larger problem with our segmentation algorithm is that noise and irregular wood texture causes coarse boundaries, it would be particularly interesting to use another algorithm to refine the already detected boundaries.

The *snake* method is commonly used for locating the object boundary by refining an initial plan. It tries to minimize an energy function that is calculated by integrating over the length of the boundary. Typically the energy function depends on the smoothness of the contour and on the image gradients. In our case the membership function might also be used in the definition of the energy function. [39]

7 LOG END RECOGNITION

7.1 Region size constraint

The segmentation algorithm assigns each pixel of the image to either object or background class. The result is stored in a registered binary image consisting of labels. Because wood color alone cannot discriminate the log end from the background perfectly, small isolated clusters of background will be given object label. The most intuitive solution is to discard regions of object labels that are smaller than some threshold.

To calculate the sizes of the regions, an algorithm that divides the object labels into connected components is needed. We implemented a connected components algorithm based on the description and source code given by Ali Rahimi [40]. It uses the disjoint-set forest data structure, described in Appendix E.

The disjoint-set forest is used to group pixels into connected components. The algorithm maintains a two-dimensional table, of the same size as the image, which contains pointers to the elements of the disjoint sets. When the algorithm finishes, every element of the table contains a pointer, and each disjoint set corresponds to one connected component of the image. If a set of pixels is connected, the corresponding pointers in the table all point to elements of a single set.

First connectivity has to be defined. Two pixels are connected if they are adjacent and homogeneous according to some criterion. Adjacency can be defined so that only horizontal and vertical neighbors are adjacent (4-adjacency), or so that also the diagonal neighbors are adjacent (8-adjacency) [2]. A connected component is then a region of an image such that every pair of adjacent pixels inside the component is connected, but none of the pixels inside the component is connected with a pixel outside the component.

We define homogeneity using the membership function given in Equation (25)—two pixels are homogeneous if they both have a positive membership value, or if neither of them has a positive membership value. We chose to use 8-adjacency. If a region has a coarse boundary because of noise, 4-adjacency may incorrectly detect some border pixels as separate regions.

In Figure 13 there are two disjoint sets with three elements, A, B, and C. The sets are represented by trees in the disjoint-set forest and elements are nodes in the trees. A and B are nodes of a single tree. B is the root of the tree, so A contains a pointer to B. C is the only node of the second tree. The connected components of the 4 × 2 pixel image have been extracted, so there is a pointer at each pixel to either A, B, or C. Because 8-adjacency is used, the trees group the pixels into two components.

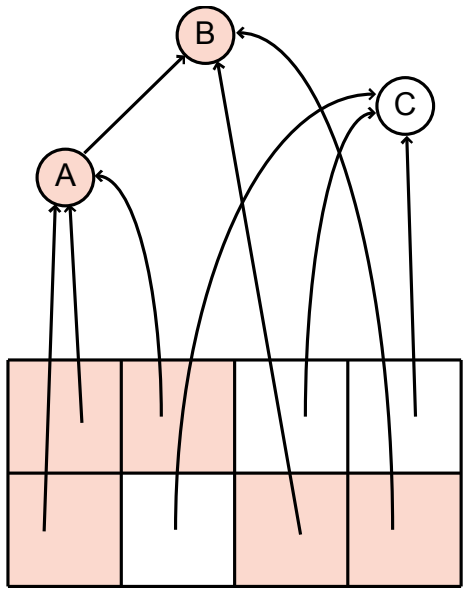


Figure 13. When the connected components algorithm finishes, each tree in the disjoint-set forest corresponds to one connected component of the image.

The algorithm needs only one pass to find the connected components and another pass to label the pixels. The image is scanned pixel by pixel. Let S denote the set of pixels that have already been scanned. If the current pixel is not connected with any of the pixels in S, a new set is created into the disjoint-set forest, and a pointer to the only node of the created tree will be stored at the current pixel. If pixel $p \in S$ is connected with the current pixel, the same pointer that is stored in p will be stored at the current pixel. If there is also another pixel $q \in S$ that is connected with the current pixel, the nodes to which there are pointers at p and q will be linked causing the nodes to become part of a single set.

The 4 × 2 pixel image in Figure 13 has been scanned from left to right, from bottom to top. During the first row three disjoint sets have been created, each of which contains one element, A, C, and B. A pointer to B has been stored at the rightmost pixel of the first row, because the pixel is connected to the previous pixel. Similarly a pointer to A has been stored at the first and second pixel of the second row. However, since the second pixel of the second row is connected to the third pixel of the first row, the elements A and B have been linked. Consequently all the pixels that point to either A or B are part of the same component. Because 8-connectivity is used, a pointer to C has been stored at the last two pixels.

Because the disjoint-set trees are not going to be modified after the image has been scanned, set-specific information may be stored in the root nodes. We store a unique number in each root node to identify the connected components. The image is then scanned for the second time and each pixel is labeled with the number that is found from the root node of the corresponding disjoint-set tree.

The histogram of the label image gives the areas of each of the connected components. Our first implementation simply disregarded the regions that covered a too small area on the grounds that they are most likely just isolated pixels of background or bark that accidentally have the same color as log ends tend to have. Now we can use the membership values to adjust the decision as to which regions to disregard. We define the size of a region as the integral of the membership values over the pixels in each region. We discard all the regions whose size is less than 0.1 times the size of the largest region. The decision will then be affected by both the size of each region and the degree to which its color matches log end color.

7.2 Region shape constraint

Especially problematic regarding the log end recognition is that large regions of bark often get assigned to object class. Often discrimination based on connected components fails inevitably because these regions are connected to the log end. Therefore we started looking for more ways to make use of the geometry in the scene. The size and the location of the log changes from image to image, but it can be localized on the grounds of its shape.

The cross section of the log should be more or less circular. Defects in wood may have different color than healthy wood, so we cannot assume that the entire log end will be assigned to object class. Usually rot appears at the center of the cross section and the circumference is healthy wood, as in the log end in Figure 2(b). However, it is also possible that a defect divides the pixels that will be assigned to object class into two separate regions. Fortunately we know that a peculiar shape usually indicates defects in the wood and the log can be marked as failed without further processing.

In Chapter 8 we will develop an algorithm that uses features of the pixels that have been classified as object pixels to find evidence of defects. Initially we had to calculate the convex hull of the object labels so that any defected interior of the log end would be assigned to object class as well. The output of the algorithm that we developed for finding circular clusters of object labels is

the location and the radius of the circle. Thus it is not necessary to calculate the convex hull anymore. After the best candidate for location and radius is found, we will use another algorithm to examine if the circle really represents majority of the log end pixels. If the circle appears to cover only a small portion of the log end, the log will be marked as failed.

Template matching is under some constraints optimal solution for the problem. Template matching has been extensively used to localize and identify patterns in images [41]. The principle is to evaluate the similarity of a known template—in this case a circle—with the image data at different locations. A commonly used measure of similarity is correlation [2]:

$$c(x,y) = \sum_s \sum_t f(s,t)w(x+s, y+t) \tag{26}$$

In the above equation $f(x, y)$ is an image and $w(x, y)$ is the template. The result, $c(x, y)$, is a measure of the similarity of the template with the image at location (x, y). If the image and the template are regarded to as vectors, correlation gives the dot product of these vectors. Hence correlation can be seen as the angle between the two vectors [41].

Correlation is also the optimal linear spatial filter for detecting a pattern in the presence of additive white Gaussian noise, in the sense that it maximizes the signal to noise ratio at the location of the pattern [41]. Unfortunately correlation may become impractical because of the required amount of computation, if the size and orientation of the template is unknown [2]. Although a circle is invariant under rotation, we would have to compute the correlation with circles of different sizes.

We first experimented with a simple algorithm that makes use of the fact that the log end is usually below other regions that have been assigned to object class. The algorithm starts from the bottom of the image and calculates the distance between the leftmost and rightmost object label on each row. When the algorithm moves up, at the lower semi-circle of the log end region the distance should increase, and at the upper semi-circle the distance should decrease. The row where the distance starts to decrease is a good candidate for the center of the log end.

If the circle has an irregular boundary because of noise, the distance could momentarily decrease before the center of the log end. To avoid making

premature decisions we compared the sum of distances from three consecutive rows, and the algorithm continued until the distance had been lower than the maximum for five times.

Frequently bark that has been assigned to object class makes the algorithm presented above to fail, because the distance will not start to decrease at the center of the log end. Figure 14(a) shows a log whose bark and inner wood have similar colors. In Figure 14(b) the pixels that the pixel-based classifier assigned to object class are tinted green. Chances are that the algorithm would fail on the image because the upper-half of the region is not circular. In the next section we will develop an algorithm that performs better in such situations.

a b

Figure 14. (a) A log whose bark and inner wood have similar colors. (b) The region that the pixel-based classifier assigned to object class is tinted green, and the result of the log end recognition algorithm is tinted red.

7.3 The final log end recognition algorithm

The final algorithm we developed for recognizing the log end shape tries to find the largest circle whose circumference contains mostly object labels. It could be regarded as an approximation of template matching. Template matching with binary images degenerates into counting the number of pixels that equal to 1 in both the image and the template. To improve the performance we do not count all the object labels at the circumference, but take 27 samples; we examine the 3 × 3 neighborhood of three points at the circumference. In addition we do not evaluate circles of every possible size at every possible location.

Again the algorithm finds the leftmost and rightmost object label on each row. One circle will be evaluated per row: the circle centered in the middle of the

leftmost and rightmost object label whose diameter is the distance between those pixels. The 3 × 3 neighborhood of three points, the top of the circle, 45 degrees left from the top of the circle, and 45 degrees right from the top of the circle, will be examined. If there are object labels in all the three neighborhoods, that circle is a candidate for log end location.

The algorithm starts from the bottom of the image moving up until the top of a circle to be evaluated is above the image boundary. The result of the algorithm is the location and diameter of the largest candidate. The right hand image of Figure 14 illustrates the result of the algorithm applied on the left hand image. The region that the pixel-based classifier assigned to object class is tinted green and the circle found using the log end recognition algorithm is tinted red.

After the circle has been found, its vicinity will be examined to verify that the circle represents majority of the log end pixels. If there is a substantial amount of object labels outside but not far from the circle, chances are that the log end recognition has failed, and the circle only covers a small portion of the log end. Therefore we count the number of object labels at the one-pixel wide circumference whose radius is 1.75 times the radius of the detected circle. If there are 25 or more object labels, the log will be marked as failed without further processing. Finally, because it is more important that there will be no bark in the object region than that all the log end pixels will be detected, the radius will be multiplied by 0.9.

8 CALCULATING EVIDENCE OF DEFECTS

8.1 Evidence function

When the log end has been found, we use statistical characteristics of the log end pixels to calculate evidence of defects. Evidence is simply a nonnegative number that indicates how certain the software is that the log contains defects. In this phase the algorithm is not required to identify the type of the defect.

The original requirements stated that each log should be labeled as passed or failed. Thus the software labeled all the logs whose evidence was above a certain threshold as failed. A person at the Mitla office graded all the failed logs. However, it turned out to be difficult to find a single threshold that would be optimal for every batch of logs.

The specifications were changed to provide for a more flexible operation: the logs in each batch are shown to the grader in order of decreasing evidence. That is, the logs that certainly contain defects will be shown first. As more and more logs have been shown, defective logs will appear less and less frequently. The grader can finish grading the batch when he or she believes that at least almost all of the defective logs in the batch have passed already.

We extracted various statistical features of the log end pixels and tried to find correlation between the feature vector and the presence of defects. The features that we calculate are:

- A: The area of the log end.
- (x, y): The coordinates of the center of the log end.
- r: The radius of the log end.
- σ^2: The variance of the intensity over the area of the log end.
- c: The relative lightness of the log end at the circumference, as described below.

We found out that the following is a reliable formula for evidence of defects:

$$h = 288 - (y + r) \tag{27}$$

$$e_y(y, r) = 0.25 \cdot \max(h - 30, 0) \tag{28}$$

$$e_\sigma(\sigma^2) = 10 \cdot \max(\sigma^2 - 8, 0) \tag{29}$$

$$e_c(c) = 3000 \cdot \max(c - 1.05, 0) \quad (30)$$

$$e_r(r) = 30 \cdot \max(10 - r, 0) \quad (31)$$

$$e(y, r, \sigma^2, c) = e_y(y, r) + e_\sigma(\sigma^2) + e(c) + e_r(r) \quad (32)$$

Equation (27) calculates the distance of the bottom of the log end from the bottom of the image (the height of the images is 288 pixels). The bottom of the log end is usually very low in the image, so the distance increases evidence. The color of clear wood varies only little. Thus σ^2 increases the evidence. The purpose of c is to measure how dark the center of the log end is compared to the circumference. Most often rot appears at the center of the log, making it darker. Therefore c also increases the evidence. A very small radius probably means that only a small part of the log end has been detected. If the radius is smaller than 10, the evidence is increased by 10 – r. The coefficients define the importance of each component.

8.2 Discussion

8.2.1 Textural features

The classification is based on tonal features. That is, we use simple statistical properties of the pixel colors to calculate the evidence. A better accuracy could be achieved if textural features were used in addition to the tonal features to calculate the evidence. Textural features are related to the spatial organization of the colors.

As an example of a textural feature that has been used for defect detection is the intensity co-occurrence function $P(i, j, d, \theta)$ [42]. It is defined as the relative frequencies with which two pixels whose intensities are i and j are at distance d from each other in the direction indicated by angle θ. All the parameters are discrete values. For example, the angle can be quantized to 45° intervals, in which case a natural distance metric is the Chebyshev distance (also called the chessboard distance [2]) quantized to 1 pixel intervals. Then for instance $P(i, j, 1, 90°)$ would equal to the number of times two intensities i and j occurred in vertically adjacent pixels in the given neighborhood.

Haralick et al. organized the co-occurrence values into matrices and computed statistical features from these matrices [42]. The matrices are computationally expensive to calculate, but they have performed well in defect detection [4].

Österberg et al. [1] analyze the log end texture by computing the local Fourier transform [2] in various points inside the log end. A neighborhood of each point is transformed to the frequency domain. The dominating frequency and its direction give estimates to the thickness and orientation of the annual rings at that point. Those estimates can easily be determined when the two-dimensional frequency spectrum is transformed to polar coordinates.

The method was able to find the position of the center of the annual rings, and draw annual ring density and orientation maps. There is a relation between the annual ring density and the strength of the wood. Annual ring density and orientation maps can also be used to detect defects. Using their algorithm more information about the quality of the wood could be acquired, but the algorithms are slower and require that the log end is more or less clean and does not contain more than a moderate amount of defects. A combined system could try to analyze the annual rings of logs where they are clearly enough visible.

8.2.2 Learning classifiers

The evidence is calculated in the interactive application that graders use from the features that the unsupervised application extracts. Currently Equation (28) is fixed and the coefficients were determined manually. An algorithm that could learn from user input would be interesting. The algorithm could automatically try to find correlation between the statistical features and the corrections that the human grader makes, and learn to interpret the features correctly.

Constructing the Bayes classifier requires that the probability density functions of the feature vectors in each class, as well as the probability of occurrence of each class, are known [2]. When the exact PDFs are unknown, the Bayes classifier can be constructed if assumptions of the shape of the PDFs are made, such as that the distribution is Gaussian with certain mean and standard deviation. One approach is to estimate these parameters from the training data.

A classifier that is able to learn from training data, but does not require any assumptions about the features, is the multi-layer perceptron. MLP is a feed-forward neural network with three or more layers. It has been used extensively in wood defect detection, among other applications, with good performance [4].

Basically an MLP consists of computing nodes, called neurons, each of which calculates the value of a decision function that is based on the linear combination of input values. A feature vector is connected to the inputs of the first layer of neurons. The outputs of the first layer of neurons are connected to the inputs of the second and so on. The final layer of neurons gives the output of the system. Each output corresponds to one class. [2]

Each neuron contains a parameter that affects its decision. Those parameters define the response of the neural network with a given input. The values of the parameters are established iteratively by training the network. Many different training algorithms have been proposed. [2]

It has been shown that the *back propagation* training algorithm gives the parameters to any MLP that minimize the mean square error to the Bayes decision function. Furthermore, each output of the MLP will approximate the probability of the input vector in the corresponding class. In other words the MLP is a method for approximating the PDFs of feature vectors in a set of classes. The accuracy of the approximation depends on the architecture of the network. [43]

9 DOUBLE FEED DETECTION

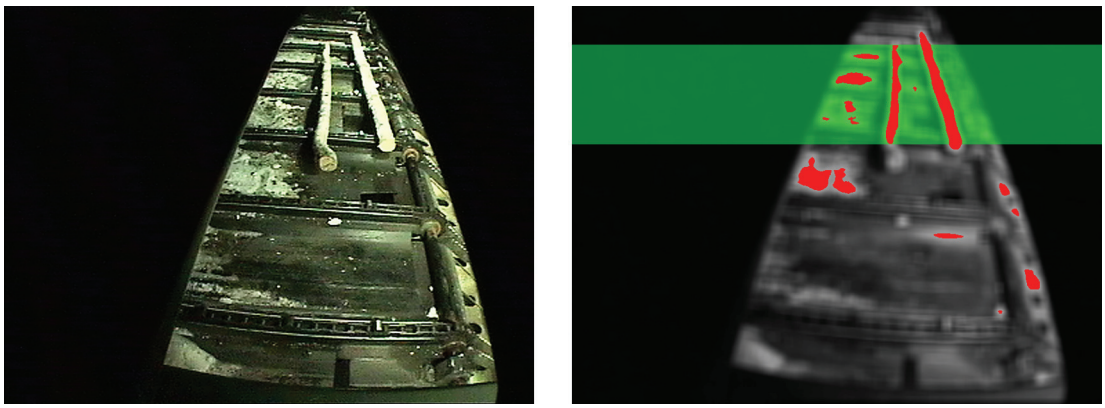
9.1 Detecting images with two or more logs

Occasionally the step feeder lets two or more logs on the transfer conveyor simultaneously. In those cases the person grading the batch has to manually add an extra log to the database. For that reason double feeds have to be detected and shown to the grader whether they contain defects or not.

A double feed is easiest to notice from the image of the overhead camera. It has been installed higher than the two cameras that take images from log ends, and gives a general image of the transfer conveyor. The algorithm uses only pixel intensities. The logs can be fairly well separated from the background by thresholding.

First an averaging filter is applied on the intensity image. Averaging filters can be used to blend small details with the background and make larger objects more discernible [2]. Usually the logs appear vertically in the image, so the average is calculated over a region that is larger in vertical than in horizontal direction. We use linear spatial filtering with a 6 pixels wide and 8 pixels high filter mask, whose all coefficients equal to 1. At the end each pixel is divided by 48. Thus each pixel is replaced by the average of its 6 pixels wide and 8 pixels high neighborhood.

Our intention is to count the number of scan lines in the overhead image that contain pixels from both logs. In case there are scan lines that contain two or more disjoint sequences of bright pixels, the sequences probably come from



a b

Figure 15. (a) A double feed that can be detected using our algorithm. (b) The averaged intensity image with bright areas tinted red and the scan lines that contain bright pixels from both logs tinted green.

different logs, or there is a fork in the log. In either case the images should be shown to the grader. Figure 15(a) shows an overhead image of a double feed. The white areas on the transfer conveyor are snow. In Figure 15(b) the image has been averaged, bright pixels have been tinted red, and scan lines that contain bright pixels from both logs have been tinted green.

One of the shafts of the transfer conveyor was too bright, so we painted it black. Also, snow forms blocks of bright pixels, but the shape of the blocks is essentially different from that of the logs. We use the same connected components algorithm that we described in Section 7.1, and calculate features that describe the shape of each region to discriminate between logs and blocks of snow:

- ***w***: The width of the bounding rectangle, i.e. the horizontal distance between the leftmost and rightmost pixel.
- ***h***: The height of the bounding rectangle, i.e. the vertical distance between the uppermost and lowermost pixel.
- ***A***: The area of the region.
- ***t***: The thickness of the region, as described below.

The shape of the log in the overhead image is long but narrow. The snow blocks on the other hand are usually wide, but the chains of the transfer conveyor limit their height. The thickness feature tries to capture this difference.

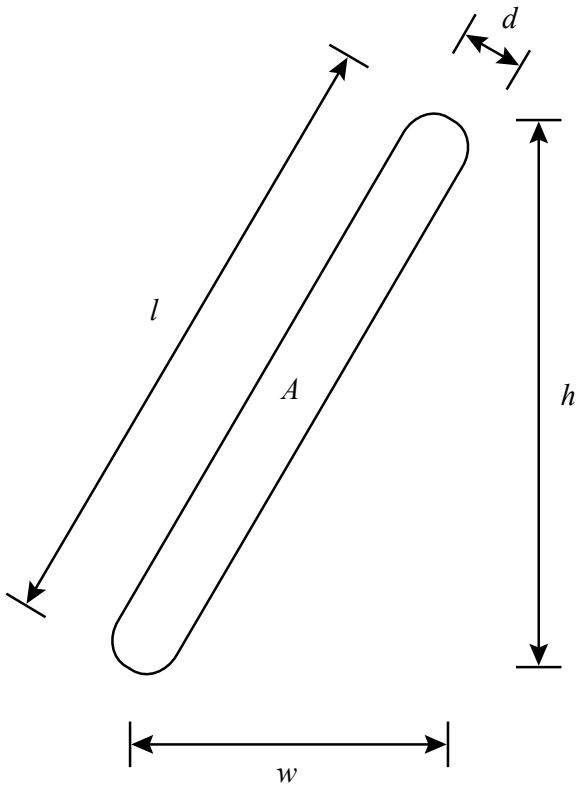


Figure 16. Features describing the shape of the log region.

Assuming that the region has a rectangular shape, we define two measures, d and l that estimate the diameter and length of the region respectively. Their definition is derived from the geometry in Figure 16:

$$l = \sqrt{w^2 + h^2} \tag{33}$$

$$d = \frac{A}{l} = \frac{A}{\sqrt{w^2 + h^2}} \tag{34}$$

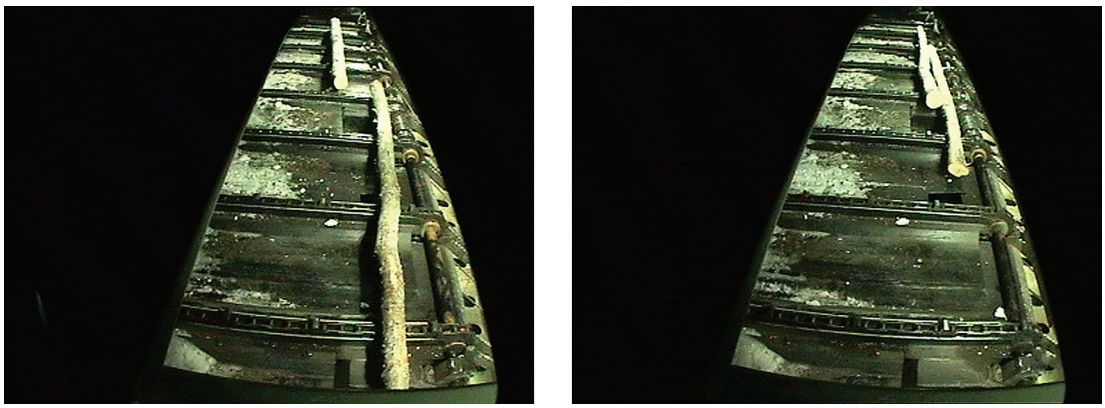
We define thickness as the ratio of d to l:

$$t = \frac{d}{l} = \frac{A}{w^2 + h^2} \tag{35}$$

Regions that are long and narrow have smaller thickness than short and wide regions. Square regions have maximal thickness, t_{max} = 1 / 2. The algorithm accepts only regions that have small thickness: t ≤ 0.185. Thickness is an unreliable discriminator when the region is small. Therefore we set a lower limit on the height of the region: h ≥ 25 pixels.

The double feed detector has performed somewhat accurately, but it is evident that the algorithm is not foolproof. Situations when the algorithm usually fails are listed below:

- Two logs enter the transfer conveyor side by side, so there are no rows in the overhead image that contain pixels from both logs. As an example of such image, see Figure 17(a).
- The logs are horizontally so close to each other that there is only one bright area. In Figure 17(b) there are only few dark pixels between the bright logs.



a b

Figure 17. Double feeds that may not be detected using our algorithm. (a) The logs enter the transfer conveyor side by side. (b) The logs are so close to each other that they appear as a single bright area.

- One of the logs has turned 90 degrees. The logs form a cross in the overhead image, so that most scan lines contains at most one continuous sequence of bright pixels.

9.2 Discussion

Ideally the size of the log in the overhead image would not depend on its position on the transfer conveyor. Unfortunately physical limitations force us to use a lens whose focal length is very short. As a result the images suffer from *keystone distortion*. That is, logs in the other end of the transfer conveyor are smaller than logs near the camera. In addition there is strong radial distortion in the images. The image could be transformed to a coordinate system that corresponds to the surface of the transfer conveyor, if the mapping from the image pixels to the surface locations was known.

The process of determining a mapping between world space points and the locations on the image plane where those points will be projected to is called *camera calibration* [44]. Usually the calibration involves taking images of calibration planes. As stated in Appendix A, assuming the simple pinhole camera model, the image plane point (u, v) where a world space point (x, y, z) is projected to is given by a matrix multiplication:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{36}$$

The unknown 3 × 4 element transformation matrix $\mathbf{T}$ defines the mapping from the world space to the image plane. We are interested in the *back-projection problem*, i.e. given an image plane point we want to know the corresponding world space point. Naturally $\mathbf{T}$ is not invertible, since a world space point anywhere in a single line of sight will be projected to a single image plane location. Fortunately we can restrict the world space points to locations (x, y) in a coordinate system that spans the surface of the transfer conveyor. When nonlinear lens distortions are neglected, the transformation relating (x, y) to (u, v) can be described by a perspective transformation [45],

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \mathbf{U} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}, \tag{37}$$

where $\mathbf{U}$ is a 3 × 3 element transformation matrix. The camera could be calibrated using coordinates taken from known locations on the transfer conveyor. Given n calibration points whose locations $\mathbf{p_i}$, on the surface of the transfer conveyor, and their corresponding image plane locations $\mathbf{q_i}$ are known, we can form the system

$$\begin{bmatrix}\mathbf{p}_1 & \mathbf{p}_2 & \cdots & \mathbf{p}_4\end{bmatrix} = \mathbf{U}\begin{bmatrix}\mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_4\end{bmatrix}. \tag{38}$$

Matrix $\mathbf{U}$ can be solved in the least squares sense using the matrix pseudoinverse [44]. If the second order terms of the measured image locations were used in the parameterization ($\mathbf{q_i} = [u^2, uv, v^2, u, v, 1]^T$), and the matrix $\mathbf{U}$ were 3 × 6, the model would account for other kinds of lens distortions as well [45].

10 RESULTS

10.1 Practical concerns

The system has been developed in cooperation with the graders who use it at the wood handling station. It was surprising how many practical concerns had to be solved before the system could be used, even though in ideal conditions the problem would be relatively easy to solve using well-known algorithms.

Feedback of the current version has been positive, although the delay caused by the software is an issue. The software is able to process 33 logs per minute on a 2.8 GHz Pentium 4 computer. The algorithm is quite fast, considering that the software has to process both log end images and detect double feeds. However, as described in Chapter 3.5, there are still unsolved real-time problems that prevent us from running the defect detection while logs are being photographed. That is the main cause of the delay.

10.2 Defect detection

The original specifications, given in Chapter 1.2, stated that the software labels each log as passed or failed and a person grades all the failed logs. The final version gives more flexibility to the grader: the logs in each batch are shown in order of decreasing amount of evidence of defects, and the grader can choose to stop at any time. It is up to the grader's judgment to stop when defects occur so rarely that it is futile to continue manual grading. The rest of the logs are assumed to be free of defects. The price of the flexibility is that measuring the performance of the defect detection is difficult.

We made a small modification to the interactive grading application to collect some statistics over a period of two and a half months. Because the defect detection is not able to run while logs are being photographed, it may take several hours after the measurement system has processed a batch, before the defect detection is ready. Sometimes the grader chooses to grade the entire batch instead of waiting for the automatic defect detection to finish. Batches that had been graded entirely manually were excluded from the defect detection statistics, leaving us statistics from 128 batches.

The application records the logs that the grader chooses to view. They can be regarded as failed logs in a sense similar to the failed logs of the original specifications—according to the judgment of the software they are the logs that contain most likely defects. However, the grader has to view more logs

than just the ones that the software considers defective: logs whose dimensions do not meet the specified standards, as well as double feeds, have to be graded in any case. They are displayed first regardless of their defect detection results.

There were 153 logs on average per batch, of which the grader had chosen to view 39.4 logs per batch. Each batch contained an average of 23.3 logs that were either detected as double feeds by the software, or whose dimensions did not meet the specified standards. To calculate the performance measures that were specified in Chapter 1.2, we would need to know which logs actually contained defects. We were unable to collect such statistics, but after several iterations of evaluation and development, the users acknowledged that the current version performs with adequate accuracy.

10.3 Double feed detection

We were able to obtain a list of logs that the graders have added to the database, and therefore the performance of the double feed detection could be evaluated. The statistics were gathered from 387 batches of logs. On average 2.53 logs per batch were detected correctly as a double feed and the grader had inserted a new log.

On average 0.243 logs that were inserted had not been detected as a double feed by the software. However, the graders did not always view all the logs in the batch, so there are probably some double feeds that neither the software nor the grader had detected. The opinion of the graders was that especially birch causes double feeds that the software does not always notice, but they are mostly situations where software detection would be difficult, and it would be more feasible to correct the flaw in the step feeder than try to improve the software.

On average 1.47 logs per batch were detected as a double feed by the software, even though no logs were inserted by the grader. The misjudgment was often caused by sunlight that was let into the room. Then again, the grader had not always inserted a log in the database even though the judgment of the software had been correct, such as in the case of a fork.

11 CONCLUSION

Computer vision can be a valuable tool for grading wood in various stages of the production chain from a forest to a finished product. In a measurement station where the photographing conditions are relatively good, computer vision can provide useful information even if the logs are untreated and regular low-end cameras are used.

We have implemented a computer vision system that reduces the work load of a human grader by automatically detecting the logs that clearly do not contain any defects. The system is in industrial use in Stora Enso wood handling terminal in Uimaharju. In order to be of practical use the software that we implemented was required to select from a sequence of frames the one where the log end appears, localize the log end from the image, examine the log end to detect defects, and detect possible double feeds.

The main contribution of this thesis was the application of well-known algorithms to a practical situation. Defect detection was separated into a real-time and a non-real-time component. Especially in real-time software simple and well known algorithms are still often preferable.

We used an image segmentation algorithm that is based on detection of wood color. Knowledge of the shape and orientation of the logs was used to discriminate the log end from the bark. Uniform and intensive illumination helps to detect defects. The images were taken in a generally dark environment, but especially snow complicates the log recognition in winter. The system was designed to tolerate lower quality images. However, because the log end may be dirty, this method will never be perfectly accurate. All the logs that the unsupervised software is unable to understand will be shown to a human grader.

The new software does not deliver any new information, but reduces the mechanical work that has so far been done by a human grader. There are methods that provide more information, but usually they demand a better image quality and their installation costs more. The speed of the grading algorithm may also be an issue.

The next step could be identification of the type of the defect. It is also plausible that the accuracy at which the defects are detected could be improved to reduce the work load of the human grader even more. Maybe in the future even in demanding environments more information can be acquired that en-

ables more accurate sorting of logs according to their quality and delivers a better value yield. With a computer vision system integrated into a harvester, the cutting of trees could be optimized and the logs could be sorted already in the forest [1]. In such environments there is no a priori knowledge of the wood species, lighting conditions may be worse, focusing the camera is an issue, and localizing the log end is difficult.

APPENDICES

Appendix A Camera model

Potmesil and Chakravarty present the basic law governing image formation through a lens [46]. An object appears exactly sharp in an image, if the following equation holds:

$$\frac{1}{D}+\frac{1}{V}=\frac{1}{F} \tag{39}$$

Here D is the distance from the lens to the object, V is the distance from the lens to the sensor, and F is the focal length of the lens (Figure 18). If an object were at infinity, its image would appear sharp at the focal distance behind the lens. In practice the lens has to be focused by increasing the distance between the lens and the sensor. Parameter f in Figure 18 represents focusing, i.e. the distance in additional to the focal length required to get a sharp image of the object.

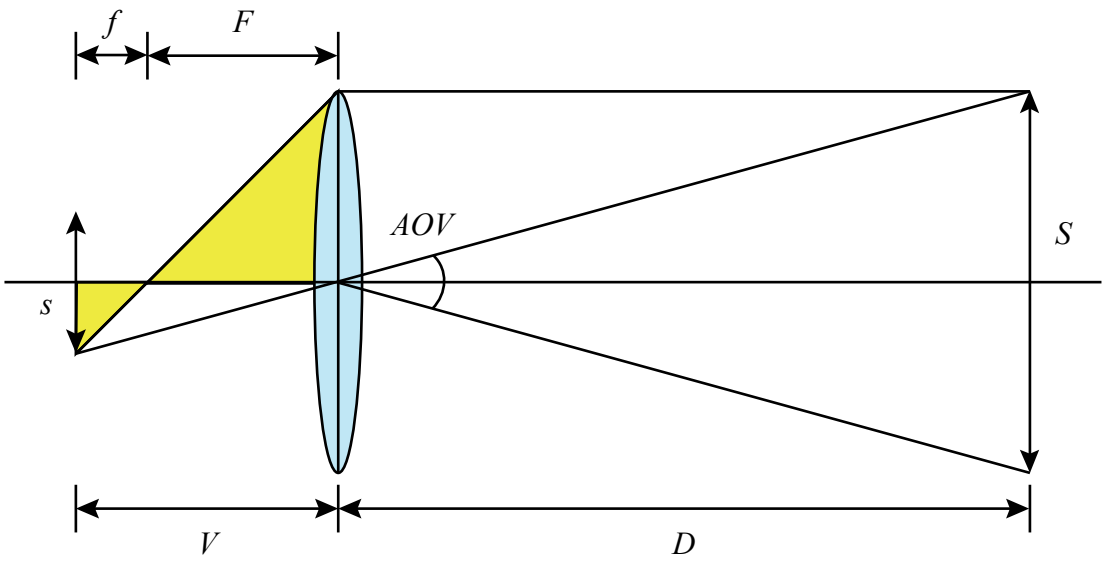


Figure 18. Image formation through a lens. An object is on the right side of the optical axis (size S), and its image is formed upside down on the left side (size s). The yellow triangles on the left side of the lens are similar.

The part of the world that is visible to the camera at any given time is called *field of view*. Its extent can be measured as the angle between its horizontal or vertical edges, called *angle of view*. A formula for horizontal or vertical angle of view can easily be derived from Equation (39) using the geometry in Figure 18, when S is considered as the maximum width or height of an object that fits into the image, and D as the distance of the object from the lens:

$$\frac{S}{D} = \frac{s}{V} = \frac{s}{F + f} \tag{40}$$

$$\tan\frac{AOV}{2} = \frac{S}{D} \tag{41}$$

$$AOV = 2 \cdot \arctan\frac{s}{F + f} \tag{42}$$

However, for our purposes the interesting quantity is the ratio of S to D. Equation (42) shows that field of view depends on the size of the sensor, the focal length of the sensor, and focusing. Because correct focusing depends on the focal length, as well as the distance of the object, the ratio becomes quite complex to calculate. From Equation (40):

$$D = S\frac{F + f}{s} \tag{43}$$

The two similar yellow triangles on the left side of the lens in Figure 18 give another equation:

$$\frac{S}{F} = \frac{s}{f} \tag{44}$$

$$S = \frac{sF}{f} \tag{45}$$

Combining Equations (43) and (45):

$$D = \frac{F^2}{f} + F \tag{46}$$

$$f = \frac{F^2}{D - F} \tag{47}$$

Substituting into Equation (40):

$$\frac{S}{D} = \frac{s}{F + \frac{F^2}{D - F}} \tag{48}$$

The formula can be significantly simplified noting that f << F. The effect of focusing to field of view is clearly insignificant with respect to the precision we need. For example, assuming F = 50 mm, and D = 6 m, Equation (47) gives f

= 0.42 mm. Approximation f = 0 (the lens is focused to infinity) yields a simple equation for field of view:

$$\frac{S}{D} = \frac{s}{F} \tag{49}$$

If the aperture size of a camera would be infinitesimally small, every object appeared sharp regardless of its distance to the camera and regardless of the distance of the sensor to the lens. This idealistic model is called the *pinhole camera model*. The pinhole model results in a simple relation between world space points and the locations on the image plane where the world space points will be projected to. Using homogeneous coordinates the mapping from the world space to the image plane can be expressed as a single matrix multiplication [46]:

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{50}$$

Here (x, y, z) is the world space point, and $\mathbf{T}$ is a 3 × 4 element transformation matrix that depends on the location and orientation of the camera, and the size of the viewing pyramid [46]. The corresponding image plane location (u, v) can be obtained by dividing each component of the resulting vector by s.

Unfortunately every real camera has a finite aperture size. The image of an object is sharp only if the camera is focused to the correct distance, i.e. Equation (39) holds. A point light source at any other distance will be projected as a blurred circle on the camera sensor, called the *circle of confusion*. The depth of the range where objects appear sharp, called the *depth of focus*, depends on how small circles the camera sensor is able to resolve. If the circle of confusion is smaller than the resolving ability of the sensor, it will be seen as a sharp point in the image.

Potmesil and Chakravarty derived a formula for depth of focus, based on the commonly accepted standard that the human visual system is not able to resolve areas whose diameter is smaller than 1/1000 of its distance from the eye [46]. The resolution of a CCD sensor is different from that of the human retina. A captured image is sharp unless the circle of confusion is larger than the size of one pixel in the sensor. The resolution of the images we take is

384 × 288 pixels, and the size of the sensor in our cameras is 1⁄3” (4.8 mm x 3.6 mm), so the width of one pixel in the sensor is 3.6 mm⁄288 = 0.0125 mm (the height of a pixel is equally large, 4.8 mm⁄384). According to Equation (49), an area whose diameter is 0.0125 mm · D⁄F, has a diameter of 0.0125 mm in the sensor. In other words, the CCD sensor cannot resolve areas whose diameter is less than (0.0125 mm⁄F) times the distance of the area.

Potmesil and Chakravarty expressed depth of focus as the far (D^+) and near (D^-) boundaries for the area that is sharp, when the lens is focused to distance D. Their formula is expressed below using the resolving ability of our CCD sensor:

$$H = \frac{F}{n} \cdot \frac{F}{0.0125mm} \tag{51}$$

$$D^+ = \frac{D}{1 - \frac{D}{H}} \tag{52}$$

$$D^- = \frac{D}{1 + \frac{D}{H}} \tag{53}$$

H is the *hyperfocal distance* of the camera—it is the nearest distance, such that if the lens is focused at that distance, the depth of focus stretches to infinity. It depends on the aperture size (the ratio of focal length to aperture number, F/n) and the permissible size for circle of confusion. Depth of focus is the distance between the far and near boundaries:

$$DOF = \frac{DH}{H - D} - \frac{DH}{H + D} = \frac{2D^2H}{H^2 - D^2} \tag{54}$$

There is no simple relation between distance, focal length, aperture size, and depth of focus, but the following trends can be seen:

- Increasing distance yields greater depth of focus.
- Decreasing focal length yields smaller hyperfocal distance and greater depth of focus.
- Decreasing the aperture size yields smaller hyperfocal distance and greater depth of focus.

Appendix B Linear spatial filtering

All the algorithms we developed perform in the *spatial domain*, as opposed to the *frequency domain*. In other words, the operations are performed directly on the pixels, instead of transforming the image first. Many elementary image processing tasks can be performed using linear spatial filter. Linear filters are often preferred, because their behavior has been extensively and theoretically studied. [2]

Linearity means that both of the following are true:

1. The result is identical if two images are first added together, and then an operation is performed on the sum image, or vice versa.
2. The result is identical if the pixels of an image are first multiplied by a constant and then an operation is performed on the image, or vice versa.

This is expressed in the following equation, where H is a linear filter, a and b are constants, and f and g are images: [2]

$$H(af + bg) = aH(f) + bH(g) \tag{55}$$

A linear filter that is defined over a local $m \times n$ pixel neighborhood is characterized by an $m \times n$ element mask. Elements of the mask are called coefficients, and the response of the filter is the sum of products of the coefficients and the corresponding image pixels, as shown in Figure 19. The value of each pixel in the filtered image is given by the response of the filter over the $m \times n$ pixel neighborhood of the corresponding location in the source image. [2]

If $w(x, y)$ is an $m \times n$ element mask and the source image is denote by $f(x, y)$, the following equation gives the result image $g(x, y)$: [2]

$$g(x,y) = \sum_{s=-\frac{m-1}{2}}^{s=\frac{m-1}{2}} \sum_{t=-\frac{n-1}{2}}^{t=\frac{n-1}{2}} w(s,t) f(x+s, y+t)\,. \tag{56}$$

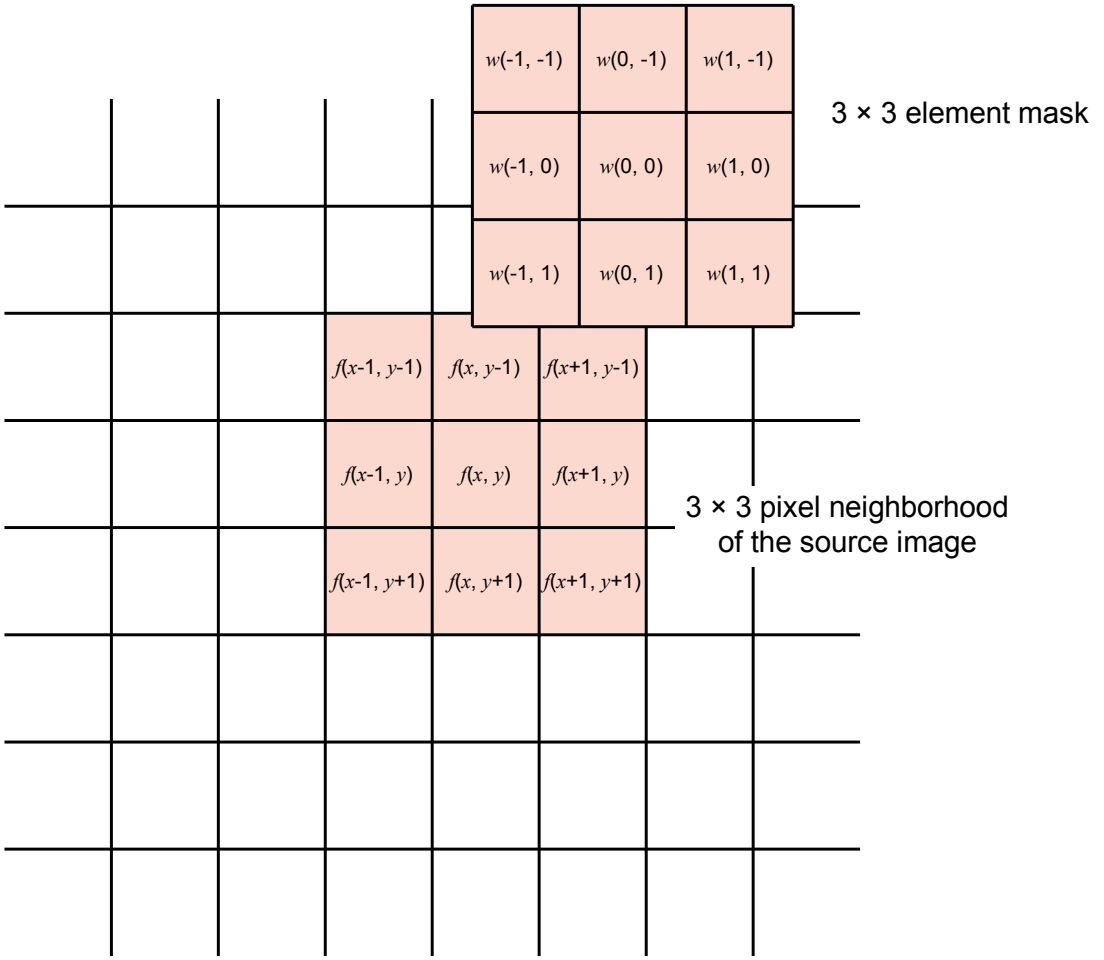


Figure 19. A filter characterized by a 3 × 3 element mask is applied to a source image. The value of the pixel at (x, y) in the resulting image is a linear combination of the corresponding 3 × 3 pixel neighborhood of the source image, with the mask elements as coefficients.

Equation (56) is not defined at the edges of the image. Sometimes it is acceptable that the filtered image simply becomes smaller than the original image. Gonzalez and Woods give two other possible solutions: the image can be filtered using only the section of the mask that falls inside the image boundaries, or the source image can be padded by adding rows and columns [2]. Both of them have identical results if the padding is done by adding zeros. Some filters are designed to produce a small response over homogeneous areas. Discontinuity at the image boundary causes them to produce a larger response even if the image is homogeneous. Thus it might be better to pad the image by duplicating the outermost pixels [2].

Appendix C Conversion from RGB to HSV color space

Conversion from RGB to HSV color space was derived by Smith [19]. Actually the HSV coordinate system Smith presented is not exactly cylindrical. It was intentionally formulated so that the transformation from RGB does not use trigonometric functions. Smith calls the coordinate system the *hexcone model*. We present the algorithm here:

$$V = \max(R, G, B) \tag{57}$$

$$\Delta = \min(R, G, B) \tag{58}$$

$$S = \frac{\Delta}{V} \tag{59}$$

If saturation equals to 0, hue is not defined. Otherwise hue depends on the order of majority of R, G, and B:

$$R \geq G > B \Rightarrow H = 1 - \frac{R - G}{\Delta} \tag{60}$$

$$G > R \geq B \Rightarrow H = 1 + \frac{G - R}{\Delta} \tag{61}$$

$$G \geq B > R \Rightarrow H = 3 - \frac{G - B}{\Delta} \tag{62}$$

$$B > G \geq R \Rightarrow H = 3 + \frac{B - G}{\Delta} \tag{63}$$

$$B > R > G \Rightarrow H = 5 - \frac{B - R}{\Delta} \tag{64}$$

$$R \geq B \geq G \Rightarrow H = 5 + \frac{R - B}{\Delta} \tag{65}$$

$$H = \frac{H}{6} \tag{66}$$

It should be noted that the transformation to HSV coordinate system does not increase or decrease the amount of information that the color coordinates contain. Assuming that the mapping from Cartesian RGB coordinates to cylindrical HSV coordinates is “one-to-one” (or more formally, assuming it is bijective), the problem can be solved in either coordinate system. Only the

specification of certain color subspaces becomes more convenient in HSV coordinates.

For the sake of completeness, we mention that in practice the computation is done using a finite precision. Because the transformation is non-linear, it introduces rounding errors. Consequently, the mapping from the discrete RGB coordinates to the discrete HSV coordinates is not exactly injective. [17]

Shih has measured the errors when colors are transformed between RGB and HSV color space, among some other perceptual spaces. He assumes 24-bit *pixel depth*[1], which is still the most commonly used precision. Real numbers were used for calculations, and results were rounded in each color space. He found out that the extreme error, when transforming colors from RGB to HSV and back, was ±3.1 for each component. Shih opines that the error is insignificant for human visual inspection, but may be worth considering, when the images are going to be processed analytically in the new color space. [17]

[1] The number of bits needed to store one pixel. 24-bit images store each of the three color coordinates as 8-bit integers.

Appendix D Membership tables

We used a fuzzy membership function to classify pixels as lying inside the log end region or not. Values of the membership function were calculated into three-dimensional decision tables, one for each sort of wood. The membership function is based on two numbers, $n_o(R, G, B)$ and $n_b(R, G, B)$, that are calculated from training patterns. They represent the number of occurrences of color (R, G, B) in log end and background pixels respectively. The membership function is shown below:

$$m_o(R,G,B) = \frac{n_o(R,G,B) - n_b(R,G,B)}{\max_{r,g,b}\left[n_o(R,G,B) - n_b(R,G,B)\right]} \tag{67}$$

We cropped log end and background regions from existing log end images and combined them into object and background training sets respectively. The background training set consists mostly of bark. A positive membership value indicates that the color has appeared more often in log end pixels than in background.

The membership tables are visualized in the contour maps in the following pages. R component is on the horizontal axis and B component is on the vertical axis. The range of possible values of each component is from 0 to 255. The grid has been calculated by integrating the membership values over the values of green component from 0 to 255. In other words the color of the contour map at coordinates (x, y) represents the magnitude of the sum

$$\sum_{G=0}^{255} m_o(x,G,y)\,. \tag{68}$$

Red areas indicate colors with a negative membership value. Those areas contain bark color and other background colors. Green areas indicate colors with a positive membership value. Those colors have appeared more often in log end pixels. A general trend that can be seen from the contour maps is that especially the color of bark pixels is very little saturated and largely independent of brightness. The color of log end pixels depends to some extent on brightness, but mostly on chromaticity. The fact that red and green areas are not well separated from each other shows that color cannot perfectly discriminate the log end from the bark.

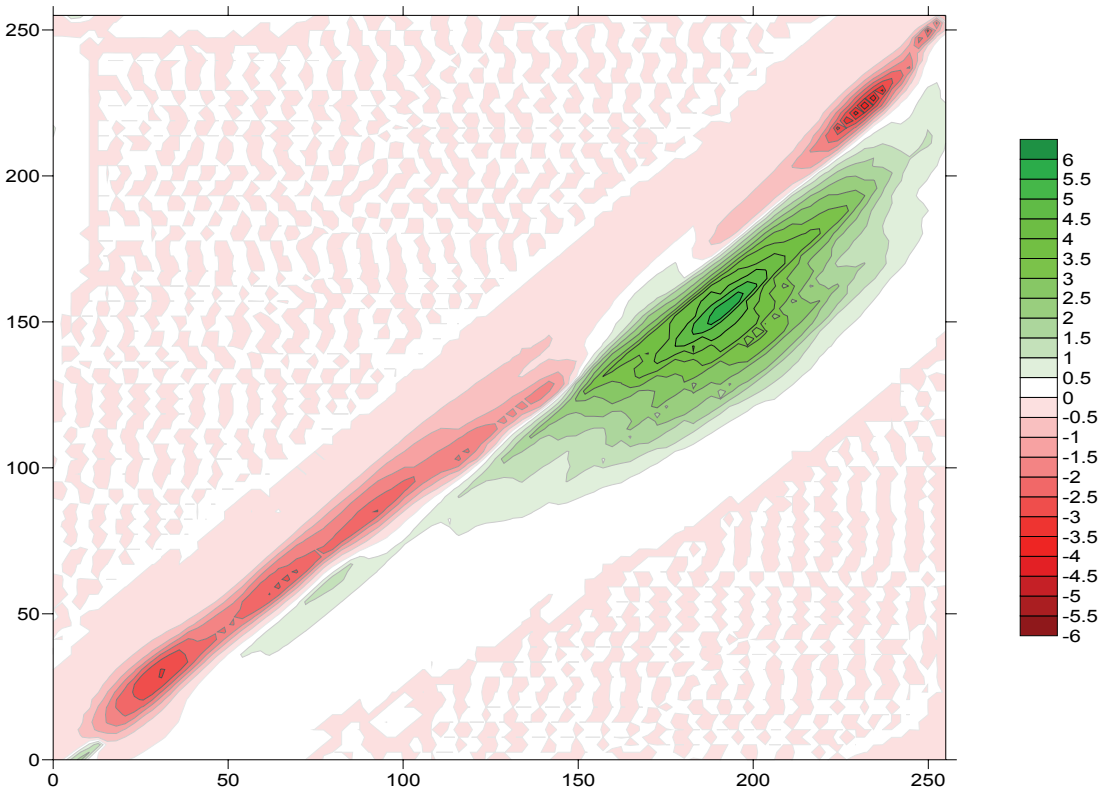


Figure 20. A cross section of the membership table for aspen.

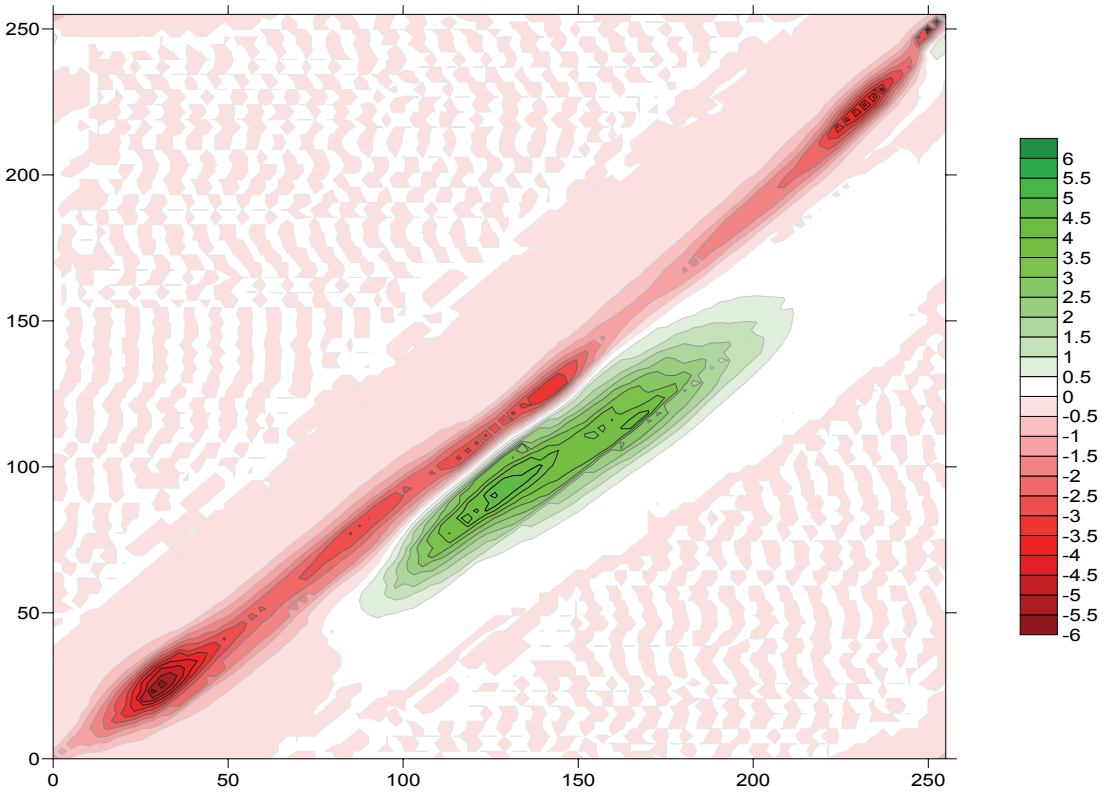


Figure 21. A cross section of the membership table for spruce.

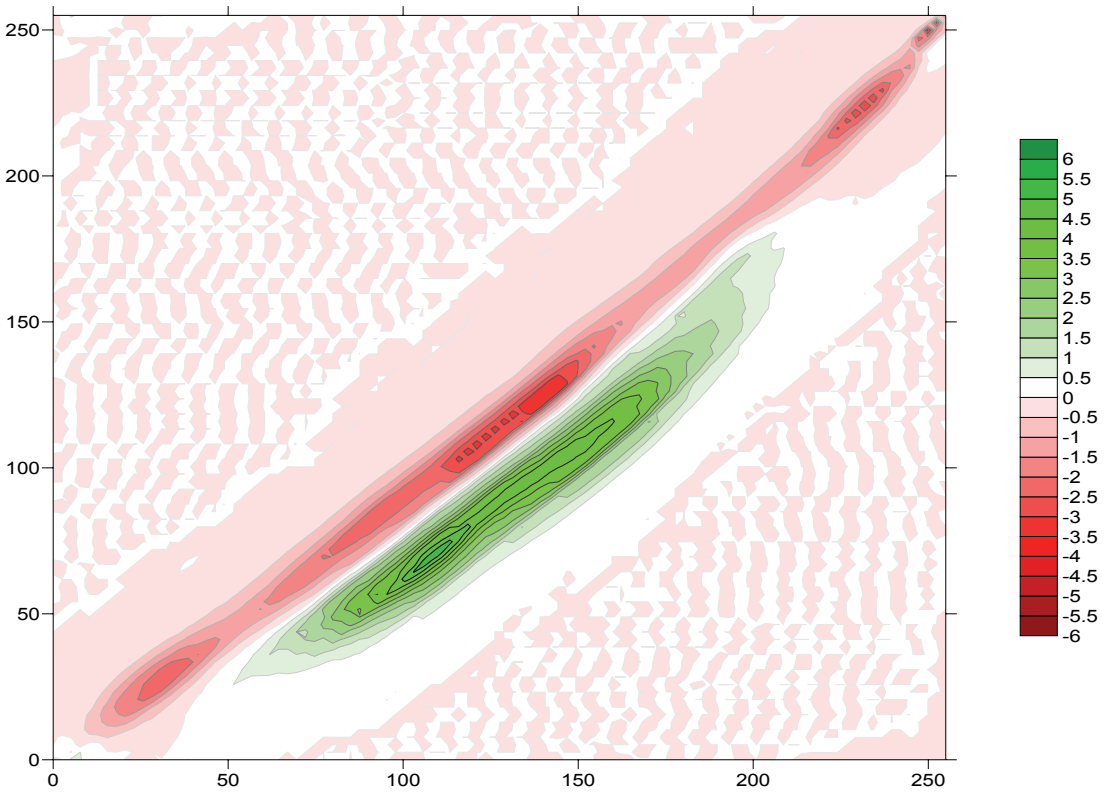


Figure 22. Cross section of the membership table for birch.

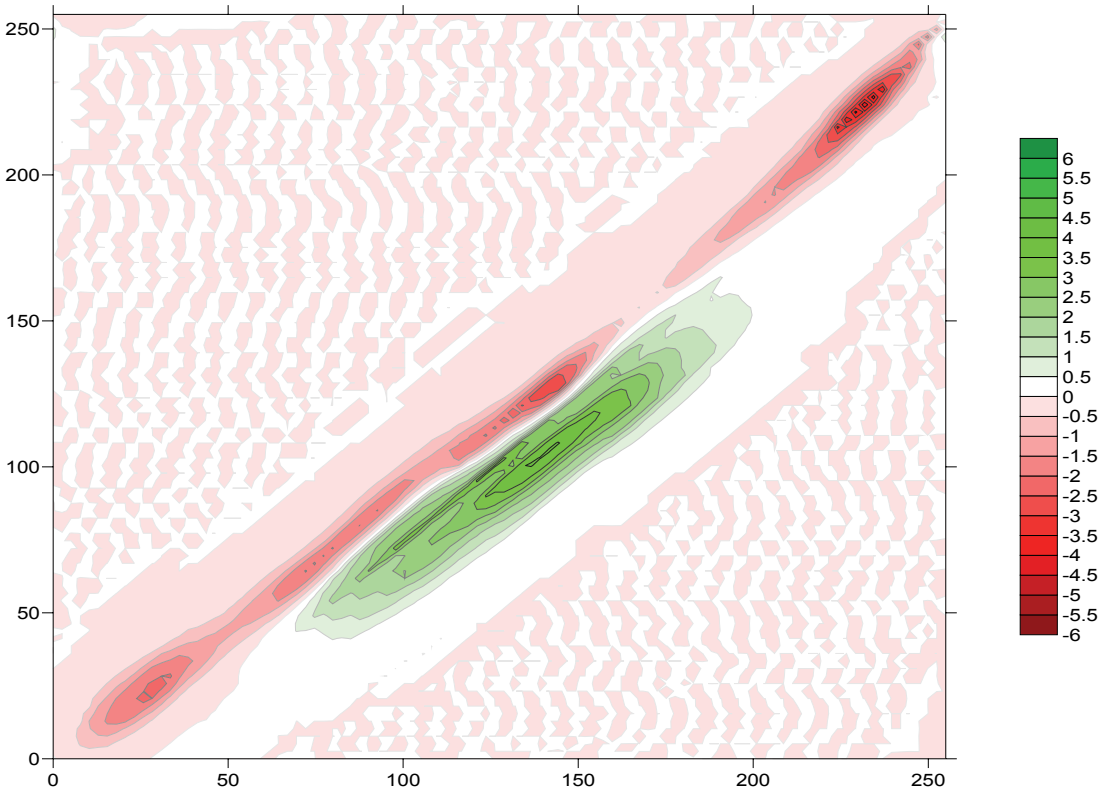


Figure 23. A cross section of the membership table for pine.

Appendix E Disjoint-set forest data structure

The connected components algorithm that we use relies on disjoint-set forest data structure. We implemented the data structure based on the explanation by Cormen et al. [47]. Disjoint-set forest is an implementation of disjoint sets with “practically” linear running time. Our implementation of the data structure is described here.

Disjoint-set forest is a data structure that maintains a collection of dynamic sets. The sets are disjoint, meaning that each element belongs to exactly one set. Each set is represented by a rooted tree. Elements of the sets are nodes of the trees, implemented as objects allocated on the heap. Each node contains a pointer only to its parent node. The parent pointer of a root node points to the node itself. To facilitate certain optimizations, a *rank* that gives an upper bound on the depth of the hierarchy below the node is also stored in each node object.

Creating a set is trivial. A node object is allocated whose parent pointer points to itself. The rank of the node is set to 0. A pointer to the object is added to a list of all the nodes on the heap, so that the memory they occupy can be released when the algorithm finishes.

The set into which each node belongs is identified by the root node of that tree. The connected components algorithm also stores set-specific information into the root node. However, it should be noted that a union operation always changes the root node of one of the sets.

The root node of the tree to which a given node belongs can be found by following the parent pointers recursively until a node is encountered whose parent pointer points to itself. Deep hierarchies make the operation slow, so each time the recursive function returns, the parent pointer of the current node will be set to point directly to the root node. This optimization is called *path compression*.

The key to an efficient connected components algorithm is *link* operation, which efficiently produces the union of two sets. Given two elements, A and B, link operation combines all the elements from the sets whose elements A and B are into a new set. If both the sets have the same root node, A and B are elements of the same set, and nothing has to be done. Otherwise the parent pointer of one of the root nodes is set to point to the other root node.

Finding the root nodes of both the sets is the most expensive procedure involved in the link operation. To keep the hierarchy as flat as possible, another optimization in addition to path compression called *union by rank* is used. Union by rank means that when linking two root nodes, the one with the lower rank is set to point to the other. If two trees with equal ranks are linked, the rank of the resulting tree will be increased.

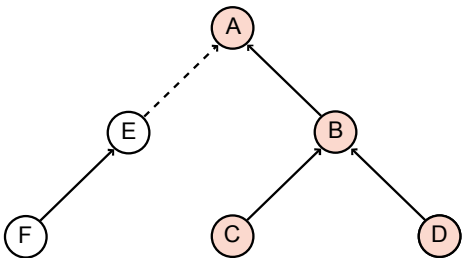


Figure 24. Two disjoint sets are linked by setting the root of one of the trees to point to the root of the other.

The resulting tree is likely to become flatter that way, since the tree whose root node has higher rank has probably deeper hierarchy. In Figure 24 one tree contains nodes A, B, C, and D, and another tree contains nodes E and F. Because the one that contains only nodes E and F has flatter hierarchy, it is more efficient to set the root of that tree (E) to point to the root of the other tree (A), than the other way around.

Appendix F Glossary of color terminology

Hue: The attribute of visual sensation that depends on the wavelength of the perceived light. Color names, such as red, yellow, green, and blue, denote hue.

Brightness: The attribute of visual sensation that depends on the intensity of the perceived light [36]. Bright light appears more intense, while dim light appears less intense. In contrast to lightness, brightness indicates the absolute intensity of light.

Lightness: Brightness in proportion to the brightness of an object perceived as “white” under similar lighting conditions [36]. Light objects appear to emit more light, while dark objects appear to emit less light. In contrast to brightness, lightness indicates how bright an object appears relative to other objects in the scene. However, lightness and brightness are often used interchangeably, and defined in different ways in some color coordinate systems.

Value: The name of the coordinate that relates to brightness in HSV color coordinate system [19].

Chroma: The attribute of visual sensation that distinguishes a chromatic light from an achromatic light of the same brightness [36]. In physical terms, this means the intensity of the dominant wavelength in the perceived color spectrum; a color that has high chroma has a high peak in the frequency spectrum, and also high saturation compared to other colors of the same brightness. In contrast to saturation, dim light has always low chroma.

Saturation: The attribute of visual sensation that distinguishes a chromatic light from an achromatic light regardless of their brightness [36]. In physical terms, this means the purity of the dominant wavelength in the perceived color spectrum; in a saturated color the energy of the light is concentrated on a narrow frequency band. In contrast to chroma, saturation indicates the colorfulness of light relative to its brightness [21]. This means that even dim light may have high saturation.

Chromaticity: An attribute that combines hue and saturation [2].

Color: An attribute that combines chromaticity and brightness.

Spectral locus: The colors that are on the boundary of the gamut of human vision. The spectral colors are seen as a horse-shoe-shaped curve in the CIE XYZ chromaticity diagram. [21]

Primaries: Color display devices create the color sensation by mixing a small number of primary colors, also called the primaries. CRT monitors use three kinds of phosphors as the primaries, each of which emits only light of a specific wavelength. The choice of the primaries affects the gamut of colors that can be produced.

Tristimulus values: The intensities of primary colors that produce a particular color sensation [2].

REFERENCES

1. Österberg, P., H. Ihalainen, and R. Ritala. 2004. *Method for Analyzing and Classifying Wood Quality through Local 2D-spectrum of Digital Log End Images*, in *Proceedings of the International Conference on Advanced Optical Diagnostics in Fluids, Solids and Combustion (VSJ-SPIE '04)*, Tokyo: The Visualization Society of Japan.
2. Gonzalez, R.C. and R.E. Woods. 2001. *Digital Image Processing*: Addison-Wesley Longman Publishing Co., Inc. 793.
3. Conners, R.W., et al. 1997. *Machine vision technology for the forest products industry*. Computer, **30**(7): p. 43–48.
4. Pham, D.T. and R.J. Alcock. 1998. *Automated grading and defect detection: A review*. Forest Products Journal, **48**(4): p. 34–42.
5. Lycken, A. 2006. *Comparison between automatic and manual quality grading of sawn softwood*. Forest Products Journal, **56**(4): p. 13–18.
6. Todoroki, C.L., R.A. Monserud, and D.L. Parry. 2005. *Predicting internal lumber grade from log surface knots: Actual and simulated results*. Forest Products Journal, **55**(6): p. 38–47.
7. Todoroki, C. 2003. *Accuracy considerations when optimally sawing pruned logs: internal defects and sawing precision*. Nondestructive Testing and Evaluation, **19**: p. 29–41.
8. Funck, J.W., et al. 2003. *Image segmentation algorithms applied to wood defect detection*. Computers and Electronics in Agriculture, **41**(1-3): p. 157-179.
9. Conners, R.W., et al. 1989. *A system for identifying defects in hardwood lumber that uses AI methods*, in *Proceedings of the IEEE Southeastcon '89*, Columbia, SC. p. 1080–1084.
10. Mantegazza, P., E.L. Dozio, and S. Papacharalambous. 2000. *RTAI: Real Time Application Interface*. Linux Journal, **2000**(72).
11. Cheng, H.D., et al. 2001. *Color image segmentation: advances and prospects*. Pattern Recognition, **34**(12): p. 2259–2281.
12. Freixenet, J., et al. 2002. *Yet Another Survey on Image Segmentation: Region and Boundary Information Integration*, in *Proceedings of the 7th European Conference on Computer Vision—Part III*: Springer-Verlag.
13. Limb, J., C. Rubinstein, and J. Thompson. 1977. *Digital Coding of Color Video Signals—A Review*. IEEE Transactions on Communications, **25**(11): p. 1349–1385.

14. Dirks, B., M.H. Schimek, and H. Verkuil. *Video for Linux Two API Specification*. 1999 [cited 2006-03-14]; Available from: http://v4l2spec.bytesex.org/spec/
15. Vila, J., et al. 2005. *SmartSpectra: Applying multispectral imaging to industrial environments*. Real-Time Imaging, **11**(2): p. 85–98.
16. Gauch, J.M. and C.W. Hsia. 1992. *Comparison of three-color image segmentation algorithms in four color spaces*. Proceedings of SPIE, **1818**: p. 1168–1181.
17. Shih, T.Y. 1995. *The reversibility of six geometric color spaces*. Photogrammetric Engineering and Remote Sensing, **61**(10): p. 1223–1232.
18. Munsell, A.H. 1905. *A Color Notation*. Boston, MA: Munsell Color Company.
19. Smith, A.R. 1978. *Color gamut transform pairs*, in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*: ACM Press.
20. Douglas, S. and T. Kirkpatrick. 1996. *Do color models really make a difference?*, in *Proceedings of the SIGCHI conference on Human factors in computing systems: common ground*, Vancouver, British Columbia, Canada: ACM Press.
21. Joblove, G.H. and D. Greenberg. 1978. *Color spaces for computer graphics*, in *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*: ACM Press.
22. Stokes, M., et al. *A Standard Default Color Space for the Internet—sRGB*. [cited 2006-03-17]; Available from: http://www.w3.org/Graphics/Color/sRGB.html
23. Meyer, G.W. and D.P. Greenberg. 1980. *Perceptual color spaces for computer graphics*, in *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, Seattle, Washington, United States: ACM Press.
24. Kasson, J.M. and W. Plouffe. 1992. *An analysis of selected computer interchange color spaces*. ACM Transactions on Graphics, **11**(4): p. 373–405.
25. Comaniciu, D. and P. Meer. 1997. *Robust analysis of feature spaces: color image segmentation*, in *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*: IEEE Computer Society. p. 750–755.

26. Jolion, J.M., P. Meer, and S. Bataouche. 1991. *Robust clustering with applications in computer vision*. IEEE Transactions on Pattern Analysis and Machine Intelligence, **13**(8): p. 791–802.
27. Adams, R.A. 2003. *Calculus—A Complete Course*. Toronto: Addison-Wesley Longman Publishing Co., Inc. 999.
28. Koschan, A. and M. Abidi. 2005. *Detection and classification of edges in color images*. IEEE Signal Processing Magazine, **22**(1): p. 64–73.
29. Tao, H. and T.S. Huang. 1997. *Color image edge detection using cluster analysis*, in *Proceedings of the 1997 International Conference on Image Processing (ICIP '97)*. p. 834–836.
30. Trahanias, P.E. and A.N. Venetsanopoulos. 1993. *Color edge detection using vector order statistics*. IEEE Transactions on Image Processing, **2**(2): p. 259–264.
31. Healey, G. 1992. *Segmenting images using normalized color*. Systems, Man and Cybernetics, IEEE Transactions on, **22**(1): p. 64–73.
32. Shafer, S.A. 1984. *Using Color to Separate Reflection Components*: University of Rochester, Computer Science Department.
33. Healey, G. 1992. *Using color for geometry-insensitive segmentation*. Journal of Optical Society of America A, **6**(6): p. 920–937.
34. Klinker, G.J., S. Shafer, and T. Kanade. 1988. *Image Segmentation and Reflection Analysis Through Color*. Proceedings of SPIE, **937**.
35. Maxwell, B.A. and S.A. Shafer. 2000. *Segmentation and Interpretation of Multicolored Objects with Highlights*. Computer Vision and Image Understanding, **77**(1): p. 1–24.
36. Wyszecki, G. and W.S. Stiles. 1982. *Color Science: Concepts and Methods, Quantitative Data and Formulae*. New York: Wiley. 950.
37. Gejguš, P. and M. Šperka. 2003. *Face tracking in color video sequences*, in *Proceedings of the 19th spring conference on Computer graphics*, Budmerice, Slovakia: ACM Press.
38. Ming-Hsuan, Y. and N. Ahuja. 1998. *Detecting human faces in color images*, in *Proceedings of the 1998 International Conference on Image Processing (ICIP '98)*, Chicago. p. 127–130.
39. Chan, F.H.Y., et al. 1996. *Object boundary location by region and contour deformation*. IEE Proceedings—Vision, Image & Signal Processing, **143**(6): p. 353–360.
40. Rahimi, A. *Fast Connected Components on Images*. [cited 2006-05-20]; Available from: http://web.media.mit.edu/~rahimi/connected/

41. Brunelli, R. and T. Poggiot. 1997. *Template matching: matched spatial filters and beyond*. Pattern Recognition, **30**(5): p. 751–768.
42. Haralick, R.M. 1979. *Statistical and structural approaches to texture*. Proceedings of the IEEE, **67**(5): p. 786–804.
43. Ruck, D.W., et al. 1990. *The multilayer perceptron as an approximation to a Bayes optimal discriminant function*. IEEE Transactions on Neural Networks, **1**(4): p. 296–298.
44. Gremban, K.D., C.E. Thorpe, and T. Kanade. 1988. *Geometric camera calibration using systems of linear equations*, in *Proceedings of IEEE Conference on Robotics and Automation (ICRA '88)*. p. 562–567.
45. Wei, G.Q. and S.D. Ma. 1991. *Two plane camera calibration: a unified model*, in *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '91)*. p. 133–138.
46. Potmesil, M. and I. Chakravarty. 1982. *Synthetic Image Generation with a Lens and Aperture Camera Model*. ACM Transactions on Graphics, **1**(2): p. 85–108.
47. Cormen, T.H., C.E. Leiserson, and R.L. Rivest. 1990. *Introduction to Algorithms*. Cambridge: The MIT Press.