CHALMERS

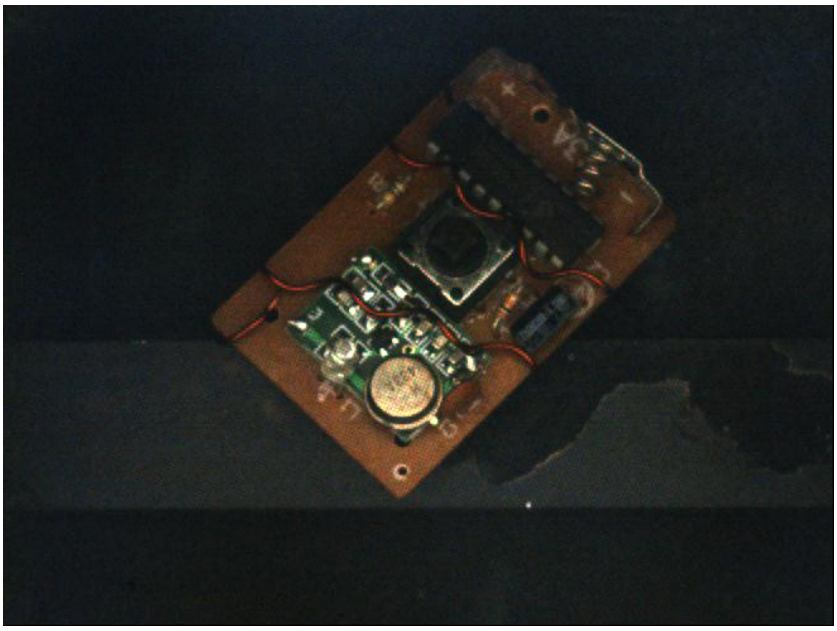

Image segmentation and pre-processing for electronic waste identification

Using OpenCV to compare different techniques for object extraction and rotation

Master of Science Thesis

ROGER LJUNGBERG
MATHIAS ANDERSSON

Department of Applied Information Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2012
Report No. 2012:85
ISSN: 1651-4769

Image segmentation and pre-processing for electronic waste identification.
Using OpenCV to compare different techniques for object extraction and rotation.

ROGER LJUNGBERG
MATHIAS ANDERSSON

Examiner: CLAES STRANNEGÅRD

Chalmers University of Technology
University of Gothenburg
Department of Applied Information Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: A circuit card being photographed and used as input to the pre-processing algorithms.

Department of Computer Science and Engineering
Göteborg, Sweden May 2012

Abstract

This is a master thesis which compares several methods for foreground segmentation and object rotation. By object rotation, it is meant in this thesis that for a given object, having images with different angles as input should ideally always output images with the object in the same angle. Finally, it is tested how this combined can make the task of object recognition easier by running the algorithms for the two tasks successively as an image pre-processing stage. It is shown that making pixel-wise background segmentation by comparing the input image to an averaged background image works well for the segmentation task, and that encapsulating the object in a minimum bounding rectangle and rotating it with the angle of the bounding box can work well for the rotation task. It is also shown that using these algorithms combined as a pre-processing stage to an object classifier may be a way of making the classification easier.

Contents

1. Introduction

1.1 Background

Electronic waste is usually fed into recycling plants in unsorted batches. It is then shredded without further pre-processing. Many products have rare earth materials which may be very valuable or perhaps may contaminate the batches. It would be beneficial to sort the content of the batches and process them in different groups according to material content. Hence, there is a need of finding and localizing each individual product at a conveyor belt.

The company Optisort is specialized in technology for identifying and sorting waste products. Their most recent accomplishment is a method for sorting batteries which is used on recycling plants. The method is based on visual recognition, and takes advantage of the limited variations in shape of batteries. The battery is fed mechanically to the front of the camera. However, when this method is extended to other sorts of waste than just batteries, the same controlled flow cannot be maintained that easily due to the large variance in shapes. Optisort has now finished their battery sorter and wants to develop more solutions for the recycle industry. Thus, in order to be able to adapt the current recognition system to electronic waste products, pre-processing of its input images has to be done in order to avoid excessive amounts of background being part of them, to avoid that several objects are taken for being a single object, and other similar issues.

1.2 Purpose

The main aim of the project is to find an algorithm that solves the two following tasks:

1) Given an image, the positions and bounding areas of each object in it are to be found.

2) Given an image of a single object, the image is to be rotated in such a way that if another image of the same object would be given, but with a different rotation, both of the resulting images would have the same rotation after being processed.

These tasks should be done in sequence, such that each output of task 1 is processed by task 2. Task 1 and 2 combined can be seen as a module, which takes one image as an input and gives a sequence of rotated sub-images as output. These output images could then be fed into a separate image recognition algorithm.

Different candidate solutions for the tasks above should be found and then tested through experiments so that their performance can be compared.

1.3 Limitations

We did not investigate different camera placements and hardware configurations; the methods we investigated just assumed that we had some picture taken and that we were going to process it.

The sample data in the experiments is limited to the kind of objects that the available prototype could produce. The prototype could only handle small objects (smaller than a mobile phone), so testing against larger objects than so has not been done. The objects used in the set of test images are chosen based on what has been of interest to Optisort. The results are thus less general and more specific to their needs.

The algorithms have been implemented using OpenCV in order to limit the implementation time, and thus solutions that have optimized support in OpenCV will be favored in the selection. The programming language used was C++, which is a language that Optisort uses in their implementations, and it is also frequently used with OpenCV. Furthermore, the fields of computer vision and image processing are broad, so we focused our study on some algorithms that are among the more well-known and that have had related use.

1.4 Questions

These are the questions that we aimed to answer in this thesis:

What are some useful algorithms that can be used to find objects in an image, and how do they perform?

What are some useful algorithms that can be used to determine the rotation of an object and how do they perform?

Can some standard pre-processing algorithms enhance the performance of the segmentation- and rotation algorithms?

Can segmentation- and rotation pre-processing enhance the performance of object classification algorithms?

2. Method

The project was carried out roughly as beginning with one month of research studies followed by two months of implementation and testing, and finally one month of report writing.

The literature studies during the first month were used to get a good understanding of which algorithms needed to be implemented and tested. A first step was to get an overview of possible candidates, and then a second step followed which was about searching for more information about the candidates.

After the literature studies, we acquired the data to be used with the experiments. The data was retrieved from Optisort, who produced it with a prototype product.

Next followed the implementation and testing phase. First some algorithms (some for object recognition and some for object rotation) were implemented in a way that they could be compared. Exactly which algorithms that were chosen was a decision based on the knowledge from the literature study. Some were more strictly based on the descriptions found, and some were based more on our own ideas.

After the implementation step was done, experiments were carried out. The purpose of these experiments was to test the algorithms in order to compare their performance and to discover how some different factors affected them. The test results were then reviewed.

The last test was done by choosing the best segmentation algorithm and the two best rotation algorithms and applying them in a pre-processing stage before they were used as input to an object classification algorithm. The purpose of this was to investigate to what degree these image processing algorithms could make the task of object classification easier.

3. Theoretical background

Here the background theory is covered which our experiments and chosen methods were based upon. This work is mainly based on theory from the related areas of Image Processing, Image Segmentation and Computer Vision. Different algorithms from these fields can work cooperatively in the sense that, for example, one might need to process an image by some enhancement method such as smoothing before segmentation can be carried out successfully.

Although we have two separate algorithms for segmentation and rotation, many of the techniques presented here are relevant for both fields. First, how an image is represented and what a computer image is will be covered in chapter 3.1. Then, image enhancement methods are presented in chapter 3.2. Next, transformation methods are described in chapter 3.3. These are primarily of interest for the task of image rotation in this report. Finally, feature detection methods are covered in chapter 3.4, and they can be helpful both when it comes to image segmentation and rotation.

3.1 Image representation and color spaces

General computer raster images are represented as matrices, where each element is a pixel (dot of an image) (Young et al., 1998, p.2).

This is a matrix of sets, where each set is the intensity value of a specific image channel. A normal grayscale image might be represented with just one color channel, while a color image might have more channels. A standard set of channels for color images is the RGB (Red, Green, Blue) set where each channel roughly corresponds to the base colors that the human eye registers.

There are several more ways a color image can be represented in (these ways go under the term color spaces), especially the HSV (Hue, Saturation, Value) / HSL (Hue Saturation, Lightness) is of interest within this field. The HSL color space is related to the RGB color space in the sense that it is a linear transformation of it (Tkalcic et al., 2005). Thus, a color represented as a red intensity, a green intensity and a blue intensity combined can be seen as a point in a three dimensional space where red, green and blue correspond to one axis each. Thus, the set of all possible colors has a form of a cube in the linear space. If this cube is uniform and has a side of length 1, then each point on the line, x = y = z, (an equal amount of each color) represents a grayscale value from black (at point x = y = z = 0), to white (at point x = y = z = 1). This is the L value of HSL. The S value is then the distance between the color point and the closest point on the x = y = z line. The H value is the angle of rotation of the color point along the x = y = z line (against some predefined reference). Note that this is just a rough explanation of how the RGB color space correlates to the HSL color space and not an exact description of how to implement conversions between the two color spaces.

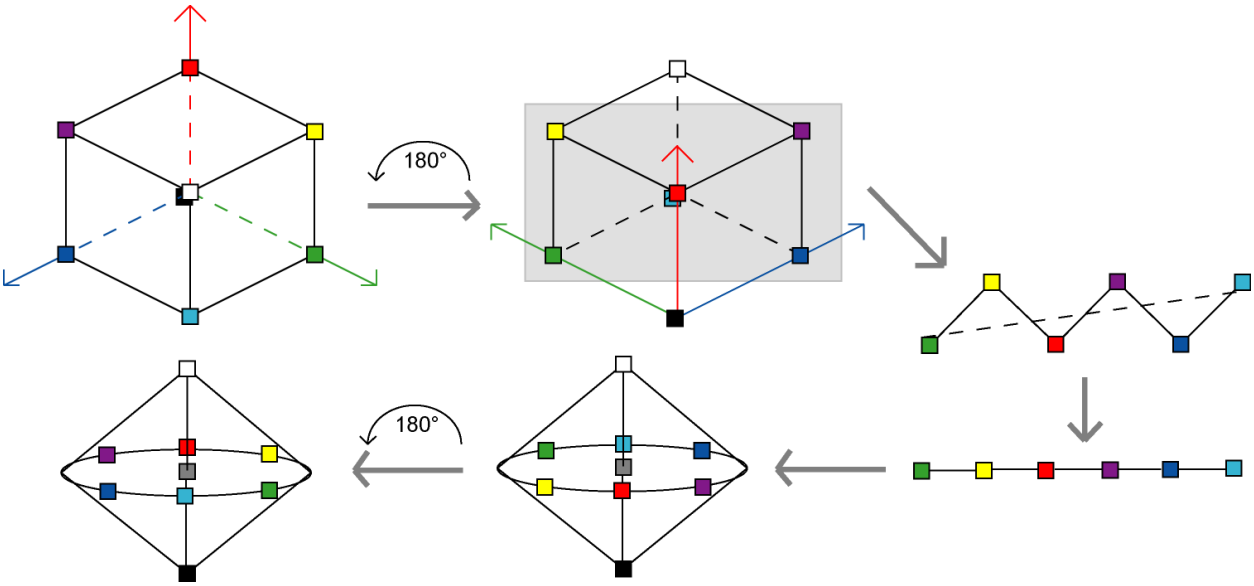

Figure 1: A visual representation of the RGB color space and a rough approximation of how it can be converted to a HSL color space. The idea is that by simply doing a three dimensional rotation of the RGB cube and flattening the color edges so that their distances to the white and the black edges are equal, a HSL style representation is acquired.

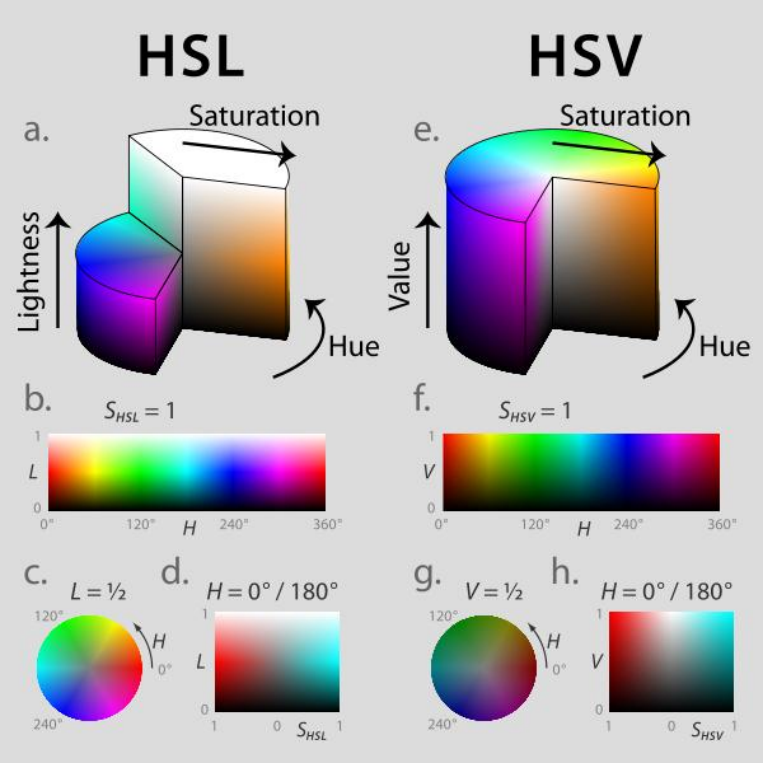

Figure 2: The HSL and HSV color spaces (Wikimedia Commons, 2010)

3.2 Image enhancement methods

A major part of image processing is the field of image enhancement. In this section image enhancement will be defined as methods to alter an image in order to enhance certain aspects of it (Wu et al., 2008, chapter 6).

There are a couple of different ways to approach image enhancement. For example, the most basic operations on a gray scale image would be multiplication and addition of the intensity values. Addition of a positive value would make the image lighter as each intensity value would increase, and addition of a negative value would make the image darker. Multiplication affects the contrast of the image, such that multiplication with a value greater than 1 increases the range of intensity values (higher contrast), while multiplication with a non-negative value less than 1 decreases the range of intensity values (lower contrast). This is a kind of operation that could be done pixel by pixel, with each pixel´s change being independent of the other pixels. On the other hand, there are also approaches that make changes to the pixels based on statistics from the collected data of neighboring pixels, which are described in sections 3.2.3 and 3.2.5 below. In section 3.2.1 and 3.2.2 histogram methods, which consider collected statistics from all the pixels and make changes more to the image on the whole rather than strictly pixel by pixel, are considered.

3.2.1 Histogram

A histogram can be seen as a graph derived from an image. For each channel of the image, the frequency of every possible intensity value is calculated. A histogram is thus a representation of the original image where the position of each pixel is lost, but instead the amount of each possible color in the color space that exists in the image is made available for analysis (Bovik et al., (2005), chapter 3). Histograms are usually visualized graphically; however a histogram is basically a table consisting of number of pixels for each intensity value in the image.

Histograms have several areas of use in the fields of Image Processing and Image Segmentation. Two of these will be presented below; namely threshold selection and histogram equalization.

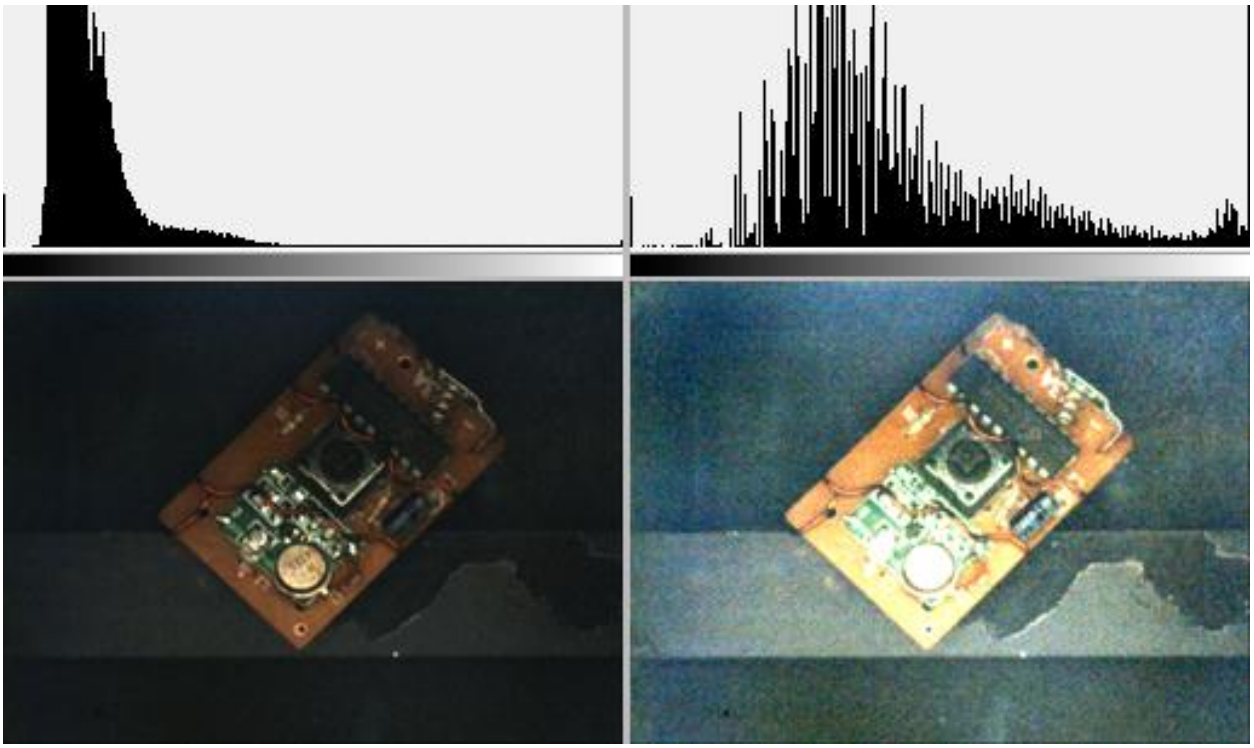

Figure 3: A histogram can be represented as a diagram. One can imagine having "number of pixels" being labeled on the y-axis, and "pixel brightness" on the x-axis. This would then group pixels with the same brightness for the respective images.

3.2.1.1 Histogram threshold selection

Histogram threshold selection is used for separating the image into different parts, where each part represents a certain brightness level. Thresholding makes the picture undertake a binary choice for each pixel; it stays or does not stay. For instance, pixels could turn white or black depending on this. The light pixels could represent foreground and the dark pixels the background, or vice versa. Thresholding is not always completely trivial, as there are often cases

where the brightness levels are evenly distributed and the image contains pixels of a large variety of nuances and colors (Shapiro et al., 2001, p. 99).

Several threshold selection methods exist for problems such as the one covered in this thesis, which use statistical information gained from the histogram in order to select suitable thresholds. The different approaches are based on shape information, space clustering, entropy information, image attributes, spatial information and local characteristics (Sezgin et al., 2004). This means that one can look at either the peaks in the histogram, clustered gray level samples, the entropy of the different parts, similarity measures, spatial probability distributions or local pixel characteristics in order to make the thresholding.

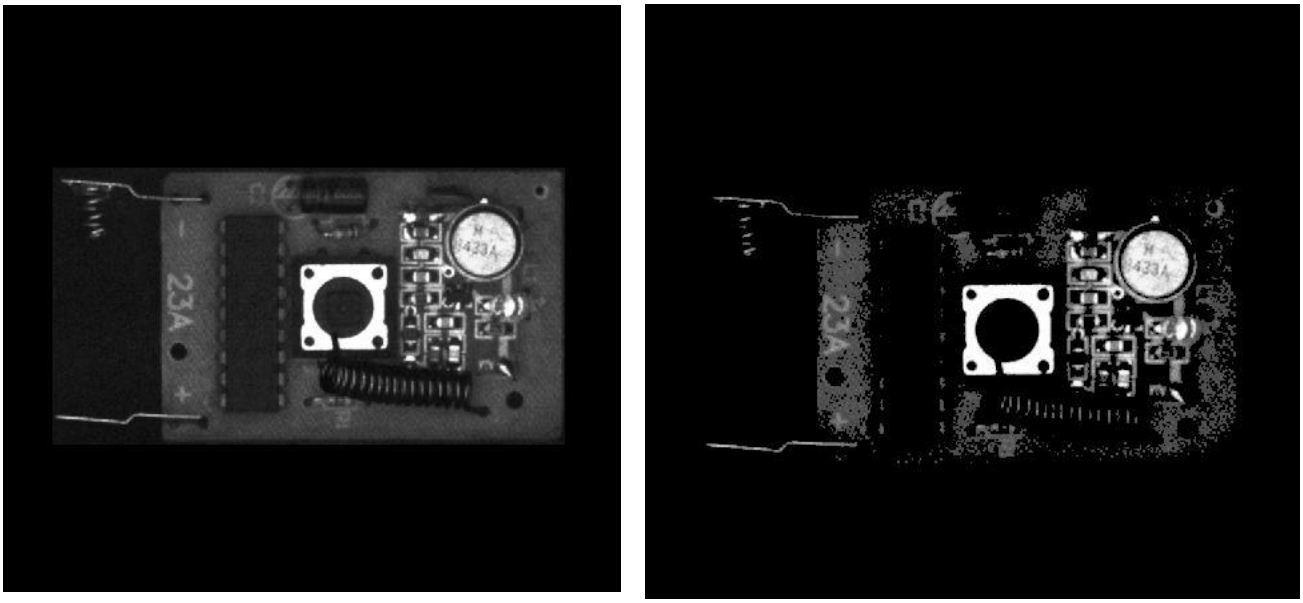

Figure 4: Before (left image) and after (right image) thresholding. The dark pixels are set to being pitch black, leaving only light pixels left in their original colors.

3.2.1.2 Histogram equalization

In a histogram the distribution can be analyzed. Histogram equalization is a method that takes an image, calculates its histogram and, given its distribution, extracts parameters and runs a function which, given the image, outputs an adjusted image. The goal is that this output image has a more evenly distributed histogram than the original image so as to heighten the contrast (Gonzalez et al., 2002). The process that ensures this is built upon the usage of a cumulative histogram with a wider range of intensity levels and lower sum of total intensity for the lower intensity scale, and a higher sum the higher up in the intensity scale (Song Ho 2006).

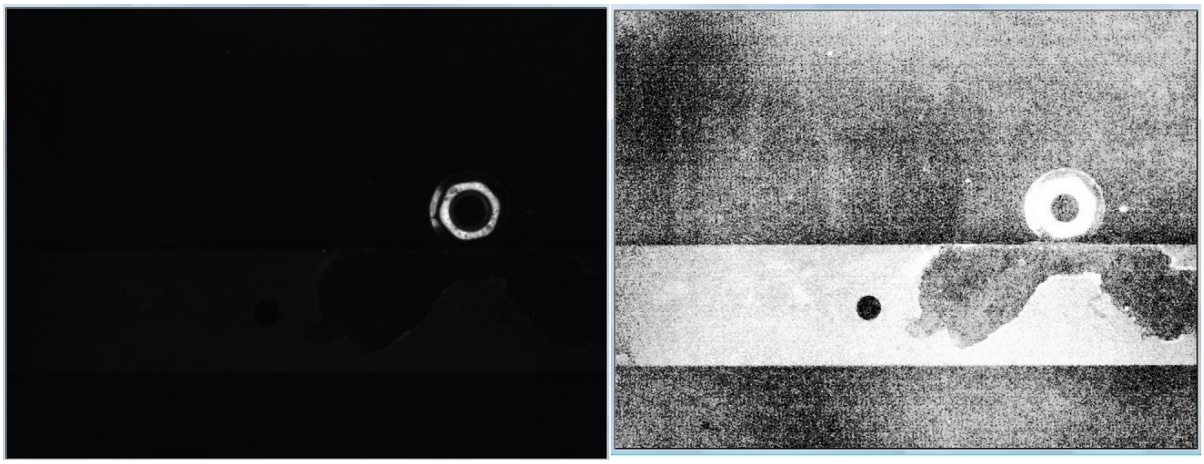

Figure 5: Before (top image) and after (bottom image) equalization. The intensity in the image is evened out, and it now appears as being clearer.

3.2.2 Filtering techniques

In practical problems, it might be needed to apply one or more filters in order to reach a certain goal, or filters might be a prerequisite for other algorithms to work well. For instance, in edge detection algorithms it might be preferable to blur the image somewhat as a pre-processing operation for more preciseness.

A common way of implementing filtering is by using convolution. This is done by using a kernel matrix, usually 3x3 in dimension, and sweeping it over the image with each pixel being the center of this matrix once. For each centered pixel, a change is made to it which depends on the computation made with the neighboring pixels being in the array. In other words, the kernel is a summation function where the output is the new pixel value and the inputs are the neighbors values individually weighted. Different kinds of filters make different changes to an image, for instance edge sharpening, noise reduction, blurring and morphology are some of the possibilities. A drawback of using convolution can sometimes be that many calculations have to be carried out, so that it might take some time (Smith, 1997, chapter 24).

Convolution filtering is not the only way of filtering, though. Median filtering, which basically is a way of ranking pixels locally, is another way of filtering. Even Fourier Transform can be seen as a kind of filtering, as it converts the image data to the frequency domain (Spring et al., 2007).

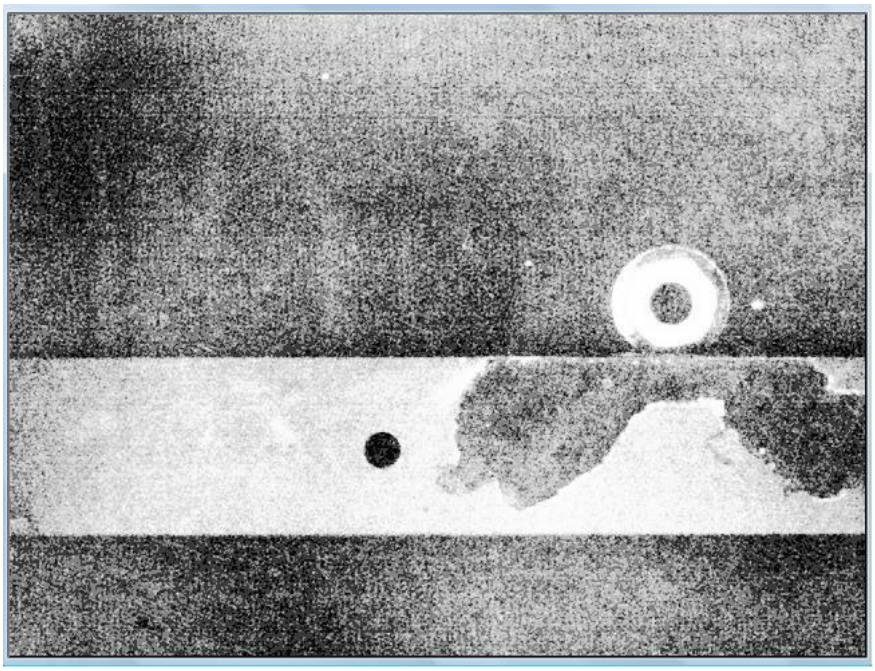

Figure 6: An original image

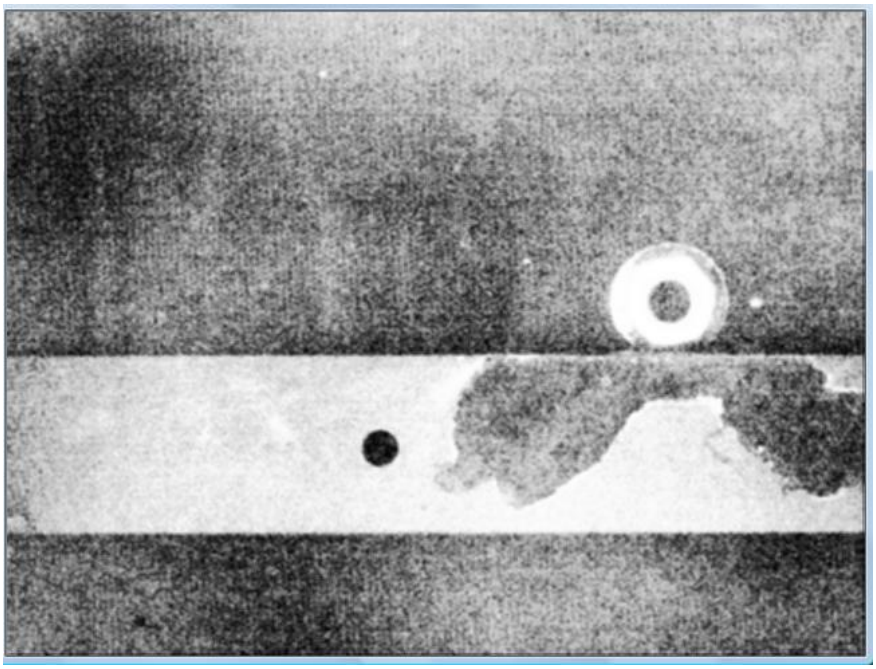

Figure 7: The same image being processed with a blurring filter.

3.2.3 Noise reduction

There are many possible sources for noise in digital images. It can for example appear at the very acquisition of the image from the sensor, from various hardware malfunctioning problems or from bad transmission (Gonzalez et al., 2002). Noise can be seen as a local deviation from the

ideal value at a pixel. Noise can be further classified by its characteristics, such as whether the pixels deviate to a certain degree or to gets an extreme value, if only a certain channel gets affected, as well as the distribution of the noise. Noise reduction is thus the process of trying to remove these deviations and getting as close as possible to the ideal noise free version of the image without too much distortion by the process. There are many ways of trying to reduce noise and the following is only a brief overview.

First, there are algorithms based on evaluating the pixels one by one, with each evaluation being based on the neighboring pixels. Using kernelization to take the median of the neighboring pixels would for example be able to reduce so called salt and pepper noise (pixels having either full intensity or no intensity) without introducing too much blur in the image (Chan et al., 2005). On the other hand, using a kernel with the average of a neighborhood would result in some form of blurring filter, and as every pixel depends on their neighbor, local pixel deviations will be evened out. Other approaches could be using a point detection algorithm that evaluates if a certain pixel differs with more than a certain threshold value from its neighbor (in fact, it is a little more advanced than this) (Gonzalez et al., 2002). On binary images a morphological operation such as erosion followed by a dilation operation could remove too small details that might be the result of noise.

Figure 8: A black image which contains small fragments of white noise.

Figure 9: Noise filtering removes most of the noise.

3.2.4 Morphological operations

Morphological operations is a kind of filtering originally for black and white images, but it can be expanded to a range of different other types, such as gray scale images. It is a filtering technique with the two fundamental operations (or really weighted combinations of) dilation and erosion. These operations could then be combined in various ways to form more advanced functions (Gonzalez et al., 2002).

Dilation works in the sense that it expands the borders of foreground objects. Using a kernel (here the usual 3x3 kernel), the idea is that when making the convolution for the image, as described in section 3.2.2, each pixel is the center of this kernel once. Then if there is some foreground pixel among the surrounding ones, the centered pixel is set to being foreground (regardless whether it was background or foreground before). Erosion is the reverse; the pixel is set to being a background pixel if there are some surrounding ones being background pixels themselves, otherwise do nothing (Fish et al., 2003).

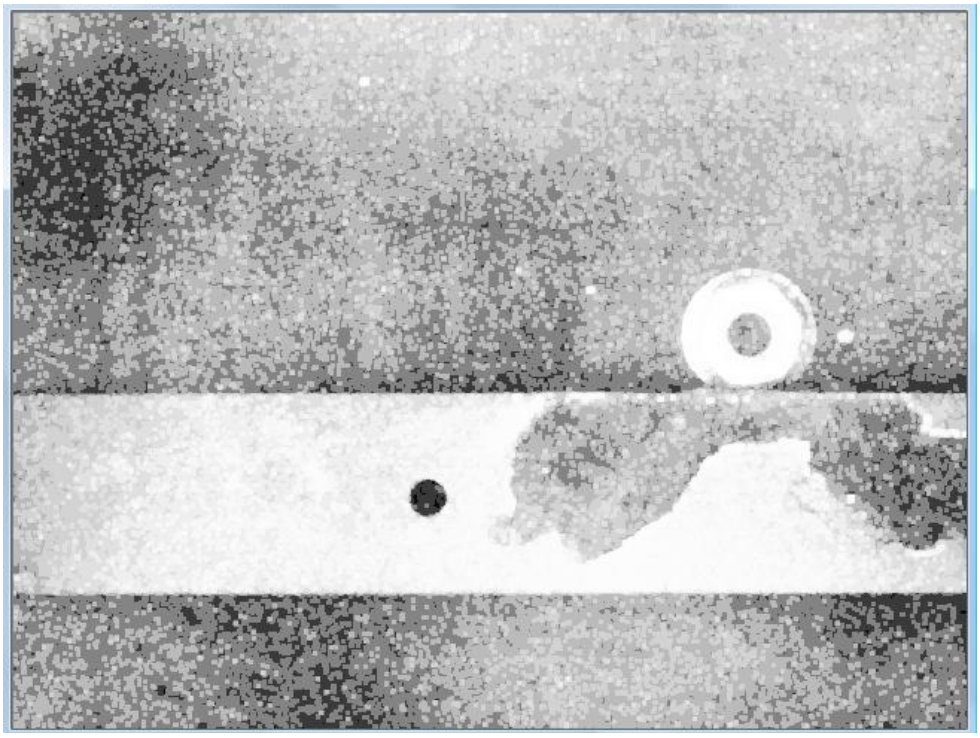

Figure 10: The image from figure 5 being dilated.

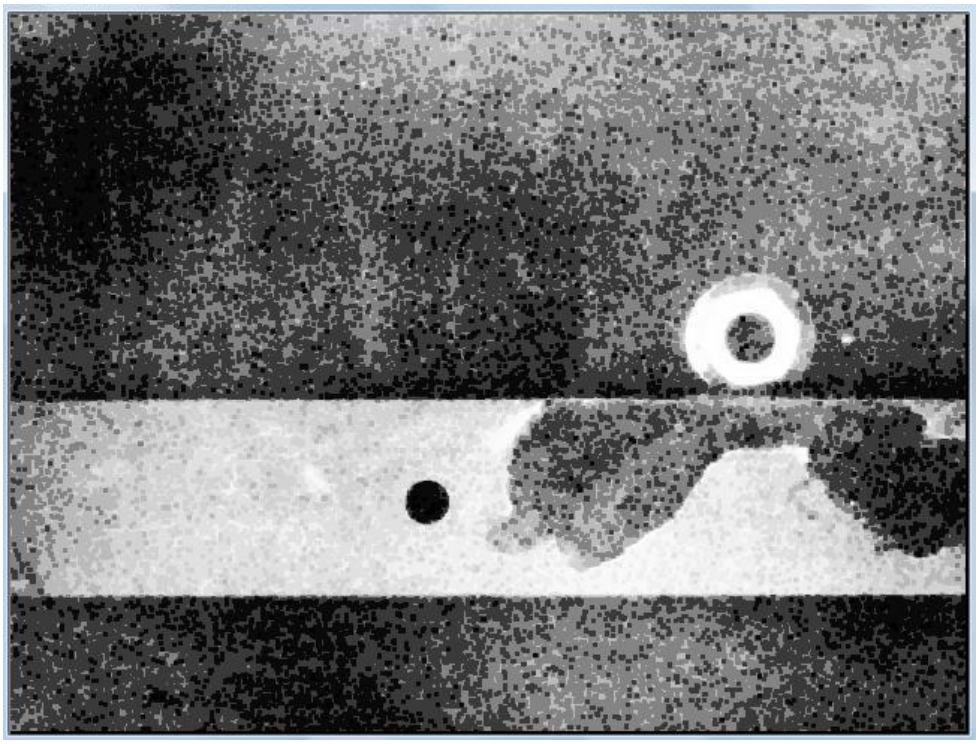

Figure 11: Here the image is eroded instead.

3.3 Image transformation methods

An image transformation returns a new picture from some given picture. The new picture will then enhance or highlight certain features of the old image (Natural Resources Canada, 2008). First and foremost alignment and spatial color distribution are such features that are of particular interest in this report, and methods will be described below which makes it possible to analyze them among others.

3.3.1 Fourier transformations

Fourier transformation is a way of representing a signal as a set of sinus waves (Sundararajan, 2001, p 18). The theory behind the Fourier transform is too complicated to be explained in greater detail here, but the transformation itself is based on decomposing the pixel intensities to a set of orthogonal functions (Owens 1997). An important property is then that one can transform the diagram back to the standard array image form after making the desired manipulations of it.

There are several variants of the Fourier transform. A common variant for digital images is the Discrete Fourier Transform (DFT) which has an advantage of being relatively fast to compute (Sundararajan, 2001, p 54). As the name indicates, DFT represents sinusoidal waves. The result of this is that given one input image, two output images (real and imaginary) can be obtained. Many filters can be defined which are based on processing these output images (which represent the frequency domain of the original image), such as noise reduction. For some pixel with coordinates (x, y) in the original image and (u, v) in the Fourier image, the equation of the transform for the DFT is (Marshall, 2001):

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x,y) e^{-2\pi i(ux/N + vy/M)}$$

The following formula is used to reverse the transform back again:

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u,v) e^{2\pi i(ux/N + vy/M)}$$

Analysis of the real output image is of special interest, as this image represents the structural information of the original image. More specifically, a normal image tends to have the frequency domain look of a body of water, in which a stone has been dropped in the middle, resulting in circular waves that are very intense at the middle and that even out the further away they reach. Thus, in the center of the frequency domain image there is usually more intensity than at the edges. Furthermore, the intensities closest to the center of this image represent the

comprehensive structure of the original image, while the further out from the center, the more the intensities represent the details in the image. A filter for removing details in an image can thus be constructed by keeping the part of the frequency domain image that is closest to the center and removing everything else, and then transforming back the image (Gonzalez et al., 2002).

Another analytical aspect of interest is finding the alignment of the image. The dominating alignment angles will be represented by the clearest straight lines in the Fourier diagram (Fish et al., 2003). Thus, one could for example use the Bresenham line algorithm (Bresenham, 1965) which approximates the drawing of a straight line between two points in a matrix (if the line is drawn on the screen image, it is essentially drawn in a matrix because of the screen resolution aspect) to search for the line with most white pixels in it.

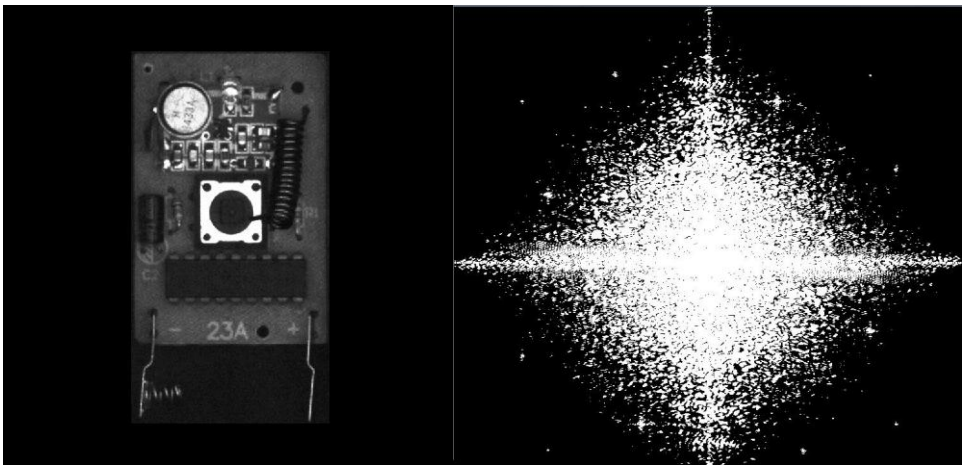

Figure 12: An image (to the left) and its Fourier diagram (to the right). One can see in the diagram that horizontal and vertical lines dominate the image, since these directions are dominating in the diagram.

3.4 Feature detection methods

This section is devoted to feature detection methods (or more concretely edge-, line-, and Points Of Interest (POI) detection methods as we have limited the content to). To make it possible to see what an image really contains, it might be necessary to make a residual image that is a simplified version of the original, for instance black and white with white as foreground and black as background, or in some other way clearly distinguish different objects and contours in an image. This is what the first section is about. Next follows line- and POI detection, which search for more specific features that stand out in some way, and mark them in the image.

As will be seen, several of the methods presented here will more or less work as algorithms for deciding the orientation angle for a given image. This is intuitive, since many of the algorithms which were eventually used for finding the rotation would be based on averaging some set of features (angle of the average line of the line detection, for instance), so this chapter basically covers algorithms for rotating the images consistently, as well as different processing algorithms. The only rotation algorithm which was tested that is not presented in this chapter is the algorithm

based on Fourier analysis (a concept described in section 3.3.1). Several of the segmentation algorithms from the experiments are also introduced here in relevant sections.

3.4.1 Edge detection

Edge detection is a class of methods to be used when one wants to find the borders of the objects in an image, or just some of the contours. The more precise definition of an edge is a sudden shift in intensity at some border (Qurechi, 2005). There are many algorithms for edge detection, and we list some of the more well-known below. The choice of algorithm can be rather subjective sometimes as most edge detectors have individual circumstances where they work either good or bad (Nadernejad et al., 2008).

3.4.1.1 Edge detection with Sobel derivates

The Sobel operator works in the sense that it approximates the gradient of an image intensity function with Gaussian smoothing (opencv dev team, 2011). Thus, the edges are assumed to be located where the jumps occur in this approximated differentiation (Qiu, 2001). More concretely, a *sobel mask* is used as a convolution kernel, and for a 3x3 mask it looks like the following:

For x axis:	For y axis:
-1 0 1	-1 -2 -1
-2 0 2	0 0 0
-1 0 1	1 2 1

(Bebis, 2003). Then as stated, convolution is used with this kernel and how that works was described in section 3.2.2.

One requirement for the Sobel edge detection to work well is that there is a regular spatial distribution of the design points (Qiu, 2001). Furthermore, there are both advantages and disadvantages with it; an advantage being that it is easy to implement and runs relatively quickly, and a disadvantage being that the edges might be thicker than necessary (compared with Canny´s edge detector for example, which is described later on) (Vincent et al., 2009).

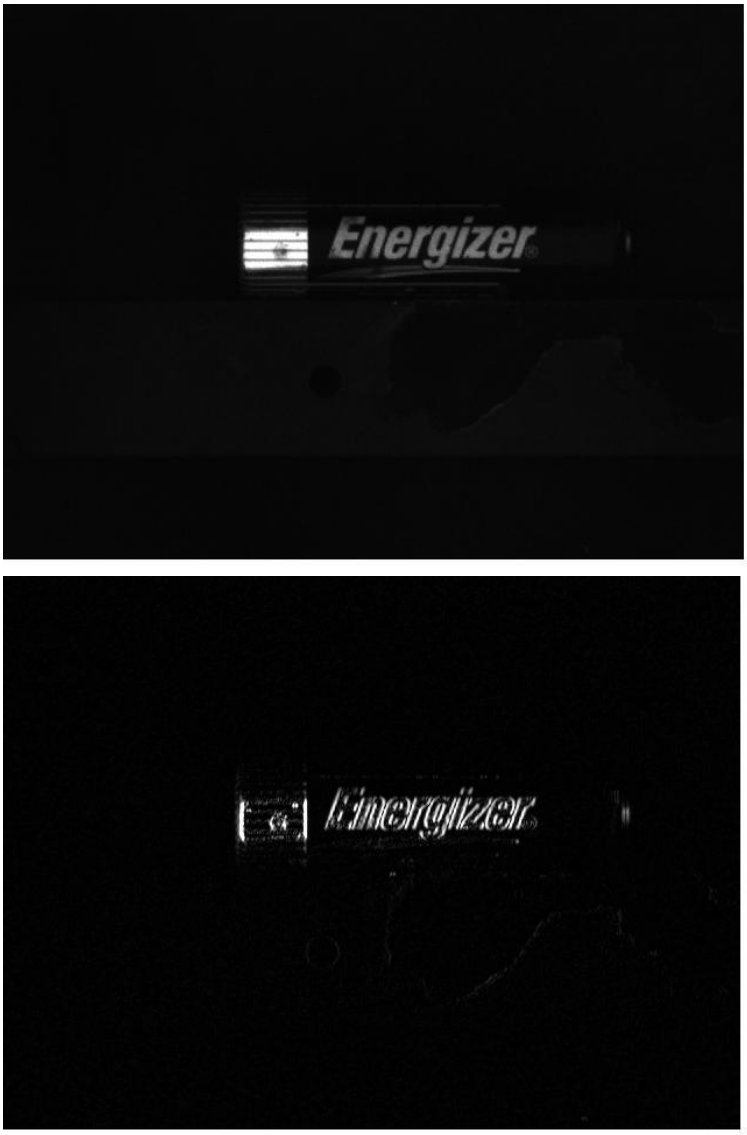

Figure 13: Sobel edge detection is used on the top image, showing the skeletonized result on the bottom image.

3.4.1.2 Laplacian edge detection

Laplacian edge detection uses calculations of the second spatial derivative to detect the regions where the intensity changes most rapidly (Fisher et al., 2003). As with Sobel edge detection, Laplacian edge detection could practically be implemented by convolution. However, the difference from Sobel edge detection is that now the second derivative is used, and not the first (the change of the slope is calculated instead of just the slope). This makes it somewhat different from the Sobel edge detection, even though much of the core concept is the same as both are based on working with gradients. An advantage of Laplacian edge detection is that it might be better at finding the localization for the edges, while a disadvantage can be that curves and corners might pose slightly more problems than for Sobel edge detection (Bhadauria et al., 2010).

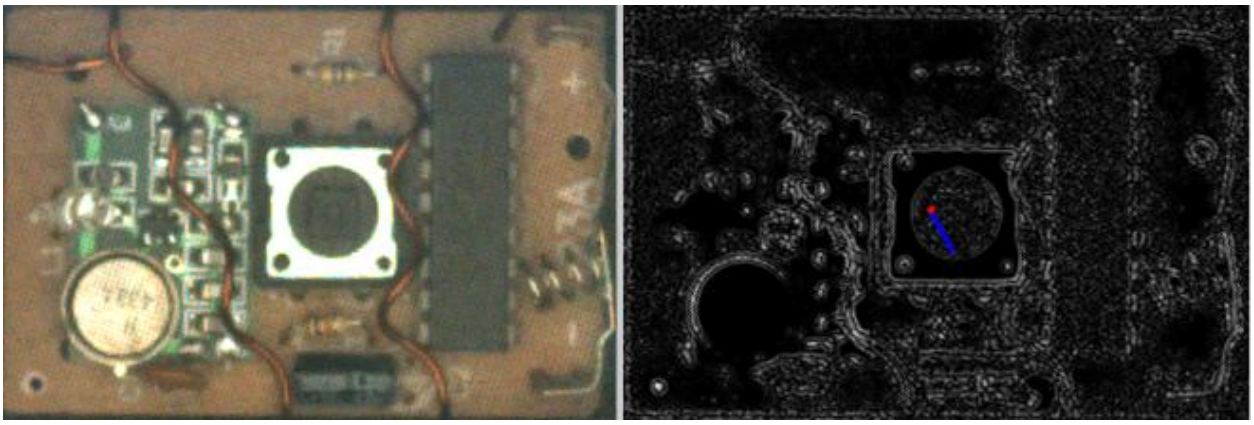
Figure 14: Laplacian edge detection is used on the image to the left, showing the skeletonized result in the image to the right (the red blue line marks the rotation angle calculated)

3.4.2 Line detection

A well-known line detection algorithm (which is of interest here since it has support in OpenCV) is the Hough transform (Duda, Hart, 1972). The method is based upon a voting system for the pixels in an image (voting in terms of number of curve intersections at certain points), which decides the parameters of line segments, and running the algorithm will eventually result in an outlining of these. While it can be an effective way of detecting the lines in an image, a drawback is the relatively high complexity of the algorithm (Fisher et al., 2003). An advantage that compensates this is the property of being robust to noise (Turkel, 2011). The interesting aspect in this report is to see whether Hough transform gives a consistent output of lines for certain objects.

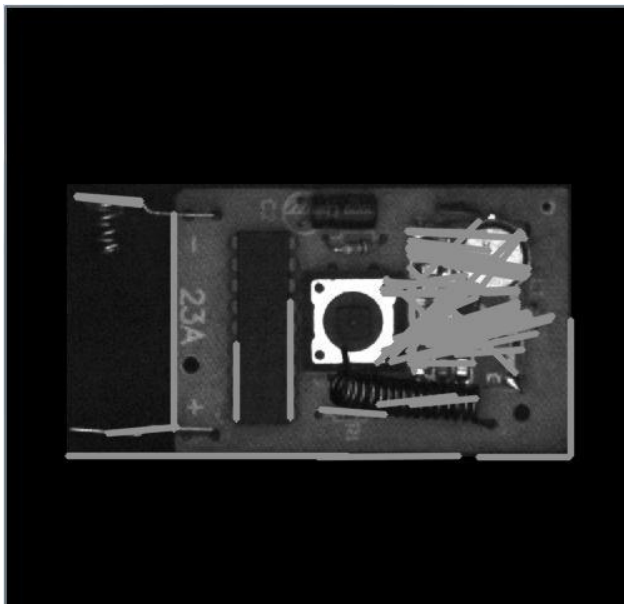

Figure 15: The Hough Lines algorithm is used to mark noticeable lines on a circuit card.

3.4.3 Thinning

In order to make the task of detecting contours and rotation for an object easier, thinning can be used in order to get lines that represent the skeletonized shape, while also preserving the topology (Palágyi). This could possibly make the process of detecting the alignment for some object easier, as lines can be easier to handle than the object itself. For instance, one way can be to view the lines as vectors, and making calculations with these (for instance, summing them could be a possibility) could be a way of finding the alignment.

Basically, thinning is a morphological operation (Fisher et al., 2003) which uses a kernel to acquire the result. There are a couple of different ways of thinning as well; one could get the whole "skeleton" of an object (the shape drawn with lines) or one could just get the corners of it. In fact, the last approach could have possible aspects of interest for finding the alignment, for instance by marking vectors between the center point and the corners, and then computing some kind of average or median angle. There are many different algorithms for thinning (or skeletonization), however using the kernel is a base approach which deletes or keeps black colored pixels depending on the amount of neighboring black pixels and their connectivity (Saeed et. al, 2010).

3.4.4 Points of interest detection

There are algorithms for finding points of interest in an image; points that make the image stand out or points that describe it in some way. It is desired that such an algorithm is robust to noise, orientation and other features when being in use. One such algorithm is the SURF descriptor which uses integral images and Hessian matrix based detection measures, among others. It has been shown to be an effective detector of interesting points and also to be relatively fast (Bay et al., 2008). Basically this is a high level function in OpenCV, and the specific usage of it was interesting in this thesis for the rotation problem.

POI detection has several applications. One such application could relate to the task in this project of finding the rotation; a mean point could be created from the interest point and a vector could be defined as going from the middle point of the image to the mean interest point. This could then stand for the orientation angle of the image.

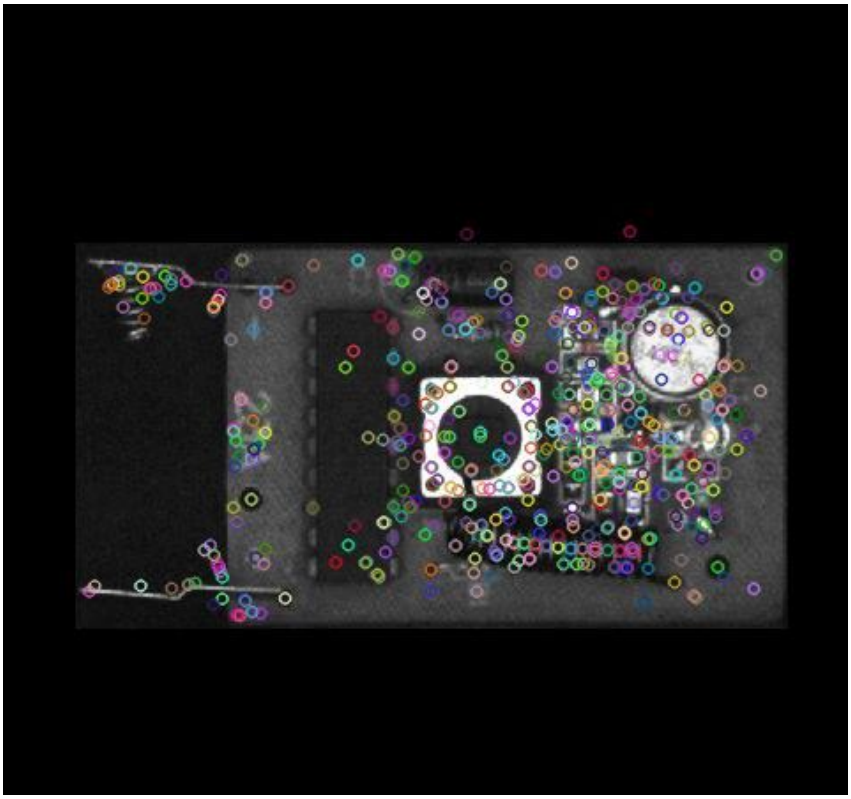
Figure 16: A number of points of interest are detected for a circuit card using OpenCV´s library.

3.4.5 Histogram of Oriented Gradients (HOG)

This section demonstrates an example of how histograms can be used to implement an algorithm for handling the rotation task in this thesis. First, some method is used to get the first order intensity derivative for each pixel (for example Sobel edge detection) (Dalal et al., 2005). This derivative basically represents the slope of the intensity change between pixels. This is done for both the x and y axis, whose results are stored separately. Next, these Cartesian coordinates are converted into polar coordinates, and these are stored in a histogram. Finally, the averaged angle is calculated for all the angles of the polar coordinates. This is basically the final angle that the algorithm outputs.

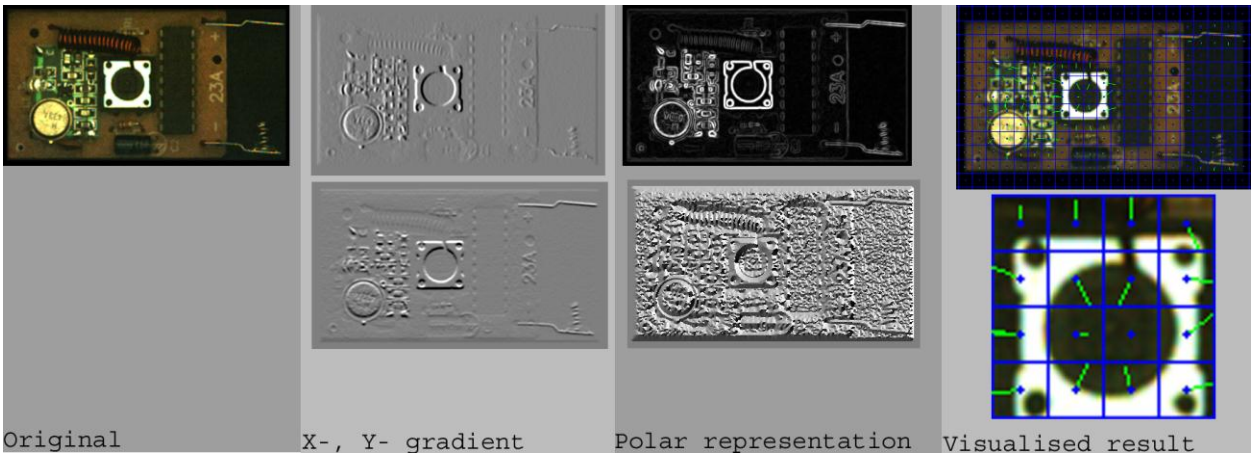

Figure 17: The image to the left is the input image to the HOG algorithm, and to the right it is gradually going through different stages. Gradients are first calculated for the X- and Y axis separately, and then the result is transformed into a representation of these with polar coordinates. The result of this is visualized in the two rightmost images, and the vectors are summed in order to calculate the final angle.

3.5 Background subtraction

In this project, what is to be regarded as the most crucial part of the image segmentation algorithm is the background subtraction. This is since once the background is gone from the picture, the rest is easy; let us suppose we know that everything but foreground objects has a single color; then we can use a simple color detection to retrieve the objects, for example. There are several approaches for background subtraction. However, in some way or another, the methods below are all based on comparing some form of statistics between the background picture with and without objects on it.

3.5.1 Pixel/histogram statistics

Statistics can be acquired to divide the image into pixels probable to be background and foreground, given a sample image as basis (opencv dev team, (2011)). This approach is popular in the computer vision area thanks to the relative simplicity in the computational aspect (Noriega et al., 2006).

A simplified description of an algorithm for background subtraction using histograms could be as follows:

Take a number of pictures on the background alone. This number depends on how dynamic and large the background is. Then, compute an image that makes a good representation of the average background by averaging over the pictures taken. Then, for each input image, find a suitable threshold value that subtracts the pixels probable to be background pixels using both the histogram for the averaged background image as well as the histogram for the input image, and do this by comparing the difference between these two histograms.

Alternatively, one can run the algorithm without using histograms, and instead comparing the images pixel by pixel. Then in the input image, one simply keeps the pixels that differ to a greater extent (because an object will probably be located there) and each pixel that is of about the same intensity in both images will be discarded.

3.5.2 Mixture Of Gaussians / Gaussian functions

A background subtraction algorithm by Kaewtrakulpong and Bowden was based on the Grimson and Stauffer background mixture modeling approach using Gaussians (Kaewtrakulpong et al., 2001). A brief, simplified overview of the theory is the following. Each color in the image is modeled as a Gaussian. The final background subtraction is made after collecting the information from each pixel and constructing these Gaussians, and this is done by keeping the pixels that differ more than some constant number of standard deviations in intensity from the rest of the distributions.

Another way is to model each pixel as a Gaussian distribution depending on what color it has had over time (in the different images in the image series). If it is seems stable, it is probably a background pixel. More uneven distributions for the pixel indicate that foreground has taken place there for some image.

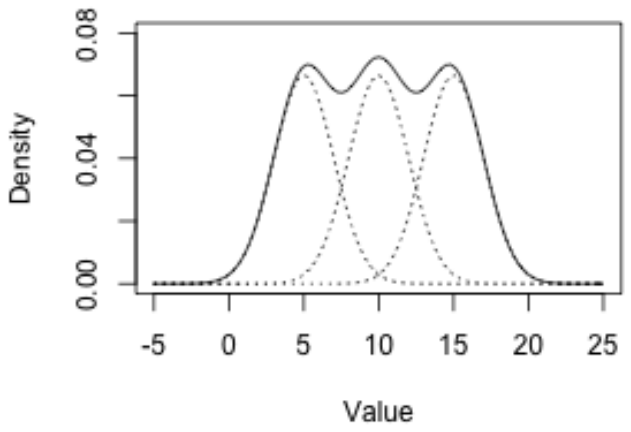

Figure 18: An example of a mixture of three Gaussian variables (Wikimedia Commons, 2009).

3.5.3 Central image moment

The centroid of an image is calculated as a mean of the image intensity in the image. So the result is a point, which has the property that the intensity is equally distributed around it throughout the image (Intelligent Perception, 2010). In other words, all pixel values can be summed up to find this point which becomes the gravitational center. The real advantage of this

method is that it is robust to minor shifts in the image - small rotations or small amounts of noise do not alter the result very much (Utkarsh, 2011).

It might be required to have the light settings exactly the same for all pictures when running this algorithm for the task in this thesis of finding the rotation of a certain object. If the light conditions are not fixed, i.e. there are reflections in some pictures, the result could alter drastically since the intensity distribution might get shifted.

3.6 Image Classification

This section is devoted to describing the method for object classification that was used in the final experiment, when testing how the pre-processing algorithms in this thesis can enhance the results when classifying objects. Because of certain time constraints for this project (about one or two weeks were devoted to the implementation of the classifier) we chose a relatively simple classifying algorithm, namely Template Matching by comparing pixel intensities.
This has been tried before with successful results (Pinto et al., 2010). There are many other choices that could have been made, but we limited the thesis to cover only this one as the field of Object Detection is broad and many algorithms are too complex to be implemented in the time frame given. Optisort recommended us to implement this algorithm because of this.

The underlying idea of the algorithm is to go through the image retrieved by the algorithm (or taken by the camera, if no pre-processing is done) pixel by pixel and compare it to the same corresponding pixel in the images stored that represent each object (the templates). In order to see which of these objects the image resembles most when all the pixels are iterated through (i.e. the least total intensity difference is for that object among the different ones).

3.7 Earlier research work of interest

It is of interest what research methods have been tried already in similar tasks to those in this report. One interesting point is that different kinds of height sensing or similar techniques resulting in a three dimensional image for analysis have often been applied in automated waste sorting and research (Mattone et al., 2000), (Dop, 1999). It is also a frequent approach to the more general object recognition task, and video processing is sometimes chosen instead of using images (Mamoru et al., 2000), (Gould et al., 2007).

Numerous methods have been used for separating foreground from background (segmentation in other words), for example the background subtraction part color modeling has been tried (Horprasert et al., 1999), and also Bayesian Rules (Li et al., 2004) which is a bit more novel than regular image processing methods. This is not to mention the approaches already written about in the section about background subtraction in this report. For example, histograms have been used as an important tool (Arifin et al., 2006). Also color-set back projection is a method that has been used, and it has worked (Doringa et al., 2010).

Fourier transformations have been used for registering images in the sense that they become rotation invariant; in other words similar problems to the image rotation problem (Makadia et al., 2003). Interestingly enough, also histograms (Villamizar et al. (2006) and wavelet transforms (Lee et al., 2002) have been used to solve problems of similar character (Villamizar et al. used gradient orientation histograms to compute image feature orientation for object detection). Thinning methods have been used as a part of rotation invariant algorithms (Ahmed et al., 2002).

It should also be mentioned that the problem of detecting skewness for printed characters has also been approached in several ways (a problem relatively similar to the rotation problem here). For example, Hough transform has been used successfully for detecting this kind of skewness (Nandini et al., 2008).

In summation, there are many approaches that have been tried to problems similar to the problems in this thesis. In this project, however, we have chosen the methods that seemed most suitable for the conditions in our specific problem. Some of these related works still worked as inspirations even though there was no direct application.

4. Experiments

The following experiments were done in order to gain understanding about what performance could be expected from the algorithms. The experiments are, as stated earlier, image segmentation, image rotation and object classification in consecutive sections 4.2-4.4. The experiments were made on a Compaq 6720s laptop with an Intel Celeron Processor with a 1.73 GHz processor and 2 GB RAM (and with other processes in the background while running them). While the choice of computer certainly may affect the performance, the idea was mainly to get a comparison of the different algorithms knowing that the final performance can probably be improved, while still getting results indicating the best algorithm choices. Even if the performance differs between different computers, it was believed that one could get results with the computer chosen here indicating whether the algorithms were probable to be effective in a real production setting. Optisort had a benchmark of 0.1 seconds; the total time that was desired to be kept as an upper limit. Then if some algorithm would take 2 seconds to complete, for example, the 0.1 second limit could probably not be managed even by a stronger computer. So getting results for the sake of comparison and realistic evaluations was the main goal.

4.1 Test set

The test sets for all the experiments below have been obtained from a prototype made by Optisort. The point with this is to make sure that the results should reflect the actual performance if used with a final product in a real environment. However, the prototype might differ from the final product, which also might be the case for the photographed test objects, so it is possible that the results might still vary. However, the main idea is to show, since the camera is to be stationed at one fixed area, that with some program settings the program can work for that environment. Because of the fixed camera position, specific light settings and a specific background (in form of a specific moving conveyor belt), one may need to adjust the program settings to fit some other conditions. In this experiment we had a specific prototype with some specific conditions to

work with. However, as explained, these conditions might be different in a release version, and thus it is possible that one needs to adjust the program settings accordingly (by changing the threshold values, for instance).

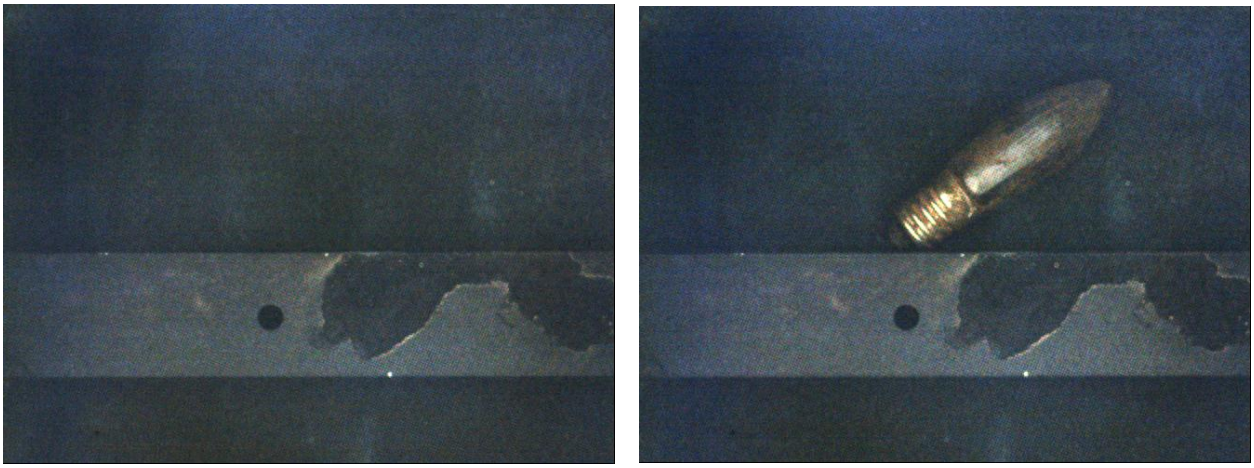

Figure 19: The left image displays the conveyor belt with no items on it. The right image shows a lamp that has appeared on the conveyor belt. This is how images in the test set would typically look.

Due to the fact that Optisort already uses high quality cameras and light settings in order to reduce errors based on image quality, comparing image quality was not considered to be a relevant aspect in the testing. A more likely source of error is dirt on equipment and objects. However, the test set was collected with a representative amount of dirt already. Thus, it is expected that the test set already represents the variation in dirt that would occur in a final product. After all, the test objects were collected at Renova, a waste sorting company, so the test objects represent data from a real live setting. A varied series of objects were used; light bulbs, clocks, circuit cards, LED lamps, screws, pencils and batteries among others. 199 images of the foreground objects were taken, and 21 background images were taken (to be used with the segmentation methods for background subtraction). As explained, the camera is fixed but the background is still partly dynamic because of the moving conveyor belt.

The images that were taken by the prototype to be used for the test set had each object's frame defined by hand in order to enable automated testing. This means that a data file with optimal results was done by hand, and the output of each test was saved in a file of the same format. A small test program made the comparison between the output file and the file with optimal results. When testing the rotation, a test set was used that basically consisted of the segmented images from the first experiment. A similar way of outputting the performance of the algorithm was used, as described for the segmentation above. This enabled automated testing, so that when acquiring the algorithms' performance, the program was run for all the test pictures and the results were written correspondingly to the output file. Then, the test software was used to iterate automatically through the results for all the pictures in the output text file, and the performance was calculated in terms of correctness (measured in number of pictures) and average time spent on each picture for the output image (that is for the segmentation task, for the rotation task the

first is instead exchanged for rotation degree variance measured for each object, not for each image).

Both accuracy and time are important aspects; accuracy alone is not enough. This is since Optisort has a requirement that the algorithm takes about a tenth of a second to complete. At the same time, this requirement is not all too strict since improvement can be made later with faster processors, images of more suitable resolution and so on. However, time is still an important aspect and it is therefore fully possible that we regard some faster algorithm as better than a slower one, even though the later might have better accuracy.

4.2 Experiment 1: Image segmentation comparison

This experiment is meant to give a general overview of how different ways of segmenting an image (finding the objects contained in it) perform on the test set. The output of each experiment run was classified by hand.

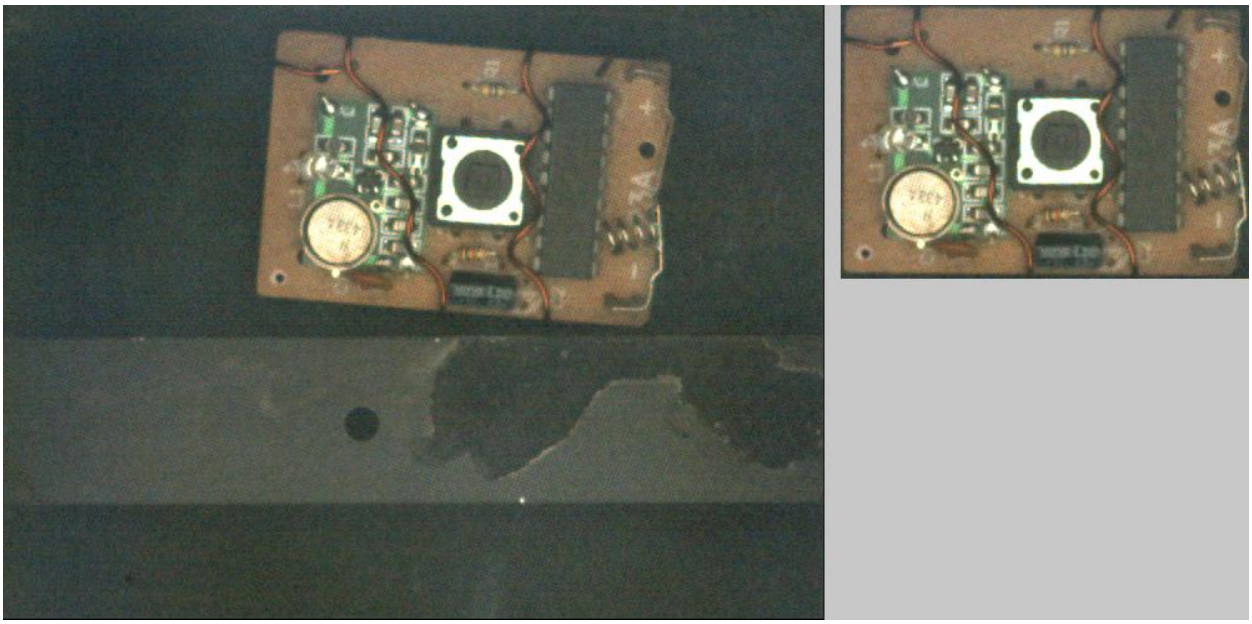

Figure 20: An image of the conveyor belt with an object on it to the left, and to the right is an image of the object being segmented in a correct way.

4.2.1 Algorithm descriptions

The main task for these algorithms is deciding whether a pixel belongs to the background or the foreground. This is done either directly or indirectly.

In general, the algorithms chosen can be divided into two categories. The first is algorithms where no knowledge based on previous images is used and thus focus on feature extraction only, such as finding contours in the original image. These algorithms generally try to extract features

based on just pixel value or by pixel neighborhoods. The second category is algorithms that use data of previous images in to perform segmentation. The simplest example is a pixel-by-pixel comparison between two objects. The background could be used as one of these images and the deviation of the given picture from this background could thus be used for background segmentation, separating the image into foreground and background sections.

The algorithms presented here generally have one step of the second category of the two ones described previously, followed by one step of the first category. It would show that it was more or less necessary to have background images sampled beforehand in order to get optimal results. This is since on the conveyor belt of the prototype, there was noise in the form of dirt, oil and iron bars that would be taken for being objects though not being desired, when having algorithms of the first category. In a realistic scenario, dealing with these problems should be quite necessary.

Once the background and foreground has been separated, usually in the form of a binary image, a segmentation of the first category can easily be used to go from separated foreground and background to a set of objects. To get a set with different objects and deciding their frame, mask and position, a most direct way has been chosen; the built in function for finding contours (Suzuki et al., 1985) and minimal bounding rectangles in OpenCV. It might be possible to optimize further for a small speed benefit.

The algorithms implemented here strongly depend on background subtraction. After the background subtraction, they all use the built in OpenCV function for finding the contours. As mentioned earlier in this report, background subtraction was required for the segmentation to work and was very much the sole challenge and the work behind the algorithms, since when a perfect background subtraction is carried out the rest is easy; the foreground is more or less segmented already. Therefore, the below descriptions are mostly about this part.

The first two algorithms below were low-level implementations while the remaining two algorithms used high level functions in OpenCV. The interesting aspect of including both kinds is that a comparison can also be made between low level implementation versus high level implementation.

4.2.1.1 4d histogram (Histogram)

One algorithm that was tested was the one described in section 3.5.1 based on background subtraction using histograms. In this implementation a four dimensional histogram was used. Three dimensions corresponded to the three color channels and the fourth dimension corresponded to the amount of pixels of some specific color found in the picture. By scaling down the intensity values, similar color values were grouped together so that the histogram did not have to store a separate value for each possible color intensity. The exact scaling might depend on the light conditions and other image qualities, and might thus need some adjustment for each specific case. Problems can arise if the histogram is scaled down too much, because then the grouping into different colors will be too coarse. Also, if the histogram is scaled down too little it will require more resources instead and be too sensitive to noise and color variation between different photos.

For the algorithm, a histogram is first created for the average background. Then, for each image, another histogram is created. The image histograms are compared to the background histogram, and each color group is either set to being part of the background or part of the foreground depending on the similarity in density. Each pixel in the given image can then be marked to foreground or background just by checking its color. The algorithm is thus sensitive to the fact that the input image should preferably have a wide spread of different color values, and ideally the background should not have colors all too similar to the objects. This might need some pre-processing based on the image qualities of the given situation. On the binary resulting image, the OpenCV algorithms for finding contours and minimum bounding rectangles are used.

4.2.1.2 Averaged background segmentation algorithm (BgDiff)

The pixel based approach was also used which is described in section 3.5.1. While the histogram approach is based on dividing the colors into foreground and background and then marking the pixels accordingly, this approach is a direct pixel-to-pixel comparison. To recap, for a set of background images, the average image is calculated. This results in an image where much noise has been removed. This can be seen as an approximation for the ideal image that would be captured if there was no noise. This image is compared to the set pixel by pixel in order to calculate the average- and maximum differences. This results in two images, the average noise level and the peak noise level. Note that these values are available on a per pixel basis. For each pixel, an approximation of the ideal value of the background, the average difference and peak difference are available. This concludes the necessary setup. For each new image, a pixel-wise comparison between the given pixel value and the approximated ideal background value with a tolerance based on the average noise level and a parameter times the peak level is performed. This is done to decide whether a pixel belongs to the foreground or the background. This algorithm is more sensitive to having the photographs taken from a fixed and stable position. On the binary resulting image, the OpenCV algorithms for finding contours and minimum bounding rectangles are used.

4.2.1.3 Gaussian segmentation algorithm (Gaussian)

The algorithm representing each pixel's intensity distribution as a Gaussian was used (described in section 3.5.2). Unlike the two algorithms described in the preceding sections, this is more of a high level algorithm. This means that a pre-implemented function is used from OpenCV´s library, so what we get is essentially a “black box” that makes the segmentation. The advantage with this is that it is easier to implement, of course, however the drawback is less control since nothing about the function can be changed except for some input parameters, however the implementation remains static.

4.2.1.4 MOG segmentation algorithm (MOG)

Another algorithm that was used was Mixture Of Gaussians (MOG) for background subtraction, described in section 3.5.2 (colors represented as a mixture of Gaussians). Basically the same

holds here as for the Gaussian segmentation; it is a high level algorithm which has its advantages and disadvantages.

4.2.2 Results

The below diagram shows the results when running the different segmentation algorithms on the test set. It is clear that the algorithms have different properties; for example, MOG made relatively many false detections (i.e. stated falsely that a part of the background was a foreground object), while still having a large amount of the objects being detected correctly. The histogram segmenter, on the other hand, missed relatively many objects compared to other algorithms.

The best performing algorithm was the pixel-wise background segmenter, with almost 90% of the objects being detected in a correct way. The Gaussian segmenter which did not miss a single object completely, but where on the other hand the majority of the detected objects were only captured partially in the segmentation, cannot be regarded as the best choice.

Overall there were not many merged detections, meaning that two objects close to each other are classified as a single object. This indicates that the algorithms are not very sensitive for close distances between objects. Also, MOG was the only algorithm to make false detections; the other algorithms did not falsely label the background as foreground.

The greatest difficulty seems to be connected with detecting the objects completely with a fine, exact bounding box that does not cut out any parts of the objects; failing with this was the most common source of errors. The main problem with only making a partial detection of some object is that when having the resulting image as input to the rotation algorithms, the fact that the object is only partially detected may alter the results considerably. As many of the rotation algorithms tend to output a result based on shapes and angles of the object, it may be a completely different result if just a little part of the object is cut off, being substituted for a sharp edge instead, whose angle will then contribute to some other resulting angle as output than what should have been if the whole object was segmented from the beginning.

Figure 21: An object being detected partially. The leftmost shows the input image, the second left image a processed version of it, and then the two rightmost images are the partial detections of the USB-stick.

Finally, time is an important aspect, and it comes clear that Gaussian and MOG segmentation takes too long time to complete. With better hardware a speed-up may be achieved, but still reaching the benchmark of less than 0.1 seconds seems unthinkable. It is important to mention that while the average time for BgDiff was 246,52 ms, a more optimized version outside the test environment managed to run at about 150 ms. Also, as a faster computer could have been used, it is believed that the algorithm has potential to run faster than 100 ms in a final version.

Algorithm	**Average speed**	**Correct detection**	**False detection**	**Missed detection**	**Partial detection**	**Merged detection**
BgDiff	246,52 ms	249	0	8	26	2
Gaussian	1106,60 ms	94	0	0	191	0
Histogram	382,17 ms	201	0	30	48	2
MOG	1626,33 ms	224	123	5	59	0

4.3 Experiment 2: Image rotation

The purpose of this experiment was to test how different algorithms perform on the following task: given a set of images of well segmented objects from the test set, rotate them in the sense that for each object, the rotation will be the same and there is no difference in the orientation regardless of which it had in the input. For instance, suppose a specific battery was photographed several times in a multitude of angles. The ideal is that after inputting the images one by one, all outputs are practically indistinguishable.

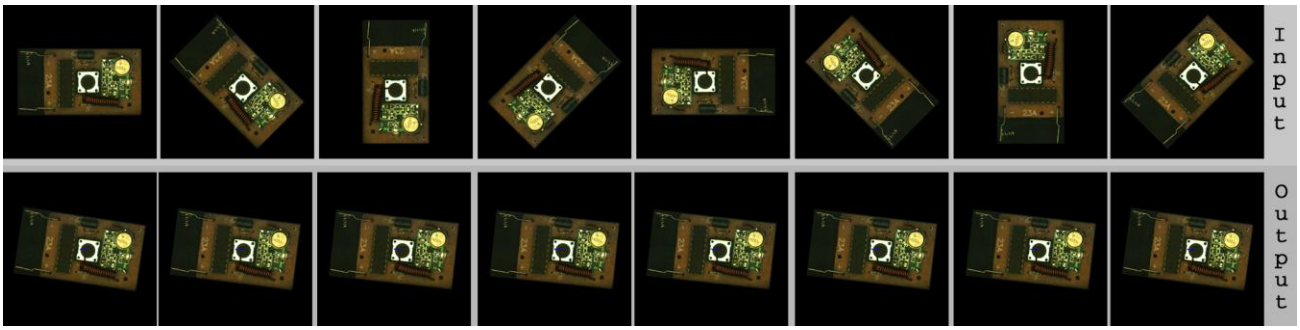

Figure 22: The upper part displays a series of input images to the rotation algorithm displaying a circuit card. The lower part shows the output images (using the Image Moments rotation algorithm) for the above respective input images. Here the algorithm succeeded very well since they look almost the same.

Each individual algorithm was tested one by one with a test set that was acquired when making the segmentation experiment; 10 different objects were tested, and images were used which represented correct segmentations of each object (average was 7 images for each object).

The main properties for testing in this experiment were speed and accuracy. The accuracy was simply measured as the variance of the rotation degree. To explain this further, we had several images of the same object in the test set. The goal is, as explained above, that each object will be rotated in the same way for all the pictures of it. Thus, accuracy is measured in terms of variance for each object and not for each individual image. The speed is measured as the average time of calculating the rotation for a certain object.

Something that should also be noted is that if nothing else is labeled, the test was done with pictures of the respective object taken on the conveyor belt of Optisort´s prototype. However, we have also included something called the Ideal Test Case. This means that for the given object, the background was black, and using Photoshop, replicated images of the exact same object were created with different rotation angles. The purpose of this was to investigate the theoretical ideal case, where the object is perfectly segmented and the light conditions do not change for the different images taken. This is probably hard to achieve with one hundred percent perfection in reality, but is something that was interesting to test anyway. This is since if an algorithm would work very well in this setting and not in the real setting, it would rather be hardware and environment posing problems than the algorithm itself. An alternative way of viewing it, of course, is that the algorithm might be too sensitive for changes, but in whichever case more conclusions could be drawn when including this test case.

4.3.1 Algorithm descriptions

The following algorithms mainly work in two different ways; working with the contour of an object or working with the surface (which is color or structure) of an object. A solution could also possibly be a combination of an algorithm based on contour analysis and an algorithm based on surface analysis. While the contour of an object is relatively easy to work with, symmetric shapes tend to cause ambiguity, such as with finding the point furthest from the center in a perfect circle; no unique solution exists here. Working with surfaces might give more opportunities to find a reference angle, but in the case of an object with a uniform surface, this could also fail. In the special case of an object that registers the image as a perfect circle with uniform surface, any angle as output by the algorithm will actually work. The hardest situations for these algorithms would then be the ones featuring objects that are very close to symmetric and have a minimal amount of surface features with direction. Several of the following algorithms passed through a pre-processing algorithm to handle the above ambiguity issue, namely an algorithm that rotated the image so that if a cross would divide the image into four equally large rectangular parts, the brightest part would be the upper left one, so that is a rotation of 0, 90, 180 or 270 degrees. This is not included in the algorithm descriptions but is instead mentioned here. Also, this pre-processing is obviously regarded as a part of the algorithm itself and the time it takes is included in the experiment results.

Something that is important as a notice is that the images were preprocessed before they were used as input to the algorithms. More specifically, Canny's - and Laplacian edge detection were used to skeletonize the image before it was used as input to the algorithms. This means that the algorithms work with the structure and the contour of the objects, and not the light nuances or colors. The reason for this is that the light conditions varied between the images depending on how close the object was to the lamp, and there was also the fact that different reflections could be depending on how the object was turned, and this was undesired since consistent results could not be acquired then. So because of this, the pre-processing was done.

Six algorithms were tested for the rotation task, and they are presented below. Both high level approaches based on pre-implemented functions (POI and Hough Lines) and low level approaches (HOG) were used. We used optimized algorithms in OpenCV in many cases where it was possible (for example, in order to perform the Fourier transform) but mostly they were combined and complemented with more thorough low level implementing in order to finalize the algorithm.

Algorithm 1:

One algorithm was based on Hough Lines. The Hough Lines algorithm was executed for the image, and then a resultant vector was created from the lines in the output. The rotation was simply the negated degree of this angle.

Algorithm 2:

Next, an algorithm was used based on Points Of Interest (POI) detection. This algorithm would use the POI detection algorithm in OpenCV (a high level algorithm based on SURF detection) to find a number of points of interest. A midpoint was then created from these points, and a vector was made by having the mid image point and the mid POI point as endpoints. So this is basically what was also described in section 3.4.

Algorithm 3:

Another algorithm which was also based on a high level function in OpenCV was the algorithm based on Central Image Moment. This is the concept described in section 3.5.6; a point of mean image intensity was calculated on the image, and again a vector made from these points would stand for the resulting rotation angle (negated).

Algorithm 4:

The fourth algorithm used the idea from section 3.3.1; it calculated the Fourier diagram of the image, and then using Bresenham´s Line algorithm, the line of most intensity was found in the diagram, and this angle would represent the angle for the image.

Algorithm 5:

The fifth algorithm was based on orientation gradients (so it was inspired by HOG, described in section 3.2.2.3), and calculated a resultant vector by summing the gradient vectors. This would then represent the angle of the image. The algorithm is referred to as the HOG algorithm though there is a slight difference from the standard approach, namely that no histograms were used; calculations were made instead by getting the gradient orientation pixel by pixel and filtering out weak gradients.

Algorithm 6:

The sixth algorithm simply used the bounding box acquired by the segmentation algorithm when segmenting the objects (the pixel-wise segmenter was used for this, as stated earlier). The angle for the bounding box was used to rotate the box back. At this stage, there is a rectangular bounding box which could be turned 0, 90, 180 or 270 degrees to maintain the property of the sides being parallel to the computer screen. First, the longest sides are turned so they represent the width of the box. Next and finally, there is a choice of rotating the bounding box either 0 or 180 degrees. This is decided by the Image Moments algorithm; if the direction of the resulting vector is to the right (between -90 and 90 degrees), the image is rotated 180 degrees. This means that for long objects, there will be no problem in choosing which side will represent the width (which poses a problem for quadratic objects, though); only the choice of the final rotation of either 0 or 180 degrees could be problematic if the light conditions are very homogeneous in the image.

4.3.2 Results

Here follows the results from the experiment. Note that the experiment was done with three different edge detection methods for each rotation algorithm, and the result is presented for each of them. The reason for testing against three different edge detection methods was that features for the individual detectors would not bias the result; they all worked a little different, so testing against all three gave a better overview. Also, the results are showed when no edge detection is used as pre-processing.

Test measuring average times for running algorithm 6

The time for segmentation is included here since the bounding box is what gives the rotation angle for this algorithm, and segmenting the objects is thus required. The segmentation should really be separate from the rotation and not included if segmentation and rotation is run in sequence, however this might not always be the case; rotation could be run alone. Therefore the time for segmentation is included; in that case it is part of this algorithm.

Here “Both” means time in total including the rotating of the images,
“Seg” means that just the segmentation algorithm was used and
“Rot” means correspondingly that only the rotation algorithm was used.

Testing average time for rotation

Algorithm	Average speed
Fourier	780 ms
HOG	44 ms
Hougline	210 ms
Moments	0.98 ms
POI	220 ms

Tests of performance of rotation algorithms with different edge detectors as pre-processing

Here the results are compared for the different rotation algorithms. As noted earlier, different edge detectors are compared, working as image pre-processing tools.

For each object in the tests, green color is used to mark the best result and red the worst result.

Fourier Rotation - Standard Deviation

	Canny edge detection	Laplacian edge detection	Sobel edge detection	No pre-processing (plain)
Battery	62,40	29,43	36,79	79,79
Long green circuit card	41,71	79,59	85,62	75,59
Green circuit card	80,55	76,96	77,62	80,13
Christmas tree lightbulb	63,32	99,54	89,95	99,83
Keychain (electronic)	43,01	63,74	30,39	40,73
Orange circuit card	49,77	23,48	19,86	51,77
Orange irregular shaped circuit card	9,68	58,83	58,95	77,60
Spotlight lamp	66,38	65,14	72,76	56,84
Oven lamp	46,45	48,61	36,00	40,39
White lamp	44,62	31,57	34,42	43,53

HOG Rotation - Standard Deviation

	Canny edge detection	Laplacian edge detection	Sobel edge detection	No pre-processing (plain)
Battery	82,69	57,33	67,07	107,68
Long green circuit card	65,11	60,57	59,29	81,44
Green circuit card	97,84	51,07	44,54	82,81
Christmas tree lightbulb	83,25	67,31	92,73	99,06
Keychain (electronic)	95,30	96,61	62,81	103,96
Orange circuit card	82,27	81,50	91,42	92,03
Orange irregular shaped circuit card	95,37	74,67	88,26	87,32
Spotlight lamp	97,88	76,15	74,81	87,36
Oven lamp	91,44	78,96	80,97	44,82
White lamp	30,95	51,18	13,06	29,82

Houghline Rotation - Standard Deviation

	Canny edge detection	Laplacian edge detection	Sobel edge detection	No pre-processing (plain)
Battery	61,80	86,45	91,88	56,64
Long green circuit card	62,57	42,73	48,38	69,70
Green circuit card	56,19	93,59	97,91	95,21
Christmas tree lightbulb	103,85	94,14	70,95	82,83
Keychain (electronic)	96,52	97,70	89,56	60,01
Orange circuit card	80,79	86,92	84,23	84,50
Orange irregular shaped circuit card	91,10	78,60	87,09	83,26
Spotlight lamp	80,64	85,50	61,95	84,51
Oven lamp	65,57	94,48	105,03	63,90
White lamp	59,74	71,65	78,45	112,52

Moment Rotation - Standard Deviation

	Canny edge detection	Laplacian edge detection	Sobel edge detection	No pre-processing (plain)
Battery	99,19	14,65	19,47	16,26
Long green circuit card	46,70	62,69	52,39	68,27
Green circuit card	71,74	49,57	45,49	66,40
Christmas tree lightbulb	39,83	68,91	60,28	74,93
Keychain (electronic)	51,67	8,18	5,10	13,12
Orange circuit	33,68	16,67	14,03	48,55
Orange irregular shaped circuit card	55,52	71,92	66,34	55,99
Spotlight lamp	99,18	34,07	40,19	50,91
Oven lamp	9,76	26,17	45,68	70,76
White lamp	24,00	1,81	2,73	17,33

POI Rotation - Standard Deviation

	Canny edge detection	Laplacian edge detection	Sobel edge detection	No pre-processing (plain)
Battery	74,39	14,60	20,92	82,59
Long green circuit card	48,13	62,50	79,76	73,65
Green circuit card	59,67	70,84	66,16	70,72
Christmas tree lightbulb	55,28	98,74	103,87	99,17
Keychain (electronic)	18,16	47,63	33,98	28,82
Orange circuit card	41,96	48,18	32,51	58,59
Orange irregular shaped circuit card	25,04	50,45	50,67	94,65
Spotlight lamp	102,47	60,82	60,37	67,48
Oven lamp	19,23	36,13	44,35	49,74
White lamp	8,20	15,24	23,31	15,81

Theoretically possible results by using a single image rotated in turns of 45 degrees using Photoshop for a total of 8 images per test set.

Here the result is shown for the Moments algorithm with the theoretical ideal test images as input. It turns out that in this setting, it performs very well.

Moments

	Standard deviation
rectangular circuit card	0,90
oval circuit card	1,06

Results of algorithm 6 using Sobel pre-processing (both results as they are and if 180 degrees rotation is allowed as in using two reference images for each object)

Here the bounding box based rotation algorithm is tested with Sobel edge detection as pre-processing for the images, and it shows that when allowing images being rotated 180 degrees (in other words, it does not matter if the image is flipped over), the algorithm performs very well.

Boxwise Rotation - Standard Deviation

	Standard	**Allowing 180 degrees rotation**
Battery	2,68	2,58
Long green circuit card	67,71	1,05
Green circuit card	0,36	0,31
Christmas tree lightbulb	63,55	1,11
Keychain (electronic)	4,70	4,76
Orange circuit card	0,26	0,22
Orange irregular shaped circuit card	112,36	9,79
Spotlight lamp	36,96	36,84
Oven lamp	64,72	5,79
White lamp	0,96	1,19

4.4 Experiment 3: Testing how the segmentation and rotation algorithms can enhance the performance of an object classifier

This experiment was done using an implementation of the object classifier described in section 3.6. The purpose of this experiment is to tie together the previous algorithms into a single applied program as a pre-processing stage to an object classifier, ideally showing that the previous results in this thesis are of benefit for real applications.

The experiment was done in the following way: using the best rotation algorithms from the previous experiment, images were sent into it that had been segmented by the pixel-wise background segmentation algorithm (which had achieved the best results among the segmentation algorithms), whose input in turn came from the test set. So in other words, images from the test sets were both segmented and rotated, and then sent into the image classifier. This was done for 20 different objects, and since the classifier would always output exactly one image from the template bank which was regarded as the one resembling the input most, the score system was completely binary for each object; it was either the right object that was suggested, or it was the wrong one.

Since the segmentation algorithm chosen here worked very well, the main interest was comparing different rotation algorithms. This was what would be the main issue in the end; assuming every object was segmented in a perfect way, rotating each object in the same way for all images in the sense that they all look more or less the same, should intuitively give good results for the classifier. The two best rotation algorithms had shown to be the one using the angle for the bounding box and having Image Moments as a secondary algorithm, and the Image Moments algorithm. These were the one used in this experiment for comparison.

4.4.1 Results

The results were the following with the following algorithms as pre-processing stages before classification:

Image rotation with bounding box angle: 12 out of 20 correctly classified images
Image rotation with image moments: 3 out of 20 correctly classified images

The results will be discussed in the Discussion section, which follows below.

5 Discussion

In this chapter, it is going to be discussed not only what algorithms worked best, but also what underlying features are important for the different tasks. Four algorithms were tested in the segmentation experiment, and five algorithms in the rotation experiment. The results could clearly show which algorithms worked best, though at the same time most algorithms had different strengths and weaknesses.

5.1 Segmentation

It turned out that background subtraction done by having an averaged background and classifying pixels in the input image with intensity above a certain threshold difference as foreground, would be the best approach for segmenting objects (the algorithm described in section 3.5.1). It appears that this is a rather straightforward approach, which works well since the background does not change very much for different images (the conveyor belt has moved, however the color nuances are still almost the same).

The histogram segmentation also works rather well, however it is a little different while still being similar, which has some consequences. The most important difference is that it does not make operations pixel-wise when deciding which pixels will be foreground and which will be background; the overall difference is compared for certain nuances. The intuitive weakness that could arise is when an object contains nuances from the background, and the comparison shows that this nuance in the input image does not differ from the background with an amount which is large enough. Then partial detections could be the resulting error, since the pixels were regarded as background. Missed detections could also be made in this way, and both kinds of errors were seen in the experiments.

The MOG and Gaussian segmentation algorithms are harder to analyze directly since they are more high level algorithms, using pre-implemented functions from OpenCV. However it becomes clear from the results that the higher level (abstraction level, to be specific) the worse the performance of the algorithm becomes. Also, with higher abstraction level, it also followed that the time it took to perform the last two algorithms increased to a degree such that it was unacceptable (more than 1 second on average per object).

It seems that the pixel-wise background subtraction worked best just because it does not complicate the task more than necessary, and all that really is needed is a pixel-wise comparison; it is a straightforward and very controlled way of segmenting the objects.

5.2 Rotation

It seems from the results that the best algorithm seems to be the sixth algorithm, which uses the bounding box of the segmentation algorithm. The main strength of this algorithm is the robustness of it compared to the Image Moments algorithm for example; the Image Moments algorithm worked well on the ideal test set with the exact same light settings and proportions but would encounter some problems when these properties were changed just a little for the real test set. Additionally, the bounding box algorithm showed to be relatively fast. There is a slight weakness though; namely handling images with a very homogeneous intensity distribution and having close to quadratic shape, since these are ambiguous in terms of orientation. It should be mentioned however, that even though some objects images in the experiment failed in the sense that there were 90- or 180 degree differences between them, these kinds of errors are not very serious from a practical point of view. While they clearly make the results in the table look worse, it can be imagined that in a practical setting with an image classifier for example, templates of a certain object could be stored with 0, 90, 180 and 270 degree rotations to solve

this problem. Thus, only irregular variations will pose a real problem where the error degree cannot be known beforehand, and these were not seen very much for this algorithm. Therefore, it can be stated that the bounding box based rotation algorithm worked best. It should also be noted that the other algorithms did not encounter the above problem in a noticeable way.

Among the other algorithms the Image Moments seemed to be the best one, with a very low running time and relatively accurate results. This was also an algorithm that would have very accurate results on the ideal test case with same proportions and light settings; in this ideal case, the variance was only 1 or 2 degrees. However for this algorithm, as well as for the remaining ones, the problem was that it was sensitive for light variations. Reflections in the object seemed to alter the resulting angle rather much. Thus it can be stated that while this algorithm works very well theoretically in a perfect setting with a very even light distribution, some problems can be encountered in a practical setting.

Another algorithm that gained decent results was the one based on Fourier analysis. The same also holds here; it performs very well theoretically, but has a weakness in the sense that the results might be inconsistent when the settings change slightly. Another weakness is the running time, which is comparatively long.

The Hough Lines algorithm did not work very well, and the problem seems to lie in the fact that it is a very high level algorithm whose output can change radically when the input is tweaked just a little, and while there are parameters possible to adjust somewhat, it is hard to gain good control of it.

All in all, it can be stated that object shape is a much more stable guideline for choosing the angle than intensity distribution or color, which can vary for different images though it is the same object with the same background. However it follows then also that in general, objects with more of a symmetrical shape are harder than unevenly shaped objects for the algorithms. Other things that could be of influence are the various image transformations and processing methods; these are operations that could make individual pixels vary for the same object in the same place for two different images, and thus make algorithms based on color or pixel intensity distributions more sensitive, so they should be applied in a careful, controlled manner.

5.3 Image classification

It would appear that when having well segmented images, rotating them with the algorithm based on the bounding box would be a helpful way of pre-processing for object classification. The reason why this works rather well (12 of 20 correct classifications) seems to be not only because the algorithm rotates the images of objects in a consistent way, but also because the angles are chosen in an intuitive way that makes the classification task easier. For example, practically all rectangular objects will be rotated in a way that they are oriented either horizontally or vertically. This is very practical since by knowing this, errors are almost only made with 90 or 180 degrees in some direction. Thus one can for instance store four pictures for each object as templates; each rotated 0, 90, 180 and 270 degrees respectively, in order to

minimize this kind of error. Then the rest is up to the image classifier; the images are rotated in a consistent way that minimizes the difficulties for it.

It seemed that the Image Moments gained considerably worse results here (3 of 20 correct classifications). Most probably, an important aspect to notice is that this did not do what was described above; it did not necessarily rotate the objects in a way so they became horizontal or vertical. The problem with this is that differences occurs naturally, and while we could correlate them easily with the previous algorithm (rotate 90, 180 or 270 degrees), this is not done as easily here as the amount the angle is wrong can be practically random. Furthermore, if we have a rectangular object and it is not rotated horizontally or vertically, the image it is contained in will have an additional amount of unnecessary black background. This will practically work as noise as it distorts the shape of the object when sending it into the classifier. Therefore it can be said that it is not only important that the objects are rotated in a consistent way; how the angle is chosen is also of importance, i.e. rectangular objects should be vertical or horizontal.

It is important to note that the results here are relative – the classifier is very simple, so 12 of 20 correct classifications is good enough to show that the pre-processing with segmentation and rotation can be helpful.

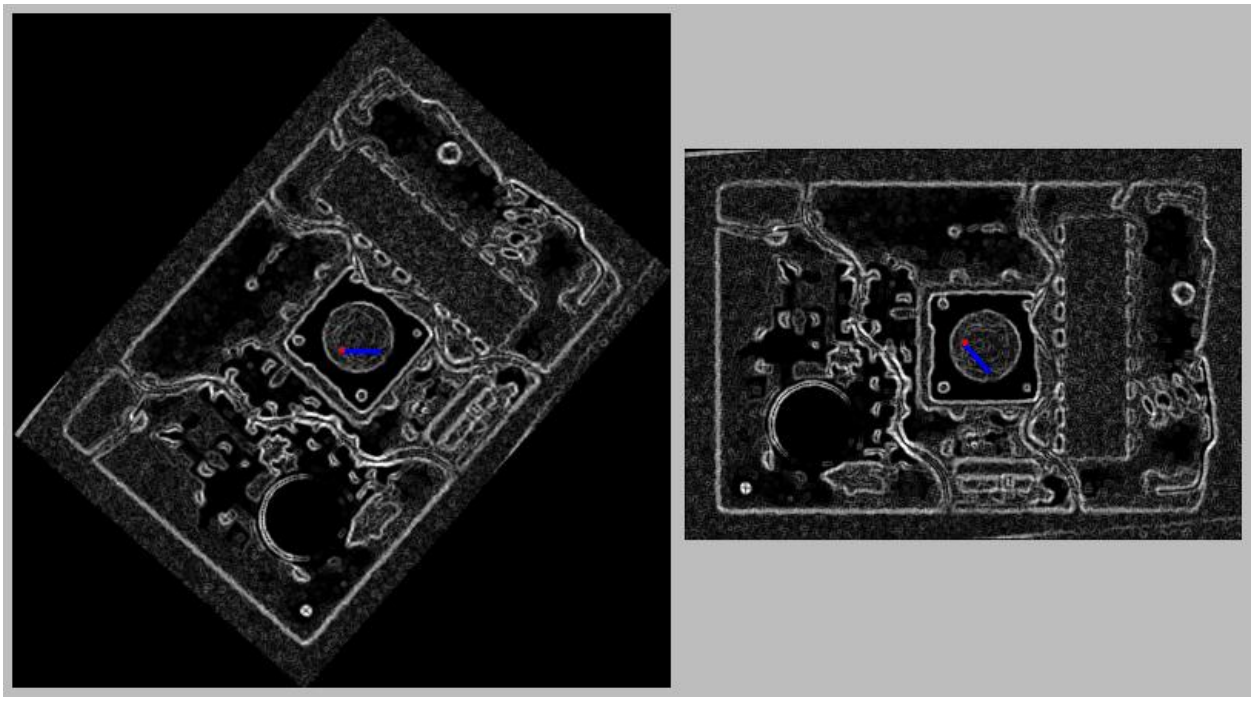

Figure 23: To the right the object is horizontal and no unnecessary background is sent into the classifier. On the image to the left, on the contrary, the object has a skew angle which adds the black background, and the result is that more pixels have to be compared by the classifier (since a saved image has to be rectangular).

6. Further work

There are several aspects that could be of interest for further work. The hardware is an aspect that is of interest; for segmentation, having a laser that could scan the height of the locations in the image would enable another kind of algorithm for background subtraction. Several versions of this has already been tried, as stated in the "Earlier Work" section, however one possibility could be to combine height scanning with pixel-wise intensity comparison. Different hardware could also be used to take the test pictures, possibly enabling different results; having a neutral light setting that is very even and causes minimal reflection could improve the results of the rotation algorithms (as edge detection maybe would not have to be used). Also, another kind of conveyor belt of larger size could be used, enabling the possibility of having larger objects.

Another aspect that would be of interest is having other frameworks than OpenCV; as an example, one could make own implementations of the Fourier Transform as an attempt to speed it up. So other ways of implementing the various algorithms in this thesis would be an interesting way of trying to get a faster running time. Furthermore, finding the ideal resolution for the images could also be interesting as a way to increase the speed while preserving as much precision as possible.

The classifier here is, as stated earlier, a very simple one that was chosen because of time constraints. Using a more complex classifier based on Neural Networks, for instance, would probably be much more effective. Something that would be of high interest would be to test having the algorithms here as a pre-processing stage for Optisort´s Object Classifier, and comparing the performance with and without this. However it can only be assumed that since effective algorithms has been found for pre-processing images before classification, they can be used in some way to enhance their results.

In the segmenter based on background difference separation, the average background has mainly been used. For the images taken by the object scanner at Optisort this worked fine. In other circumstances such as when using a conveyor belt with strong stains and tears, it might be beneficial to treat the background differently. Photographing the whole belt part by part and synchronizing the current photo with the corresponding photo of the background conveyor belt might be a solution.

Another remark is that the current approach is based on taking single photos, however a different approach could be to use video input. By tracking objects in a video sequence, there is less risk of having the same object analyzed multiple times or the picture taken a bit too early or too late and thus leaving part of the object outside the frame (making it harder to recognize a partial object). The approach used in this report should still be valid. Still, the possible precision benefits should be properly weighed against the extra costs in processing introduced, and it might be possible that video recognition could add effectiveness if only these added costs seem reasonable.

7. Conclusions

It has been shown that Foreground Segmentation can be done effectively in an environment where the background is partly static and partly dynamic. The method for this can be chosen as calculating an averaged background image based on a multitude of different background images, and then making a pixel-wise threshold selection of pixels probable of being foreground because of the intensity difference, and finally marking dense areas of such pixels as foreground objects.

Calculating angles for an object in the above setting in a way such that for every image of it, rotating them with the respective angles calculated will result in images looking almost the same can be done in the following way. Use a good image segmentation algorithm (like the one described previously) and rotate the objects with the angles of the bounding boxes. Then, use simple rules like having the longest side representing the width and the brightest side up, in order to avoid mirrored images.

To answer the question if pre-processing can be helpful for the algorithms presented in this thesis it seems that for the rotation task, edge detection might be more or less necessary if the light conditions are uneven.

Finally, it has been shown that using effective algorithms for segmentation and rotation can be helpful for the task of object recognition. It is possible to implement pre-processing algorithms that segments and rotates objects in a sense that they are likely to match templates previously stored in the classifier of the same object. This could then be a possibility of making object recognition easier.

References

Ahmed, Ward, (2002), *A Rotation Invariant Rule-Based Thinning Algorithm for Character Recognition*, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 24, NO. 12

Arifin, Asano, (2006), *Image segmentation by histogram thresholding using hierarchical cluster analysis*, Pattern Recognition Letters 27 1515-1521

Bay, Ess, Tuytelaars, Gool, (2008), *SURF: Speeded Up Robust Features*, Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346--359

Bebis, (2003), *Edge detection*, Available: < http://www.cse.unr.edu/~bebis/CS791E/Notes/EdgeDetection.pdf >, [2012-03-30]

Bhadauria, Dewal, (2010), *Comparison of Edge Detection Techniques on Noisy Abnormal Lung CT Image before and after Using Morphological Filter*, Roorkee: IIT Roorkee

Bovik (ed), Alan C., (2005), *Handbook of image and video processing*, [Books24x7 version] Available from http://common.books24x7.com/toc.aspx?bookid=25360.

Bresenham, (1965), *Algorithm for computer control of a digital plotter*, *IBM Systems Journal*, Vol. 4, No.1, pp. 25–30

Chan, Tony F. & Shen, Jianhong, (2005), *Image processing and analysis: variational, pde, wavelet, and stochastic methods*, [Books24x7 version] Available from http://common.books24x7.com/toc.aspx?bookid=23051.

Dalal, Triggs, (2005), *Histograms of Oriented Gradients for Human Detection*, INRIA Rh.one-Alps, 655 avenue de l'Europe, Montbonnot 38334, France

Doringa, Mihai, (2010), *Comparison of Two Image Segmentation Algorithms*, Second International Conferences on Advances in Multimedia

Duda, Hart, (1972), *Use of the Hough Transformation to Detect Lines and Curves in Pictures*, Comm. ACM, Vol. 15, p. 11–15

E. Dop, (1999), *Multi-sensor object recognition: The case of electronics recycling*, University of Twente

Fisher, Perkins, Walker, Wolfart, (2003), *A to Z of Image processing*, Available: < http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm, http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm, http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm, http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm, http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>, [2012-03-23]

Gonzalez, Woods, (2002), *Digital Image processing Second Edition*, New Jersey: Prentice-Hall Inc. p. 91-94, 147-219, 222, 523-527, 569-570

Gould, Arfvidsson, Kaehler, Sapp, Messner,
Bradski, Baumstarck, Chung, Ng, (2007), *Peripheral-Foveal Vision for Real-time Object Recognition and Tracking in Video*, Stanford University

Horprasert, Harwood, Davis, (1999), *A Statistical Approach for Real-time Robust Background Subtraction and Shadow Detection*, University of Maryland

Intelligent Perception, (2010), *Centroid*, Available: < http://inperc.com/wiki/index.php?title=Centroid >, [2012-03-30]

KaewTraKulPong P, Bowden R, (2001), *An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection*, Video based surveillance systems: Computer Vision and Distributed Processing, Kluwer Academic Publishers.

Lee, Pun, (2002), Rotation and scale invariant wavelet feature for content-based texture image retrieval, Journal of the American Society for Information Science and Technology Volume 54, Issue 1, pages 68–80

Li, Huang, Yu-Hua Gu, Qi Tian, (2004), *Statistical Modeling of Complex Backgrounds for Foreground Object Detection*, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 13, NO. 11, 1459-1471

Makadia, Daniilidis, (2003), *Direct 3D-Rotation Estimation from Spherical Images* via a generalized shift theorem, University of Pennsylvania

Mamoru, Katsuhisa, (2000), *Investigation of Image Sensing Technique for Recyclable Waste Sorting*, Japan: Mekatoronikusu Koenkai Koen Ronbunshu

Marshall, (2001), *The Discrete Fourier Transform (DFT)*, Available: < http://www.cs.cf.ac.uk/Dave/Multimedia/node228.html >, [2012-05-25]

Mattonea, Campagiornia, Galatib, (2000), *Sorting of items on a moving conveyor belt. Part 1: a technique for detecting and classifying objects*, Robotics and Computer-Integrated Manufacturing 16 (2), pp. 73-80.

Nadernejad, Sharifzadeh, (2008), *Edge Detection techniques: Evaluations and Comparisons*, Babol: Applied Mathematical Sciences, Vol. 2, 2008, no. 31, 1507 – 1520

Nandini, Srikanta, Kumar, (2008), *Estimation of Skew Angle in Binary Document Images Using Hough Transform*, World Academy of Science, Engineering and Technology 42 2008

Natural Resources Canada, (2008), *Image Transformations*, Available: < http://www.nrcan.gc.ca/earth-sciences/geography-boundary/remote-sensing/fundamentals/2062 >, [2012-05-14]

opencv dev team, (2011), *Sobel derivatives, Back projection,* Available: < http://opencv.itseez.com/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html#sobel-derivatives, http://opencv.itseez.com/doc/tutorials/imgproc/histograms/back_projection/back_projection.html #back-projection >, [2012-03-23]

Owens, (1997), *Fourier transform theory*, Available: < http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT4/node2.html >, [2012-05-25]

Palágyi, *Skeletonization,* Available: < http://www.inf.u-szeged.hu/~palagyi/skel/skel.html#Thinning >, [2012-03-30]

Pinto, Baikerikar, Sridhar, Deshmukh, Kavthekar, Mudhliyar, (2010), *Template Matching Using Sum and Difference in Pixel Intensities*, The 3rd International Conference on Machine Vision

Qiu, (2001), *Some recent developments on edge detection and Image Reconstruction based on local smoothing and Nonparametric Regression*, Minneapolis: University of Minnesota

Qureshi, Shehrzad, (2005), *Embedded image processing on the tms320c6000 dsp: examples in code composer studio and matlab*, [Books24x7 version] Available from http://common.books24x7.com/toc.aspx?bookid=30945.

Saeed, Tabedzki, Rybnik, Adamski, (2010), *Km3: A universal algorithm for image skeletonization and a review of thinning techniques*, Applied Mathematics and Computer Science 20(2): 317-335

Sezgin, Sankur, (2004), *Survey over image thresholding techniques and quantitative performance evaluation*, Journal of Electronic Imaging 13(1), 146–165

Shapiro, Stockman, (2001), *Computer vision*, Upper Saddle River: Prentice Hall

Smith, (1997), *The Scientist and Engineer's guide to Digital Signal Processing,* California: California Technical Pub

Song Ho, (2006), *Histogram*, Available: < http://www.songho.ca/dsp/histogram/histogram.html >, [2012-03-22]

Spring, Russ, Turchetta, Parry-Hill, Long, Fellers, Davidson (2007), *Introduction to Digital Imaging in Microscopy,* Available: < http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/kernelmaskoperation/ >, [2012-03-22]

Sundararajan, (2001), *Discrete Fourier Transform: Theory, Algorithms and Applications*, Singapore: World Scientific Publishing Co. Ptc. Ltd

Suzuki, Abe, (1985), *Topological Structural Analysis of Digitized Binary Images by Border Following*, COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 30, 32-46

Tkalcic, Tasic (2005), *Colour spaces - perceptual, historical and applicational background*, Ljubljana: Faculty of electrical engineering

Turkel, (2011), *Hough Transform*, Available: < http://www.math.tau.ac.il/~turkel/notes/HoughTransform.pdf >, [2012-03-30]

Utkarsh, (2011), *Image moments*, Available: < http://www.aishack.in/2011/06/image-moments/ >, [2012-03-30]

Vincent, Folorunso, (2009), *A descriptive algorithm for Sobel Edge Detection*, Information Science IT education

Wikimedia Commons, (2009), *Example density of the mixture of three gaussian random variables.*, Available: http://commons.wikimedia.org/wiki/File:Gaussian-mixture-example.png >, [2012-05-17]

Wikimedia Commons, (2010), *HSL-HSV Models b*, Available: < http://upload.wikimedia.org/wikipedia/commons/1/16/Hsl-hsv_models_b.svg >, [2012-05-15]

Wu, Qiang & Merchant, Fatima & Castleman, Kenneth, (2008), *Microscope image processing*, [Books24x7 version] Available from http://common.books24x7.com/toc.aspx?bookid=28062.

Young, Gerbrands, Van Vliet (1998), *Fundamentals of Image Processing*, Deen Haag: CIP-Data Koninklijke Bibliotheek.