Erik Veijola

COATING PROCESS MONITORING USING COMPUTER VISION

Case Metso Paper Inc.

COATING PROCESS MONITORING USING COMPUTER VISION

Case Metso Paper Inc.

Erik Veijola
Bachelor’s thesis
Spring 2013
Degree Programme in Information Technology
Oulu University of Applied Sciences

PREFACE

This Bachelor’s Thesis was done at Oulu University of Applied Sciences, School of Engineering, Raahe Campus during the Spring of 2013. The work was done for Metso Paper Inc.

I would like to thank my thesis instructor Mr Juha Huhtala for the guidance and helpful tips he has provided, Mr Juha Räty and Mr Janne Räihä for the help during the work, Mr Seppo Parviainen of Metso Paper Inc. for the opportunity to do this work, and the staff of Oulu University of Applied Sciences, School of Engineering, Raahe Campus, for the wonderful years I spent in Raahe.

Last, but definitely not least, a special thank you to my family, and friends for the support during the long process.

Raahe, April 2013
Erik Veijola

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Tekijä: Erik Veijola
Opinnäytetyön nimi: Coating Process Monitoring Using Computer Vision
Työn ohjaaja: Juha Huhtala
Työn valmistumislukukausi ja -vuosi: Kevät 2013 Sivumäärä: 40 + 3

Tämän pinnäytetyön tavoitteena oli tehdä Metso Paper Inc. -yhtiölle prototyyppiohjelma paperirullan päällystämisen valvomista varten. Jos päällystys tehdään huonosti ja siinä on virheitä, päällystysprosessi pitää uusia. Tämä taas laskee yrityksen tuottoja, koska prosessi on aika kallis. Työn aihe tuli Seppo Parviaiselta joulukuussa 2012.

Valmiin järjestelmän tarkoitus oli hälyttää prosessia valvovaa henkilökuntaa virheistä päällystysjärjestelmässä joka valuttaa hartsin paperirullan päälle. Virheitä ovat esimerkiksi venttiilin tukkeutuminen tai liika ilma järjestelmässä. Virhe voi johtua myös jostain muusta syystä.

Tämän projektin tekemisessä käytettiin ilmaisia avoimen lähdekoodin konenäkökirjastoja ja muuta materiaalia. Järjestelmä tehtiin toimimaan Windows-ympäristössä ja sen tekemisessä käytettiin OpenCV-kirjastoja ja C++-ohjelmointikieltä. Järjestelmä myös testattiin Windows-ympäristössä.

Lopullinen järjestelmä toimii kuin odotettiin, ja asiakas on tyytyväinen lopputulokseen. Tutkimus osoittaa, että tämän tyyppistä prosessia on mahdollista valvoa konenäköä soveltaen. Järjestelmä ei suorita objektin tunnistusta aikarajoitusten takia. Järjestelmää voidaan kehittää edelleen esimerkiksi objektin tunnistuksen lisäyksellä, käyttämällä parempia konenäkö algoritmeja, kameran valinnalla yms. Tässä työssä ei ollut tarkoitus kehittää uusia algoritmeja, tai tutkia mikä olisi paras kamera tämän kaltaisessa järjestelmässä.

Asiasanat: kuvankäsittely, kuva-analyysi, OpenCV, C++, konenäkö

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Erik Veijola
Title of Bachelor's thesis: Coating Process Monitoring Using Computer Vision
Supervisor: Juha Huhtala
Term and year of completion: Spring 2013 Number of pages: 40 + 3

The aim of this Bachelor's Thesis was to make a prototype system for Metso Paper Inc. for monitoring a paper roll coating process. If the coating is done badly and there are faults one has to redo the process which lowers the profits of the company since the process is costly. The work was proposed by Seppo Parviainen in December of 2012.

The resulting system was to alarm the personnel of faults in the process. Specifically if the system that is applying the synthetic resin on to the roll gets blocked, there is too much air in the system, or there is some other reason why the nozzles are blocked.

In the making of the project I used freely available open source computer vision libraries and materials. The system was made for Windows systems and was implemented with OpenCV libraries and C++ programming language. The system was also tested on a Windows system.

The final system works as planned and the customer is satisfied with the results. The research shows that it is possible to make a system for monitoring this type of process. The system was not implemented with object detection because of time constraints. The system can be developed further by adding object detection, better image processing and image analysis algorithms, cameras etc. My work was not to create these kinds of algorithms, or research what is the best camera for the purpose.

Keywords: Image Processing, Image Analysis, OpenCV, C++, Machine Vision, Computer Vision

TABLE OF CONTENTS

ABBREVIATIONS 8

1 INTRODUCTION 9

1.1 Ability to See 9

1.2 OpenCV 10

2 COMPUTER VISION THEORY 11

2.1 Seeing with Computers 11

2.2 Mathematics of Computer Vision 11

3 OPENCV BASICS 13

3.1 Image Containers and Colour Models 13

3.1.1 Drawing and Writing on Images 16

3.2 Image Processing 17

3.2.1 Smoothing 17

3.2.2 Image comparison 23

3.2.3 Colour detection 24

3.3 Image Analysis 24

3.3.1Contour and line detection 24

2.3.2 Feature detection 25

3.3.3 Object detection 27

3.3.4 Histograms 28

3.4 Image Understanding 28

3.5 GPU acceleration 29

3.6 OpenNI 29

4 CASE METSO PAPER INC. 30

4.1 Research 32

4.1.1 First prototypes 32

4.1.2 Colours and Contours 33

4.1.3 Objects and features 34

4.2 Final Solution 34

4.3 Problems 35

5 TESTING 36

6 FURTHER DEVELOPMENT 38

7 CONCLUSION 39
LIST OF REFERENCES 40
APPENDICES 41
Appendix 1: Sample application with use of webcam as input 41
Appendix 2: Sample application with an image as input 42

ABBREVIATIONS

AI	Artificial Intelligence
BGR	Colour Model (Blue Green Red)
C/C++	Programming Language
CUDA	Compute Unified Device Architecture
CvMat	Default image container in OpenCV 2 and onward
FAST	Features from Accelerated Segment Test
GPU	Graphics Processing Unit
GRAY	Colour Model
HOG	Histogram of Oriented Gradients
HSL (HLS)	Colour Model (Hue, Saturation, Lightness)
HSV	Colour Model (Hue, Saturation, Value)
Hue	“The degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow.”
IplImage	Default image container in older versions of OpenCV with C
OpenCV	Open Source Computer Vision Library
OpenNI	Open Natural Interaction
RGB	Colour Model (Red Green Blue)
SIFT	Scale-Invariant Feature Transform
SURF	Speeded Up Robust Features

1 INTRODUCTION

The topic of this Bachelor's Thesis came from Seppo Parviainen of Metso Paper Inc., a Finnish technology company. The aim of the thesis was to research image processing for "real time" process monitoring and to make a sample application to monitor a paper roll coating process. The work was based on OpenCV, Open Source Computer Vision Library, which is a cross-platform open source computer vision library originally developed by Intel Corporation. I chose C++ as the programming language due to the fact that I'm quite familiar with it already and it is easy to find a lot of examples and tutorials for it online.

This thesis is intended for students, professionals, and people interested in computer vision and programming. A basic understanding of some programming language and University level mathematics are needed before reading this thesis.

1.1 Ability to See

For humans, the ability to see and to understand what you see is so normal that we never think about it. When making a computer vision application, or doing research on the subject, the process which happens in the brain is mimicked as to our best understanding of how thought is formed. We have been, just in recent years, able to see how a thought happens in the brain of a fish, but it is a far cry off the human brain in size and complexity. When a human sees with their eyes they see colours and depth, since we have two eyes. Based on the colours and depth changes we can tell that we see some objects and people, the human brain can distinguish a person by just seeing a part of that person. For a computer to recognize a face millions of calculations have to be made just to see that there is a face in a picture, to recognize the person millions more have to be made. The computers do not really understand. They do not have the capability, or the processing power to truly have a conscious idea of what an

image holds within. There have been important achievements in AI to give some thought to computers. One of the most important and well known ones is Watson, developed by IBM to compete in the quiz show Jeopardy! Still Watson does not have the capability to understand, it can only answer to questions.

In some situations the computer can be faster or more precise at detecting certain characteristics, for example the computers are immune to optical illusions. A good example of it is that you are shown an image, which you know is a still image and not a video, but you still perceive a movement within the image. Another example of it is that you are shown an image which can be perceived in two ways, or you are shown colours which look different, but are actually the same, or you find something in an image which is not there.

1.2 OpenCV

OpenCV is an open source library which can be used for commercial and academic purposes to develop computer vision applications. It is cross-platform and supports multiple programming languages. The library contains hundreds of algorithms for different purposes, such as filtering and object detection algorithms. (OpenCV Reference Manual 2012, 1.)

OpenCV was originally developed by Intel Corporation and more recently by Willow Garage and Itseez. The library was not developed with the intent of making new algorithms, but to optimize currently known methods for real time computer vision and make them available to everyone.

2 COMPUTER VISION THEORY

In this chapter I will introduce some basic theories behind computer vision. Some of the theory of computer vision can also be applied to digital signal processing. Parts of this chapter will be repeated in the following chapters and some new theory will be introduced too.

2.1 Seeing with Computers

What is computer vision? A short answer, it is a field of computer science that specializes in making computers understand the content of an image. Computers can only understand zeros and ones, everything else is a combination of them. For example number 7 in a 4 bit system is 0111, and number 8 is 1000, nowadays we use 32 or 64 bit systems, but the idea is the same. So, when even the simplest of things such as numbers and easy sum calculations are complicated, you cannot even begin to realize how complicated the computer vision algorithms and calculations are that allow the computer to make decisions. Of course they are made easy to use with current operating systems and development tools, but digging deeper makes everything more complex and abstract. For humans, seeing with our eyes is easy, we do it all the time without thinking or even paying attention to it. We do not completely understand how the brain does it, but the topic is being researched jointly in the fields of neuroscience and computer science.

2.2 Mathematics of Computer Vision

The hardest part in computer vision is probably to understand the mathematics behind the algorithms and methods. To understand the mathematics you should have knowledge of university level mathematics.

An example of a linear filters equation, the value of g(i,j) is calculeted by the weighted sum of f(i+k,j+l) as shown in Figure 2.1. (Szeliski 2010, 111)

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

FIGURE 2.1 Linear filter example

The equation of a normalized box filter; in which the value of the pixel is found by calculating the mean of the surrounding pixels as shown in Figure 2.2. (OpenCV Reference Manual 2012, 228)

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

FIGURE 2.2 Normalized box filter

The feature detectors are even more complex when compared to simple filters. For feature detection we need to identify changes in colour, or depth, in the image. Based on the results, we can identify for example characteristics, such as edges, blobs, lines, or corners which can be used for object detection.

3 OPENCV BASICS

In this chapter I will introduce you to some basics of images and image processing algorithms with OpenCV. Image processing is a type of signal processing where an image is taken as an input and the output is a different image. All the algorithms can be put in one of the three different categories: image processing, analysis, and understanding. In all sorts of algorithms an image is the input, but the output differs in them. In image processing the output is another image, in analysis the output is simple measurements, and in understanding the output is high level measurements. Most computer vision applications made for the industry rely on the last one. This is because of the robustness and low error rates.

3.1 Image Containers and Colour Models

A computer sees an image as a grid of values, each one of these values represents a pixel. In OpenCV the basic image container is Cv::Mat. It can have a number of dimensions and the pixel values can be represented in many different ways. The number of dimensions and pixel representations depend on the colour model in which the image is presented. Different colour models are RGB, BGR, GRAY, HSV etc as shown on Figures 3.1, 3.2, 3.3, and 3.4. You can even create your own colour model. HSL and HSV are not represented on a Cartesian coordinate system, RGB is, but rather in a cylinder like coordinate system. Before the CvMat the basic image container was IplImage. This was not so different from CvMat, but the differences were quite important ones, for example when using IplImage you needed to do manual memory management. This should not be a problem in smaller programs, but in large applications the problem starts to grow. This problem was due to the nature of the C language.
(OpenCV Reference Manual 2012, 2 – 22.)
When doing some image processing, analysis, or understanding, you might not want to use the whole image as an input, for this kind of situations you can use

a region of interest to define the area that you want to process or extract information from.

Hue is a quite important subject, but also quite difficult to explain. “The degree to which a stimulus can be described as similar to or different from stimuli that are described as red, green, blue, and yellow” (Fairchild 2004, 44).

The following images are examples of the same image in different colour models. The first image, Figure 3.1, came with the OpenCV libraries. The rest of the images, Figures 3.2, 3.3, and 3.4 were made by my program from the first image.

FIGURE 3.1 RGB

FIGURE 3.2 Grayscale

FIGURE 3.3 HSL

FIGURE 3.4 HSV

3.1.1 Drawing and Writing on Images

There are ready functions to draw different kinds of shapes and write text onto the image containers. These kinds of operations may seem easy, but in fact they are not. To draw something onto the image you have to change the pixel values of the image only in the positions where you want to add something to.

3.2 Image Processing

Image Processing is used to modify the content of the image or create new images based on input. Filters are the most common algorithms. They are used to extract information, or modify the content of an image. Some of the algorithms are used to change the colour model of the images. The filtering operations can be linear or non-linear. Filtering operations originate from signal processing, but in image processing instead of a signal the input is an image. I will introduce some of the most commonly used filters. These kinds of operations can be done for many reasons, for example image restoration where a damaged image can almost be restored to the original state, or as preprocessing before the feature or object detection. (OpenCV Reference Manual 2012, 1.)

3.2.1 Smoothing

Smoothing is one of the most basic image processing algorithms. It can be used for removing noise and small mistakes from the image. Different kinds of smoothing algorithms have different effects on the image, but the basic idea behind them all is the same. You have a filter which is applied to the image in all positions. You can write your own algorithms for doing only parts of the image, or you can extract the part which you want to work with ROI. (Fisher, Date of Retrieval 29.4.2013.)

Regular smoothing is done by running a filter over the image. The function blur(InputArray src, OutputArray dst, Size ksize, Point anchor, int borderType) is a smoothing operation in which the output, dst, is created by applying a filter, the size of ksize, over src. The anchor point is the pixel, which is being evaluated. If the value of this is a negative value, the center of the kernel is the anchor point. BorderType defines the mode that extrapolates the surrounding pixels. This type of filtering is the simplest one, the output values are the mean of the neighboring values. The result of applying the blur method on the reference image can be seen on Figure 3.5. (OpenCV Reference Manual 2012, 228.)

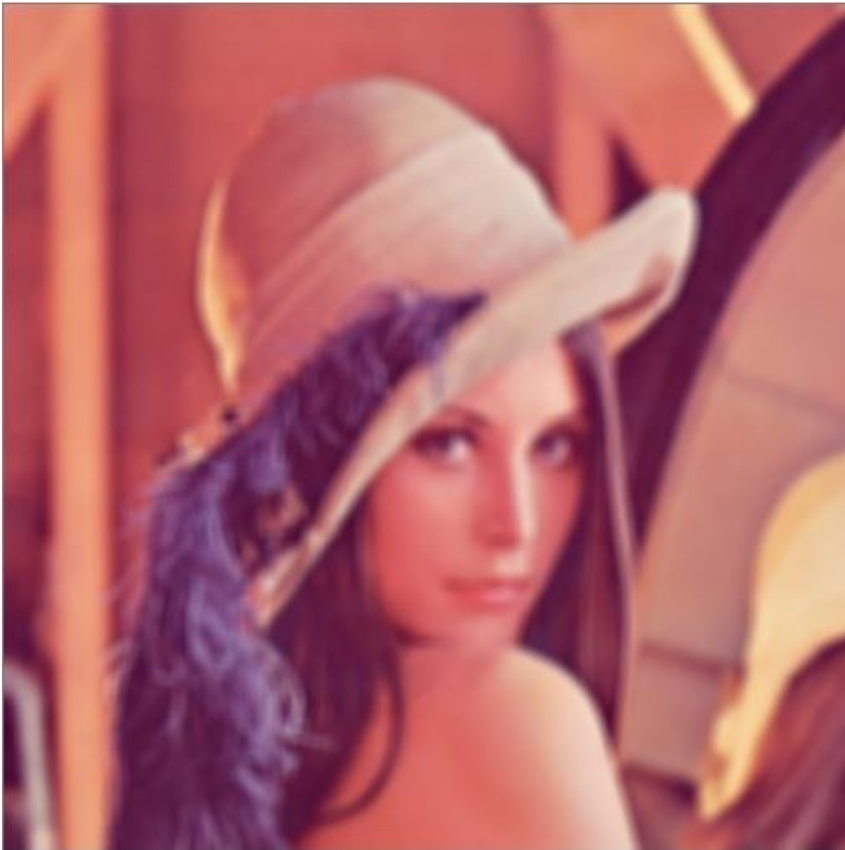

FIGURE 3.5 Blur with kernel size 9, 9

Median Blur, like the name tells, calculates the median of the neighbouring pixels. In some cases this gives better results than mean filtering although it is slower due to the sorting of the values of the surrounding pixels. The function for this operation is medianBlur(InputArray src, OutputArray dst, int ksize). The ksize must be odd and greater than 1. The result of applying median blur on the reference image can be seen on Figure 3.6. (OpenCV Reference Manual 2012, 240-241.)

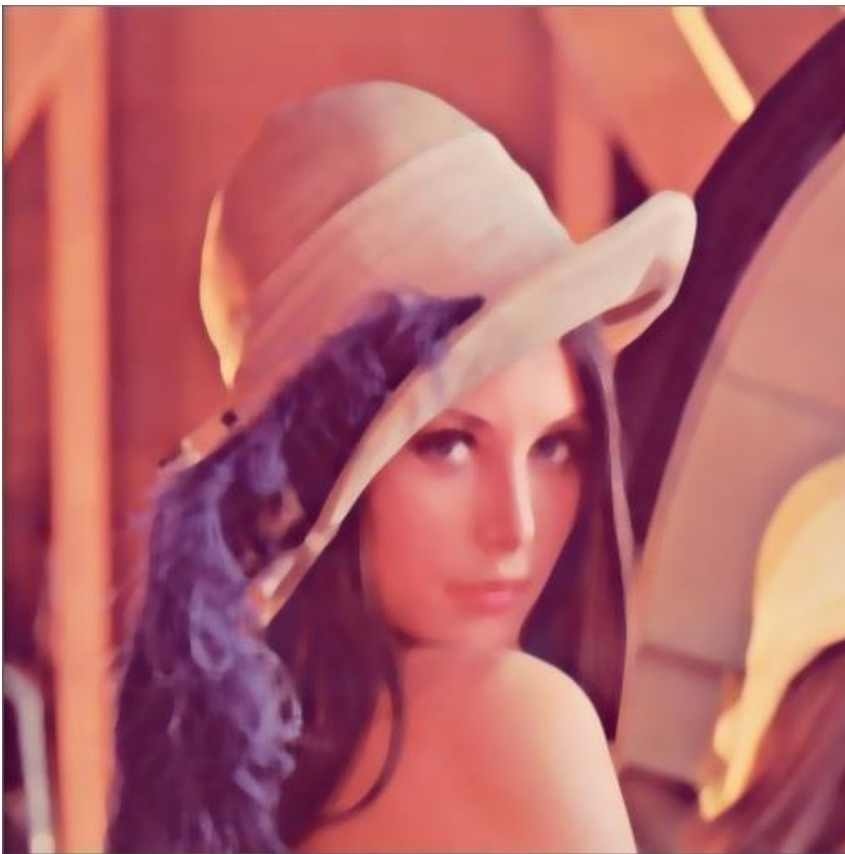

FIGURE 3.6 Median blur with kernel size 9

Gaussian blur is a sophisticated smoothing algorithm that reduces noise, but also reduces details such as edges. A good example of a Gaussian blur is to take a scanned image which has not been printed with a high pixel density and apply the filter to get a more lifelike image. The filter makes the image look like you were watching it without eyeglasses, if you need them. The result of applying Gaussian blur on the reference image can be seen on Figure 3.7. (OpenCV Reference Manual 2012, 237-238.)

FIGURE 3.7 Gaussian blur with kernel size 9, 9

Bilateral filtering is used to remove noise from an image without affecting the edges. It is, however, slower than some other filters, but due to the good functionalities it is often used. The filter works just like any other, but it takes the surrounding pixels into account in a smarter way. The result of applying bilateral filter on the reference image can be seen on Figure 3.8. (Fisher, Date of Retrieval 29.4.2013.)

FIGURE 3.8 Bilateral filtering with parameters 27, 54, 13.5

Dilation and erosion are similar functions, the difference is that when dilating we calculate the maximum values and in erosion, the minimum values. This means that when you dilate an image the brighter areas will grow and darker areas will shrink and the opposite thing happens in erosion. When using either algorithm, you have to create an element which determines the area which is being affected. The result of applying dilate and erode methods on the reference image can be seen on Figures 3.9 and 3.10. (OpenCV Reference Manual 2012, 235-236.)

FIGURE 3.9 Dilate with MORPH_RECT, Size (5, 5)

FIGURE 3.10 Erode with MORPH_RECT, Size (5, 5)

3.2.2 Image comparison

Comparing images takes 2 images as an input and outputs the difference of them based on what kind of comparison you want to do. This kind of image analysis, even though it is image processing not analysis, is not so good compared to object and feature detection. Based on my tests and experience, it does not work very nicely with webcam images due to the high distortion of the output images. The method for comparison is compare(InputArray src1, InputArray src2, OutputArray dst, int cmpop). The integer value cmpop is used to determine what kind of checking operation you want to do. For example CMP_NE defines that the pixel values, which are being checked, are not equal and CMP_EQ defines that they are equal. (OpenCV Reference Manual 2012, 727-728.)

3.2.3 Colour detection

In colour detection, an image and two scalar values are given as input to get an output image, which is a binary image, where each pixel is either black or white. The colour of the pixel is white if the input image's pixel is within the two scalar values. The method for detecting colour is inRange(InputArray src, Scalar low, Scalar high, OutputArray dst). The values of the two scalars can be between 0 and 255 for each colour and for alpha. It does not matter if the first values are greater or less than the second or vice versa. (OpenCV Reference Manual 2012, 137-138.)

3.3 Image Analysis

Image analysis is done to extract low level information from an input image such as lines, contours, or corners. For the computer to be able to do any kind of intelligent decisions, these results have to be accurate. This is one of the biggest challenges in computer vision. The results of any kind of detector on a computer are much less accurate and even much slower than that of a human brain. To achieve better results, you need to understand how the brain works and these days we can just guess.

3.3.1Contour and line detection

Contour and line detection, in OpenCV can only be done on a binary image where the pixel values are either 0 or 1. To achieve this you need to use some sort of edge detection algorithm first. It is of course also possible to use colour detecting algorithms to create an image. This might become a problem if you detect lines from the image. Because you might have large unified areas, you could find the edges after colour detection and you could use that as an input to get rid of that problem. (OpenCV Reference Manual 2012, 287-290 & 310-314.)

The results of both detectors can be used for object detection and shape analysis.

2.3.2 Feature detection

Feature detection is done to find certain kinds of characteristics in the images, such as edges, corners, and blobs. Most commonly used feature detectors are edge detectors, as they are all basis for object detection. Canny and Sobel are quite good for detecting edges. Technically, Sobel is not an edge detection algorithm since it can calculate the intensity of the edge rather than just find edges like the Canny algorithm does. Edge detection relies on colour or depth changes. Object detection could be implemented solely on detecting colours, but that would be problematic in some cases. Sobel and Canny are both used for edge detection. Sobel is better than Canny for detecting and displaying only strong edges. You can use both in the same program for good accuracy. (Szeliski 2010, 206-214.)

Other feature detectors include FAST, SIFT, and SURF. These algorithms do not detect edges, but they can be used to detect corners and blobs. The SIFT generates vectors from the input image and then compares the results to a set of samples. SURF is faster than SIFT and also claims to be more robust against different image transformations. FAST is used to detect corners.

The following images, Figures 3.11 and 3.12, are the results of Sobel and Canny methods applied to the reference image. Before the applying the methods, the images were converted from RGB to GRAY colour model.

FIGURE 3.11 Sobel

FIGURE 3.12 Canny

3.3.3 Object detection

Object detection can be done with many off the shelf algorithms, or you can create your own algorithms. Object detection relies on edge, blob, or corner detection, so in a way object detection is trying to understand the results of feature detection. OpenCV has ready functions for Haar Cascade Classifier, which is good for object detection. The function implements Viola-Jones object detection framework, which was developed mainly for face detection. To use this sort of detector, you have to first teach it what to detect. For this, you need to input large quantities of positive and negative samples. Another object detector Latent SVM, based on Dalal-Triggs detector, is used for detecting objects. The detection is done by a single filter on HOG. It goes through the image and is applied to all positions on it. (OpenCV Reference Manual 2012, 315-316.)

3.3.4 Histograms

Histograms are a visual way to represent data in statistics, and in computer vision it is used to represent the amount of pixels of a certain intensity such as the different intensities of colour red. Figure 3.13 shows the histogram of the reference image. (OpenCV Reference Manual 2012, 275-277.)

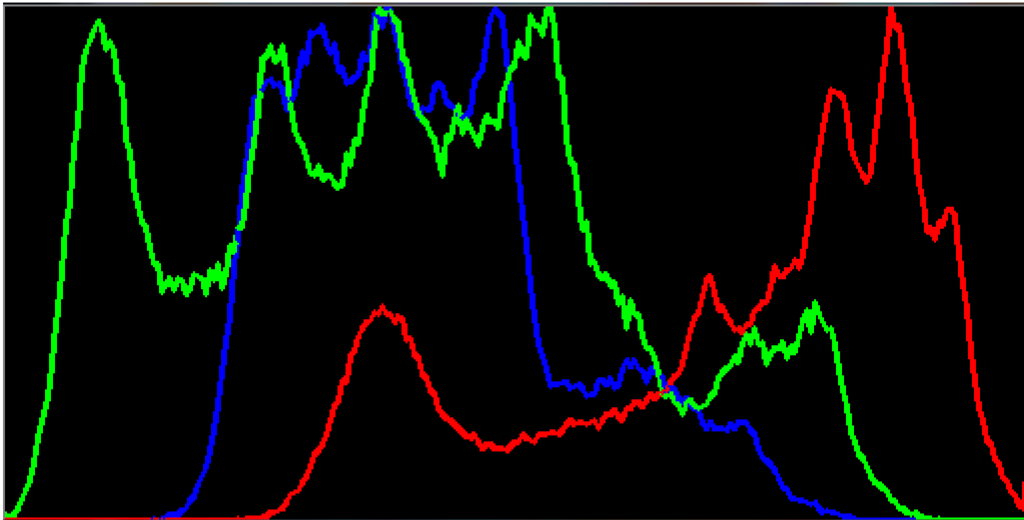

FIGURE 3.13 Histogram of Lenna

3.4 Image Understanding

Image Understanding has been developed for the past 30 or so years and is still a growing field in computer science, mainly because now computers are faster and we have a better understanding about how the brain works and how ideas are formed. If a human sees a picture of something familiar, they know what is in the image even though they have never seen that specific image before. For example, you get a post card from a friend and a picture of a copied Eiffel tower is displayed on it, you know that it is the Eiffel tower just not the real one. But a computer might need billions of calculations to understand that it is in fact a copy opposed to millions that it takes to notice that the image has the Eiffel tower in it. For a computer to understand these kinds of concepts, which seem so simple and every day for a human, is the result of years of research and

development by teams that not only consist of computer scientists, but also include neuroscientists and neurobiologists among others.

3.5 GPU acceleration

The GPU of the computer can be harnessed to do processing in the OpenCV GPU module, which is implemented with NVidia CUDA Runtime API. Unfortunately, it only supports NVidia GPUs. The GPU is much faster than the conventional CPU in highly parallel multitasking and it can be used to speed up some high-level algorithms. This is due to the high number of slower processing cores in a GPU versus a lower number of faster processing cores of the CPU. For example, Image Understanding can be made dozens of times faster when the processing is done by a GPU rather than a CPU. When using the CUDA libraries with the OpenCV, you need not have prior knowledge and skills with CUDA. It is recommended that you do have some experience to achieve a higher performance on the system. You can implement the CUDA support into your application and still use it in PCs without an NVidia GPU by modifying very little of code. It is also possible to use multiple GPUs to achieve an even higher performance. (OpenCV Reference Manual 2012, 489-490.)

3.6 OpenNI

OpenNI is a software development kit for creating user interfaces where the controller is the person’s body or voice instead of a keyboard, mouse, or a gamepad. A XBOX 360 Kinect controller is such a device and can be used instead of a conventional webcam to get both image and distance views and also sound. These kinds of devices allow the system to receive multiple kinds of input for a greater accuracy and extended functionalities.

4 CASE METSO PAPER INC.

In this chapter I will explain what is the problem addressed by my system and the results of my work. The paper roll coating process is rather simple. You have a steel roll which can be used multiple times. You coat the roll with synthetic resin and fiber glass. Once the roll has been used for long enough, it is sent back to the factory to be coated again.

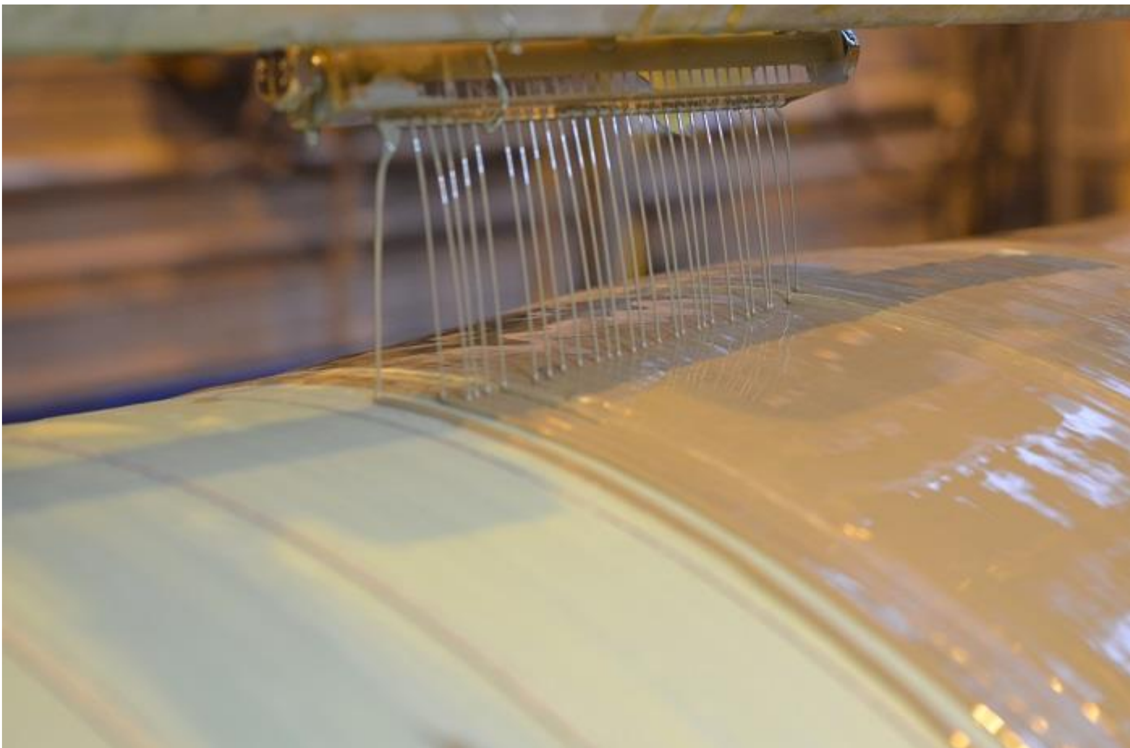

FIGURE 4.1 Nozzles

FIGURE 4.2 Image of the roll with a mistake

During the process one of the vents could get blocked, or there is some other problem, as shown on Figure 4.1. This might cause an error in the manufacturing process and make the roll unusable. Or if the roll is used in producing paper, the final product can be faulty. Another danger is that if the roll breaks during the making of the paper, the whole factory can be at risk. This is why it is important to monitor the process, so that no mistakes are made.

The point is to detect the streams of the synthetic resin and check if there is enough of them. If the amount of resin, which is put on top of the fiber glass strengthener, is too low, the fiber glass can be dry on some spots as seen in Figure 4.2. This will cause problems and the coating process has to be done again. The process costs hundreds of thousands of euros, which means that it is not a good thing to redo it many times.

4.1 Research

The first thing was to find out how to make a computer vision application. This was made easy by my supervisor who suggested a few different libraries and websites with information. For my project I chose OpenCV. The reason was the C++ interface and ready functions that can be used for fetching the images and processing them and the fact that you can use any OpenNI device as a camera. This would be one of the things that could be developed afterwards along with AI. Once I had the libraries installed, I started to make prototype programs that detected colours, shapes, edges, lines, contours and objects. To make the program as simple and error proof as possible, I decided not to do any object detection because it would have required some sort of artificial intelligence. Implementing the AI would have taken too long, so I started to try methods to extract contours and lines from the images. In the final version of the program only edges are detected and the lines are detected from those edges.

At first, getting the libraries to work was a little problematic because the instructions to install the libraries were outdated. Thankfully, other people had been able to make theirs work and wrote instructions online. It is important to remember to do this step correctly, otherwise you cannot use any of the OpenCV algorithms.

4.1.1 First prototypes

The first programs I created were to test the functionalities of OpenCV. The first idea was to take a picture and compare it to the next one. The results I got were quite odd, I had expected them to be quite white since there was no visible difference in the two images. The reason for this is that the cameras sense extremely small differences that the human brain filters away, such as colour changes due to the lights. It would have been possible to make something useful out of this, but I decided to move on.

The next was edge detection with Canny. I found nice examples made by Noah Kuntz in 2009 and decided to try those things out myself. I noticed that in those examples he used IplImage rather than Mat as the container for the image. I wondered why there would be two different kinds of containers for the images and found out that IplImage was the old way to store images. It is worse than Mat, mostly because of its memory management due to the C programming language. Anyway, the basic ideas at his site were useful to me and I started to test the Canny algorithm. It was rather interesting to see the results for the first time. The algorithm worked nicely, but how would I get some kind of output that I could use to determine if the process is working fine. Again I tried the image comparison. The results were still not what I thought to be useful, but the Canny algorithm seemed like a good thing to keep for further prototypes.

4.1.2 Colours and Contours

Again, I found myself thinking what would be a simple, efficient and accurate way to determine the number of streams in the process and then I thought about colours. The synthetic resin is quite yellowish and not so transparent. Maybe if there were something behind the streams, I could detect the colour of the resin and then count the number of the coloured areas. The finding of the colour areas proved to be quite simple, but how to count them proved to be quite difficult.

Contours were the next thing I started experimenting on. It was not too difficult to extract them from an image after a colour model change, blur, and Canny. It was easy to count them and draw the found ones on an image. But the problem was that I found too many contours. With the eyes you can see that there should only be two of them, but the program found 3 or 4 of them. I then pointed my attention to Hough lines and Hough lines probabilistic, which are algorithms for extracting lines rather than contours from the images. The output of the

canny operation had to be straight so you could find a line. The results of the line transformations were pretty similar to the contours.

4.1.3 Objects and features

After some trial and error I started to look into AI, object detection and feature detection. It looked like a lot of work to make them work properly, but I gave it a try anyway. There were many ways to implement the wanted application with feature or object detection, but I only researched HAAR, SIFT, and SURF. The problem might have been that they cannot detect all objects, and I needed to detect many streams in one image. Of course, you could have detected the whole area in which the resin “streams” are, but that might have caused some problems in the detection of dripping and if the “stream” were not straight. The difference between the algorithms was quite mathematic and hard to understand. The application would have been possible to make with Haar, but I would have needed to make a ton of positive and negative images for the training and this would have taken too long. It would have been quite nice to try this, but time is of the essence.

I stumbled onto video analysis when researching image processing. This would also be nice to implement, but would also take too long time.

4.2 Final Solution

All the solutions I had made so far needed some sort of AI to give out proper results, or I would have had to use too much time to make them, so I checked the streams one by one. This was kind of a breakthrough idea because for the first time I was able to tell, with almost 100 % certainty, if one of the streams was missing from the image. This, of course, was not the ideal solution for the problem, but it worked well and it did not give too many false errors. During the testing, the alarm was not on when all the streams were visible and when there

were some disturbance to the streams, the program turned on the alarm. During the testing it was also suggested to me that one should be able to change the alarm settings, such as what is the size of the buffer and what is the percentage at which the alarm should go on.

So the finalized application has two loops, one for the application and one for the image analysis. This is because in the application loop you can adjust the settings before starting the process. After the process starts you will have to press ‘a’ on the keyboard to start the image analysis loop. Inside both loops, an image is taken from the webcam feed. In the application loop, this is done to ensure that the settings, such as camera and ROI positions and the number of streams from the nozzles, are correct. In the image analysis loop the structure is more complicated and the details of it will not be included in this paper.

4.3 Problems

During the development, I noticed that the camera is creating some optical distortion to the images when it is used too close to the object which is targeted. This caused some headache during the testing because you cannot get too close to the target for greater accuracy. This was not a really big problem because the camera supported Full HD stills and video. The size of the image might become a problem if the computer is changed to a less powerful one.

For the testing purpose a camera holding system was created, which made it easier to hold the camera at the right distance.

Changes in the environment also caused problems. This could be partially got rid of by using blinds and light sources to keep the environment stable and insensitive to bright lights and camera movements.

5 TESTING

During the whole process I tested all the different kinds of applications I had made. For colour detection, I just placed the webcam to face a room and adjusted the low and high values to find certain objects, the colour of which was within the limits. For other kinds of testing I used images I found online or took myself. In most cases, where a still image was needed, I used the “industry standard” image, Lenna, which was taken from the November 1972 issue of Playboy, displaying Lena Söderberg. When testing any application that uses OpenCV algorithms or resources with a compiler, you have to remember to check that you have installed the OpenCV libraries and have linked them correctly.

When I finished the final system I tested it with water, because we did not have the necessary equipment to test it with the synthetic resin. Even when using water in a quite low light environment, the application worked fine. For testing with the synthetic resin, a test bench was assembled at Tuotantostudio, a research laboratory for mechanical, metal, and energy technologies. The test bench was designed to be as simple as possible, only featuring a stand for the camera, a back plate, and a source for resin. The results of this kind of testing were similar to those done with water.

For the testing I made a modified program of the final version, so you can see the amount of images and also the amount of errors that were found during the run. While testing, it was brought to my attention that we should be able to change the alarm settings. The alarm should not be turned on by just one error, which it does not do in my program, and you should also be able to change the amount of images from which the percentage is calculated from. This is not such a big thing so I will add it to the finalized program.

FIGURE 5.1 Testing system

As you can see from the image of testing system, Figure 5.1, the synthetic resin was quite green. The actual resin that would be applied to the paper roll would be more yellowish.

6 FURTHER DEVELOPMENT

As stated earlier, further development is possible and suggested. Things like AI, object detection, feature detection, and OpenNI would be first on my list due to the nature of the process which is being monitored. It would also be possible to make the application work automatically with no need for the user to change the settings by first finding the area with object or feature detection and adjusting the ROI according to it. With more resources, it would be possible to make my own algorithms which might be more efficient. Also it would be possible to do some preprocessing on a separate system, such as a small real time embedded system. Choosing better camera equipment would also make the system more robust. Currently the camera cannot be put too close to the system due to optical distortion. This could be fixed by replacing the lens of the camera.

Video analysis would also be a good thing to implement, but with the current limitations in open source computer vision libraries, it is not feasible.

Of course, comparing different image processing and computer vision libraries together and choosing one of them based on the requirements of the application, is one thing that could be done to make the system more robust or faster.

7 CONCLUSION

Prior to this project, I had no experience with computer vision. During this project I gained valuable information and skills of this very interesting subject.

With the time and resource limitations that I worked with, I was able to research and create a prototype application to analyze webcam images and alarm the user if there is something wrong with the process. With more time and resources, it would have been possible to make the application better in terms of usability and correctness of the results. The current prototype requires the user to manually position the region of interest and the sub regions to work correctly. Those limitations would be possible to get rid of by detecting the area, which is being monitored, automatically.

I think that with the time available this was the best I could do and I feel like I have fulfilled the requirements set by the client. This was a great opportunity to learn and I am sincerely grateful for the opportunity.

The system worked with almost 100 % accuracy. This is due to the simplicity of the entire system and the fact that the alarm will turn on only with continuous errors. For academic and testing purposes a new system could be developed with all, or some, of the further development ideas. The system is not completely insensitive to changes in the light, position of the camera, and other changes in the environment.

I hope that this paper was of use for you as a reader interested in computer vision or image processing.

LIST OF REFERENCES

1. Itseez. 2012. The OpenCV Reference Manual 2.4.3. Date of retrieval 28.12.2013, http://docs.opencv.org/opencv2refman.pdf.

2. Szeliski, R. 2010. Computer Vision: Algorithms and Applications. Springer. Date of Retrieval 13.3.2013, http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf.

3. Fisher R. Bilateral Filtering for Gray and Color Images. The University of Edinburgh, School of Informatics. Date of retrieval 29.4.2013, http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MANDUCHI1/Bilateral_Filtering.html.

4. Fairchild, M., Professor, Rochester Institute of Technology, Center for Imaging Science. Color Appearance Models: CIECAM02 and Beyond. IS&T/SID 12th Color Imaging Conference, 11/4/04. Date of retrieval 24.4.2013, http://www.cis.rit.edu/fairchild/PDFs/AppearanceLec.pdf.

APPENDICES

Appendix 1: Sample application with use of webcam as input

```
// A simple demonstration of an OpenCV application where input is webcam feed
#include <opencv\cv.h>
#include <opencv\highgui.h>

using namespace std;
using namespace cv;

int main()
{
		// Image container
		Mat src;

		// Video capture device
		VideoCapture cap;

		// Window where to display the webcam feed, not neccessary to
		// initialize
		namedWindow( "Source" );

		// Container for the key which is pressed
		int keyPressed = 0;

		// Open the video capture device, you can have multiple devices
		// You can choose the device by the number you put in
		cap.open(0);

		// Set the capture device dimensions
		// if set higher than max dimensions then use max dimensions
		cap.set(CV_CAP_PROP_FRAME_WIDTH, 1920);
		cap.set(CV_CAP_PROP_FRAME_HEIGHT, 1080);

		// Program loop, if esc key is not pressed continue
		while( keyPressed != 27 )
		{
				// Get image from device
				cap >> src;

				// Show the image in window, you don't need to
				// initialize
				// the window earlier
				imshow( "Source", src );

				// Check if some key is pressed
				keyPressed = waitKey(1);
		}

		// Destroys the window
		// not neccessary at the end of program since the execution
		// will be stopped anyway
		cvDestroyWindow( "Source" );

		// Destroy all windows
		//cvDestroyAllWindows();
}
```

Appendix 2: Sample application with an image as input

```
// A demonstration of an OpenCV application where input is an image
#include <iostream>
#include <opencv\cv.h>
#include <opencv\highgui.h>

using namespace std;
using namespace cv;

int main()
{
            // Image containers
            Mat src, dstBlur, dstMedian, dstGaussian, dstSobel, dstCanny,
                        dstGray, dstHsv, dstHls, dstBilateral, dstErode,
                        dstDilate;
            Mat sobelX, sobelY;

            // Load source image
            src = imread("lena.jpg");

            // regular blur with kernel size 9
            blur(src, dstBlur, Size(9,9) );

            // median blur with kernel size 9
            medianBlur(src, dstMedian, 9 );

            // Gaussian blur with kernel size 9
            GaussianBlur( src, dstGaussian, Size(9,9), 0, 0 );

            // bilateral blur with d value 27, sigmaColour value 54, and
            // sigmaSpace 13.5
            bilateralFilter( src, dstBilateral, 27, 54, 13.5 );

            // change to gray colour
            cvtColor( src, dstGray, CV_BGR2GRAY );

            // change to HSV
            cvtColor( src, dstHsv, CV_BGR2HSV );

            // change to HLS(HSL)
            cvtColor( src, dstHls, CV_BGR2HLS );

            // Canny edge detector from gray image
            Canny( dstGray, dstCanny, 50, 100 );

            // horizontal Sobel on gray image
            Sobel( dstGray, sobelX, CV_16S, 1, 0, 3 );
            convertScaleAbs( sobelX.clone(), sobelX );

            // vertical Sobel on gray image
            Sobel( dstGray, sobelY, CV_16S, 0, 1, 3 );
            convertScaleAbs( sobelY.clone(), sobelY );

            // combine vertical and horizontal Sobels with weight 0.5 on both
            addWeighted( sobelX, 0.5, sobelY, 0.5, 0, dstSobel );

            int s = 2;

            // filter for dilating and eroding
            Mat element = getStructuringElement(
```

```
                MORPH_RECT,
                Size( 2 * s + 1, 2 * s + 1 ) );

    // dilate with filter
    dilate( src, dstDilate, element );

    // erode with filter
    erode( src, dstErode, element );

    // Show images
    imshow( "SRC", src );
    imshow( "Blur", dstBlur );
    imshow( "Median", dstMedian );
    imshow( "Gaussian", dstGaussian );
    imshow( "Bilateral", dstBilateral );
    imshow( "Gray", dstGray );
    imshow( "Canny", dstCanny );
    imshow( "Dilate", dstDilate );
    imshow( "Erode", dstErode );
    imshow( "HSV", dstHsv );
    imshow( "HLS", dstHls );
    imshow( "Sobel", dstSobel );

    // "Program loop" if Esc key is not pressed wait for a key press
    // if Esc is pressed exit loop
    int keyPressed = 0;
    while( keyPressed != 27 ){

                keyPressed = waitKey(1);

    }

    // Destroys the window not neccessary at the end of program since
    // the execution will be stopped anyway
    // cvDestroyWindow( "SRC" );

    // Destroy all windows
    cvDestroyAllWindows();

}
```