João Cachulo Soares

Valve and Steam Trap Component Recognition Using Machine Vision in an Industrial Application

September 2016

Universidade de Coimbra

FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE DE COIMBRA

Valve and Steam Trap Component Recognition Using Machine Vision in an Industrial Application

João Cachulo Soares

Coimbra, September 2016

Valve and Steam Trap Component Recognition Using Machine Vision in an Industrial Application

Supervisor:

Dr. Cristiano Premebida

Co-Supervisor:

Prof. Urbano José Carreira Nunes

Jury:

Prof. Urbano José Carreira Nunes

Prof. Jorge Manuel Moreira de Campos Pereira Batista

Prof. Vítor Manuel Mendes da Silva

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, September 2016

Acknowledgements

I would firstly like to thank my beloved family: parents, brother and sister for their continuous support throughout my time in college and life in general. I'd also like to express my sincere gratitude to my supervisor Dr. Cristiano Premebida for his huge patience, guidance and support. I also thank Professor Urbano Nunes and ISR for hosting me and providing the necessary resources and conditions that allowed me to accomplish my goals. I also thank my friends and colleagues in the Institute of Systems and Robotics as well as all the others who by one way or another interfered positively in my life. Last but not least I'd like to thank my girlfriend for always being there for me.

Resumo

No presente trabalho é apresentada uma solução para um sistema autónomo de identificação/reconhecimento capaz de classificar componentes de válvulas e purgadores, numa aplicação industrial de pintura, usando reconhecimento supervisionado de padrões baseado em visão. O sistema de visão aqui proposto tem por objectivo servir de base para uma solução a ser instalada numa unidade fabril de uma empresa especializada no fabrico de equipamentos para vapor, por forma a complementar a modernização e e automação do processo. Este processo passaria a contar como robôs para proceder à pintura dos produtos ao invés de pessoas, utilizando programas específicos chamados de acordo com o resultado do processo de identificação do produto, realizado à priori.

Começou-se por criar um conjunto de dados que incluiu o grupo dos produtos mais produzidos/vendidos pela empresa, recolhendo imagens num setup semelhante àquele que poderíamos montar no ambiente industrial. O passo seguinte consistiu no pré-processamento das imagens extraídas. De seguida são aplicadas técnicas de processamento de imagem para o tratamento e binarização das imagens. Nesta etapa é ainda desenvolvido um algoritmo para a remoção das pinças que penduram as peças em posição para pintura. Neste momento estamos na presença de imagens binárias com *blobs* que representa exclusivamente os produtos. O passo seguinte consistiu na implementação de dois métodos de extração de características das imagens. O primeiro método é baseado na extração características da forma dos *blobs*, seguido de uma implementação de um descriptor HOG. Ambas as técnicas são posteriormente usadas nas imagens resultantes do pré-processamento, sendo que as características extraídas são utilizada para treinar um classificador discriminativo e generativo, respetivamente um SVM (máquina de vectores de suporte) para classificação de múltiplas classes e um NBC (classificador bayesiano ingênuo).

No que diz respeito aos resultados de classificação, o SVM provou ser a melhor solução em termos de desempenho, velocidade e robustez quando comparado com o NBC. Relativamente à escolha entre as features geométricas baseadas em formas e as features extraídas ao utilizar o descritor HOG, concluiu-se que as primeiras mostraram melhor resultados no que diz respeito ao reconhecimento de maior número de imagens, mostrando precisões de 100% para toda a gama de *thresholds*. Os resultados para a revocação foram igualmente elevados, neste caso para *thresholds* abaixo dos $0.65 - 0.70$.

Palavras Chave: Reconhecimento Supervisionado de Padrões, Sistema de Visão Industrial, Classificação de Componentes de Válvulas e Purgadores

Abstract

In this work an autonomous identification/recognition system capable of classifying valve and steam trap components in an industrial painting application was implemented, using vision-based supervised pattern recognition. The proposed vision system has the main objective of being a foundation for a solution to be installed in the manufacturing facilities of a company specialized in steam equipment, in order to complement the modernization and automation of the process. The process would rely on robots instead of human beings, using specific programs which would be called depending on a prior product identification result.

The first step corresponds to the creation of a dataset with a group of the best-selling/most produced products, grabbing frames from a image acquisition scenario similar to the one possibly built in the industrial environment. The following step consists in pre-processing, where image processing techniques are introduced to threshold and treat the images as well as removing the claw that holds the products in position for painting. At this point the image contains a blob that exclusively represents the products. The following step consists in the implementation of two feature extraction methods. Firstly blob features based on shape and overall geometric characteristics, followed by a HOG implementation. Both feature extraction techniques are then used on the post-processing images and are trained on a discriminative and generative classifier, respectively a multiclass Support Vector Machine (SVM) and Naive Bayes classifier (NBC).

In terms of classification results, the SVM proved to be the best solution in terms of performance, speed, and robustness, outclassing the NBC. Regarding the choice between blob features or HOG features, it was concluded that the blob features would do a better job in describing the objects, showing results with 100% precision for all possible threshold values, and recalls equally high for thresholds below $0.65 - 0.70$.

Keywords: Supervised Pattern Recognition, Machine Learning, Valve and Steam Trap Component Classification.

"O homem comum é exigente com os outros; o homem superior é exigente consigo mesmo."

— Aurélio, Marco

Contents

List of Acronyms

AUC	Area Under the Curve
ANN	Artifical Neural Network
ATPS	Automatic Trajectory Planning System
CAD	Computer Aided Design
FS	Feature Selector
HOG	Histograms of Oriented Gradients
ISR	Institute of Systems and Robotics
NBC	Naive Bayes Classifier
PCA	Principal Component Analysis
RAM	Random Access Memory
RFID	Radio Frequency Identification
ROI	Region of Interest
SIFT	Scale Invariant Feature Analysis
SURF	Speeded Up Robust Features
SVM	Support Vector Machine

List of Figures

List of Tables

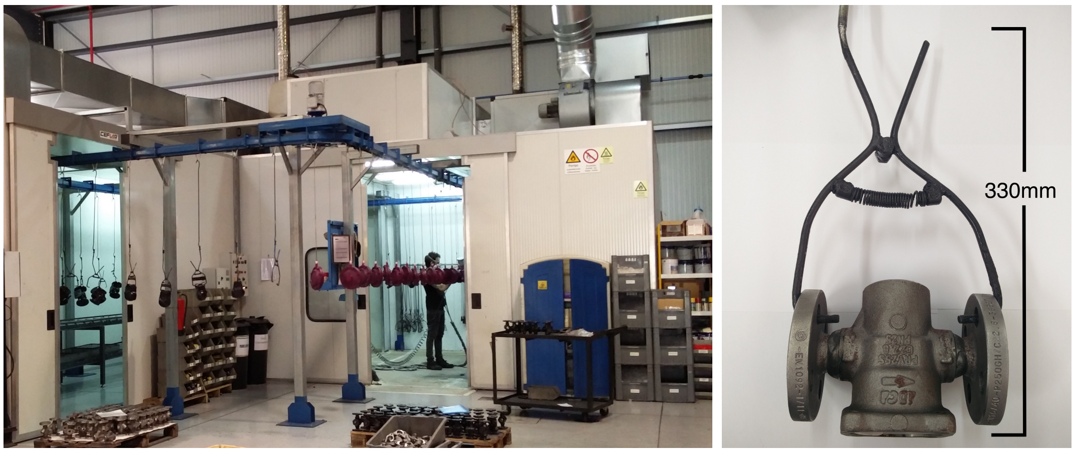
1 Introduction

This thesis involves the proposition of a system for the recognition of valve and steam trap components in the context of a industrial painting application, using vision-based supervised pattern recognition.

1.1 Motivation and Context

The existing conveyor where valve and steam trap components [1] are transported to a painting cabinet, installed in the manufacturing facilities of a company specialized in steam equipment, is shown in Fig. 1.1(a). A employee is in charge of placing each component in a claw, depicted in Fig. 1.1(b), attached to the conveyor of the cabinet. The conveyor is then activated in order to move the components towards the painting section. Here, the employee applies paint coats in each one, activating the conveyor occasionally to move new products in, and freshly painted products out to the drying section. This process is repeated until all the products in the conveyer are painted and dried, moment in which they are removed from the same.

[1]The word "product" will also be used when referring to valve and steam trap components.



(a) (b)

Figure 1.1: (a) The existing painting cabinet in the manufacturing facilities of a company specialized in steam equipment; (b) A valve body and the claw that holds it in place for painting.

In order to increase the level of automation in this process and to replace human beings in the painting task by robots, it is necessary to automatically recognize each product, before it enters the painting cabinet. The system would be in charge of outputting the type of product to be sent to the robot's controller. An automated painting cabinet offers various advantages: robots are fast, reliable and able to apply the same high-quality finish time after time without "tiring". Robots increase finish quality, consistency and throughput while lowering operating costs, they also provide paint savings over a manual process, and gain in speed. Finally, robots improve safety by reducing the exposure of human workers to paint fumes and other environmental risks, as well as reducing repetitive motion injuries.

1.2 Goal

The goal is to integrate machine learning techniques, in particular vision-based supervised pattern recognition, in the process of creating an automated recognition system for a new painting cabinet project. The products will be recognized and painted by robots which will execute specific movements depending on the object, in order to paint it thoroughly. This movements can be planned using an automatic trajectory planning system (ATPS) that uses techniques based on CAD-guided models[48] [10]. There are several solutions for this kind of application (automated product finishes and coating solutions) however, when dealing with a really high range of products (each one with their own particular shape and dimension; e.g. the case of steam equipment), the majority of solutions using sensors or tags [15] (e.g. RFID) might not be suited. Thus, this thesis proposes a solution based on machine vision.

In summary, the purpose of this project is to create an autonomous identification/recognition system for different manufactured and non-painted products, as they hang on the transportation conveyor of a painting cabinet. Such products are components of valves and steam traps manufactured by a specialized company, located in the central part of Portugal.

1.3 Setup, Dataset and Software

A setup was built and arranged in order to mimic a plausible image acquisition scenario for the industrial environment, as shown in Fig. 1.2. A backlight was positioned behind a claw that is similar to the ones used in the existing painting cabinet. The claw is fixed on a wooden frame high enough in order to keep the products hanging. A PointGrey Grasshopper2 GS2-FW-14S5C camera, with a F1.4/8.2mm lens is then used to capture images on a Linux machine with appropriate software.

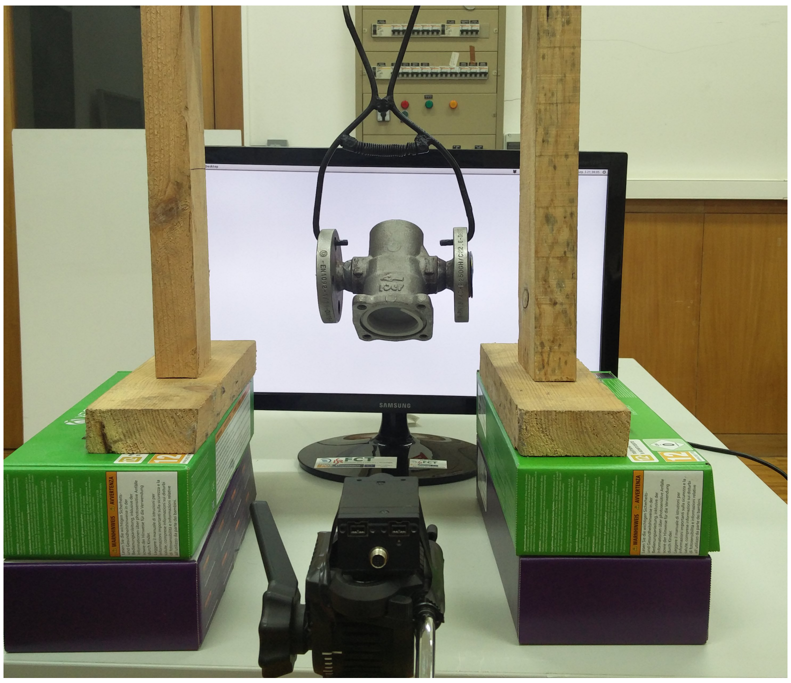


Figure 1.2: Image acquisition setup scheme.

Regarding the dataset, it was selected a group of seven of the best-selling/most produced products in the company (see Fig. 1.3) and one hundred images were taken for each one, giving a dataset with a total of seven hundred images.

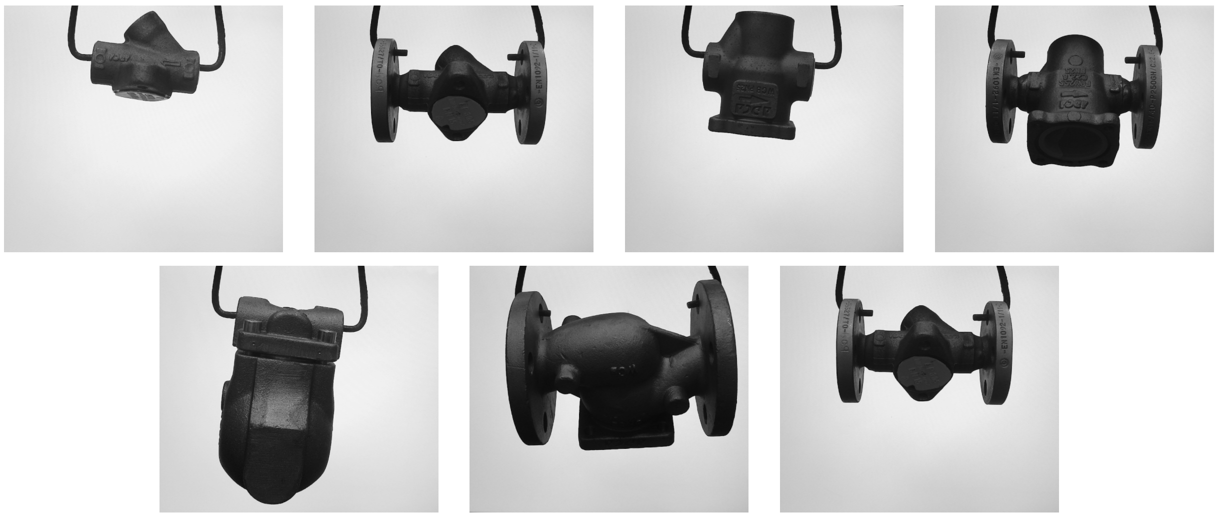


Figure 1.3: Raw image examples of each of the products that belong to the dataset.

In practice the products would be positioned in specific poses, front or back, with slight variance in rotation, and the claw would be always fixed in the same hanging spot of the product and maintain a fixed distance to the camera (no significant variance to scale). Therefore, the images, of size 800×706, were taken with consideration to what would occur in the industrial environment. This includes images with products on a front facing pose, back facing pose as well as others with slight horizontal and vertical rotations in relation to each of the two mentioned poses. Frames were also grabbed with purposely applied noise, such as slight distortion, shadowing or reflexion, see Fig. 1.4.

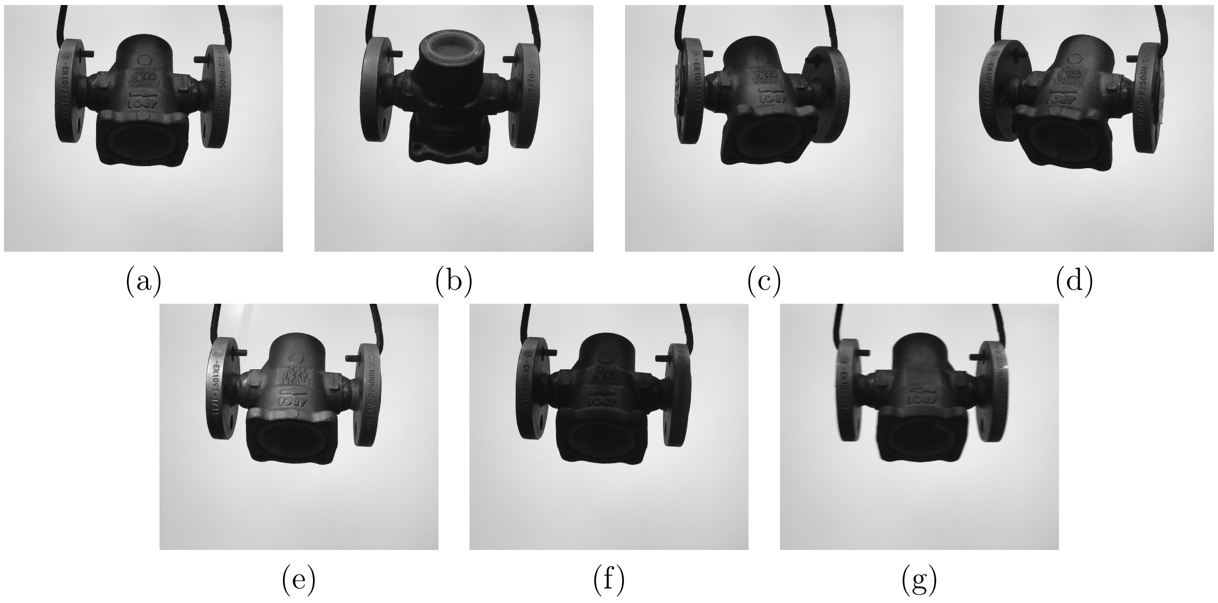


Figure 1.4: Some examples out of the 100 images taken for a certain product: (a) Front facing pose; (b) Back facing pose; (c) Front facing pose with slight horizontal rotation to the right; (d) Front facing pose with slight horizontal rotation to the left; (e) Front facing pose with reflexions; (f) Front facing pose with shadowing; (g) Front facing pose with blur.

The code written in this project was made using C++ language in conjunction with the open source OpenCV 2.4.11 library, in a machine running Ubuntu 14.04. Part of the experiments where also conducted in a Matlab environment.

2 State of the Art

Throughout the literature, automated visual systems, i.e. machine vision systems, are used in many industrial situations for a huge range of tasks. In terms of quality inspection, it is used for detecting defects in surfaces like wood [54], concrete [41] and steel [38]. In the food industry it's used for quality evaluation of fruits [2], rice [53] and food manufacturing inspection in general [12]. Automated visual inspection is also used in food packaging, for applications such as can-end inspection [11] in beverage companies. In the industry of electric components, machine vision can be used for inspection during the production of printed circuit boards [17]. In other manufacturing industries we see machine vision being used for bearings defect inspection [45], and fastener/bolt recognition [44]. Some examples in textile industry are fabric pattern and garment recognition [35].

Based on the study of the above related works, it can clearly be concluded that regardless of the application, in general, current state of the art machine vision systems have a pipeline with common modules. Such pipeline may be described as the architecture for a typical automated machine vision system, using machine learning [42], and it's shown in Fig. 2.1. The system starts by a camera, or more, that is used to capture a image at a certain key point in the process. This raw image is then sent for preprocessing, in which it's cropped into a ROI (Region of Interest), and then further processing steps may be performed, such as: binarization, normalization, morphological operations, segmentations, scaling, and so on. A set of features are then extracted to compose a feature vector which is then sent to a previously trained classifier in order to determine the category/class of an unseen image.

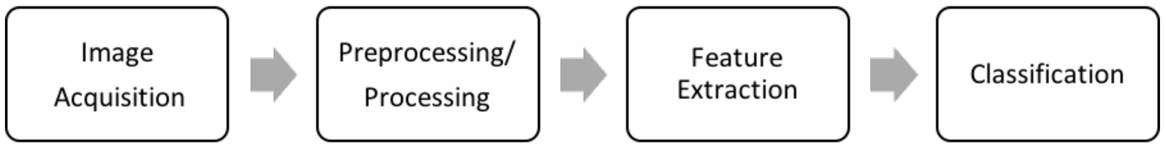


Figure 2.1: Typical supervised recognition based machine vision system architecture.

The sections bellow contain brief descriptions of each of the pipeline stages which represent the machine vision system implemented in this thesis, with references to state of the art techniques found in the literature.

2.1 Image Acquisition and Preprocessing

Image acquisition and image preprocessing represent two really large and very vast fields in machine vision, with a lot of theory, background material, approaches and techniques. Although it is beyond the scope of this thesis to provide a description and a literature review as it deserves, a concise and focused description of some of the techniques will be provided in the sequel.

In a vision system there is nothing more important than image acquisition, since any deficiencies on the original images are likely to cause great problems with image analysis and interpretation. In image acquisition, topics like the choice of camera (number, position, frame-rate, type of sensor, output standard, robustness, cost, size, etc), the lens (telecentric, multifocal, etc) and illumination technology and techniques are deeply discussed. For further readings please check [16] [32] [43] [20] [28].

Image preprocessing involves operations with images at the lowest level of abstraction. The aim of pre-processing is, in plain words, to improve of the image data, which is done by suppressing unwanted distortion and enhancing important image features. In image preprocessing topics like image filtering and enhancement, image compression, morphological processing and image segmentation are discussed. For further readings please check [47] [43] [31] [23].

2.2 Object Description and Feature Extraction Techniques

The selection of a proper and "optimal" feature set is probably the single most important factor in achieving high performance in object recognition systems. Recognition of image regions is an important step on the way to understanding image data, requiring an exact region description in a form which is suitable for a classifier. The description will be used to generate a feature vector characterizing properties of the region such as shape, texture, color, motion and rotation.

Shape descriptors may be sorted according to whether they're based on the object boundary information (called contour-based, giving external description) or whether they're based on object region information (called region-based, giving internal description). Under each class, different methods are further divided into structural approaches and global approaches. This sub-class is based on whether the shape is represented as a whole or represented by segments/sections [46]. Both contour-based and region-based shape descriptors differ in sensitivity and invariance to translation, scaling and rotation. Table 2.1 represents a list of some shape descriptors found in the literature [1] [39] [47] [43]. In [52] we may find a survey of shape feature extraction techniques.

Contour-Based	**Region-Based**
Simple geometric border representation:	Simple scalar region descriptors:
Boundary length/Perimeter	Area
Curvature	Euler's number
Bending energy	Projections
Signature	Elongatedness
Chord distribution	Height, width
Elongation	Eccentricity
Compactness	Rectangularity
Corner Points	Direction
Area	Compactness
...	...
Chain codes	Invariant Moments
Fourier Shape descriptors	Convex Hull
B-splines	Region Decomposition and Region Neighborhood graphs

Table 2.1: List of different contour-based and region-based shape descriptors.

The descriptors mentioned above are called high-level descriptors since they rely on high-level features, extracted from shape information (information about spatial relationships). Low-level descriptors on the other hand are defined by low-level basic features that can be directly extracted from an image without any shape information. However, the function of low-level feature extraction is oftentimes to provide information for a later higher level analysis.

Some low-level features that might be extracted from images are, for example, edges (using edge detectors with operators; e.g. Prewitt, Sobel, Canny and MarrHildreth) and curvature (e.g. Harris operator). Carefully designed and dense descriptors such as SIFT [36], SURF [3] and HOG [14] are also usually used as "low-level" descriptors. However, the later descriptors are rarely, not so say never, covered in the traditional machine vision literature since those descriptors (SIFT, SURF, HOG and others) were proposed for more general purpose applications in object detection, intelligent systems, machine learning, and others.

2.3 Object Classification

The theory of supervised pattern recognition and object classification is thoroughly discussed in various references [21] [4] [27] [51].

In any object recognition system we obviously need classification, which involves a decision machine (one classifier or an ensemble of classifiers) that allows the determination of the object's category/class based on its extracted features. In order to design a supervised system, one needs labelling in part of the data, so that it may be used as training set to train a classifier. Other methods, ignoring labeling (unsupervised learning), are also used, but the solution proposed in this thesis depends on supervised classification and hence unsupervised classifiers are not the scope of this work. Classifiers may also differ in the type of output they produce. In the probabilistic case the output corresponds to the probability of a new object belonging to a particular class. This type of output brings the great advantage of assigning confidence levels for the classification which is important in industrial machine vision systems, since it helps to prevent misleading classifications when having low confidence levels. On the other hand, the output of a non-probabilistic classifiers corresponds to the assignment of the object to a class. Classifiers can also be categorized as discriminative and generative [34]. Generative classifiers learn a model of the joint probability, $p(x, y)$, of the input x (in this case a feature vector) and label y, making their predictions based on rules to estimate the posterior $p(y|x)$, and then choosing the most likely label y. The Naive Bayes classifier (NBC) is an example of a simple generative (and probabilistic) classifier, based on applying the Bayes' theorem with the assumption that all features are conditionally independent given class labels [37]. Discriminative classifiers on the other hand are based upon the direct mapping of inputs x to class labels y. As examples of this types of classifiers we have Support Vector Machines (SVM) and Neural Networks. Support Vector Machines are based on the conception of decision hyperplanes that define margins (minimal distances from the separating hyperplanes to the closest data points) and separate between a set of objects that carry different class memberships. The SVM learning machine seeks for the optimal separating hyperplane, where the margin is maximal. An important feature of this approach is the fact that the solution is based only on the points which are in the decision boundaries. This points are called support vectors [7] [8]. A Neural network, more properly referred to as an artificial neural network (ANN), consists in a type of artificial intelligence that attempts to imitate the way a human brain works. The principle of AAN is based on creating connections between processing elements, the computer equivalent of neurons. The weights and organization of the connections determines the output [26] [27]. ANN have gained much attention recently because of the promising and success of deep learning architectures (e.g. convulational NN) in many problems of machine vision.

3 Image Preprocessing

This chapter presents the preprocessing techniques implemented and used in this dissertation. Firstly a thresholding algorithm is used in order to create a suitable binary image, followed by a morphological operation called "closing". The next step is the implementation of an algorithm for removing the claws that hold the objects, and finally a filling algorithm is used to fill undesirable gaps from the resulting blob. The images will be, at this point, ready for feature extraction.

3.1 Thresholding

Thresholding consists in a binarization method of image segmentation, allowing the transformation of a grayscale image into a binary image [23]. Otsu's method, named after its inventor Nobuyuki Otsu, is a "benchmark" on this field, being a very effective and popular binarization algorithm. The method involves iterating thoughout all the possible threshold values and calculating a measure of distribution for the pixel levels in each "side" of the threshold, i.e. the pixels that fall in the category of background or foreground. The goal is to find the threshold value where the foreground and background distribution sum is minimal [40]. This means that for a bimodal image (an image whose histogram has two peaks, which represent foreground and background) the Otsu's method is clearly an appropriate and robust algorithm for binarization. Therefore, since the images acquired in this project meet the requirement for being bimodal, the Otsu's method was the chosen algorithm for binarization, see Fig. 3.1.

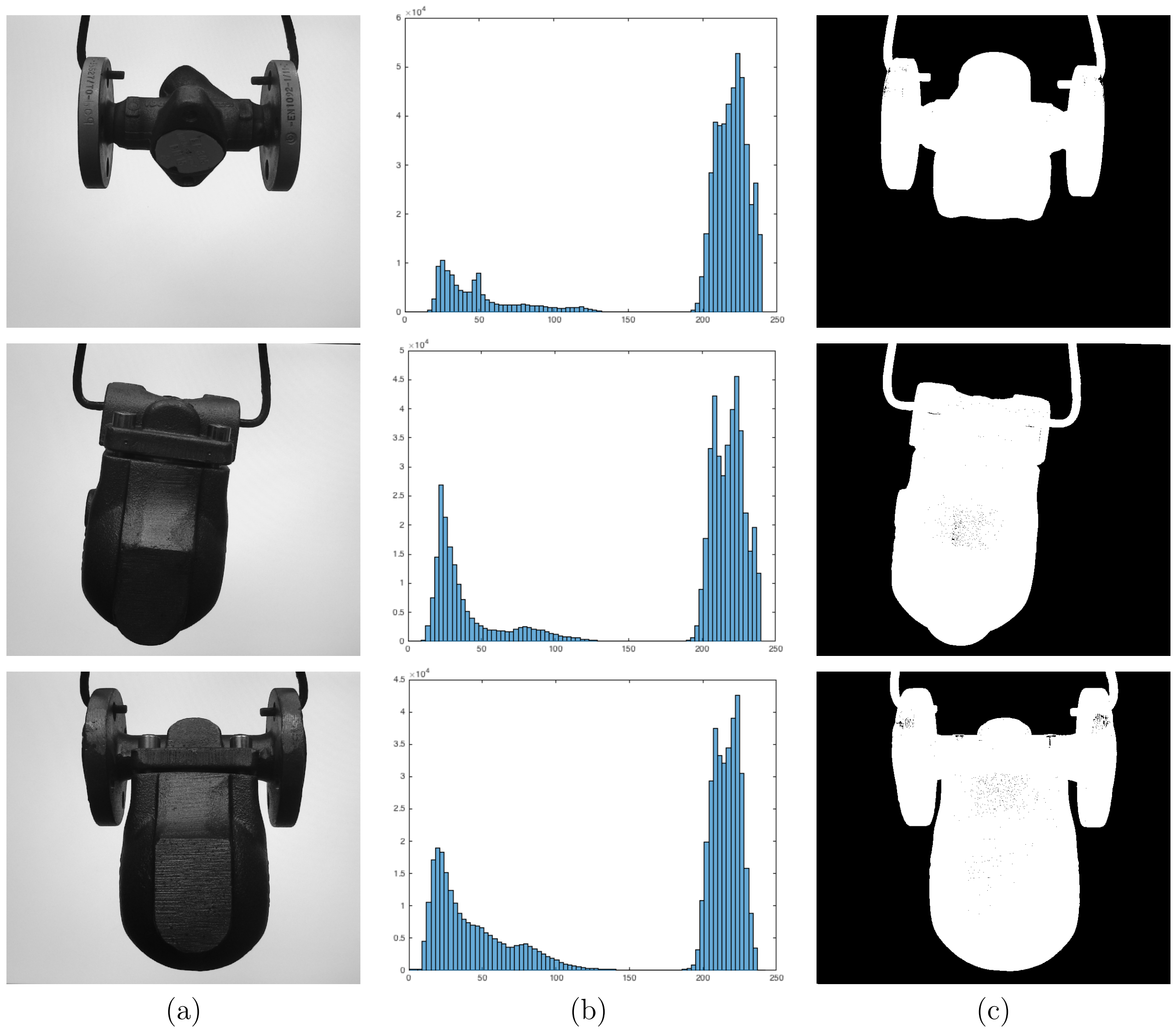


Figure 3.1: (a) Raw image examples; (b) Grayscale histograms of each image represented in (a); (c) Otsu's thresholding method results.

3.2 Morphological Closing Operation and Flood Fill

A brief analysis of the thresholding results show good binary representations of the objects of study, with preservation of the objects shape and outline. However it also shows presence of noise which is the result of reflections caused by illumination, as well as some distortion purposely applied to some images of the data set. Several filters (e.g average, Gaussian, median) may be introduced before the segmentation step in order to minimize this noise. Another way of achieving this goal is by applying morphological operators, but this time after segmentation.

Morphological operations are based on the image shape characteristics and are normally performed on a binary image. The image is then used alongside a structuring element, or kernel, which correspond to the two needed inputs for the transformation. The structuring element is a small matrix of pixels, with values zero or one, which is shifted over the image and positioned at all possible locations. In each of this locations, the structuring element is compared with the corresponding neighborhood of pixels and, depending on the operations, it's tested whether the elements "fit", "hit" or intersect the neighborhood [22]. The two most basic morphological operators are called erosion and dilation, followed by variant forms like opening and closing.

The closing operator, which corresponds to a dilation followed by erosion, connects objects that are close to each other, filling up small holes, and smoothing the object's outline by filling up narrow gulfs. The images in 3.2 illustrate the results obtained by applying this operator. In order to prevent the operator from smoothing the object's outline and affecting the objects shape, which is an important feature to maintain for the next pipeline stage, a small 3x3 structuring element is used.

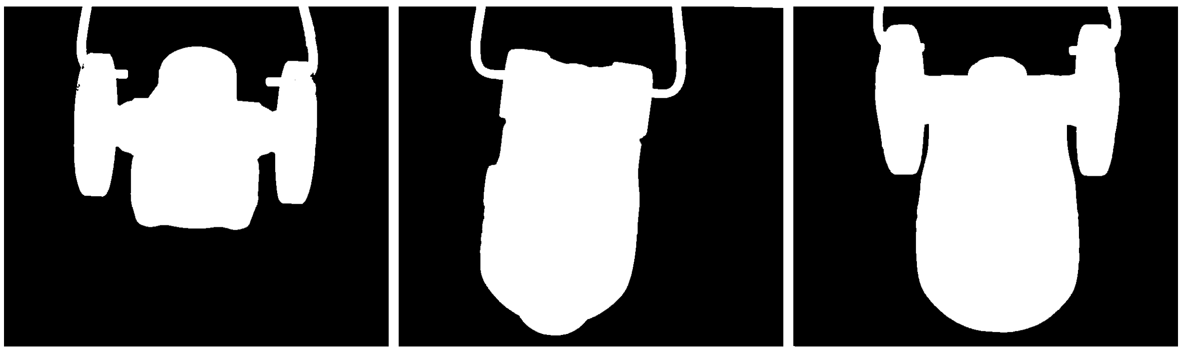
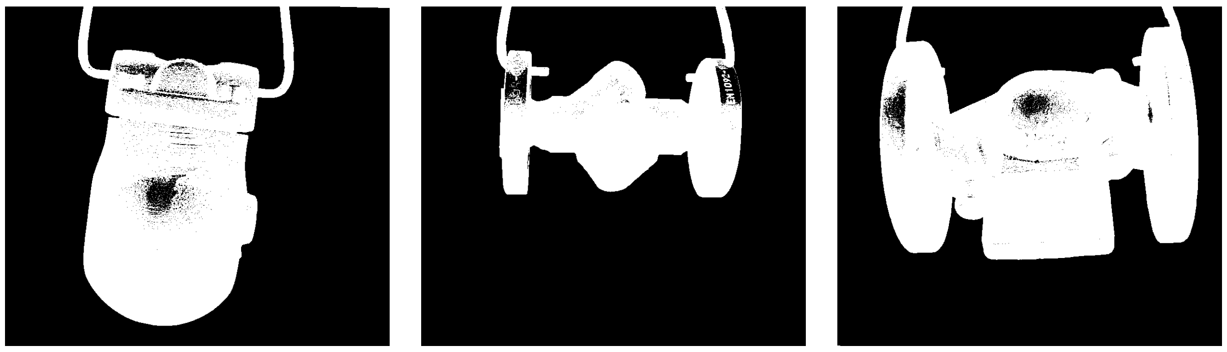


Figure 3.2: Closing operation results for the images shown in Fig. 3.1.

As seen in the last figure, the operator is successful in eliminating small pixel dots in the images. However if the images present bigger holes (high noise concentration in specific areas), like the ones shown in Fig. 3.3, the operator is not able to remove them fully.


(a)

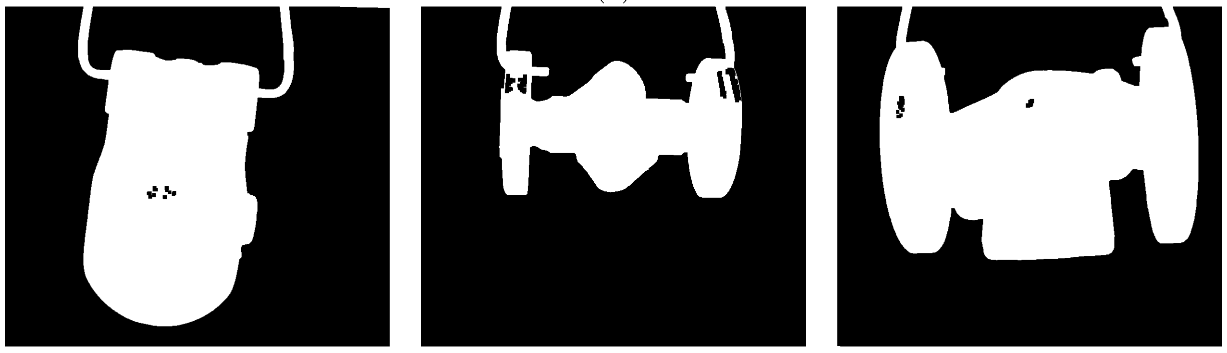

(b)

Figure 3.3: (a) Thresholding results for images with high amount of noise; (b) Closing operation results for each image shown in (a), with a 3x3 structuring element.

A solution to this problem is the usage of a flood fill algorithm [6], to complement the morphological operation results. The algorithm is responsible for filling up a given contour with a specific color. The images in Fig. 3.4 show the results obtained after applying the algorithm, on the problematic example images shown in Fig. 3.3.

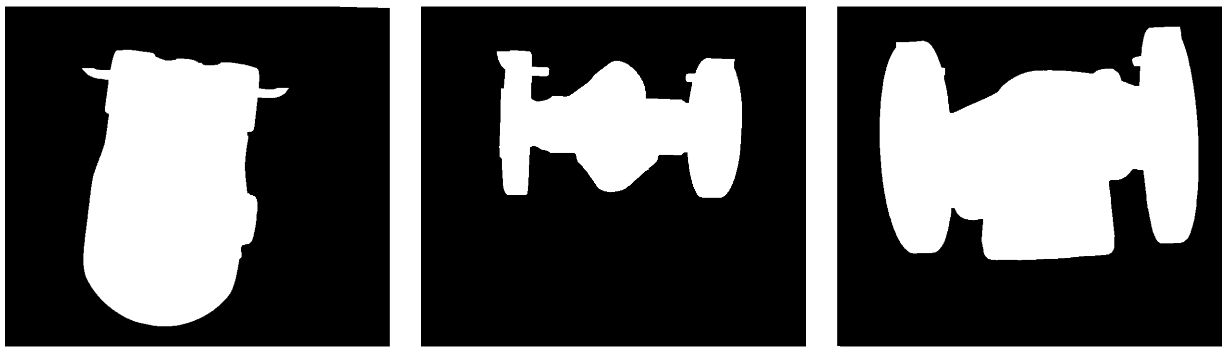


Figure 3.4: Flood fill results for the images in Fig. 3.3.

Note: The flood fill algorithm is applied after the claw removal step, discussed in the next section, since it's the stage where we end up with a blob that exclusively represents the object, see Fig. 3.5.

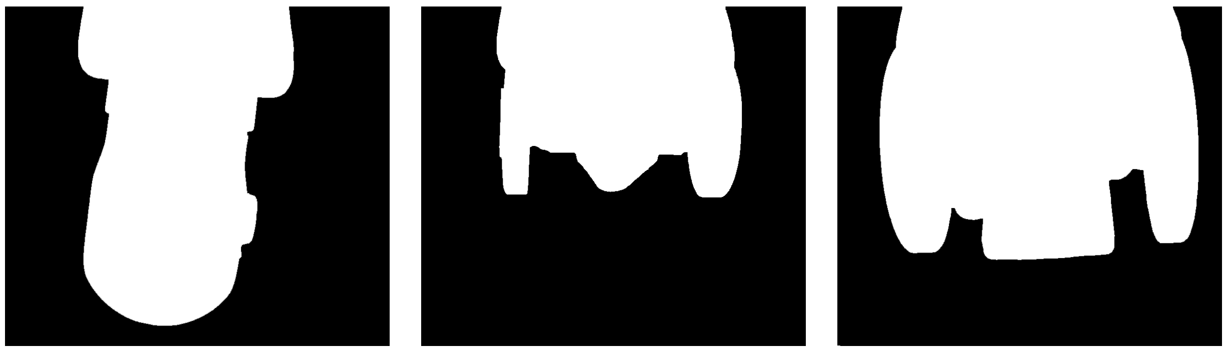


Figure 3.5: Flood fill results for the images in Fig. 3.3, when applied before the claw removal process.

3.3 Claw Removal

This section presents the algorithm responsible for the removal of the majority of the visible claw sections present in the images of the data set. The algorithm can be divided in two stages. In the first stage the average claw piece lengths[1] are calculated based on the upper section of the image, the "region of interest". At this stage we are also able to determine a estimated initial horizontal position for each claw piece. In the second stage two masks (one for each claw piece) are slided along the image, performing comparisons between the mask values and the underlying image pixel values. A decision is then made whether or not to unset a section of those underlying pixels, based on the amount of "hits" between the mask and the current image region thats being analyzed, allowing the removal of each claw piece segment by segment.

In order to understand how the average length of each claw piece is calculated, lets first take a look at Fig. 3.6 which illustrates the upper sections of the images shown in Fig. 3.2.

[1]The word "length", "dimension" and "size" is used throughout the thesis when referring to the number of pixels. Measurement units such as the millimeter or inch are not used.

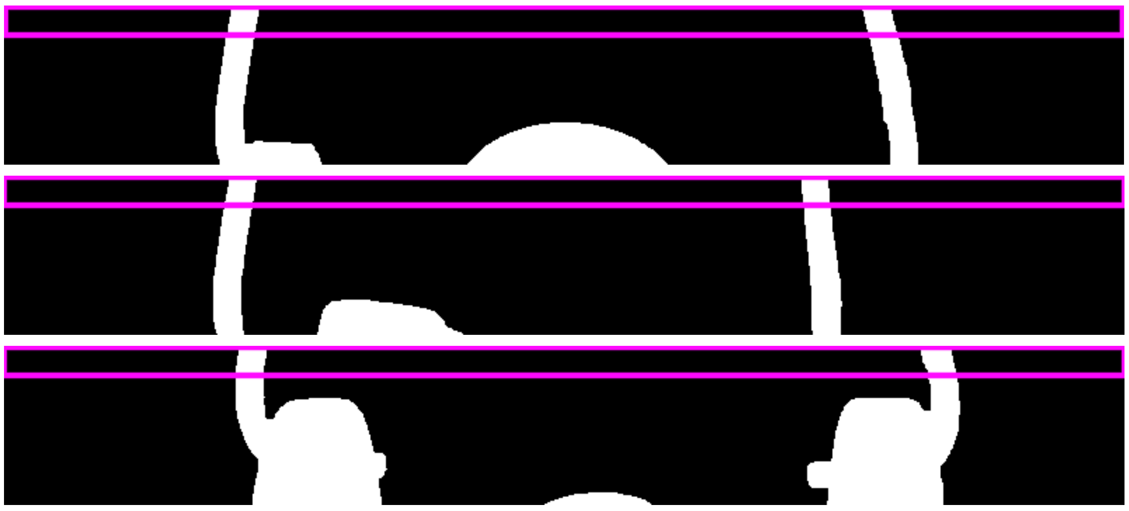


Figure 3.6: Upper section of the images in Fig. 3.2. The purple rectangles outline a set of rows, corresponding to our "region of interest".

In the pixels rows inside the "region of interest" there is exclusive presence of claw segments, without presence of the object that it holds, see Fig. 3.7. This means that for each of this rows all the information extracted is related to the claw. The result is a fairly straightforward calculation of the left and right claw piece segment lengths in each of those rows.

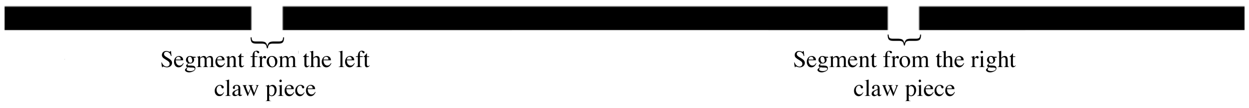

Segment from the left claw piece
Segment from the right claw piece

Figure 3.7: Example of an image row inside a "region of interest".

The algorithm starts by performing a scan from left to right in each row inside the "region of interest", *row_set*, in order to analyze each of its pixels. The scan seeks for a transition from an unset pixel to a set pixel, which will correspond to the left edge of a left claw piece segment, followed by a transition from a set pixel to an unset pixel, which will correspond to the adjacent edge of that segment, see Fig. 3.8. The number of pixels between each of those transitions gives the length of the left claw piece segment in that row. The same is done for the right claw piece segment, but this time with a scan from right to left. This process is repeated for each row in *row_set* and the average segment length of each piece is calculated. The algorithm also estimates the initial horizontal axis position of the left and right claw pieces, *est_left_claw_axis* and *est_right_claw_axis*, respectively. The position is only based on the first row, *row_set*[0], and is calculated by relating the points where the pixel transitions occur with the length of each claw piece in that row.

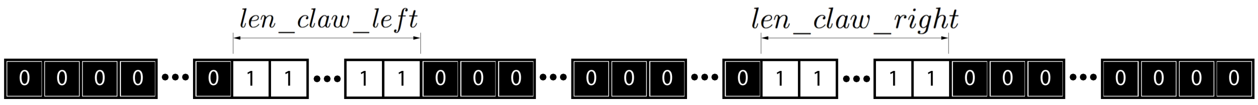


Figure 3.8: Representation of a row of pixels inside a "region of interest". The unset pixels are represented by the number '0' and the set pixels by '1'.

As said before, the claw removal process is based on masks. For simplicity reasons it's considered that a single mask, shown in Fig. 3.9, is used for the removal of both claw pieces. The masks consists on a window with two sections of unset pixels, Γ_L and Γ_R, of sizes ρ_L and ρ_R respectively, and one section, Γ_C, of size ρ_C. All three sizes are based on the average claw piece length value previously calculated. In reality there are two masks, $claw\_mask\_left$ and $claw\_mask\_right$, one for each claw piece.

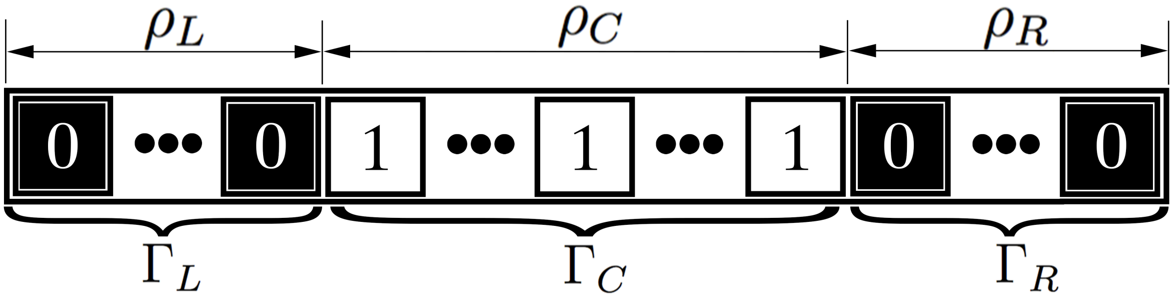


Figure 3.9: Illustration of the mask used for the claw removal process.

The mask is used as a slide window, which moves across the image row by row, firstly from left to right, as the algorithm searches for the left claw segment, and vice versa, as it searches for the right one. In both situations the slide window moves inside a rectangular validation region of size $2 \times (\rho_L + \rho_C + \rho_R)$.

As the slide window moves from left to right, the algorithm calculates the amount of pixel "hits" in each of it's locations (i.e. H_L, H_C and H_R). If that amount is greater or equal to a defined percentage of pixels for each section (i.e. θ_L, θ_C and θ_R) it means that a claw piece segment was found in that row. At that point, all the corresponding Γ_C section pixels in that particular row of the image are removed, see Fig. 3.10 . The measured left claw piece segment axis position of the current row, $obs\_left\_claw\_axis$, is stored, and coincides with the central pixel of the Γ_C removed section. The hole process is repeated for the right claw piece but this time the slide window moves from right to left, removing the right claw piece. In this case the measured claw piece segment axis position is stored in $obs\_right\_claw\_axis$.

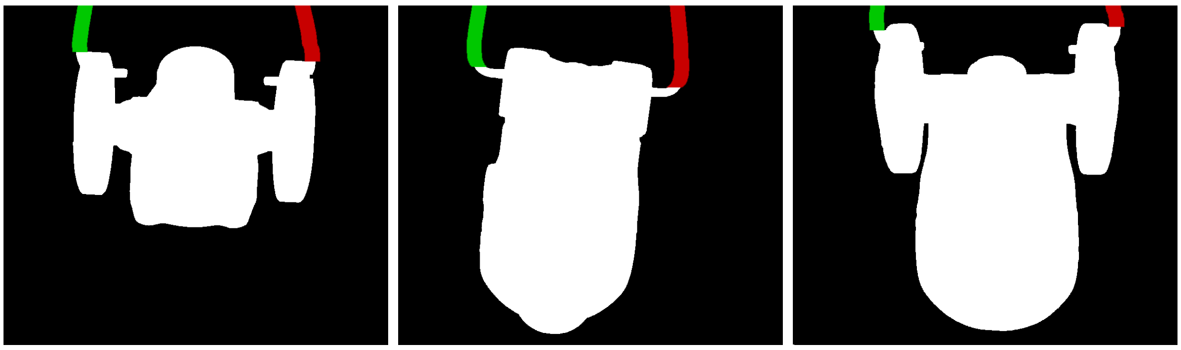


Figure 3.10: Claw removal process results. The removed pixels from the left and right claw pieces are represented in green and red respectively. The colors were chosen for illustrative purposes, in reality the pixels are unset.

The validation region aforementioned corresponds to the set of columns where the mask is positioned, and it's used as a claw segment tracking mechanism. The region starts by being centered on the estimated initial horizontal axis position value of each piece calculated in the first stage. A one dimensional Kalman filter [50] is then applied in order to estimate the next axis positions, $est\_left\_claw\_axis$ and $est\_right\_claw\_axis$, based on the current observed positions and the previous estimated position. In other words, the goal is to estimate the state of the current time step k, which corresponds to $x \in \Re$, based on the state of the previous time step $k-1$, or x_{k-1}, where x is the axis position.

The discrete-time controlled process is governed by the linear stochastic difference equation

$$x_k = Ax_{k-1} + \omega_{k-1} \tag{3.1}$$

with a measurement $z \in \Re$ that is

$$z_k = Hx_k + v_k \tag{3.2}$$

and considering a state matrix $A = 1$, a output matrix $H = 1$, a process noise covariance $Q = 0.1$ and a measurement noise covariance $R = Q \times 0.1$.

The full algorithm responsible for the claw removal process discussed in this section is shown bellow (see Algorithm 1).

Algorithm 1 Claw removal algorithm

```
 1: procedure ClawRemoval($Src$, $claw\_mask\_left$, $claw\_mask\_right$, $row\_set$)  ▷ Src: Input image with claws
 2:     $len\_claw\_left = \{\}$;
 3:     $len\_claw\_right = \{\}$;
 4:     //Stage 1: Claw piece segment length calculations and initial horizontal claw piece axis position estimations:
 5:     for each $Src$ row in $row\_set$ do
 6:         Search for left claw piece segment;
 7:         $len\_claw\_left \leftarrow$ Left claw piece segment length for the current row;
 8:         if $Src$ row= $row\_set[0]$ then
 9:             $est\_left\_claw\_axis$=Left claw piece estimated axis value (column), for the current row (row 0);
10:         end if
11:         Search for right claw segment;
12:         $len\_claw\_right \leftarrow$ Right claw piece segment length for the current row;
13:         if $Src$ row= $row\_set[0]$ then
14:             $est\_right\_claw\_axis$=Right claw piece estimated axis value (column), for the current row (row 0);
15:         end if
16:     end for
17:     $claw\_mask\_left \leftarrow mean(len\_claw\_left)$;  ▷ Average $len\_claw\_left$ value calculation
18:     $claw\_mask\_right \leftarrow mean(len\_claw\_right)$;  ▷ Average $len\_claw\_right$ value calculation
19:     //Stage 2: Claw removal:
20:     Estimation of the rectangular validation regions position based on $est\_left\_claw\_axis$ and $est\_right\_claw\_axis$;
21:     for $Src$ row=0 to (Number of rows in $Src$)/3 do
22:         for each $Src$ column inside the rectangular validation region of the left claw piece do
23:             Center $claw\_mask\_left$ in the current column;
24:             Calculate $H_L$, $H_C$ and $H_R$;
25:             if $H_L >= \rho_L \times \theta_L$ & $H_C >= \rho_C \times \theta_C$ & $H_R >= \rho_R \times \theta_R$ then
26:                 Unset all pixels in the $\Gamma_C$ section of the image;  ▷ Left claw segment removal
27:                 $obs\_left\_claw\_axis$= Measured left claw piece current axis value (column);
28:             end if
29:             Kalman filter is used to estimate $est\_left\_claw\_axis$ based on it's last value and $obs\_left\_claw\_axis$;
30:             Estimation of the left rectangular validation region position for the next row, based on $est\_left\_claw\_axis$;
31:         end for
32:         for each $Src$ column inside the rectangular validation region of the right claw piece do
33:             Center $claw\_mask\_right$ in the current column;
34:             Calculate $H_L$, $H_C$ and $H_R$;
35:             if $H_L >= \rho_L \times \theta_L$ & $H_C >= \rho_C \times \theta_C$ & $H_R >= \rho_R \times \theta_R$ then
36:                 Unset all pixels in the $\Gamma_C$ section of the image;  ▷ Right claw segment removal
37:                 $obs\_right\_claw\_axis$= Measured right claw piece current axis value (column);
38:             end if
39:             Kalman filter is used to estimate $est\_right\_claw\_axis$ based on it's last value and $obs\_right\_claw\_axis$;
40:             Estimation of the right rectangular validation region position for the next row, based on $est\_right\_claw\_axis$;
41:         end for
42:     end for
43: end procedure
```

4 Feature Extraction

This chapter presents the feature extraction methods used in this work's implemented system. We start by presenting a set of geometric features based on the objects shape characteristics followed by a HOG implementation suited for simple blob description.

4.1 Blob Features Based on Shape

Each image of the data set and after post preprocessing, contains a blob that represents a object. A way of obtaining relevant features from this kind of structure involves the extraction of it's shape characteristics. The following is a list of features pertaining to a given blob, together with their description:

1. **Area:** Number of pixels of the blob. For an image with a single object, it corresponds to the total number of set, or white, pixels in it.
2. **Perimeter:** Length of the blobs contour in pixels.
3. **Straight Bounding Box:** Minimum straight (i.e. not considering the rotation of the object) rectangle which contains the blob. In other words it's the smallest straight rectangle that contains every point in the shape.
4. **Rotated Bounding Box:** Minimum rotated rectangle which contains the blob, representing the true minimum enclosing rectangle.
5. **Minimum Enclosing Circle:** Minimum circle which contains the blob.
6. **Line Fit:** Line that best fits the blob. The line passes through the blob's geometric center and has orientation given by the direction in which the blob is oriented. A way of fitting a line is by minimizing $\sum_i \rho(r_i)$ where r_i is a distance between the i^{th} point, the line and $\rho(r)$ is a distance function:

$$\rho(r) = r^2/2 \quad \text{(the simplest and the fastest least-squares method)} \qquad (4.1)$$

The algorithm is based on the M-estimator [55] technique that iteratively fits the line using the weighted least-squares algorithm. After each iteration the weights w_i are adjusted in order to be inversely proportional to $\rho(r_i)$.

7. **Convex Hull:** The convex hull of a blob B is the smallest convex polygon that contains all it's points.

   This first seven features are illustrated in Fig. 4.1.

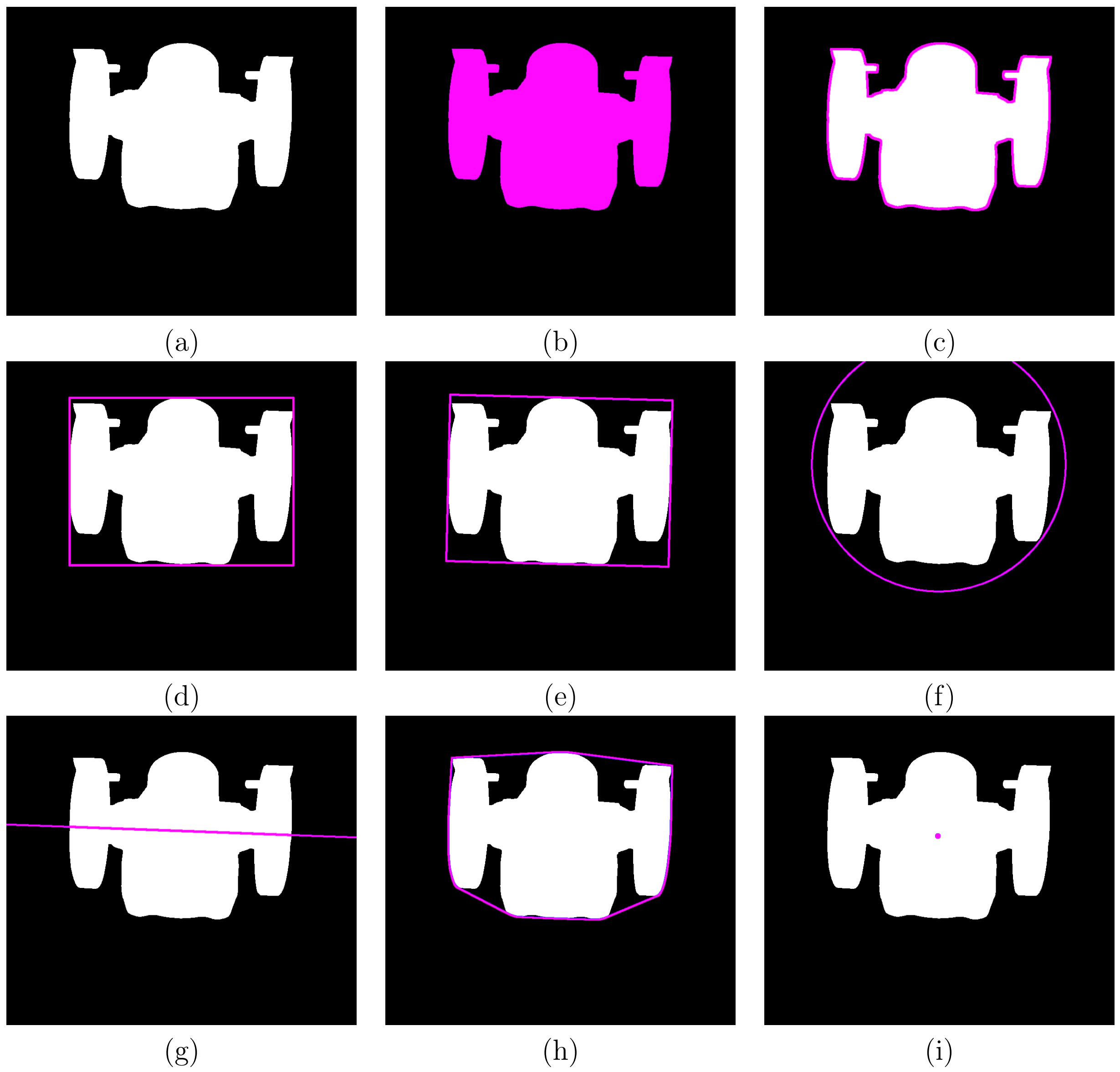


Figure 4.1: (a) Post-processing example image; (b) Area; (c) Perimeter; (d) Straight Bounding box; (e) Rotated rounding box; (f) Minimum enclosing circle; (g) Line fit; (h) Convex Hull; (i) Centroid.

8. **Convexity:** Convexity is a measure of how close a region is to being convex. Convex regions have convexity equal to 1.0, while the more concave the region is, the closer to 0.0 is its convexity. The same applies to *blob's*. The convexity of a blob B is calculated

as follows:

$$Convexity(B) = \frac{Area(B)}{Area(ConvexHull(B)} \tag{4.2}$$

9. **Invariant Moments:** Invariant moments are a very popular shape-based statistical feature. The 2-D moments of order (p, q) of a pdf function $f(x, y)$ are defined by

$$m_{pq} = \int\int x^p y^q f(x, y) dxdy \tag{4.3}$$

and the central moments of oder (p, q) are defined as

$$\mu_{pq} = \int\int (x - \bar{x}^p)(y - \bar{y}^q) f(x, y) dxdy \tag{4.4}$$

where

$$\bar{x} = \frac{\mu_{10}}{\mu_{00}} \qquad \bar{y} = \frac{\mu_{01}}{\mu_{00}} \tag{4.5}$$

For a digital image, the integrals are replaced by summations to get

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{4.6}$$

Given the normalized central moments as denoted by η_{pq},

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{4.7}$$

where

$$\gamma = \frac{p + q}{2} + 1 \text{ for } p + 1 = 2, 3, ... \tag{4.8}$$

A set of seven invariant moments, also called Hu moments [30], with respect translation, rotation and scale can be derived from the second and third moments as

$$\begin{aligned}
&\phi_1 = \eta_{20} + \eta_{02}, \\
&\phi_2 = (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2, \\
&\phi_3 = (\eta_{30} - \eta_{02})^2 + (3\eta_{21} - \eta_{03})^2, \\
&\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2, \\
&\phi_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\
&\quad + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2], \\
&\phi_6 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}), \\
&\phi_7 = (3\eta_{21} - \eta_{03})(\eta_{30} - \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} - \eta_{03})^2] \\
&\quad + (3\eta_{12} - \eta_{30})(\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{4.9}$$

10. **Centroid:** The centroid, also called center o mass, corresponds to the arithmetic mean ("average") position of all the points in the blob. It's value derives directly from raw moments, as shown in Equation 4.5.

11. **Compactness:** Ratio between the *blob's* area and the area of it's minimum bounding box.

$$Compactness(B) = \frac{Area(B)}{Area(RotatedBoundingBox)} \tag{4.10}$$

12. **Bounding Box Ratio:** Indicates the elongation of the blob.

$$BoundingBoxRatio = \frac{width(RotatedBoundingBox)}{height(RotatedBoundingBox)} \tag{4.11}$$

13. **Circularity:** Represents how circular a blob is, and may be given by:

$$Circularity(B) = \frac{Perimeter(B)}{2\sqrt{\pi.Area(B)}} \tag{4.12}$$

14. **Form Factor:** Provides a measure that describes the shape of the blob. Mathematically, the form factor of a blob is given by:

$$FormFactor = \frac{4\pi Area(B)}{\sqrt{Perimeter(B)}} \tag{4.13}$$

15. **Features Resulting from Intersections between a blob and a Circular Beam of Line Segments:** A feature extraction technique inspired in works found in the literature [9] [24], and based on the extraction of information regarding relations between

the centroid of shapes and their contour points, is here proposed. The technique will be first introduction, following by the explanation of an algorithm capable of extracting features, based on it.

The angle between a straight line segment and the horizontal axis x is given by θ. This segment is traced from the centroid C of a blob B and has length R, given by the radius of it's minimum enclosing circle plus a certain constant D. Considering N segments, with different angles θ_N, given by $\frac{k.(360/N).\pi}{180}$ where $k = 1, ..., N$, are traced over a circle of radius R. For each line segment it is extracted information regarding their intersection with the blob B's points.

This technique allows us to extract interesting features. e.g. the Euclidean distance r_N between C and the first contour point ω_N, the number of B's sections ns_N, and number of pixels p_N that each line intersects with, as shown in Fig. 4.2.

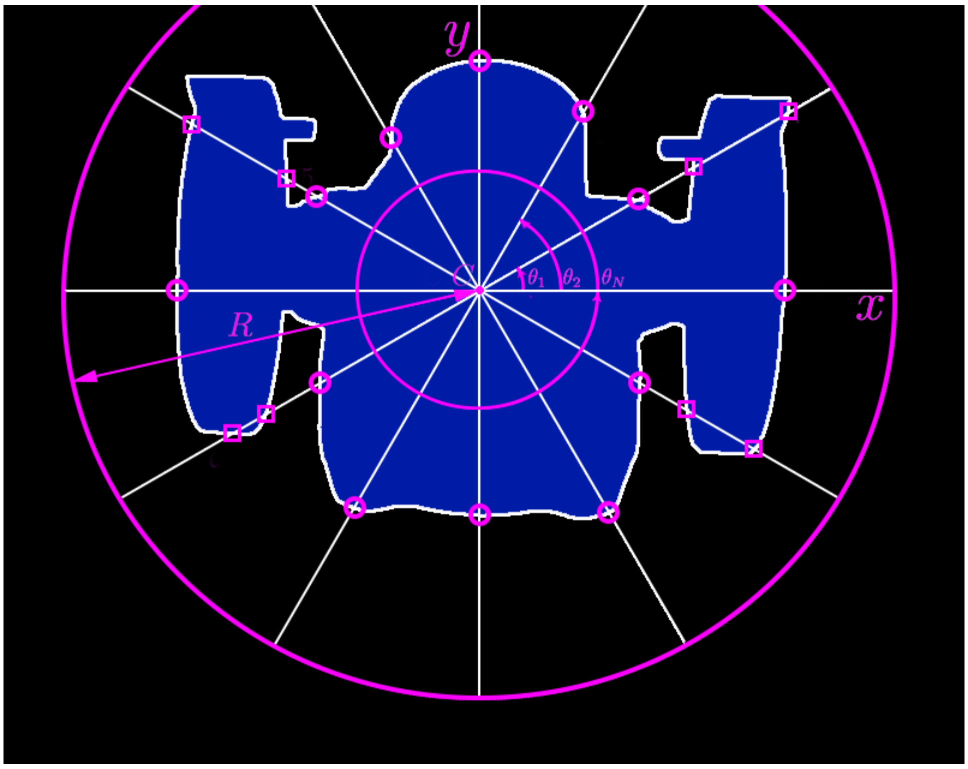


Figure 4.2: Geometrical representation of the line segments for $N = 12$, and contour intersection points.

The figure above illustrates the positions of the contour points of intersection between a blob B, and the N line segments. The small pink circles represent each of the ω_N points, and the squares represent the contour points of intersection for the cases where the lines intersect in regions with multiple blob sections. In the case of the image above $ns_N = 2$ in $N = 1, 5, 7$ and 11. This information provides details about the shape of B and may be extracted in the form of feature vectors, $f_r = \{r_1, ..., r_N\}$,

$f_{ns} = \{ns_1, ..., ns_N\}$ and $f_p = \{p_1, ..., p_N\}$. Algorithm summarizes the approach used to extract the mentioned features from a binary image with a single blob B.

Algorithm 2 Extraction of f_r, f_{ns} and f_p

```
1: procedure LineBeamIntersectionFeatures($Src$, $N$, $t1$, $t2$)    ▷ Src: Input binary image with a
   single blob $B$
2:     C=centroid($B$);
3:     R=Radius(MinimumEnclosingCircle($B$))+$D$;
4:     $f_r = \{\}$; $f_{ns} = \{\}$; $f_p = \{\}$;
5:     for k=1 to N do                                        ▷ For each line segment
6:         $it = 0$;
7:         $lpp_k = \{\}$; $lpv_k = \{\}$;
8:         for it=1 to segment ending $(x, y)$ position do    ▷ For each line segment pixel starting at $C$
9:             $lpp_k \leftarrow$ current $(x, y)$ position of the segment in relation to the image;
10:            $lpv_k \leftarrow$ current pixel value for the current $(x, y)$ position;    ▷ '0': unset; '1': set
11:        end for
12:        Extract $\omega_k$ by knowing $lpp_k$ and $lpv_k$, and taking into account $t1$;
13:        $f_r \leftarrow$ Euclidean distance between $C$ and $\omega_k$;
14:        Search for groups of set elements in $lpv_k$ taking into account $t2$;
15:        $f_{ns} \leftarrow$ Number of groups;
16:        $f_p \leftarrow$ Total number of elements with the value '1' in $lpv_k$;
17:    end for
18: end procedure
```

The algorithm is based on the calculation of the intersections between the line segment pixels and the underlying image pixels, performed by iterating over the line segment pixel (x, y) positions. The line segment iterator it start at C and ends on the opposite side of the line, being that each intersection result and corresponding positions in the image is stored in arrays, lpv_k and lpp_k respectively.

The arrays allow the extraction of the position of the first contour point of intersection ω_k. In theory, this position corresponds to the first transition between a set and an unset pixel in lpv_k however, since possible presence of noise has to be taken into account, a threshold $t1$ in used. The threshold considers the following pixels in the array in order to decide whether the outside contour of B was reached or if it simply reached a small hole resulting from noise. As soon as the algorithm decides on the contour point it proceeds on calculating r_k using the Euclidean distance formula.

Additionally the arrays also allow the extraction of ns_k and p_k, where ns_k is given by the number of groups with the value '1' in lpv_k. The groups are considered sections of the image if they include a number of elements greater than a certain threshold $t2$. The total number of elements with the value '1' in lpv_k gives p_k.

This process is repeated for each of the $k = 1, ..., N$ line segments, filling up the feature vectors f_r, f_{ns} and f_p which will ultimately have size N.

4.2 HOG Implementation for Simple Blob Description

The gradient of an image consists in a vector that points in the direction with the biggest increase in scalar values in the neighborhood. It provides two pieces of information, the magnitude, which tells us how quickly the image is changing, and the direction, which tells us the direction in which the image is changing most rapidly. For an image I, the gradient vector at a certain point (x, y) is given by:

$$\bigtriangledown I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right] \tag{4.14}$$

The gradient magnitude is given by

$$|\bigtriangledown I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{4.15}$$

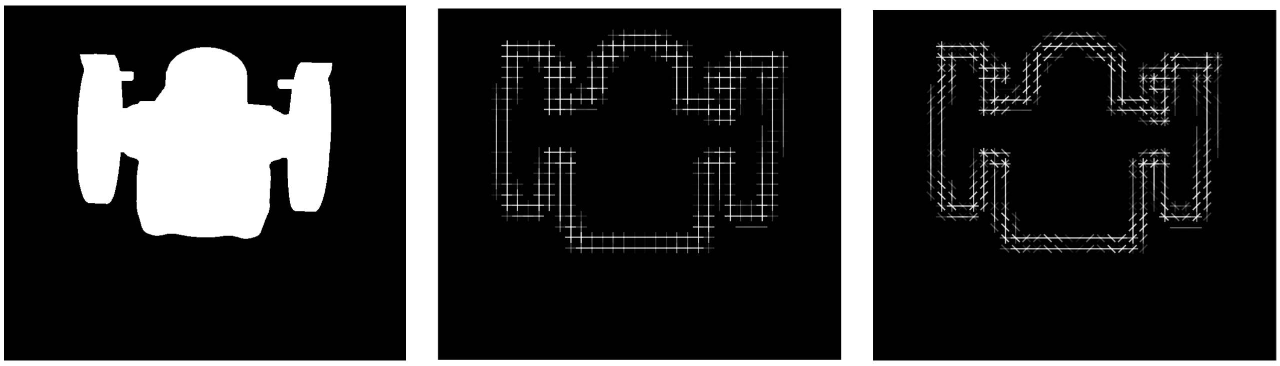
and the orientation by

$$\theta = \arctan\left(\frac{\frac{\partial I}{\partial y}}{\frac{\partial I}{\partial x}}\right) \tag{4.16}$$

The thought behind Histograms of Oriented Gradients (HOG) is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions, even without clear information about the corresponding gradient or edge positions. The image is divided into small spatial regions called cells, and for the pixels within each cell, a one-dimensional histogram of gradient is composed. The descriptor is the concatenation of these histograms. In order to improve accuracy, each local histogram can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called block. The results are used to normalize all of the cells in the block [14].

Dalal et al. (2007) concludes, in his paper regarding HOG in human detection [14], that for a good performance, many orientation bins should be used, as well as well as moderately sized overlapping descriptor blocks. However, in simple blob description, there is no real need for such complexity.

Fig. 5.1 represents the computed HOG descriptor in binary test images with *blobs*, using few orientation bins and relatively large block sizes. Piotr Dollár's implementation [18] of the HOG features is used in this work.



(a) (b) (c)

Figure 4.3: (a) Post-processing example image; (b) Computed HOG features with 2 bins and blocks of size 20; (c) Computed HOG features with 4 bins and blocks of size 20.

5 Object Classification

This chapter includes an explanation of each of the classification techniques used in this work. A generative and a discriminative algorithm are used, namely a NBC and multiclass SVM, in order to test one of each of this classifier types. Other reasons for the choice are the fact that the first one consists in a really simple probabilistic algorithm, and the second one in a powerful algorithm, as well as a standard for data classification for the past years, with proven results throughout the literature [14] [45] [54] [41].

5.1 Naive Bayes Classifier (NBC)

A vast group of classifiers may be viewed as computing a set of discriminant functions of the example, one for each class, and assigning it to the class whose function is maximum [21]. Considering E as the example, and $f_i(E)$ as the discriminant function corresponding to the ith class, the chosen class C_k is the one for which

$$f_k(E) > f_i(E), \forall i \neq k. \tag{5.1}$$

Suppose an example is a vector with A attributes and let v_{jk} be the value of attribute A_j, a possible set of discriminative functions is

$$f_i(E) = P(C_i) \prod_{j=1}^{n} P(A_j = v_{jk} | C_i). \tag{5.2}$$

The classifier obtained by using this set of discriminant functions is called Naive Bayesian classifier. The reason for being called "naive" is due to the assumption that the attributes are independent given the class, which mean this classifier can easily be shown to be optimal, in the sense of minimizing the misclassification rate or zero-one loss, by a direct application of Bayes' theorem. If $P(C_i|E)$ is the probability that example E is of class C_i, zero-one loss is minimized if, and only if, E is assigned to the class C_k for which $P(C_k|E)$ is maximum

[21]. This means that, using $P(C_i|E)$ as the discriminant functions $f_i(E)$ is the optimal classification procedure. By Bayes' theorem

$$P(C_i|E) = \frac{P(C_i)P(E|C_i)}{P(E)} \tag{5.3}$$

which in plain English, using the Bayesian probability terminology, can be written as

$$\text{Posterior probability} = \frac{\text{Class Prior Probability} \times \text{Likelihood}}{\text{Predictor Prior Probability}} \tag{5.4}$$

$P(E)$ can be ignored since it's the same for all classes, not affecting the relative values of their probabilities. If the attributes are independent given the class, $P(E|C_i)$ can be decomposed into the product $P(N_1 = v_{1k}|C_i)...P(N_n = v_{ak}|C_i)$, leading to $P(C_i|E) = f_i(E)$, as defined in equation 5.2 [19]. In this work $P(A_j = v_{jk}|C_i)$ follows a normal (Gaussian) distribution.

5.2 Support Vector Machine (SVM)

The basic idea behind a Support Vector Machine is that of finding a hyperplane which separates the d-dimensional data perfectly into its two classes (SVM only deals with binary classification). However, since example data is often not linearly separable, SVM's introduce the notion of a "kernel induced feature space", casting the data into a higher dimensional space where the data is separable. Casting into such a space would normally cause problems with overfitting and computation, however this concerns are eliminated due to a key insight used in SVMS's, which tell us that the higher-dimensional space does not need to be dealt with directly. As we will see next, only the formula for the dot-product in that space is required.

Assuming l training examples $\{\boldsymbol{x_i}, y_i\}$, $i = 1, ..., l$, where each example has d inputs ($x \in \Re^d$), and a class label with one of two values ($y_i \in \{-1, 1\}$), all hyperplanes in $\Re^d$ are parameterized by a vector $\boldsymbol{\omega}$ and a constant b, expressed in the equation

$$\boldsymbol{\omega} \cdot \boldsymbol{x} + b = 0 \tag{5.5}$$

where $\boldsymbol{\omega}$ is the vector orthogonal to the hyperplane, see Fig. 5.1.

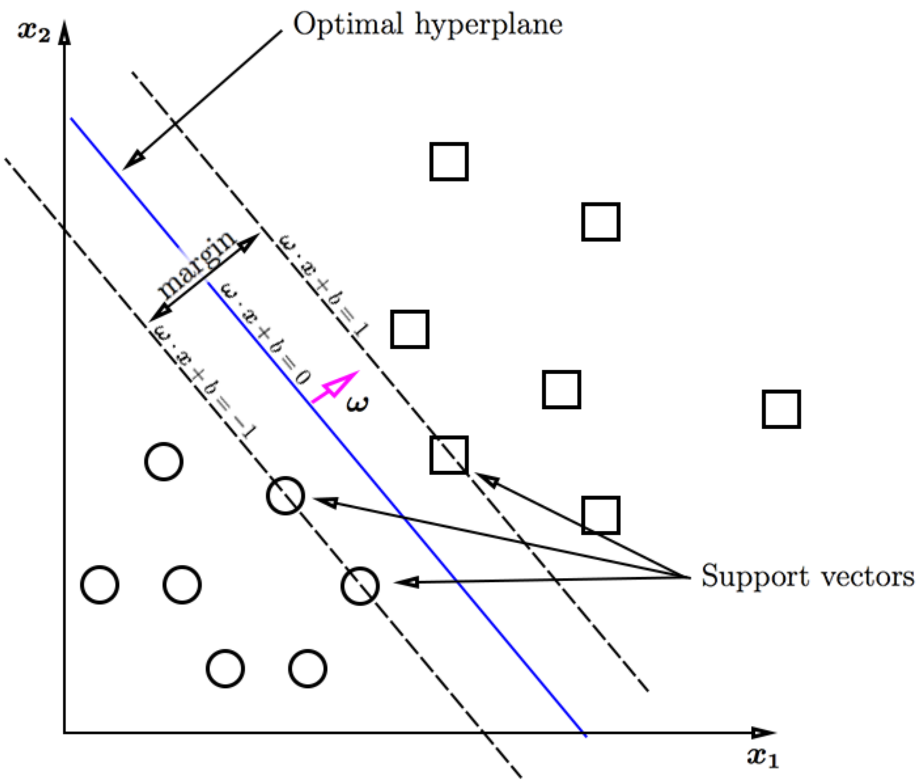


Figure 5.1: Maximum-margin hyperplane and margins for a linear SVM, trained with samples from two classes.

A given hyperplane represented by ($\boldsymbol{\omega}$,b) may be expressed by all pairs $\{\lambda\boldsymbol{\omega},\lambda b\}$ for $\lambda \in \Re^+$. So we define the canonical hyperplane as the one that separates the data from the hyperplane by a distance of at least 1 (in fact, we require that at least one example on both sides has a distance of exactly 1). This means that we consider those that satisfy:

$$\begin{aligned} \boldsymbol{x_i} \cdot \boldsymbol{\omega} + b \geq 1 \text{ when } y_i = 1 \\ \boldsymbol{x_i} \cdot \boldsymbol{\omega} + b \leq -1 \text{ when } y_i = -1 \end{aligned} \tag{5.6}$$

which is equivalent to

$$y_i(\boldsymbol{x_i} \cdot \boldsymbol{\omega} + b) \geq 1 \ \forall i \tag{5.7}$$

It's also true that for a given hyperplane $\boldsymbol{\omega}$,b), all pairs $\{\lambda\boldsymbol{\omega},\lambda b\}$ define the same hyperplane, but each has a different functional distance to a given data point. To obtain this geometric distance, we must normalize the hyperplane by the magnitude of $\boldsymbol{\omega}$, giving the distance:

$$d((\boldsymbol{\omega}, b), \boldsymbol{x_i}) = \frac{y_i(\boldsymbol{x_i} \cdot \boldsymbol{\omega} + b)}{\|\boldsymbol{\omega}\|} \geq \frac{1}{\|\boldsymbol{\omega}\|} \tag{5.8}$$

Intuitively, we want the hyperplane that maximizes the geometric distance to the closest data points. From equation 5.8 it's clear that this is accomplished by minimizing $\|\boldsymbol{\omega}\|$ (subject to the distance constraints). The main method of doing this is with Lagrange multipliers (see [13] [49] for derivation details), transforming the problem into:

$$\begin{aligned} &\text{minimize: } W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j (\boldsymbol{x_i} \cdot \boldsymbol{x_j}) \\ &\text{subject to: } \sum_{i=1}^{l} y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C \ \forall i \end{aligned} \tag{5.9}$$

where α is the vector of the l non-negative Lagrange multipliers to be determined and C is a constant.

Additionally, from the derivation of these equations, the optimal hyperplane can be written as:

$$\boldsymbol{\omega} = \sum_i \alpha_i y_i \boldsymbol{x_i} \tag{5.10}$$

which means that the vector $\boldsymbol{\omega}$ corresponds to a linear combination of the training examples.

It can also be shown that

$$\alpha_i(y_i(\boldsymbol{\omega} \cdot \boldsymbol{x_i} + b) - 1) = 0 \ \forall i \tag{5.11}$$

which means that when the functional distance of a certain example is greater than one, $y_i(\boldsymbol{\omega} \cdot x_i + b) > 1$), then $\alpha_i = 0$. This means that only the closest data points contribute to $\boldsymbol{\omega}$. These training examples for which $\alpha_i > 0$ are called support vectors, and they are the only ones needed in defining, and finding, the optimal hyperplane [5].

Assuming we have the optimal α from which we build $\boldsymbol{\omega}$, b is determined by

$$\begin{aligned} (\boldsymbol{\omega} \cdot \boldsymbol{x}^+ + b) &= 1 \\ (\boldsymbol{\omega} \cdot \boldsymbol{x}^- + b) &= -1 \end{aligned} \tag{5.12}$$

where $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are respectively the positive and negative support vectors. Solving these equations gives

$$b = -\frac{1}{2}(\boldsymbol{\omega} \cdot \boldsymbol{x}^+ + \boldsymbol{\omega} \cdot \boldsymbol{x}^-) \tag{5.13}$$

6 Experiments and Results

This chapter includes the presentation and discussion of the classification results obtained while testing the complete pipeline with a multiclass linear SVM and NBC, using the dataset presented in section 1.3 and labeled as shown in Fig. 6.1. As said before SVM performs binary classification. In order to achieve multi class classification, *libsvm* [8], which corresponds to the chosen open source library for the SVM algorithm used in this work, performs what is called "One vs All" [29] which involves training a single classifier per class.

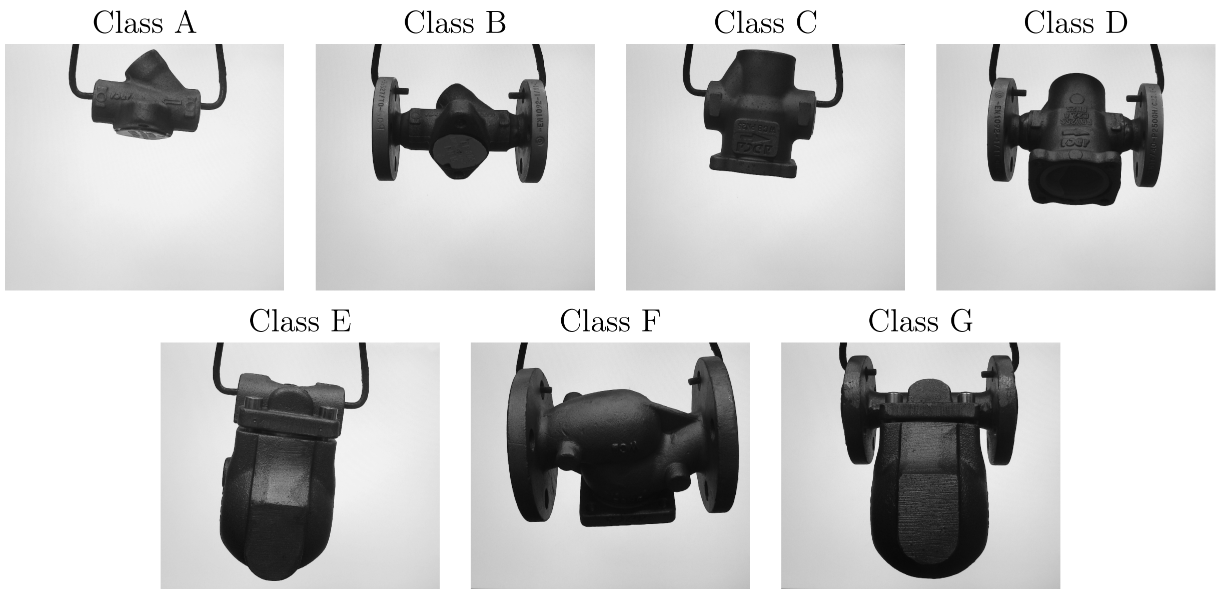


Figure 6.1: Dataset image examples with labels.

The pipeline will also be tested with both the geometric features, which will be called "blob features", and the features obtained from the HOG algorithm, or "HOG features". In order to give insight on how the models would generalize to an independent dataset as well as trying to limit problems like overfitting it was used 4-fold cross validation. This means that the dataset, composed by 100×7 images (100 images times 7 classes) in random order, is divided in 4 subsets, and in each of 4 trials, one of the subsets is used for training (25%, i.e. 25 images per class) and the other 3 subsets are put together to form a set for testing (75%, i.e. 75 images per class). The presented testing results are the average of all 4 trials and are returned as probability estimates. The reason for a smaller training set and larger testing set as opposed to what usually happens in k-fold cross validation, resides in the fact that we are trying to generalize for what would actually happen in reality, on a manufacturing facility. It's important to add that the classifier trains with a number of samples equally distributed between classes, preventing biases towards specific classes. Focusing on the overall performance of the classifiers it is made an analysis on precision, recall and F-measure for different thresholds values. In order to understand which images are being misclassified and to compare similarities between the classification results and the actual classes of the images, there will be presented confusion matrices followed by their analysis. The matrices are related to the sum of all four cross validation trials (75×4 images per class). The matrices were built for specific thresholds.

6.1 Blob Features

The features mentioned on section 4.1 allowed the assembly of a vector with a total size of 146 different features. From non numeric features; e.g. the bounding box, minimum enclosing circle and convex hull, there were extracted numeric features such as areas, circle radius, rectangle width and height among others. Suitable parameters where then chosen for the thresholds $t1$ and $t2$ used in algorithm 2 as well as the number of line segments N, which in this case ended up with the value $N = 36$. The reason for this value resides in the fact that in testing, greater values proved to results in redundancy, with some features presenting null variance. Lower values on the other hand showed lower classification performance.

6.1.1 NBC

Before proceeding with the presentation of the attained results for the application of a NBC on the extracted blob features it's important to firstly assess certain issues regarding the classifier. Equation 5.2 corresponds to the posterior probability function used in the NBC. This function, as shown before, is basically a likelihood function times a prior distribution, divided by a normalization element. The number of data points corresponds to the number of terms in the product. Those numbers are likelihood values, which range close to zero, and if enough of them are multiplied together, the result will be an awfully small number to represent in a floating point. This means the calculation will eventually underflow to zero. To avoid this problem sums of logarithmic probabilities are used instead of products of probabilities, transforming equation 5.2 in:

$$\log\left(f_i(E)\right) = P(C_i)\sum_{j=1}^{n}\log\left(P(A_j = v_{jk}|C_i)\right). \tag{6.1}$$

A second problem with the NBC, which is actually related with the previous one, resides in the fact that it has a really high tendency of favoring a specific class in detriment of others. This favoring tends to grow as the number of features increases, meaning that the difference between likelihoods ends up becoming prohibitive. As a result, the probability estimates end up being 1 for the class with the larger probability (rank) and 0 for the others, rather than what the actual probability is, which doesn't allow a thorough analysis of the results. In order to overcome this problem a regular Min-Max normalization strategy is applied to the values returned by equation 6.1. This results in more conservative probability estimates, which in turn allowed the creation of the following graphics:

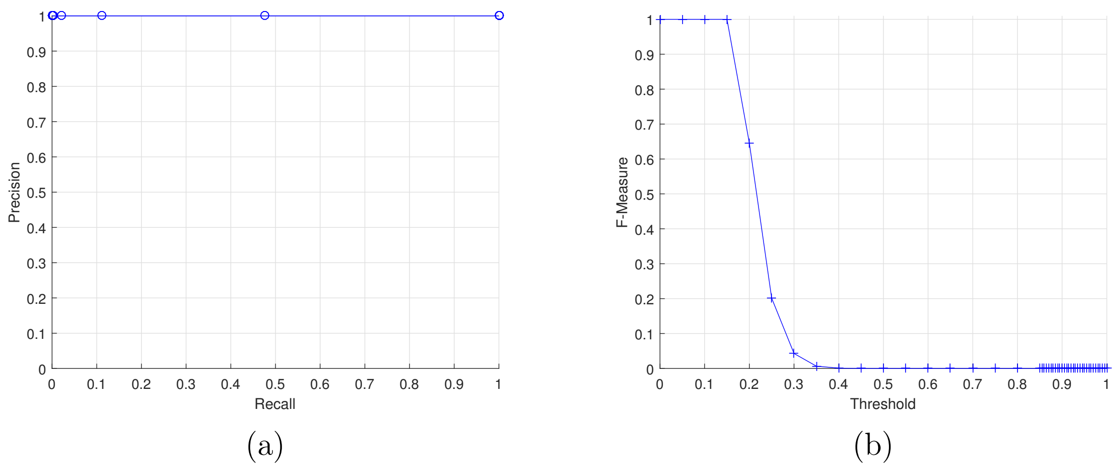
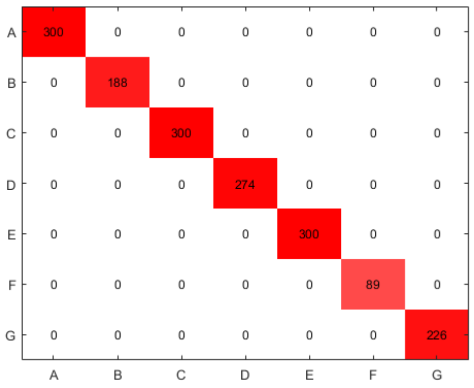


Figure 6.2: P-R curve (a) and F-measure curve by threshold (b) for a NBC using blob features.

From the results shown above, it's clear that the classifier preforms remarkably well in terms of precision since it has maximum value for all the threshold values, meaning that the classifier doesn't output false positive values. It does however start giving false negatives for thresholds above 0.15, which is the result of decrease in recall. After a 0.4 threshold the classifier has recall (i.e. true positive) equal to zero. Regularly the confusion matrix of a classifier is done for thresholds of 0 or 0.5, which in our case brings a dilemma on figuring out which threshold to choose (see Fig. 6.2(b)). In the case of 0 threshold the result is a confusion matrix with the main diagonal with the value 300 for each class, and 0 on the rest; otherwise for a threshold of 0.5, the matrix will only have null values. For a situation like this the solution does not reside in selecting the point for higher threshold since we do hit maximum precision and recall, i.e. maximum F-measure, at a certain point, and thus it would end up like the first case. For this reason it was decided to select a threshold value on a point where the curve is already start dropping. This allows the analysis of which classes are the first to lose true positives, and where did they get misclassified in case of false positives.



	A	B	C	D	E	F	G
A	300	0	0	0	0	0	0
B	0	188	0	0	0	0	0
C	0	0	300	0	0	0	0
D	0	0	0	274	0	0	0
E	0	0	0	0	300	0	0
F	0	0	0	0	0	89	0
G	0	0	0	0	0	0	226

Table 6.1: Confusion matrix for a NBC using HOG features, considering a threshold of 0.17.

By analyzing the confusion matrix for a threshold of 0.17, it's clear that the class where the classifier has worse performance is class F, followed by class B. However, we do see no false positive results which is expected since the classifier presents constant precision of 100% (see Fig. 6.2(a)).

6.1.2 SVM

The obtained results after the application of a linear SVM classifier on the blob features, are shown in the following graphics:

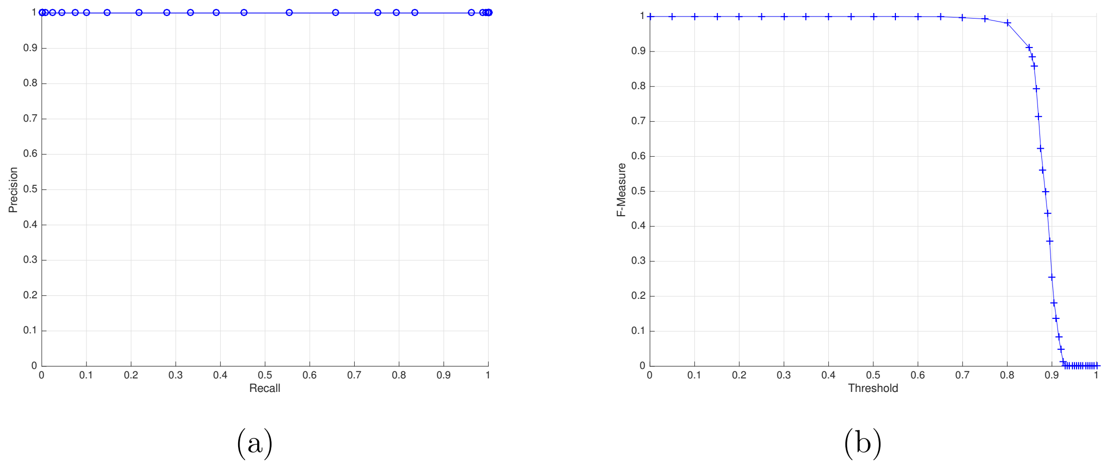
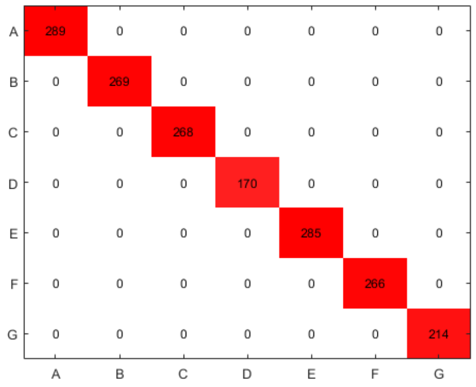


Figure 6.3: P-R curve (a) and F-measure curve by threshold (b) for a SVM using blob features.

The P-R Curve shows perfect precision results throughout all the threshold values even as the recall starts to fall. It also shows that the greatest accumulation of threshold values is close to 100% recall which is reflected in the F-measure curve. This curve presents maximum value for thresholds bellow 0.7 meaning that the the classifier hits in all the testing classes. After 0.7 the curve starts falling quite drastically until the threshold reaches it's maximum which means the classifier stops hitting all the set even though we continue with no false positive values (maximum precision). The F-measure eventually reaches null value moment in which the recall is also null.



	A	B	C	D	E	F	G
A	289	0	0	0	0	0	0
B	0	269	0	0	0	0	0
C	0	0	268	0	0	0	0
D	0	0	0	170	0	0	0
E	0	0	0	0	285	0	0
F	0	0	0	0	0	266	0
G	0	0	0	0	0	0	214

Table 6.2: Confusion matrix for a SVM using blob features, considering a threshold of 0.85.

The confusion matrix for a threshold of 0.85 shows presence of true negative results and absence of false positives which makes sense, once again, due to the constant precision of 100% (see Fig. 6.3(a)). In this case the class showing a bigger decrease in terms of performance is class D followed by class G.

6.2 HOG Features

6.2.1 NBC

In order to extract HOG features there are three key parameters that have to be settled, which correspond to the number of orientation bins for the histograms, the block sizes and the cell sizes. Since Piotr Dollár's implementation of the HOG descriptor is being used, the number of cells is fixed at half the size of a block, meaning that in each block a total of four histograms are computed. In order to determine the best block size we developed experiments using blocks of size 10×10 to 200×200 in increments of 10×10, calculating the F-measure values for different number of bins ranging between 1 an 4. The following step was the calculation of each of the areas under the curve (AUC) of the F-measure (see Fig. 6.4).

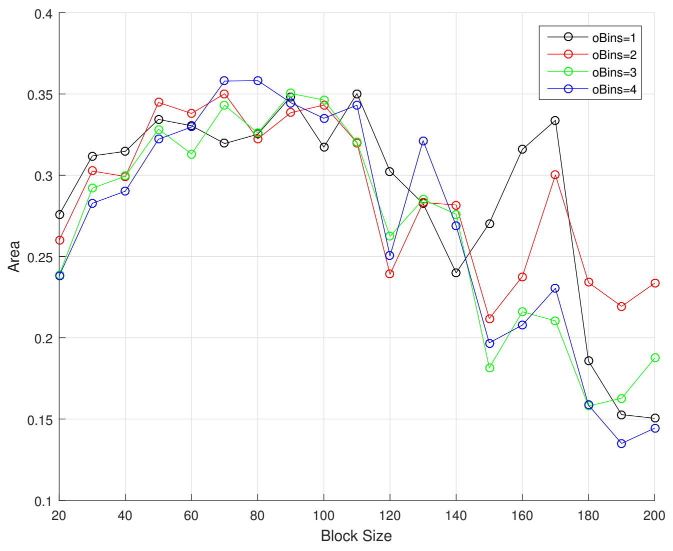


Figure 6.4: AUC of the F-Measure curve for a NBC in function of the increase in HOG block sizes considering different number of orientation bins.

From the analysis of the graphic above it was concluded that block sizes of 80×80 with 4 orientation bins per histogram consists in the best combination of parameters when using HOG features and a NBC, in this application. This means 8×6 blocks[1] and 16×12 cells for each image (each image has size 800×706). For each of this cells a histogram with 4 bins is computed. The results is a vector of 768 features for each image. It's important to note that this methodology is conditioned to the classifier. Other methods include the usage of a feature selector algorithm [25] to rank the features.

The graphic also allows the conclusion that smaller and larger sized blocks tend to worsen the results. This might have to do with the fact that, since each blob occupies a large portion of its corresponding image, then a greater area of the image might be needed in order to gather meaningful information. Thus, smaller sized blocks result in too few pixels per cell for a relevant histogram. On the other hand, if the blocks are too large, then we end up losing too much of the spatial information.

The following is an image with graphic representations of the attained results for the NBC with the optimal values previously referred (see Fig. 6.5).

[1]Consult [18] for more information on Piotr Dollár's implementation of HOG

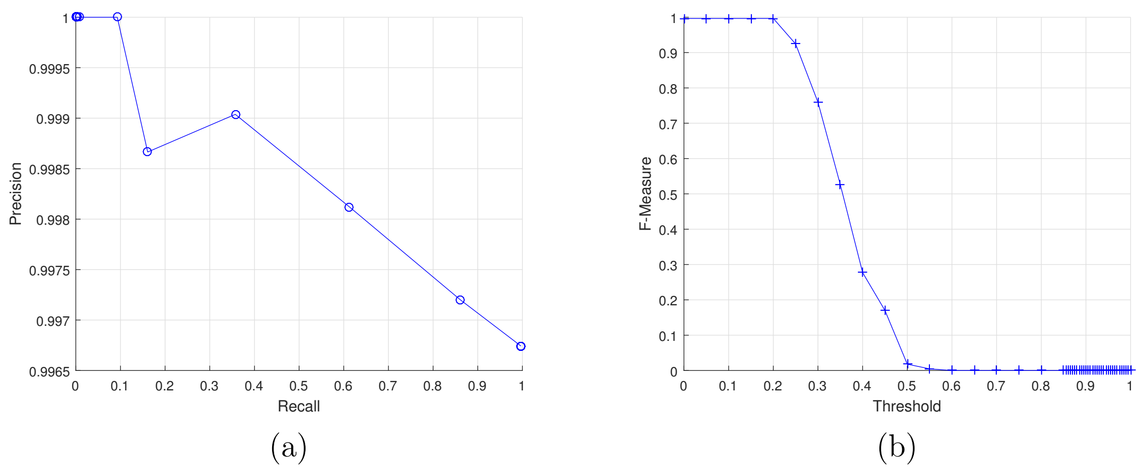
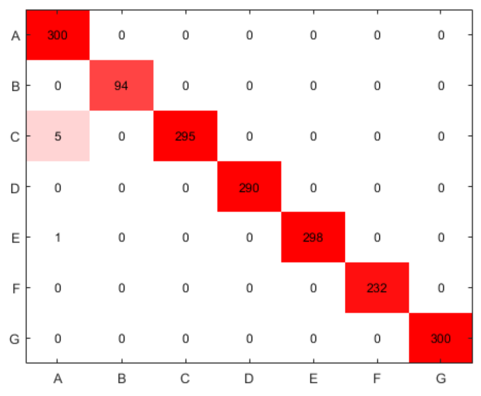


Figure 6.5: P-R curve (a) and F-measure curve by threshold (b) for a NBC using HOG features.

The graphics show that in the moment which the classifier has it's higher precision, with values extremely close to 100%, the recall is close to null. It's also visible that recall values drop quite drastically for thresholds above 0.20, which by result drops the F-measure value. The precision values reduce a little as recall increases indicating that a few false positive values start to appear. From 0.60 threshold forward the classifier gives no true positive results.



	A	B	C	D	E	F	G
A	300	0	0	0	0	0	0
B	0	94	0	0	0	0	0
C	5	0	295	0	0	0	0
D	0	0	0	290	0	0	0
E	1	0	0	0	298	0	0
F	0	0	0	0	0	232	0
G	0	0	0	0	0	0	300

Table 6.3: Confusion matrix for a NBC using HOG features, considering a threshold of 0.25.

The referred false positives are reflected in the confusion matrix shown in Fig. 6.3, as expected. In this case the classifier misclassifies 5 images of class C, labeling them as being part of class A. It also has some problems classifying classes as B, probably due to similarities between both classes B and D.

6.2.2 SVM

For the SVM classifier we used the same process for determining the optimal HOG parameters (see Fig. 6.6).

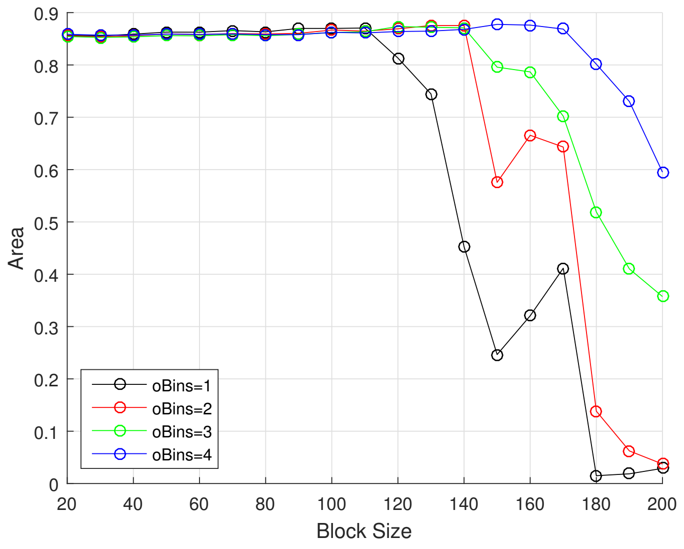


Figure 6.6: AUC of the F-Measure curve for a SVM in function of the increase in HOG block sizes considering different number of orientation bins.

The F-measure area shows a tendency to maintain rather steady for block sizes bellow 110 × 110 pixels. From that point on the curve that represent 1 orientation bin starts dropping, followed by the curves that represent 2 and 3 orientation bins respectively. The curve representing 4 orientation bins has a slight different behavior, continuing a slow increase in area value before reaching it's maximum point corresponding to a block size of 150 × 150. After reaching this point the curve drops with less intensity than the rest. With this results in mind, it's concluded that 4 orientation bins and block sizes of 150 × 150, are the optimal block size and number of orientation bins for this problem. This translates in 3 × 2 blocks and 6 × 4 cells in each image, and thus vectors with 96 features per image.

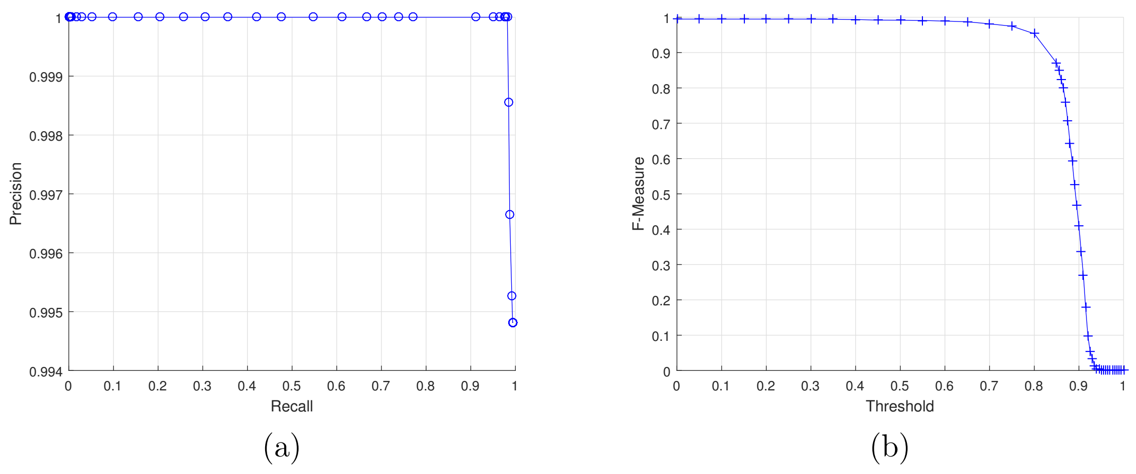
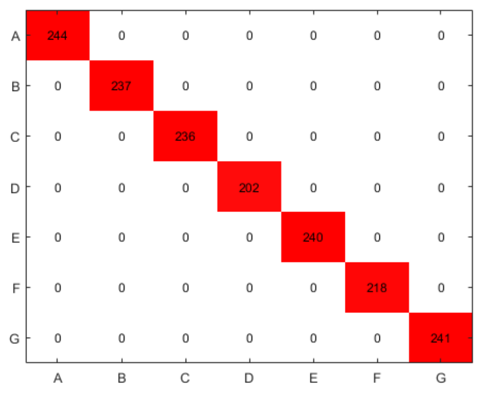


Figure 6.7: P-R curve (a) and F-measure curve by threshold (b) for a SVM using HOG features.

Fig. 6.7 represents graphical results for the SVM classifier using HOG features with the optimal parameters. The P-R curve shows once again tremendous precision, in this case maximum, for close to all the recall values. However, the recall is never 1 which translates to a F-measure curve that also doesn't reach that value. The classifier manages to maintain good values until it reaches thresholds of about 0.6%, starting and exponential fall due to a great increase on false negative values, and stoping at around 0.93 where we start getting no true positives whatsoever.



	A	B	C	D	E	F	G
A	244	0	0	0	0	0	0
B	0	237	0	0	0	0	0
C	0	0	236	0	0	0	0
D	0	0	0	202	0	0	0
E	0	0	0	0	240	0	0
F	0	0	0	0	0	218	0
G	0	0	0	0	0	0	241

Table 6.4: Confusion matrix for a SVM using HOG features, considering a threshold of 0.85.

For a threshold of 0.85 the classifier is still able to correctly determine most of the images (see Table.6.4). On top of that, it continues presenting no false positives meaning that for the images its able to classify, there are no false predictions. The classifier shows a tendency for not being able to classify as much images of class *D* as it does the others, which is similar with what happened to the SVM when using blob features. This might have to do with the fact that class *D* and *B* have a lot of similarities which means that the classifier might have a slightly lower certainty when classifying class D.

6.3 Discussion

From the developed tests using blob features, the SVM classifier proved to be much more robust than the NBC, which is reflected by a much higher degree of certainty while presenting true positive results. This is the main reason why the F-measure curves are presented in function of threshold, since they allow an easier interpretation of the actual values for which the classifiers start worsening their results, thus allowing a more straightforward comparison between classifiers. It is important to mention that even though the NBC is less robust it does give good results in terms of decision, so long as the correct class is the one more probable than all the others. What this means is that, regardless of whether the probability estimate is slightly or even exceptionally inaccurate, the classifier can be robust enough even under its underlying naive probability model. However, the NBC also proved its drawbacks, in comparison with the SVM, in terms of computation speed and resource consumption due to the numeric computations involved in the NBC. When training the HOG features with small block sizes and 4 bins, the NBC took almost teen times more time to test when compared to the SVM, using all the RAM storage resources of the machine.

The SVM classifier also showed better results when using HOG features. From the moment we choose the HOG parameters for both classifiers, we start seeing big contrast between the SVM's ability of managing much higher block sizes than the NBC. Meaning that the SVM is able to give better results with much less features, 96 in comparison with the NBC's 768, which translates in a lower degree of complexity involved in the classification, a thus lowering the tendency of affecting generalization and reducing the change of overfit. The SVM also shows greater precision and recall for higher threshold values, meaning once again that it's more robust than the NBC. As the NBC starts lowering it's stats for thresholds above 0.2 the SVM continues with good results, only starting it's drop, due to a increase in false negatives, as it reaches thresholds of 0.6. For this reasons, the SVM seems a better choice of a classifier.

The next step consists in the comparison between both blob and HOG features, in order to determine which group of extracted features is best on solving the problem discussed in this work (see Fig. 6.8).

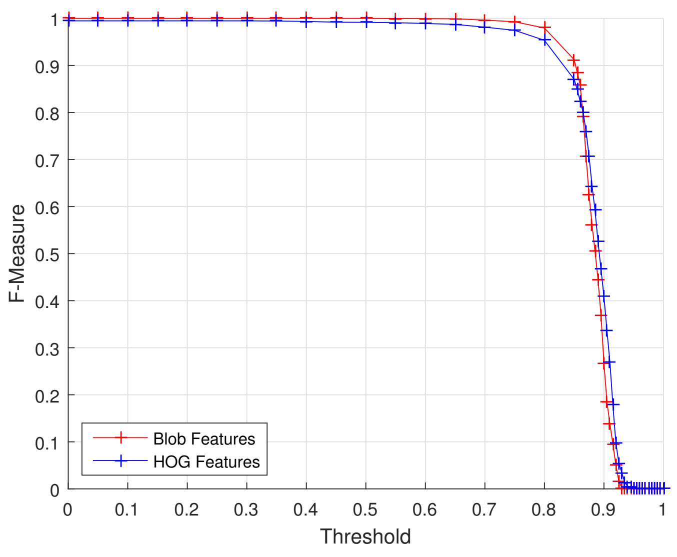


Figure 6.8: F-measure curve by threshold for a SVM using HOG and Blob features.

The previous image shows the combined graphics which represent the F-measure curves while using a SVM classifier with both HOG and blob features. Straight away it's visible that both curves maintain high F-measure values for much of the threshold range. However, the curve related with the HOG features tends to keep slightly worse values than the one related with the blob features, which has to do with the output of a few false negative results. As the threshold reaches 0.6 the curve representing the HOG features start falling followed by the the one representing the blob features at around $0.65 - 0.7$. Both curves fall with similar slopes. The P-R curves shown in Fig. 6.3(a) and 6.7(a) display another interesting for comparison, which is related with the fact that the HOG features are never able to present SVM results with maximum recall for all thresholds contrary to what happens with the blob features. Therefore the SVM achieves a better performance when aligned to the blob features.

7 Conclusions

Throughout this work a system able to accurately distinguish each class of the dataset was developed, following with stringency each step of an automated machine vision pipeline, using machine learning. In the image preprocessing stage, feature processing techniques were used to successfully create binary images. This techniques were also implemented to extract most of the noise created by reflections on the surface of the products as well as blur. A well performed system capable of removing the majority of the claws that hold each product was also implemented, allowing the isolation of the same in the form of a blob. Regarding feature extraction two different methods were implemented, firstly using basic features based on the blobs shape and overall geometric characteristics, followed by a HOG implementation. Each of the two feature extraction techniques were then used on the post-processing images and combined with a multiclass SVM and NBC for classification. This corresponded to the final stage of the pipeline.

During the course of this project several important conclusions were attained. When testing both SVM (discriminative classifier) and NBC (generative classifier) it was concluded that that, even though the NBC gives good results regarding decision, it's far away from the robustness of a SVM. This results are identical to the theoretical predictions found in the literature regarding de performance of generative versus discriminative classifiers [34], which indicate that generative algorithms should initially do better as the number of training examples increases, but for discriminative algorithms to eventually catch up and likely overtake the performance quite easily. The NBC is also far slower than the SVM classifier, especially as the feature vector size grows, which in a industrial environment, where quick decisions matter, is not ideal. Regarding the choice between HOG and blob features, the results turned out slightly better, in terms of performance, in favor of the blob features. The blob features showed results with 100% precision and recall for all possible threshold values, outclassing the state of the art HOG descriptor which presented marginally worse results. This means the blob features would ultimately be the best solution out of both. However, it is important to mention that the feature vector corresponding to the blob features is slightly bigger then the one corresponding to the HOG features that ended up performing well for blocks of size 150×150 and 4 orientation bins. The sizes correspond to 146 and 96 respectively for the blob and HOG features, meaning a slightly lower complexity in the classification with the HOG features. A solution for this problem is a feature selector algorithm, which might help remove possible redundant features, decreasing the blob feature vector's size.

In conclusion, this work presents a system able of classifying steam trap and valve components on a transportation conveyor of a painting cabinet with extremely high accuracy, which is a huge step in the right direction in terms of creating a autonomous painting station.

In the future one can investigate the possibility of using a feature selection (FS) technique, such as Principal Component Analysis (PCA) [33]. One can also search for different discriminatory features of valves and steam traps that could help improve even more the performance of the classifiers, or even test both HOG and blob features together and look for improvements after applying the FS. It would also be interesting to test the system in a real time scenario inside the manufacturing facilities to see the results in practice.

8 Bibliography

[1] Mark S Nixon; Alberto S Aguado. *Feature extraction and image processing.* Academic press, 2nd ed edition, 2008.

[2] V. Ashok and D. S. Vinod. Automatic quality evaluation of fruits using probabilistic neural network approach. 2014.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[4] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128, 2006.

[5] Dustin Boswell. Introduction to support vector machines.

[6] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[7] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[8] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[9] Chin-Chen Chang, SM Hwang, and Daniel J Buehrer. A shape recognition scheme based on relative distances of feature points from the centroid. *Pattern Recognition*, 24(11):1053–1063, 1991.

[10] Heping Chen, Thomas Fuhlbrigge, and Xiongzi Li. Automated industrial robot path planning for spray painting process: a review. In *2008 IEEE International Conference on Automation Science and Engineering*, pages 522–527. IEEE, 2008.

[11] Tiejian Chen, Yaonan Wang, Changyan Xiao, and QM Jonathan Wu. A machine vision apparatus and method for can-end inspection.

[12] M. M. Chetima and P. Payeur. Automated tuning of a vision-based inspection system for industrial food manufacturing. *Instrumentation and Measurement Technology Conference (I2MTC), 2012 IEEE International*, pages 210–215, 2012.

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[14] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[15] Babilla Borine D'Angelo and Jean Resende. Automation and control of automotive painting process using the rfid technology automatização e controle do processo de pintura automotiva utilizando a tecnologia rfid. Technical report, SAE Technical Paper, 2007.

[16] E Roy Davies. *Computer and machine vision: theory, algorithms, practicalities.* Academic Press, 2012.

[17] Alexandre R de Mello and Marcelo R Stemmer. Inspecting surface mounted devices using k nearest neighbor and multilayer perceptron. In *2015 IEEE 24th International Symposium on Industrial Electronics (ISIE)*, pages 950–955. IEEE, 2015.

[18] Piotr Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). `https://github.com/pdollar/toolbox`.

[19] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3):103–130, 1997.

[20] Kevin J Dowling, George G Mueller, and Ihor A Lys. Systems and methods for providing illumination in machine vision systems, May 9 2006. US Patent 7,042,172.

[21] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification.* John Wiley & Sons, 2012.

[22] Nick Efford. *Digital image processing: a practical introduction using java (with CD-ROM).* Addison-Wesley Longman Publishing Co., Inc., 2000.

[23] Rafael C Gonzalez and Richard E Woods. *Digital image processing.* 2008.

[24] Lalit Gupta and Mandyam D Srinath. Contour sequence moments for the classification of closed planar shapes. *Pattern Recognition*, 20(3):267–272, 1987.

[25] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[26] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004), 2004.

[27] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[28] Kurt Hecht. Integrating led illumination system for machine vision systems, March 29 2005. US Patent 6,871,993.

[29] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425, 2002.

[30] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE transactions on information theory*, 8(2):179–187, 1962.

[31] Anil K Jain. *Fundamentals of digital image processing.* Prentice-Hall, Inc., 1989.

[32] Ramesh Jain, Rangachar Kasturi, and Brian G Schunck. *Machine vision*, volume 5. McGraw-Hill New York, 1995.

[33] Ian Jolliffe. *Principal component analysis.* Wiley Online Library, 2002.

[34] A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

[35] Christos Kampouris, Ioannis Mariolis, Georgia Peleka, Evangelos Skartados, Andreas Kargakos, Dimitra Triantafyllou, and Sotiris Malassiotis. Multi-sensorial and explorative recognition of garments and their material properties in unconstrained environment. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1656–1663. IEEE, 2016.

[36] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[37] Andrew McCallum, Kamal Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[38] Roberto Medina, Fernando Gayubo, Luis M González-Rodrigo, David Olmedo, Jaime Gómez-García-Bermejo, Eduardo Zalama, and José R Perán. Automated visual classification of frequent defects in flat steel coils. *The International Journal of Advanced Manufacturing Technology*, 57(9-12):1087–1097, 2011.

[39] Babu M Mehtre, Mohan S Kankanhalli, and Wing Foon Lee. Shape measures for content based image retrieval: a comparison. *Information Processing & Management*, 33(3):319–337, 1997.

[40] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[41] P. Prasanna, K. J. Dana, N. Gucunski, B. B. Basily, H. M. La, R. S. Lim, and H. Parvardeh. Automated crack detection on concrete bridges. *IEEE Transactions on Automation Science and Engineering*, 13(2):591–599, 2016.

[42] S Ravikumar, KI Ramachandran, and V Sugumaran. Machine learning approach for automated visual inspection of machine components. *Expert systems with applications*, 38(4):3260–3266, 2011.

[43] John C Russ. *The image processing handbook*. CRC press, 2016.

[44] Nashlie Sephus, Sravan Bhagavatula, Palash Shastri, and Erica Gabriel. An industrial-strength pipeline for recognizing fasteners. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 781–786. IEEE, 2015.

[45] Hao Shen, Shuxiao Li, Duoyu Gu, and Hongxing Chang. Bearing defect inspection based on machine vision. *Measurement*, 45(4):719–733, 2012.

[46] Thomas Sikora. The mpeg-7 visual standard for content description-an overview. *IEEE Transactions on circuits and systems for video technology*, 11(6):696–702, 2001.

[47] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[48] S-H Suh, I-K Woo, and S-K Noh. Development of an automatic trajectory planning system (atps) for spray painting robots. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1948–1955. IEEE, 1991.

[49] Vladimir Vapnik. *The nature of statistical learning theory*. Springer Science & Business Media, 2013.

[50] Greg Welch and Gary Bishop. An introduction to the kalman filter, 2001.

[51] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[52] Mingqiang Yang, Kidiyo Kpalma, and Joseph Ronsin. A survey of shape feature extraction techniques. *Pattern recognition*, pages 43–90, 2008.

[53] M. Yao, M. Liu, and H. Zheng. Exterior quality inspection of rice based on computer vision. *World Automation Congress (WAC), 2010*, pages 369–374, 2010.

[54] Si yuan Wu, Zhao Zhang, and Liang Feng. Statistical feature representations for automatic wood defects recognition research and applications. 2009.

[55] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76, 1997.