UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI
CORSO DI LAUREA TRIENNALE IN INGEGNERIA MECCANICA E MECCATRONICA

TESI DI LAUREA

ROBUST VISION-BASED 3D LOCALIZATION OF PLANAR OBJECTS FOR INDUSTRIAL BIN-PICKING SYSTEMS

Relatore: Ch.mo Prof. ENRICO PAGELLO

Correlatore: Dott. Ing. ALBERTO PRETTO

Laureando: NICOLA COVALLERO

Matricola 1023586-IMC

ANNO ACCADEMICO 2013-2014

Abstract

This project presents an algorithm for planar objects detection for Bin-Picking systems with a single 2D camera. Bin-Picking systems are automated systems that have to hold and drop objects from a bin where these objects are randomly placed, to, for example, a conveyor belt, or must simply hold them for other possible manufacturing processes. These systems allow to automatize the process to grasp objects that come from a certain manufacturing process and locate them for the next manufacturing process that they have to undergo.

The present project is a reimplementation of the algorithm presented by Pretto et al. [1]. In particular, a different approach for the object detection stage has been implemented.

The algorithm presented here is a robust and flexible vision system for a 3D localization of planar objects. The system of Pretto et al. is able to work with nearly any planar object randomly placed in a bin or in a conveyor belt. The system has 6 degrees of freedom, is able to detect and handle planar objects not only placed on an ideal layer but also when they are sloping.

The important innovation with respect to other Bin-picking systems is the use of a single 2D camera, differently from the majority Bin-Picking systems that use 3D cameras, or laser triangulation systems, or laser range finders, that make the price of the systems to grow up a lot. The presented algorithm can work with any planar objects, with a thickness from some millimetres up to several centimetres, without handle parameters, since the only effort required is to give as input the CAD model of the object that the user is looking for. The localization process is based on a two step strategy:

- a candidates selection
- refinements to get the best matches

In this paper the attention is focused on both points of the procedure, precisely the candidates selection is quite similar to the one used in [1], while the refinements have been made with a different algorithm. Instead of using a Hough Like voting scheme to match the objects the concept of the *chamfer distance* [2] has been used.

At the end of the development, the new implementation, that use the standard chamfer distance to detect objects, showed to be really robust and quite fast, and it could be considered for a implementation of a real Bin-Picking system. However it is bit slower than the original one presented by Pretto et al.

Contents

List of Figures

Chapter 1

Introduction

One of the key challenge in automated robot-aided manufacturing is the capability to automatically identify and locate objects that the robot can grasp and handle in an accurate way. Generally parts are randomly placed in a bin therefore it is required a sophisticate system to locate searched objects, this perception task is well-known as the "bin-picking" problem. It is easy to understand how much this is important for robot manufacturing and that is why there are a lot of studies about this, hence the interest of the thesis in this subject.

Basically the algorithm works on a robot aided to handle objects for different purpose, the main stage of the robot is to grasp and manipulate searched objects. Getting the model CAD of the searched object, it can locate the objects in a few seconds from a 2D image taken by a 2D industrial camera, then the robot will grasp the object and handle it for different purposes. A configuration of a Bin-Picking system, by Pretto et al. [1], is showed in figure Fig. 1.1.

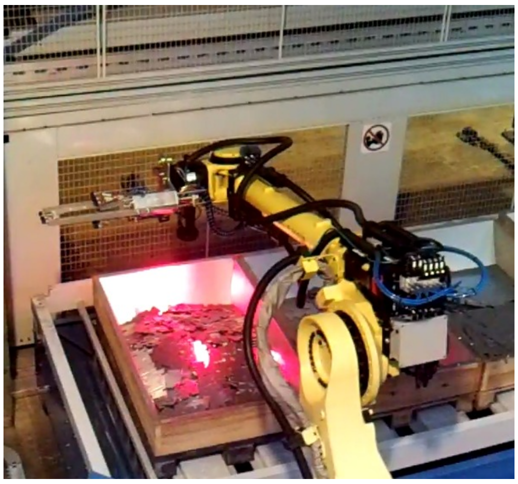


Fig. 1.1: Configuration of a Bin-Picking system with the camera mounted on the robot arm. (Image taken from [1])

This system uses a single 2D high resolution camera while the most Bin-Picking systems use other systems, such as 3D high definition camera, laser range finders or laser triangulation systems. These systems offer the big advantage of locating really precisely objects, which is the main goal, but the use of this systems lead to rise the price of the single robot. In fact, the cost of these devices for detection is from 3 to 10 times higher than a conventional high resolution industrial camera and need to be moved. The devices of moving are expensive and the detection of thin planar objects, usually disposed in smooth hills, imposes a high precision in depth estimation that requires very expensive sensors.

In order to present a cheaper Bin-Picking system a 2D approach has been thought to use. Moreover, since the science of computer vision is quite young, there are a lot of studies published every year about object detection that can make reliable an object detection even with a 2D camera. The solution of using a 2D camera provides a reliable, cost-effective and less invasive system to be installed in existing robotic cells, making this the best choice in some cases, such as the detection of planar objects that have a thickness starting from $0.5mm$ up to several centimetres.

The whole work here presented is based on the work of Pretto et al. [1]. My study was focused on almost the whole process with the main goal to introduce with, Alberto Pretto himself, an other technique for planar object detection.

Her the procedure of the work of Pretto et al. is presented. First the localization starts smoothing the noise introduced by sunlight or artificial lights using a High Dynamic Range image (HDR) [3] and then a tonemapping process [11]. Later through a highly engineered edge-based vision algorithm called LSD [12] (Line Segments Detector) the algorithm locates edges in the 2D image, what will be applied on a scaled version of the input image in order to reduce false detections due to the image noise. Later, a list of objects pose hypothesis is collected. For each hypothesis they extract the rigid body transformation that relates the object frame with the camera frame. Then, they use the Generalized Hough Transform to evaluate the match, and for each hypothesis they perform a coarse-to-fine registration based on a constrained optimization procedure that exploits specific image gradient based cost-functions. The best match is selected among all registered best matches using a robust gradient direction-based scoring function. Obviously the whole process is iterated until no matches are found.

My study focuses on the first part of High Dynamic Range and Tonemapping, and on the second part of the LSD algorithm for edge detection. Moreover, the study focuses on the Hough Generalized Transform and *Chamfer Distance* [2] for object detection with the main goal of replacing the Hough-like voting scheme with a *chamfer distance* voting scheme. To evaluate the matches the same scoring function used in [1], based on gradients direction, has been used. Lastly some improvements that can make the *chamfer distance* voting scheme faster and more precise will be commented.

The whole algorithm has been developed using several libraries that will be commented, but the most used is the open source library OpenCV [10], that offers a lot of functions about computer vision, especially with the last version OpenCV 3.0.0 published in October 2013. The work has been always matched with a lot of tests, the most meaningful of them will be reported and commented.

All the elapsed times and results commented in this thesis refer to a **C++** implementation in a **2.2 GHz** machine with a **dual-core processor**.

Chapter 2

High Dynamic Range and Tonemapping

2.1 Introduction

Today digital cameras have a limited dynamic range, using 8 bits per channel, which is lower than one encounters in the real world. This limits the dynamic range of the device to two orders of magnitude (actually 256 levels), while human eye can adapt to lighting conditions varying by ten orders of magnitude. In high dynamic range scenes, a picture will often turn out to be under or over exposed.

High Dynamic Range (HDR) imaging works with images that use more than 8 bits per channel (usually 32-bit float values), allowing much wider dynamic range.

Basically the HDR images let to take off all annoyances due to presence of sunlight, or an artificial light, or the absence of these ones in the main scene. In this section is described the whole process of HDR imaging, why there are annoyances due to the lights and how HDR imaging can take them off.

The need to use a HDR image is due to the fact that when we take a photo of a scene with some lights, for example sunlight or artificial lights, there are some parts of the scene that are exposed to a stronger irradiance than the others that camera can not recognize; the result is an image with some parts saturated, that are brighter than other ones. A typical example is a scene with a window, such as the one in Fig. 2.1, where ,at a certain exposure time, zones of the scene that are far from the windows have an irradiance value quite corresponding to the real one but the ones near the windows have values much higher than the real ones, since they are saturated.

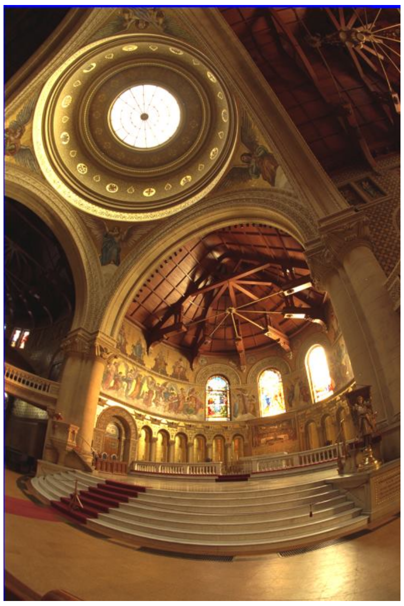


Fig. 2.1: Example of an image of a scene with sunlight, the sunlight produces annoyances near windows

A bracketed exposure sequence allows to acquire the full dynamic range, and can be turned into a single high dynamic range image. In this chapter will be presented a method to use these photographs to get a HDR image. Upon display, the intensities need to be remapped to match the typically low dynamic range of the display device, through a process called *tone mapping*.

HDR images find usefulness in some modern applications such as: image-based modelling and rendering, image processing, image compositing. In the particular the most image processing operations, such as blurring, edge detection, color correction, and image correspondence, expect pixel values to be proportional to the scene radiance. Because of non-linear image response, especially at the point of saturation, these operations can produce incorrect results for conventional images. Hence makes the use of HDR image fundamental in this project.

2.2 Image Aquisition

To understand because sometimes working with LDR images leads to have noise in the image is important to understand how the whole process of taking a photo works, a summary of the process is explained in the paper [3].

When we photograph a scene, either with film or an electronic imaging array, and digitize the photograph to obtain a two dimensional array of "brightness" values, these values are rarely true measurements of relative radiance in the scene. For example, if one pixel has twice the value of another, it is unlikely that it observed twice the radiance. Instead, there is usually an unknown, non-linear mapping that determines how radiance in the scene becomes pixel values in the image.

This non-linear mapping is hard to know beforehand because it is actually the composition of several non-linear mappings that occur in the photographic process. In a conventional camera (Fig. 2.2), the film is first exposed to light to form a latent image. The film is then developed to change this latent image into variations in transparency, or density, on the film.

The film can then be digitized using a film scanner, which projects light through the film onto an electronic light-sensitive array, converting the image to electrical voltages. These voltages are digitized, and then manipulated before finally being written to the storage medium.

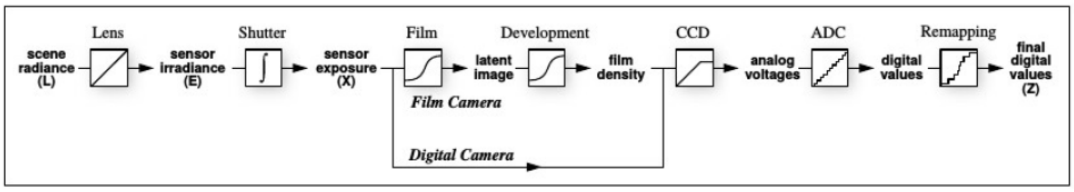


Fig. 2.2: Image aquisition of a Digital Camera (from [3])

In the first stage of the process, the film responses to variations in exposure X (the quantity of light reaching the film, which is $E\Delta t$, the product of the irradiance E that the film receives and the exposure time Δt), that is a non-linear function, called the "characteristic curve" of the film. Noteworthy in the typical characteristic curve is the presence of a small response with no exposure and saturation at high exposures.

The development, scanning and digitization processes usually introduce their own non-linearities which compose to give the aggregate non-linear relationship between the image pixel exposures X and their digitized values Z. Digital cameras, which use charge coupled device (CCD) arrays to image the scene, are prone to the same difficulties. Although the charge collected by a CCD element is proportional to its irradiance, most digital cameras apply a non-linear mapping to the CCD outputs before they are written to the storage medium.

This non-linear mapping is used in various ways to mimic the response characteristics of film, anticipate non-linear responses in the display device, and often to convert 12 bit output from the CCD's analog-to-digital converters to 8-bit values commonly used to store images. As with film, the most significant non-linearity in the response curve is at its saturation point, where any pixel with a radiance above a certain level is mapped to the same maximum image value.

The obvious difficulty is of limited dynamic range one has to choose the range of radiance values that are of interest and determine the exposure time suitably. Sunlight scenes, and scenes with shiny materials and artificial light sources, often have extreme differences in radiance values that are impossible to capture without either under-exposing or saturating the film.

2.3 Algorithm

As said before, the High Dynamic Range image of a certain scene can be constructed starting from a series of photographs with different exposure times, but how can we combine them into a composite radiance map?

In this project has been seen two ways to get an HDR with two different algorithms:

- Debevec [3]
- Exposure Fusion, so called *Merge Mertens* [13]

The next step, since a HDR image uses more than 8 bits per channel, the HDR image obtained has to be handled in order to display and treat it as a normal image, so called Low Dynamic Range image (LDR), through a tone mapping process.

Merge Mertens method actually does not create a HDR image but directly creates a LDR image with the same result of a tone-mapped HDR image, that is to say it takes light annoyances off. But obviously the two methods presented do not give the same LDR image in output.

2.3.1 Debevec and Tone Mapping

Since the camera response is the non-linear function that produces the non-linear mapping of the scene, it has to be found, trough an algorithm explained briefly in this section, in order to allow to reconstruct the image of the scene with a linear film response. Once the film response is known constructing the High Dynamic Range Radiance map is easy.

Camera Response Once a photo has been taken with a known exposure time Δt, its development gives us a value Z for each pixel of the image and the value Z is a non-linear function of the original exposure X of the pixel, defined as the product $E\Delta t$. Let's call this non-linear function f which is the composition of the characteristics curve of the film as well as all the non-linearities introduced by the later processing steps.

Knowing the function f it is possible to compute the original exposure X of each pixel simply using the inverse function f^{-1} :

$$X = f^{-1}(Z)$$

Debevec et al. make the assumption that the function f is monotonically increasing, so its inverse is well defined. Now that are known the exposure time Δt and the exposure X the irradiance E is given simply by:

$$E = \frac{X}{\Delta t}$$

It is important to highlight that this irradiance value should be weighted with the spectral response at the sensor site, this because the sensor weights the exposure value according to its spectral response. Therefore the irradiance E can be taken as proportional to radiance L of the scene.

They make the assumption that the scene is static and that this process is completed quickly enough that lighting changes can be safely ignored. Then it can be assumed that the film irradiance values E_i for each pixel i are constant. As input of the algorithm there are a number of digitized photographs with an exposure time Δt_{ij}. The subscript i refers to pixels while the subscript j refers to exposure time Δt_j . So the film reciprocity equation is:

$$Z_{ij} = f(X) = f(E_{ij}\Delta t_j)$$

Since has been made the assumption that f is monotonic it is invertible, therefore:

$$f^{-1}(Z_{ij}) = E_{ij}\Delta t_j$$

and taking the natural logarithm to both sides:

$$\log f^{-1}(Z_{ij}) = \log E_{ij} + \log \Delta t_j \tag{2.3.1}$$

The unknowns terms are the irradiance E and the inverse function f^{-1}. Is possible to recover the function f^{-1} and the irradiance E of the equation 2.3.1 in a least-squared error sense. Recovering g, where $g = \log f^{-1}$, only requires recovering the finite number of values that $g(Z)$ can take, since the domain of Z, pixel brightness values, is finite. Letting Z_{min} and Z_{max} be the least and greatest pixel values (integers), N be the number of pixel locations and P be the number of photographs, they formulate the problem as one of finding the $(Z_{max} - Z_{min} + 1)$ values of $g(Z)$ and the N values of $\ln E_i$ that minimize a quadratic objective function (for more details refer to [3]).

Constructing the High Dynamic Range Radiance Map Once the response function is calculated, from the equation 2.3.1 is possible to know the true value of the irradiance in the pixel i with the exposure time j:

$$\ln E_{ij} = g(Z_{ij}) - \ln \Delta t_j$$

To robust the algorithm Debevec et al. consider all the exposure times and weight the irradiance values taken from different exposure times with a weighting function.

Combining the multiple exposures has the effect of reducing noise in the recovered radiance values and reduces the effects of imaging artefacts such as film grain.

Tone Mapping Tone mapping is a strong technique for the display of high-dynamic-range images, which reduces the contrast while preserving detail. It is based on a two-scale decomposition of the image into a base layer, encoding large-scale variations, and a detail layer. Only the base layer has its contrast reduced, thereby preserving detail. The base layer is obtained using an edge-preserving and noise-reducing filter called the bilateral filter. This is a non-linear filter, where the weight of each pixel is computed using a Gaussian in the spatial domain multiplied by an influence function in the intensity domain that decreases the weight of pixels with large intensity differences. For more details about tone mapping refer to [11].

2.3.2 Merge Mertens

The method of Mertens et al. [13] consists of skipping the step of computing a high dynamic range image, and immediately merges the multiple exposures image into a high-quality low dynamic range image, ready for display (like a tone-mapped picture). This avoids camera response curve calibration and it is computationally efficient.

The idea behind this approach is the computing of perceptual quality measures for each pixel in the multi-exposure sequence, which encode desirable qualities, like saturation and contrast. Guided by these quality measures, "good" pixels will be selected from the sequence and combined into the final result. As the Debevec acquisition approach, is assumed that the images are perfectly aligned.

Merge Mertens approach has several advantages, first of all, the acquisition pipeline is simplified, no in-between HDR image needs to be computed, and the result obtained is a well detailed image. On the downside, this cannot extend the dynamic range of the original pictures, but instead it directly produces a well-exposed image for display purposes.

Quality Measures Many images in the stack contain flat, colorless regions due to under and overexposure. Such regions should receive less weight, while interesting areas containing bright colors and details should be preserved.

They use the following measures:

- **Contrast**: indicated by C. Using a Laplacian filter to the gray-scale version of the input image and taking the absolute value of the filter response. It tends to assign high values to important elements such as edges.
- **Saturation**: indicated by S. When a photo is taken with a long exposure time, the resulting colors become desaturated and eventually clipped. Saturated colors are desirable and make the image look vivid.
- **Well-exposedness**: indicated by E. Looking at just the raw intensities within a channel, reveals how well a pixel is exposed. We want to keep intensities that are not near zero (underexposed) or one (overexposed). So each intensity is weighted basing on how close it is to 0.5 using a Gauss curve: $e^{-(\frac{i-0.5}{2\sigma^2})}$ where σ is equal a 0.2 in their implementation and in the ours.

For each pixel, they combine the information from the different measures into a scalar weight map using multiplication. Similar to weighted terms of a linear combination, is possible to control the influence of each measure using a power function:

$$W_{ij,k} = (C_{ij,k})^{\omega_C} \times (S_{ij,k})^{\omega_S} \times (E_{ij,k})^{\omega_E}$$

where C, S and E, are the contrast, saturation and well-exposedness indices, and ω_C, ω_S and ω_E are weighting exponents, usually setted to 0. The subscript ij, k refer to pixel(i, j) in the k-th image. The final pixel weight $W_{ij,k}$ will be used to guide the fusion process.

Fusion The fusion step consists of computing a weighted average along each pixel to fuse the N images, using weights computed from the early gained quality measures. To obtain a consistent result, the values of the N weight maps will be normalized such that they sum to one at each pixel (i, j):

$$\widehat{W}_{ij,k} = [\sum_{k'=1}^{N} W_{ij,k'}]^{-1} W_{ij,k}$$

Where normalization, in image processing, is a process that changes the range of pixel intensity values. The resulting image R can then be obtained by a weighted blending of the input images:

$$R_{ij} = \sum_{k=1}^{N} \widehat{W}_{ij,k} I_{ij,k} \tag{2.3.2}$$

with I_k the k-th input image in the sequence. Unfortunately, just applying eq. 2.3.2 produces an unsatisfactory result. Wherever weights vary quickly, disturbing seams will appear. This happens because the images we are combining, contain different absolute intensities due to their different exposure times.

To avoid this problem they use another technique. First, the input images are decomposed into a Laplacian pyramid. A Laplacian pyramid is very similar to the Gaussian pyramid with the alteration that it uses a Laplacian transform instead of a Gaussian one. Where Gaussian pyramid is a technique that involves creating a series of images which are weighted down using a Gaussian average (Gaussian blur) and scaled down. When this technique is used multiple times, it creates a stack of successively smaller images, with each pixel containing a local average that corresponds to a pixel neighbourhood on a lower level of the pyramid.

Blending is then carried out for each level separately. Let the l-th level in a Laplacian pyramid decomposition of an image A be defined as $L\{A\}^l$, and $G\{B\}^l$ for a Gaussian pyramid of image B. Then, we blend the coefficients (pixel intensities in the different pyramid levels) in a similar fashion to 2.3.2:

$$L\{R\}^l_{ij} = \sum_{k=1}^{N} G\{\widehat{W}\}^l_{ij,k} L\{I\}^l_{ij,k}$$

Each level l of the resulting Laplacian pyramid is computed as a weighted average of the original Laplacian decompositions for level l, with the l-th level of Gaussian pyramid of the weight map serving as the weights. Finally, the pyramid $L\{R\}^l$ is collapsed to obtain R. For more details about Laplacian and Gaussian pyramids refer to [14].

2.4 Tests

Both techniques have been tested in order to choose the best one for the object detection purpose, based on reliableness and fastness. The OpenCV 3.0.0 has implemented both techniques but the prof. Pretto has given available another implementation of them, so both has been tested with the functions of OpenCv 3.0.0 and the implementation given available by prof. Pretto.

Moreover the tone mapping implemented by prof. Pretto is not the same used by OpenCV, which refers to [11], but it is based on the *gamma correction*. This method consist of an exponential correction, by a factor γ of the input image I_{in} and can be simply described with the relation:

$$I_{out} = A \cdot I_{in}^{\gamma}$$

where $A > 0$ and $0 < \gamma < 1$. The function maps the luminance I_{in} in the domain $[0, 1/A^{1/\gamma}]$ to the output range $[0, 1]$. The input image is weighted by the exponent γ, which regulates the contrast on the image. This tone mapping method increases the exposure of underexposed parts of the image while at the same time, if $A < 1$, it can decrease the exposure of overexposed parts enough to to prevent them from being overexposed. How Fig. 2.3 shows, pixels are considered with their values normalized to 1, and their value will change according to the gamma correction function.

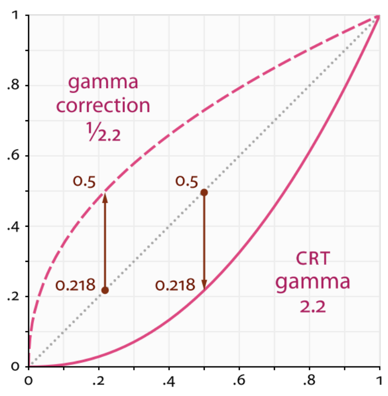


Fig. 2.3: Gamma correction graph with pixels that have values normalized to 1, A factor equal 1. (Image taken from [4])

In Fig. 2.4 the input images of a scene inside a church are showed, taken with different exposure times. The resulting images with the two methods, both with the OpenCV 3.0.0 implementation and the Pretto's implementation, are showed in Fig. 2.5.

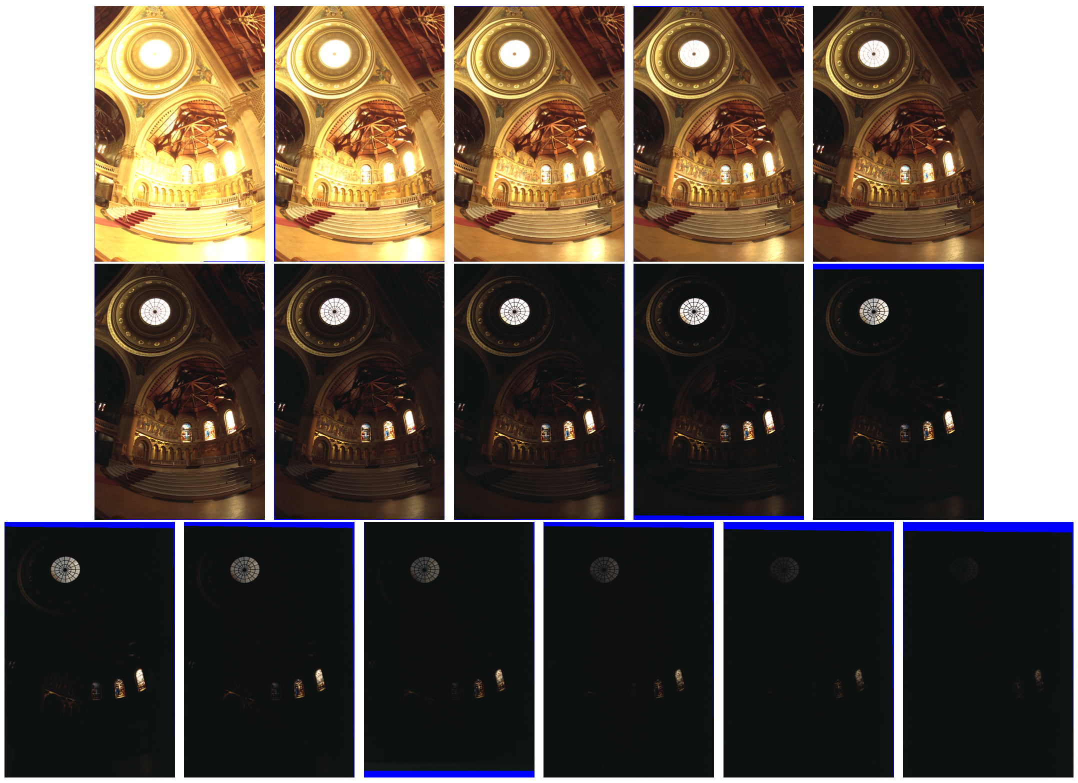
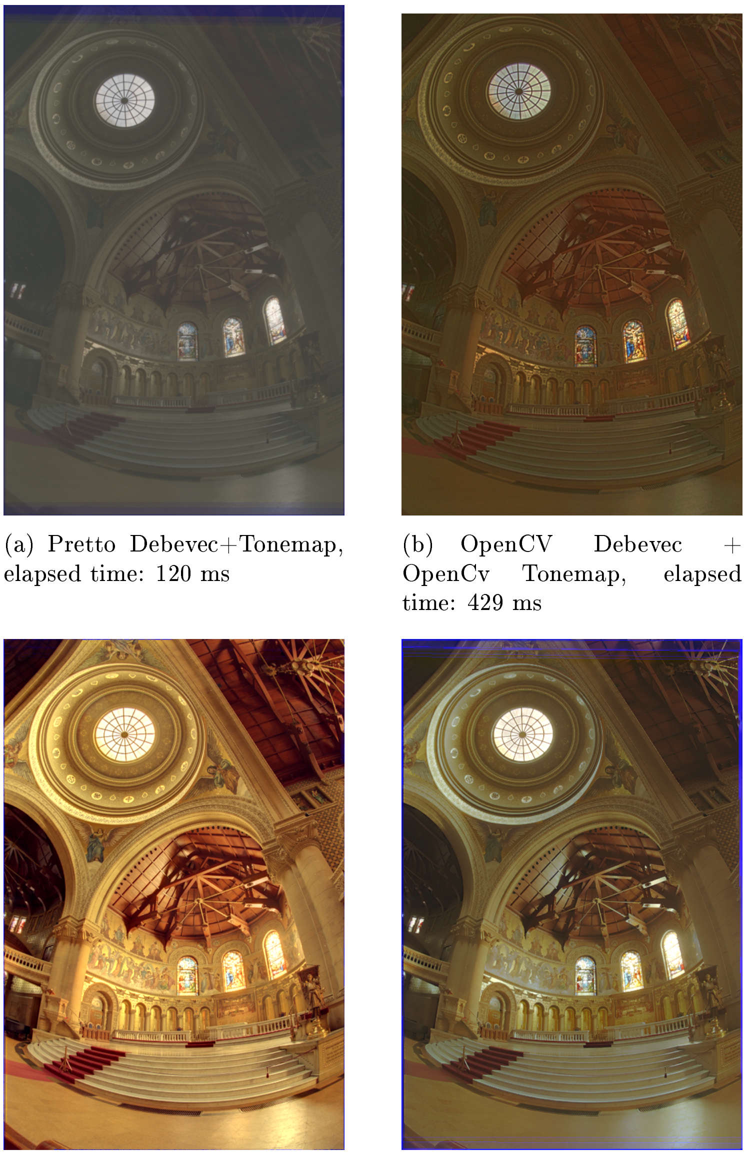


Fig. 2.4: Series of photographs of a church with different exposure time.

How Fig. 2.5 shows, the merge mertens method can preserve a lot of details more than debevec method, at the expense of a longer computational time. Since in the object detection stage it is important to have more possible details, one could think to use merge mertens in this first stage to take annoyances off.

In a real bin-picking practical case, where shot scenes are "less complex" than the church, as the one in Fig. 2.6 (note that input images are gray and not coloured images), different results have been noticed from the previously test of the church, as Fig. 2.7 shows. Note that in the elapsed time indicated is not taken into account the elapsed time to calculate the camera response, since it is a constant of the camera, so can be computed



(a) Pretto Debevec+Tonemap, elapsed time: 120 ms

(b) OpenCV Debevec + OpenCv Tonemap, elapsed time: 429 ms

(c) Pretto Merge Mertens, elapsed time: 639 ms

(d) Opencv Merge Mertens, elapsed time: 799 ms

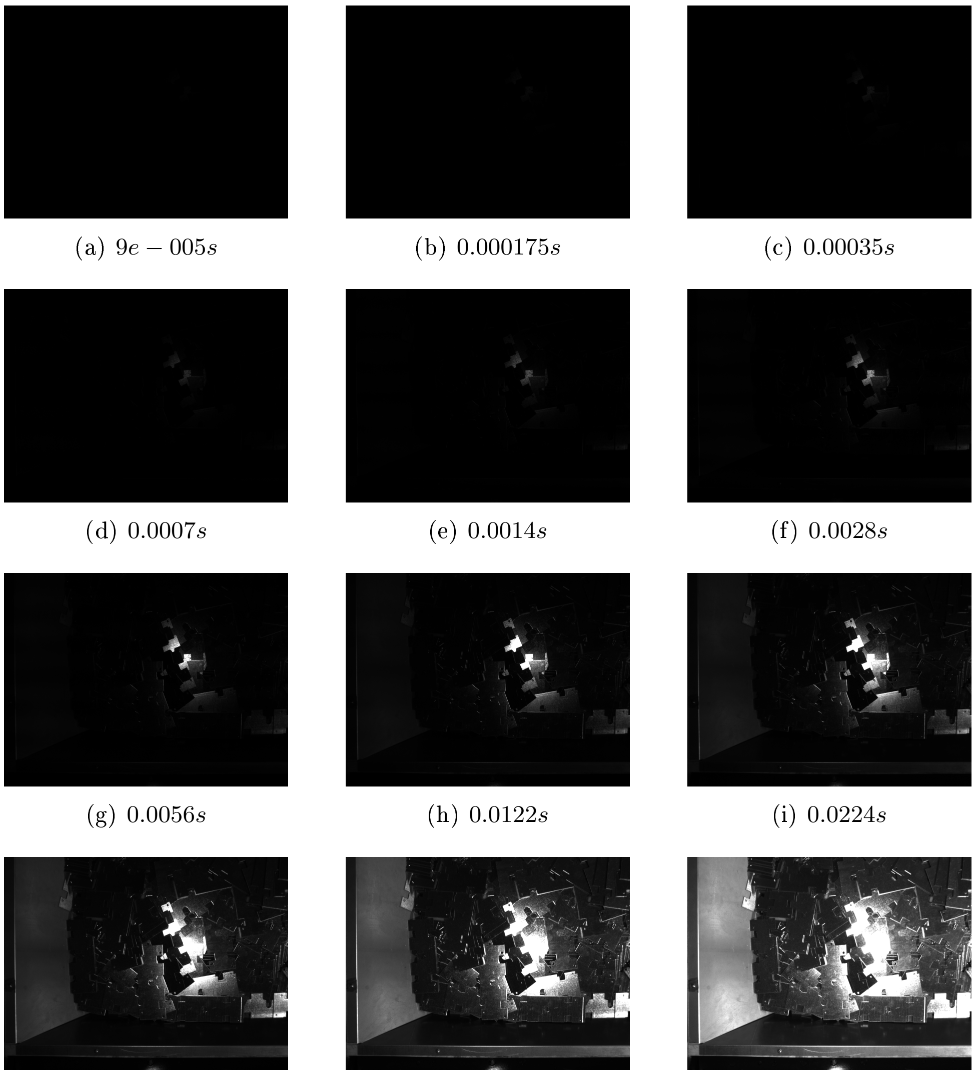
Fig. 2.5: Result of different approaches for HDR images from the memorial in Fig. 2.4

off-line without calculating it every time the camera takes a photo of a scene.

Tab. 2.1: Elapsed times with input images ones showed in Fig. 2.7

Algorithm	$\Delta t[s]$	Line segments detected
Pretto Debevec + Tonemapping	0.357	1381
OpenCV Debevec + Tonemapping	3.8829	1169
Pretto Merge Mertens	3.041	1417
OpenCV Merge Mertens	3.278	1263

The results have been evaluated in terms of execution time and reliableness, where reliableness has been evaluated using on resulting images the algorithm LSD, explained in chapter 3, in order to check how many line segments are detected, since for the detection



(a) $9e-005s$ (b) $0.000175s$ (c) $0.00035s$

(d) $0.0007s$ (e) $0.0014s$ (f) $0.0028s$

(g) $0.0056s$ (h) $0.0122s$ (i) $0.0224s$

(j) $0.0448s$ (k) $0.0896s$ (l) $0.1792s$

Fig. 2.6: Series of photographs of the scene with different exposure time.

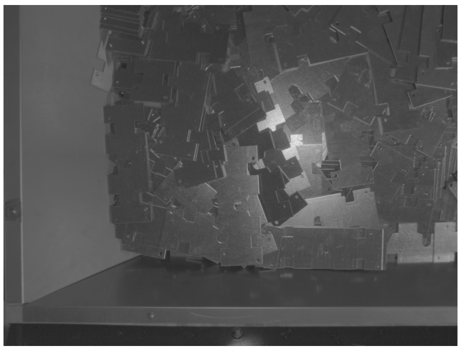
stage will be used only line segments as representatives of the scene. This final step is important since HDR image has been used only in order to take annoyances off from the input image in order to detect reliably line segments over all the image.

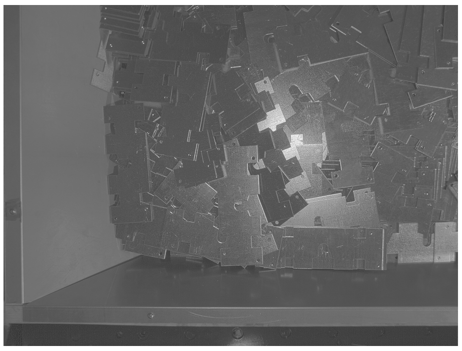
The Fig. 2.8 shows well that the methods tested are similar, output images are really similar and so the number of line segments detected, with a maximum of $\approx 12\%$ of difference. But it is interesting to notice how the Debevec's method implemented by Prof. Pretto is faster than the others, as showed in Tab. 2.1.

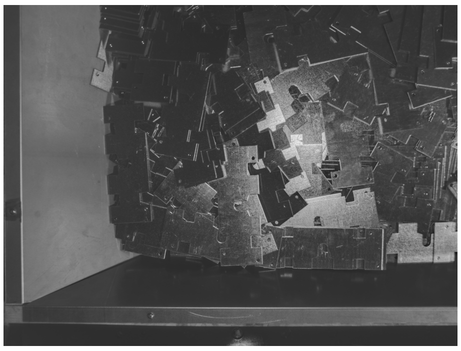
The implementation of Pretto is faster than the one of OpenCV because he used the API *OpenMP* (Open Multi-Processing) [15]. This API supports multi-platform shared memory multiprocessing in several programming languages, such as C/C++, and basically can run some parts of the same program simultaneously. This will be traduced in a faster way to execute the program, depending on the number of CPU's cores.

Moreover let notice that the execution time of the Debevec and tone mapping process implemented in OpenCV need more time to be execute than Merge Mertens method because they are optimized for coloured images (3 channels) and not for gray images (1 channel), such as the tested ones.

Merge Mertens method still preserves more details than Debevec method, but only


(a) Pretto Debevec+Tonemapping


(b) Opencv Debevec+Tonemapping


(c) Pretto Exposure Fusion

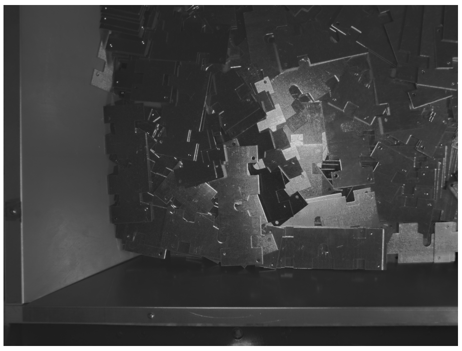
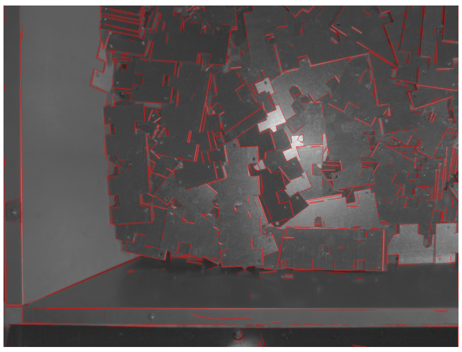
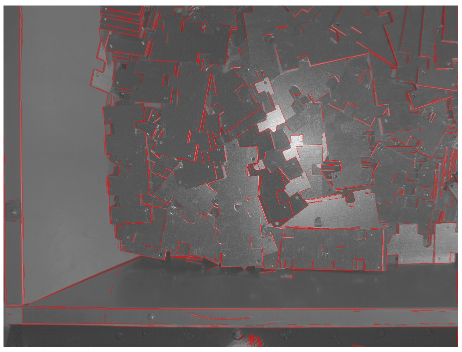
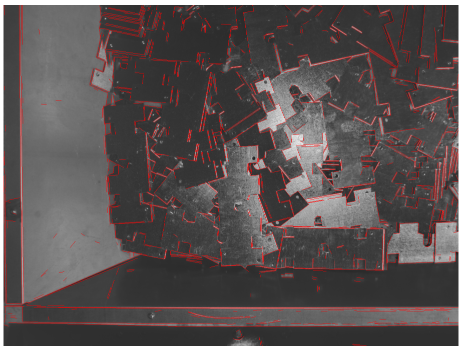

(d) Opencv Exposure Fusion

Fig. 2.7: Result of different approaches for HDR images


(a) Pretto Debevec+Tonemapping


(b) Opencv Debevec+Tonemapping


(c) Pretto Exposure Fusion

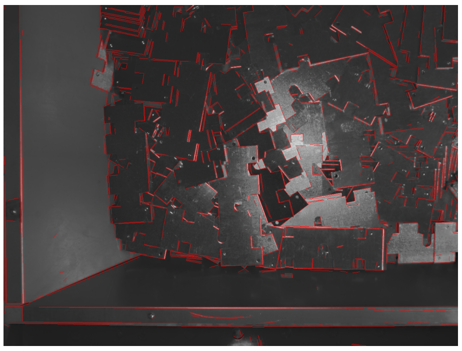

(d) Opencv Exposure Fusion

Fig. 2.8: Application of LSD algorithm on the image showed in Fig. 2.7

few of them are preserved in real cases wherein the Bin-Picking system has to work, at the expense of an execution time higher. Since in the Bin-Picking systems fastness is the second most important thing to consider, after precision, the Debevec method implemented by Prof. Pretto has been chosen for this first stage.

Chapter 3

Line Segments Detection

The next step, after having used HDR and tone mapping to take the annoyances off from the image, consists in line segments detection. This part is really important since the information caught from this step will be used to match the model in order to find correct matches to the input object model in the scene.

3.1 Introduction

Line segments give important information about the geometric contents of images for two main reasons:

- most human-made objects are made of flat surfaces
- a lot of these objects, but even nature objects, accept an economic description in terms of straight lines. Therefore line segments can be used to extract low-level features from images and in particular they can be used for the main problem of computer vision: the objects detection.

Before proceeding with the reading it is advisable to have some basics knowledge about the gradient of images; if not so, the theme is briefly debated in appendix A.

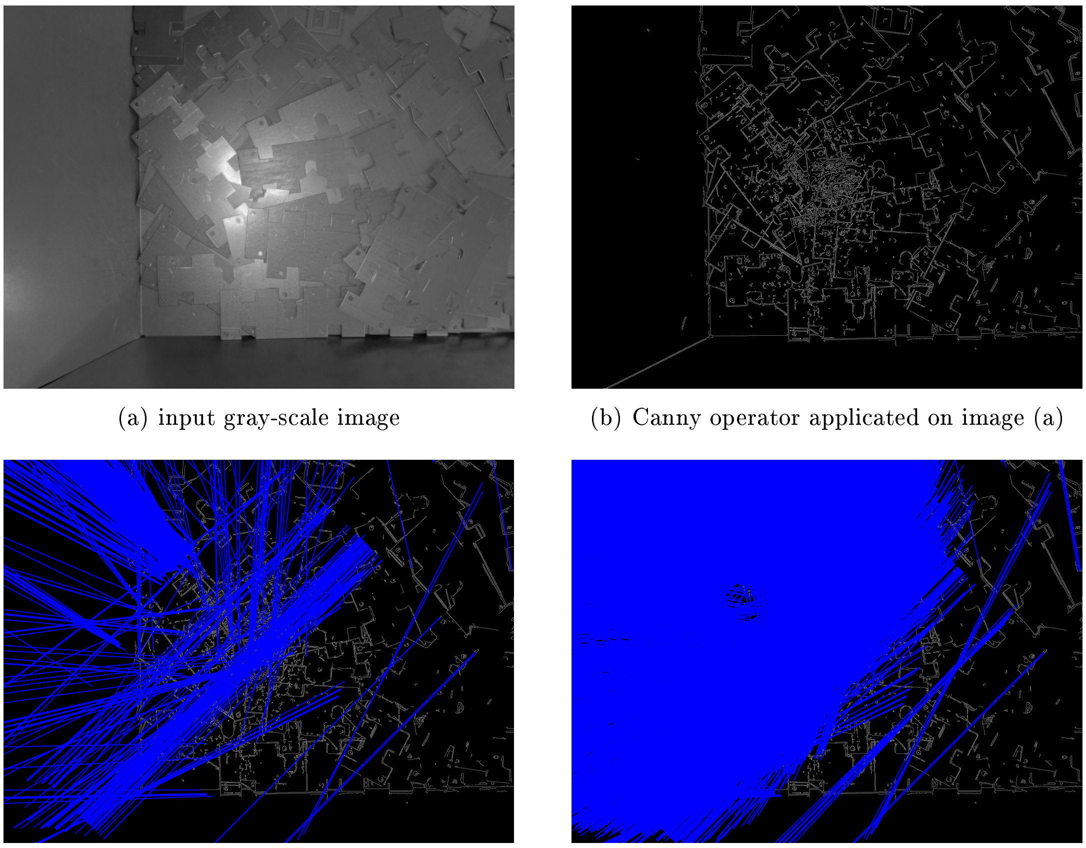
Ideally one would like to have an algorithm that accurately detects line segments presented in an image, without false detections, and without the need to manually tune parameters for each image. The line segments detection is an old and recurrent problem in computer vision, a standard approach is combining the most known and standard method for edge detection that is the Canny Edge Detector [16] followed by the application of an Hough Transform [8] extracting all lines that contain a number of edges points exceeding a threshold; these lines are thereafter cut into line segments by using gap and length thresholds.

The Canny Edge Detector follows 4 mains steps:

- filtering by a Gaussian Filter to reduce noises
- getting the image gradient
- non-maximum suppression on the image gradient, in order to remove pixels that are not considered part of an edge
- hysteresis: trough two thresholds, the algorithm allows to establish if a pixel make part of an edge or not

The Hough Transform is a strong algorithm that can recognize some geometric figures, simply by verifying if the pixel of the image, in this case the image resulting from canny edge detector, matches with the geometric figure considered, in this case a line.

The main drawback of this standard procedure to detect line segments is due to textured regions that have a high edge density which can cause many false detections. Moreover this procedure considers only the magnitude gradient and not the information of the direction, therefore such algorithm obtains line segments with aberrant directions.



(a) input gray-scale image

(b) Canny operator applicated on image (a)

(c) Standard Hough Line Transform with threshold setted to 150

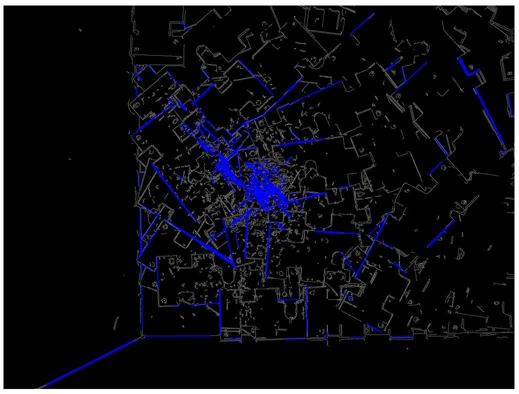
(d) Standard Hough Line Transofrm with a threshold setted to 100

Fig. 3.1: Application of Canny and Standard Hough Line Transform to a gray-scale input image with differents values of the threshold. Blue lines are the lines detected.

Another fundamental problem for all detection methods is the setting of thresholds. In fact the use of fixed thresholds can lead to a significant number of false positive or false negative detections. Therefore the thresholds have to be regulated for each image in step with environment conditions, as illustrated in Fig. 3.1.

Along the years, some good algorithms for line segments detection have been developed, but all of them have some benefits and some disadvantages that make them not completely satisfactory for the line segments detection problem. One of them has been developed by Etemadi [17] which is able to detect simultaneously line segments and arcs but it is not completely satisfactory. Another one has been developed by Burns et al. [18], this algorithm was quite innovative because it ignores gradient magnitude and actually works with gradient direction and this one lead to have a good detection but still the problem of setting every time a threshold remains. A good improvement has been made from Matas et al. [19] with the Progressive Probabilistic Hough Transform (PPHT) which accelerates the computing time with the use of gradient direction information and introduces a false detection control. The Fig.3.2 shows a clear improvement over the Standard Hough Transform Method. Nevertheless, the false detection control used is not completely satisfactory, because it considers only the whole line and not line segments, and this is a problem since a lot of lines are not detected as whole lines but as sum of more segments.

The threshold question was thoroughly analysed by Desolneux et al. [20]. Their line segment detection method succeeds in controlling the number of false positive. The method


(a) PPHT applicated on Fig. 3.1(a)with a threshold of 150

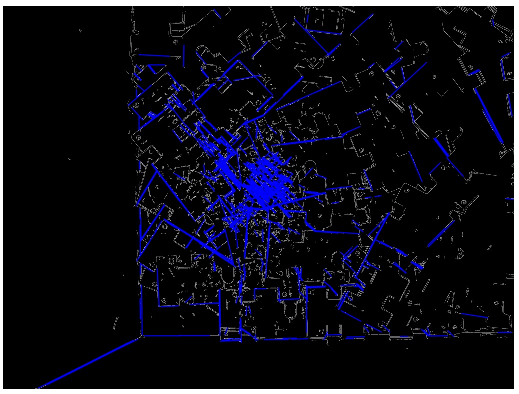

(b) PPHT applicated on Fig. 3.1(a)with a threshold of 100

Fig. 3.2: Application of Canny and Probabilist Progressive Hough Line Transform (PPHT) to the gray-scale input image Fig. 3.1(a) with differents values of the threshold

counts the number of aligned points and finds the line segments as outliers in a non-structured a contrario model based on the Helmholtz principle.

The main problem was to create an algorithm that did not need a threshold setting, that could be reliable with the most common environment conditions, and that could detect reliably line segments. The algorithm discussed in this chapter, and presented in [12, 5] in 2010, has gone forward this direction, combining improvements of Burns et al. and Desolneux et al. Actually it just requires setting some parameters, that could be left to a default setting.

In this chapter will be commented the LSD algorithm, which is really reliable and fast and it has been the best choice for the phase of line segments detection. This chapter also comments the LSD parameters setting and a comparison between the implementation proposed by developers and the one by OpenCV 3.0.0 library.

3.2 Line-Based Representation

The most of standard algorithms to detect objects need a point representation of the scene, but actually, with the introduction of new reliable algorithms to detect line segments, some works evidenced that it is better a representation of the scene through line segments.

The edge map of a scene is not an unstructured binary pattern. On the contrary, the object contours are made with certain continuity constraints that can be retained by combining line segments of various lengths, orientations, and translations. Based on this observation an image can be represented as a collection of m line segments. Compared with a set of points which has cardinality n, its line-based representation is more concise.

Encoding an edge map using the line-based representation requires only $O(m)$ memory size, where $m << n$. When the searched object exhibits a curved contour, more segments are required for good approximation, but the line base representation is still more concise than the set of edge pixels.

The algorithm LSD presented in this chapter only retains edge points with continuity and sufficient support, therefore it acts also like a filter that filters noise and isolated edges out. In addition, the directions recovered through line-segments detector are more precise than ones obtained using local operators. An example of the line-based representation is given in Fig. 3.3, where a set of 88,205 points is modelled with 1,396 lines segments.

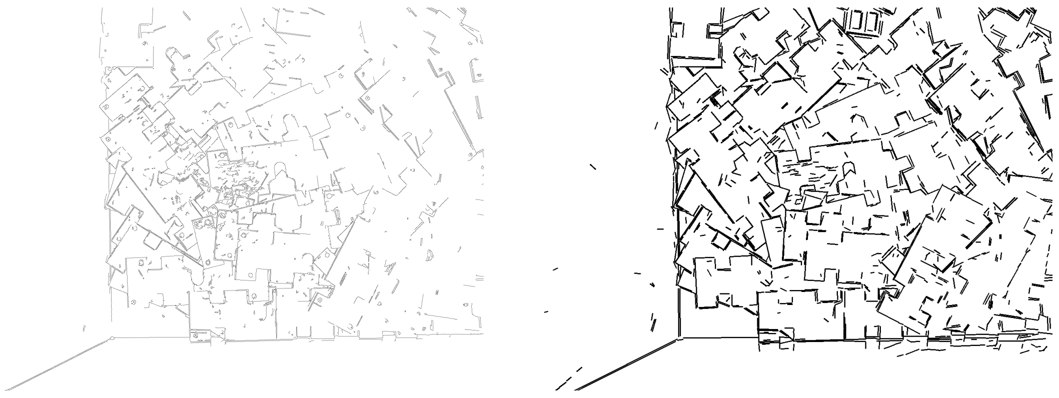

(a) (b)

Fig. 3.3: Line-based representation of input image 3.1(a): (a) Edge map (obteined with Canny operator with a threshold of 14). The image contains 88,205 points (b) Line-based representation of the edge map. The image contains 1,396 line segments.

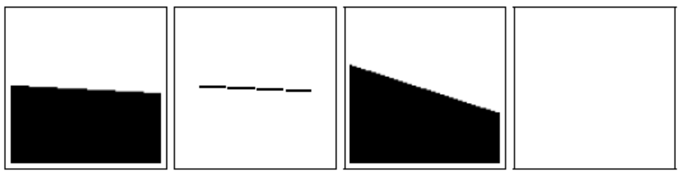


Fig. 3.4: Example of the scaling problem: the first image and the third one are the input images while the second one and the fourth one are the output images. (Image taken from [5])

3.3 Line Segment Detector

LSD is a linear-time Line Segment Detector giving subpixel accurate results since it controls its own number of false detections: on average, one false detection is allowed per image. LSD has been designed as an automatic image analysis tool. As such it must work without requiring any parameter tuning. The algorithm actually depends on several parameters that determine its behaviour; but their values were carefully devised to work on all images.

The LSD [12, 5] is actually a fusion and improvement of Burns et al. method about line segments finding and the Desolneux et al. method about the validation criterion of the line segments found in the first step of the algorithm.

The whole process of this detector can be simplified in three basics steps:

- Looking for line support region
- Approximation of that region in a rectangle
- Validation criterion

In the following will be commented all the phases of the Line Segments Detection.

Image Scaling The result of LSD is different when the image is analysed at different scales or if the algorithm is applied to a small part of the image, that is why there are different information about details of the image when scaled.

An example of this different result is illustrated in Fig. 3.4, that shows two discrete edges at different angles, both presenting the staircase effect, and the result of LSD on

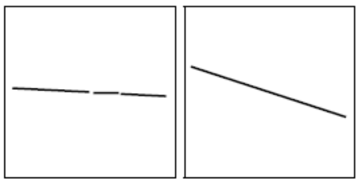


Fig. 3.5: Output images of the input images of Fig. 3.4 scaled by a factor of 0.8 . (Image taken from [5])

those images not scaled. Instead Fig.3.5 shows the result on those image using the 80% scaling, both edges are detected with the right orientation. In this project the input image of 2592x1944 pixels is scaled using a 50% scaling factor, in order to reduce image noise.

Gradient Pseudo-Ordering LSD is a greedy algorithm and the order in which pixels are processed has an impact on the result. Pixels with high gradient magnitude correspond to the more contrasted edges. Normally, in an edge, central pixels have the highest gradient magnitude, therefore makes sense to start to look for line segments at these central pixels. To sort these points, a simple pixel pseudo-ordering in linear-time is used. To this aim, 1024 bins are created corresponding to equal gradient magnitude intervals between zero and the largest observed value on the image, later pixels are classified into the bins according to their gradient magnitude value. LSD starts to look for line segments from the largest magnitude value's bin until having tested all its pixels, then LSD tests pixels of the bin with the second largest gradient magnitude value and so on until exhaustion of all bins.

Gradient Threshold Pixels with small gradient magnitude correspond to flat zones or to an ideal background, and usually present even a higher error in the gradient computation due to the quantization of their values. The algorithm therefore rejects all pixels that have a gradient magnitude minor than a certain value p and will not used in the construction of line-support regions. The threshold p is setting in based on the value of parameter q that is a bound of the possible error in the gradient angle value due to quantization effects.

Line Support Regions Contrary to classic edge detectors, the Burns et al. method defines a line segment as an image region, called "line support region", that is to say a straight region whose points share roughly the same image gradient angle, such line segments are roughly oriented along the average level-line direction. The Burns et al. algorithm extracts line segments in three steps:

- Partition of the image into line-support regions by regrouping connected pixels that share the same gradient angle up to a certain tolerance
- Find the line segment that best approximates each line-support region
- Validate or not the line of the second step through a validation criterion

The LSD algorithm has caught the principle idea of first two steps from Burns et al. method while the step of validation is based on the Desolnoux et al.

The improvements made by Grompone von Gioi et al [12] on the first step of Burns et al. method is the use of a region growing algorithm as illustrated in Fig. 3.6.

Each region starts with just one pixel and the region angle is setted to the level-line angle at that pixel, which is orthogonal to the pixel gradient direction. Then pixels adjacent to the region are tested and the ones with the same level-line orientation of the

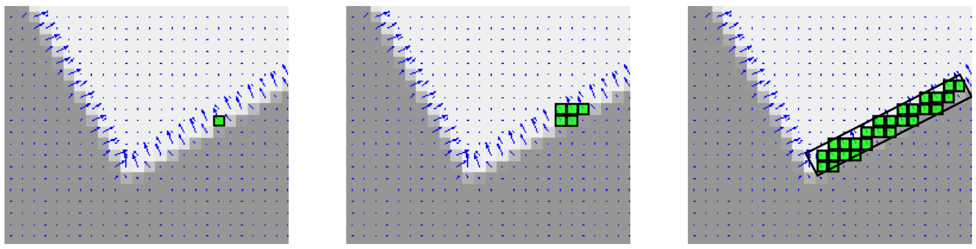


Fig. 3.6: Illustration of growing line-support region. (Image taken from the website [6])

region angle, up to a certain precision, are added to the region. At each iteration the region angle is updated to consider the influence of new pixels that make part of the region. The orientation of the region is defined as

$$arctan\left(\frac{\sum_i \sin ang_i}{\sum_i \cos ang_i}\right)$$

The process is iterated until no new point can be added to the region. Since pixels with large gradient magnitude have more influence and are more likely to belong to straight edges, they are tested first. Once a point is added to the region the point is marked and never visited again.

Rectangular Approximation of Regions Prior to the validation step, each line-support region has to be associated to a line segment, that actually is a rectangle. Therefore input image is partitioned in some regions like in Fig. 3.7. Normally a line is determined by its starting and ending points (where starting and ending point here are used to refer to extreme points), in this case even by its width. Or equivalently, by its center, angle, length and width. The rectangular approximation includes all these parameters, since has to take the whole information necessary to describe a line.

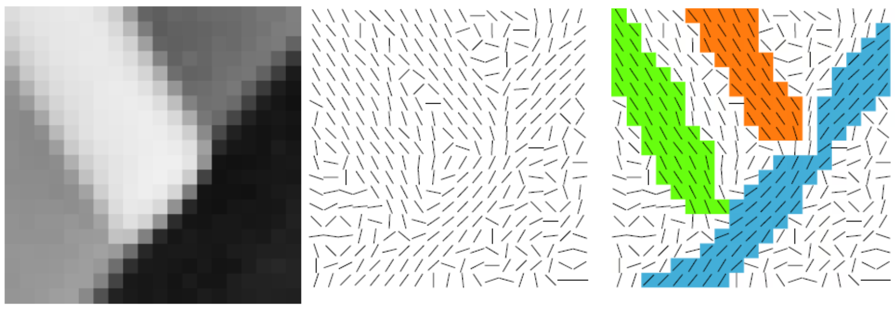


Fig. 3.7: Line support regions in a generic input image. (Image taken from [5])

The angle of the line is not taken by simply calculating the angle of an ideal line that connects the start and end points of the region, because this would lead to a wrong result. The right procedure has been suggested by Kahn et al. that refers to calculate the center of mass, which will be used to select the center of the rectangle, and the first inertia axis, to select the rectangle orientation. Where the mass of pixels corresponds to their gradient magnitude. Then the length and width are chosen in order to cover the line-support region. The figure Fig. 3.8 shows the process of rectangle approximation.

Line Segment Validation The two key points of the Desolneux et al. method are the use of the gradient orientation and a new framework to deal with parameter settings. From the gradient of the input image only the direction is considered in the validation step.

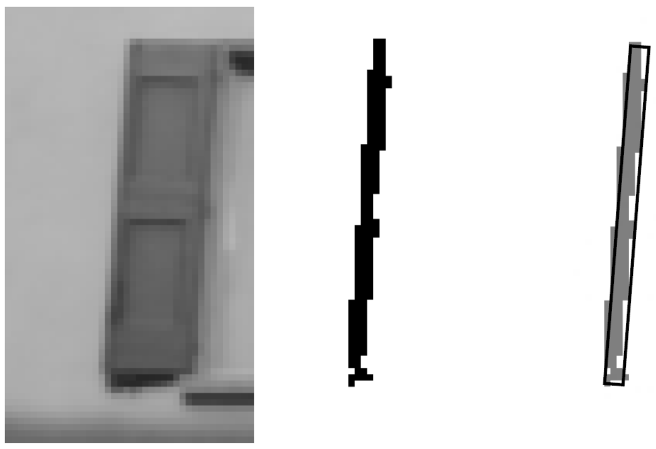


Fig. 3.8: Approximation of the line support region in a rectangle. (Image taken from [5])

The orientation is used to counts the number of aligned points to the level-line orientation, up to a certain tolerance τ . All potential line segments are tested and those that satisfy a threshold criterion based on their length l and their number of aligned points k are kept as valid segments.

On natural images often the gray-level transition, corresponding to edges, is made of many pixels. On the algorithm of Desolneux et al. this needed a lot of effort to identify that edge as a line segment, this step has been simplified considering no line segments but rectangles.

The framework used is based on the concept of white noise. White noise is a term used to represent an image where gradient angles of its pixels are uniformly distributed over all angles direction in the whole image. This is an important representation of an image because, as Desolneux et al. showed, a suitable background model, is just one in which all gradient angles are independent and uniformly distributed. Also flat zones, that have no a variable intensity, have gradient angles distributed uniformly. Therefore for the line segment detection problem an image can be thought like an overlapping of lines above the background.

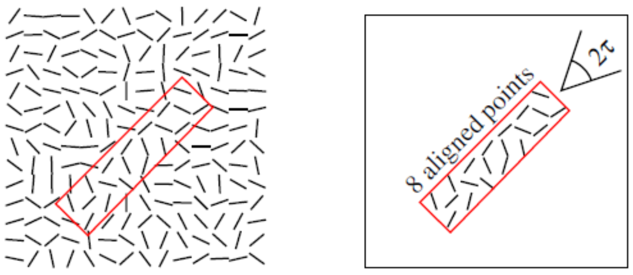


Fig. 3.9: Candidate line made of an isotropic process. (Image taken from [5])

It is clear that the zones of the image where there are lines are zones whose pixels have the same gradient angles, up to a certain tolerance, these zones are called *anisotropic zones*. Thus, in practice, a set of pixels will not accepted as a line segment if it could have been made of an *isotropic process*. In fact is possible that a candidate line segment is made of an isotropic process, cause to the rectangle approximation that, in order to cover all the line support region, is made even of other pixels with an aberrant orientation. The Fig.

3.9 illustrates an event of a candidate line made of an isotropic process.

The validation step is based on the a contrario approach and the Helmholtz principle proposed by Desolneux, Moisan, and Morel [20]. The so-called Helmholtz principle states that no perception (or detection) should be produced on an image of noise. Accordingly, the a contrario approach proposes to define a noise on a contrario model H_0, equivalent to a white noise image, where the desired structure is not present. Then, an event is validated if the expected number of events as good as the observed one is small on the a contrario model. In other words, structured events are defined as being rare in the a contrario model H_0.

The a contrario model used for line segment detection is defined as a stochastic model of the level-line field satisfying the following properties:

- $LLA(j)_{j \in Pixels}$ is composed of independent random variables
- $LLA(j)$ is uniformly distributed over $[0, 2\pi]$

where $LLA(j)$ is the level-line angle at the pixel j of the image.

In the case of *line segments* the interest is on aligned points. We consider the event that a line segment in the a contrario model has as many or more aligned points, as in the observed line segment. The threshold k_r must be fixed in a way that guarantees a control of expected number of false alarms under H_0.

The Number of False Alarms (NFA) associated with the rectangle r corresponds to the expected number of rectangles which have a sufficient number of aligned points to be as rare as r under H_0. When the NFA associated with an image rectangle is large, this means that such event is expected on the a contrario model, that is to say the event is common and thus not a relevant one, in other words it is not a line. On the other hand, when the NFA value is small, the event is rare and probably a meaningful one. A threshold ϵ is selected and when a rectangle r has $NFA(r, i) \leq \epsilon$ on the image i, the rectangle is called $\epsilon - meaningful\ rectangle$ and produces a detection.

Rectangle Improvements Before rejecting a line-support region for being not meaningful ($NFA > \epsilon$), the algorithm tries some variations of the rectangle's configuration initially tested in order to get a valid one. The reason of this rectangle improvement is due to the fact that width of the line segments is the worst estimated parameter on the first rectangular approximation, but also a very influential one. An error that makes the rectangle one pixel thicker adds a large number of non-aligned points, as many as length of the line segment. This can increase the NFA value, rising the non-detection risk. Also the precision p is another very influential parameter, defined as $p = \frac{\tau}{\pi}$, since the precision control the line-support growing process.

The initial precision used, corresponding to the region growing tolerance, is large enough so only testing smaller values makes sense. If the pixels are well aligned, using a finer precision will keep the same number of aligned points, but a smaller p yields a smaller (and therefore better) NFA. In a similar way, it only makes sense to try to reduce the rectangle's width because the initial width was tuned to cover the whole line-support region. Often, reducing by one pixel the width may reduce the number of aligned points by only a few units while reducing the total number of pixels by a number equal to the length of the rectangle, see Fig. 3.10. This may decreases significantly the NFA value. If a meaningful rectangle is found ($NFA \leq \epsilon$) the improvement routine will stop. The refinements simply works trying to reduce the precision or the width, not the both at the same time.

Aligned Points Density Sometimes the tolerance τ used can lead to wrong interpretation of lines. In fact could be that a rectangle actually includes into itself two lines

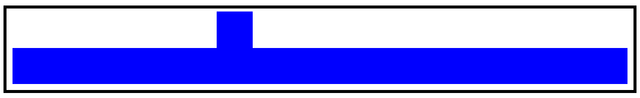


Fig. 3.10: Example of a line-support region that needs a refinement.(Image taken from [5])

with a different orientation, as showed in the Fig. 3.11. This problem can appear when two straight edges are present in the image forming an angle between them smaller than tolerance τ.

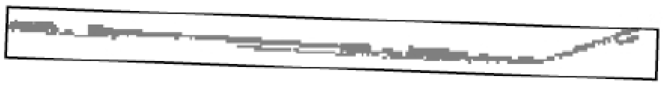


Fig. 3.11: Example of a line-support regions containing two lines. (Image taken from [5])

This problem is handled by detecting problematic line-support regions and cutting them in two, or more, smaller line-support regions. The detection of this angle problem is based on the density of aligned points, where there is a difference in the orientation the density in that zone has a low value. Therefore the point corresponding to the intersection of the two researched lines is indicated by the point with the lowest density value of aligned points.

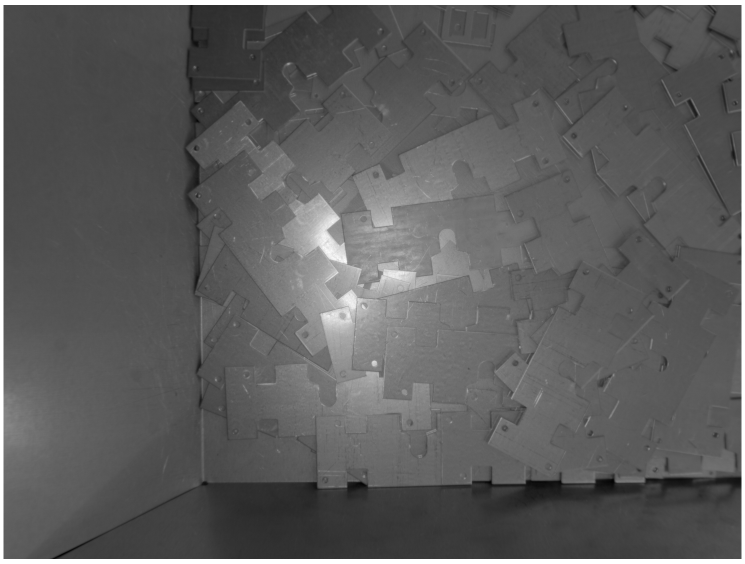
3.4 LSD - Standard Parameters and Settings

The LSD algorithm actually depends on some parameters, described above, which determine its behaviour, that are (on the side are written the values suggested by developers):

- Scale factor S: 0.8
- Quantization error for the gradient norm q: 2.0
- Tolerance τ: 22.5 degrees
- Density limit d: 0.7
- The number of false detections admitted, ϵ: 1
- The number of bins n used to order pixels on their gradient magnitude n: 1024

In [5] all the parameters are well described and there are some tests which justify the choice of parameters' value of the developers. In our test of detecting line segments we needed to handle these parameters suggested by the developers in order to get more line segments. In fact the figure Fig. 3.12 reports an input image and the result of LSD , with the default parameters setting, on the input image. Clearly, the lines detected, the purple ones, are too few to be sufficient for a robust object detection.

Since the phase of object detection needs more possible information about the input image, and since some line segments, that were clearly edges, were not detected, we have handled these parameters in order to get more possible line segments. Parameters tuning


(a) Input gray-scale image

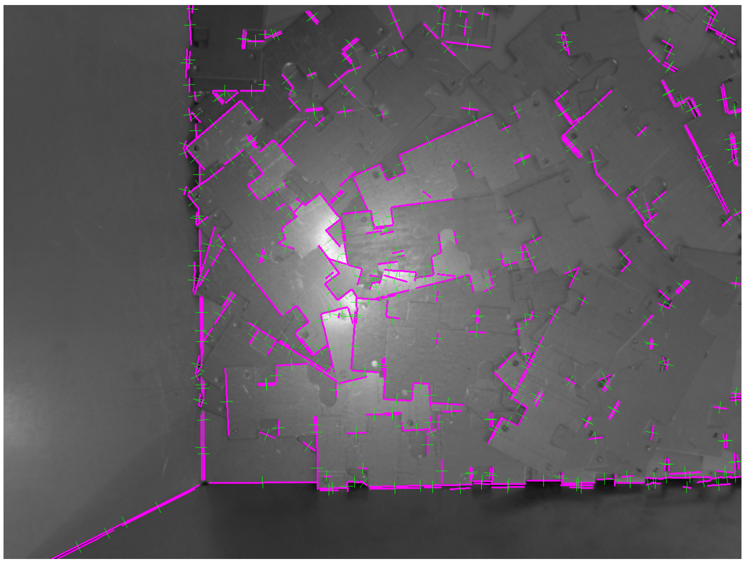

(b) LSD with default parameter values

Fig. 3.12: Application of LSD algorithm with the default parameters value

has been performed with a trade-off between precision and number of line segments detected. Handling these parameters is possible to detect more line segments, some of which are line segments that does not correspond to any real edge. An arising number of line segments detected leads the next phase of the object detection to make a relevant effort in terms of efficiency, that is to say it would be too low in computation.

The results of tests lead us to consider the following combination of parameters:

- Scale factor S: 0.5
- Quantization error q: 0.5
- Tolerance τ: 22.5 degrees
- Density limit d: 0.5
- Number of false detection ϵ: 1

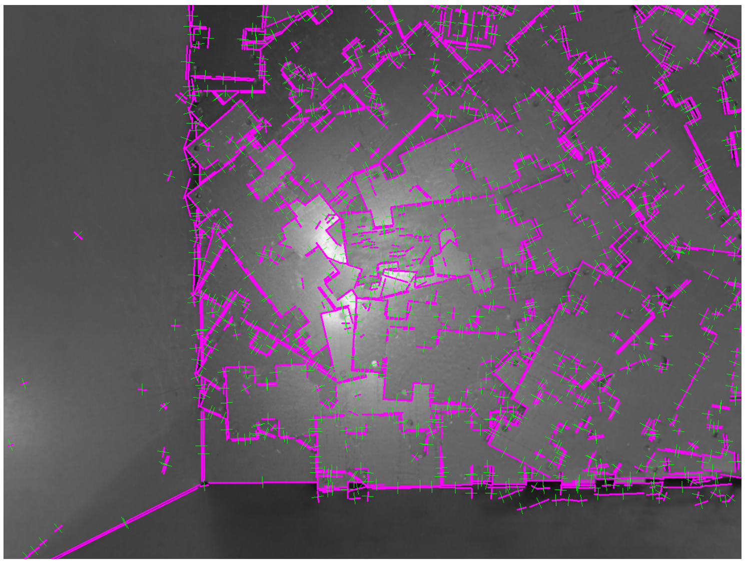


Fig. 3.13: Application of LSD algorithm with the chosen parameters to input image Fig. 3.12(a).

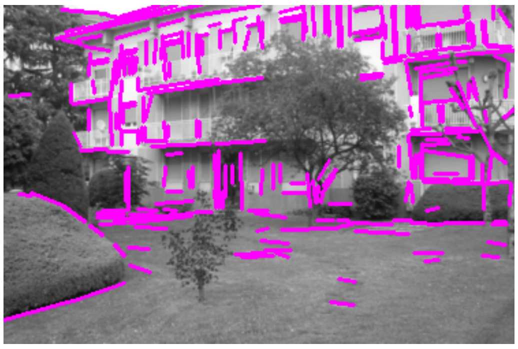
- Number of bins n: 1024

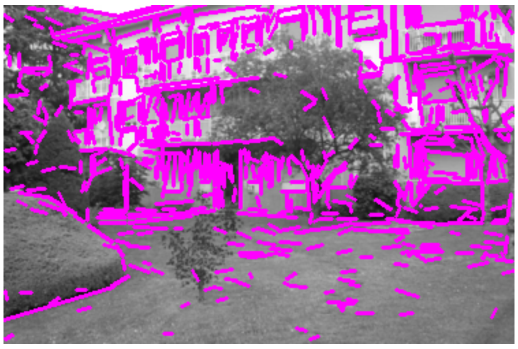
The result of LSD with this new parameters setting on the input image Fig. 3.12(a), and reported in Fig. 3.13, is really more satisfying than the original one. A lot of the real edges are detected at the expense of an increase of line segments that does not represent edges, leading the next object detection phase more precise but a little bit more slowly.

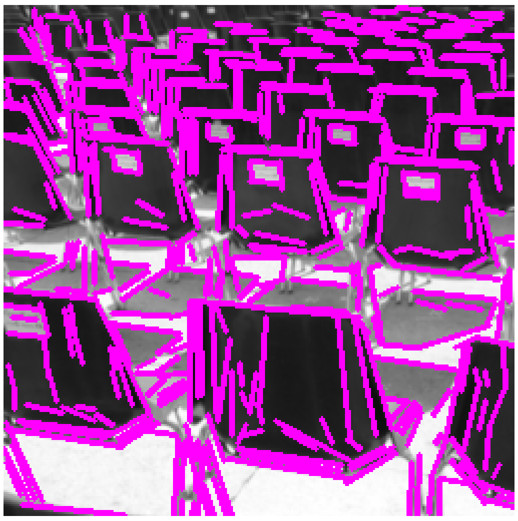
3.5 LSD in OpenCv

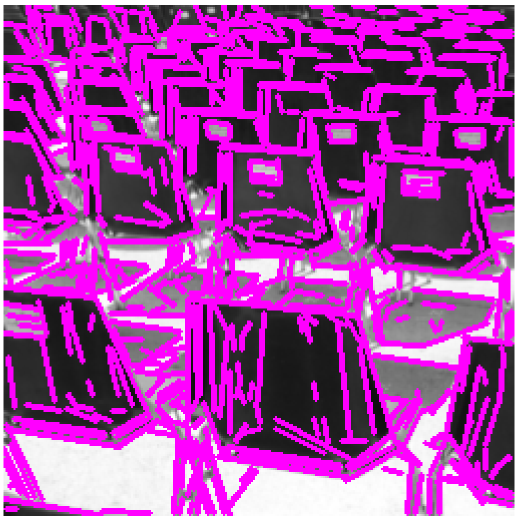
In the version 3.0.0 of OpenCV library the algorithm LSD [12] has been implemented with some improvements in terms of fastness. The Fig. 3.14 reports some tests and the relative execution time, with LSD code provided by von Gioi et al. and the equivalent function of OpenCV 3.0.0 , with the same parameters setting.

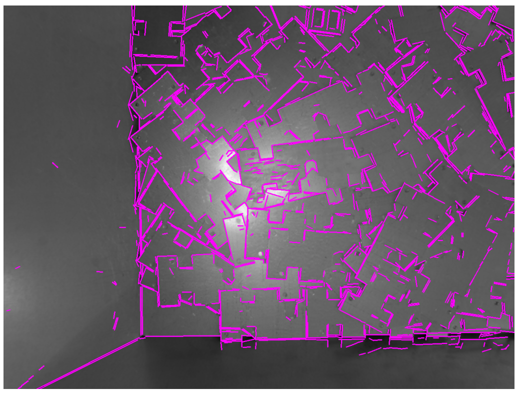
Notice the good increase of speed, about 50% (it depends on the number of line-support regions to validate and the size of the input image), of the algorithm with the function LSD implemented in OpenCV 3.0.0 at the expense of a lower precision, there are a lot of line segments detected that do not correspond to any relevant edge, this actually is a deterioration of input image data. Since the number of line segments detected will affect the speed of the object detection phase, that requires a lot of efforts and time, we have chosen to use the original implementation of von Gioi et al., which is more precise than the one of library OpenCV 3.0.0.


(a) 374 x 248 Elapsed time: 108,992 ms


(b) 374 x 248 Elapsed time: 45,4814 ms


(c) 256 x 256 Elapsed time: 57,5579 ms


(d) 256 x 256 Elapsed time: 32,0993 ms


(e) 1296 x 972 Elapsed time: 438,738 ms

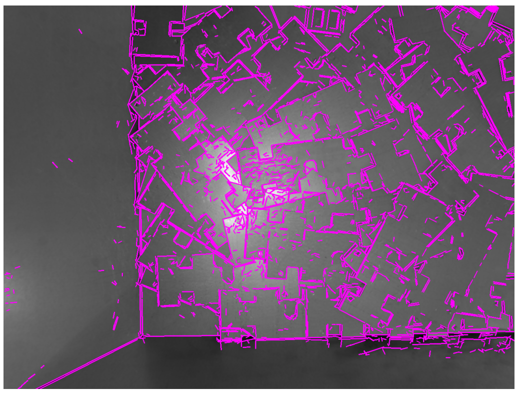

(f) 1296 x 972 Elapsed time: 283,647 ms

Fig. 3.14: Some tests of fastness of LSD, with the same parameters setting, using the implementation of von Gioi et al. [(a),(c),(e)] and the one of OpenCv 3.0.0 [(b),(d),(f)].

Chapter 4

Object Detection

4.1 Introduction

Object detection is the main challenge in computer-vision systems, in fact in recent decades a considerable amount of work has been done on automating the process of part assembling through vision systems. These are successful in identifying, inspecting, and locating parts in carefully engineered manufacturing settings, but it remains a great challenge to extend their applicability to more general, unconstrained settings.

Several authors have proposed shape representations and similarity measures that aim to be invariant to object deformations, and that achieve good performance in object recognition. However, they require a clean segmentation of the target object, therefore they need a good extraction of line segments from the image, and this makes them less suitable for dealing with unstructured scenes due to the difficulty in foreground-background separation.

A lot of improvements have been proposed with excellence results in shape matching, even in cluttered images, but at the expense of a high computational complexity that makes them unsuitable for time-critical applications, as Bin-Picking.

A great solutions was to improve an old algorithm proposed decades ago, that is to say the Chamfer Matching algorithm [21], that is a good method when speed and accuracy are required.

An evolution of Chamfer Matching is the Generalized Hough Transform [8], but recently in 2012 there has been some improvements of Chamfer Matching by Liu et al. and Shotton et al. [9, 2] that lead us to prefer this one to the modified Generalized Hough Transform, originally used in [1].

In this thesis the standard Chamfer Matching algorithm for the detection stage has been implemented, with the idea to implement the improvements mentioned in the future starting from this work.

In this phase of object detection the line segments, extracted from the initial image and the CAD model of the searched object, are given as inputs. The CAD model will be rasterized with a pitch of some millimetres (the pitch will affect speed as well as precision of the detection) and will be used for the shape matching. In order to make the chamfer distance more robust, a phase of optimizations has been used in order to improve the precision and another phase to score the matched object.

Note that to get the correct position of the object it is necessary to do a calibration of the camera before proceeding with matchings. This calibration is really important since it leads the algorithm to understand how far the objects are in the bin from the camera lens and where they are located in the scene.

In this chapter all the procedures of this detection phase will be discussed, with a mention to camera calibration, with a study of the Hough Generalized Transform [8] and Chamfer Matching, and the results will be discussed. In the next chapter the improvements on Chamfer Matching suggested by Liu et al. and Shotton et al. will be discussed.

Axis Disposition In order to make the explanation more understandable let's comment the axis disposition. The axis to consider are:

- x axis: it corresponds to the horizontal axis of the shot plane. Rotations around this axis are called "roll" rotations.

- y axis: it corresponds to the vertical axis of the shot plane. Rotations around this axis are called "pitch" rotations.
- z axis: it corresponds to the camera axis, normal to the bin plane. Rotations around this axis are called "yaw" rotations.

and are well showed in Fig. 4.1.

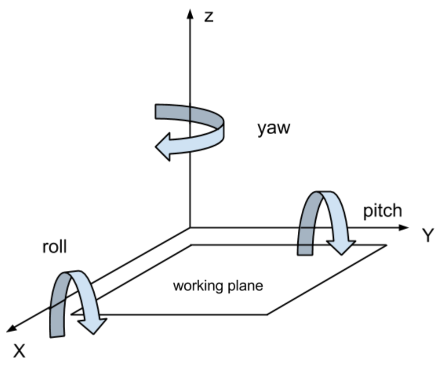


Fig. 4.1: Axes disposition.

Note that, thanks to the camera calibration, all coordinates are stated respect to the camera frame.

4.2 Camera Calibration

Camera calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images. In this section we only limit to hint at this process phase.

Unfortunately the camera is affected to a significant distortion, that depends on the fact that camera projects 3D world points onto the 2D image plane. The calibration of the camera finds the quantities internal to the camera that affect this imaging process.

Furthermore, with calibration you may also determine the relation between the camera's natural units (pixels) and the real world units. Since this algorithm is for a Bin-Picking system it is obvious that actually the main goal of the algorithm is not to detect objects in an image but to detect objects in the bin, in the real world, and since the camera is the device that realizes the connection between digitized world and real world, it has to be calibrated.

According to the calibration method, Zhang et al. [22] classify calibration techniques roughly into 2 categories:

- **Photogrammetric calibration.** Camera calibration is performed by observing a calibration object whose geometry in 3D space is known with very good precision. Calibration can be done very efficiently. The calibration object usually consists of two or three planes orthogonal to each other. Sometimes, a plane undergoing a precisely known translation is also used. These approaches require an expensive calibration apparatus, and an elaborate setup.
- **Self-calibration.** Techniques in this category do not use any calibration object. Just by moving a camera in a static scene, the rigidity of the scene provides in general two constraints on the camera's internal parameters from one camera displacement by using image information alone. Therefore, if images are taken by the same camera with fixed internal parameters, correspondences between three images are sufficient to recover both the internal and external parameters which allow to reconstruct 3-D

structure. While this approach is very flexible, it is not yet mature. Because there are many parameters to estimate, and is not always possible to obtain reliable results.

The algorithm used to calibrate the camera is the one presented in the a toolbox for Matlab [7], based on the paper [22]. This approach lies between the photogrammetric calibration and self-calibration, because it uses 2D metric information rather than 3D or purely implicit one. It is a flexible technique that does not require any knowledge in computer vision in order to work with it and, overall, does not require expensive equipments. In order to calibrate the camera it just requires the camera to observe a planar pattern shown at a few different orientations (at least two, more different orientations are tested more the calibration is robust). This pattern can be easily printed and attached to a planar surface. To take photos of the planar pattern the camera can move or, otherwise, the pattern can be moved, and the motions need not be known. Compared with self-calibration, it gains considerable degree of robustness.

This calibration method is the same used in [1] and has not been implemented in this project, prof. Pretto just gave available the camera intrinsic matrix.

The Algorithm in Brief

Intrinsic Parameters Considering a 2D point denoted by $m = [u, v]^T$, a 3D point denoted by $M = [X, Y, Z]^T$, and their augmented vector by adding 1 as the last element denoted by $\tilde{m} = [u, v, 1]^T$ and $\tilde{M} = [X, Y, Z, 1]^T$. A camera is modelled by its pinhole, the relationship between a 3D point M and its projection m in the photo, that is given by:

$$s\tilde{m} = \boldsymbol{A}[\boldsymbol{R}\ t]\tilde{M}, \tag{4.2.1}$$

where s is an arbitrary scale factor, $[\boldsymbol{R}\ t]$, called the *extrinsic parameters*, are the rotation and translation which relate the world coordinate system to the camera coordinate system, and $\boldsymbol{A}$, called the *camera intrinsic matrix*, is given by:

$$\boldsymbol{A} = \begin{pmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.2.2}$$

with (u_0, v_0) the coordinates of the principal point, which would be ideally in the centre of the image, α and β the scale factors in image u and v axes, and γ the parameter describing the angle between the two image axes.

The algorithm detects the feature points in the images, estimates the five intrinsic parameters that construct 4.2.2, estimates the coefficients of the radial distortion, that affects the point projection, by solving a linear least-squares problem and finally it refines the parameters by minimizing an objective function.

In order to tell to computer what is the pattern used for the calibration, the user has to click manually on one of the images shot the four corners of the pattern, as showed in image 4.2(a). Later the algorithm automatically will recognize the squares that compose it, and from the images with different orientations will be able to extract the intrinsic parameters by minimizing the mentioned objective function.

Extrinsic Parameters Extrinsic parameters are the ones which denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates, that is to say it consists in finding $[\boldsymbol{R}\, t]$ for each point. Considering a set of n 3×3 rotation matrices R_i and a set of n 3×1 translation vector T_i where $i = 1, 2, \ldots, n$ and n is the number of images taken with the camera. Let consider the calibration grid $grid_i$ attached to the i-th image, and to make more simple the explanation let consider the image $1 \rightarrow i = 1$. Let P

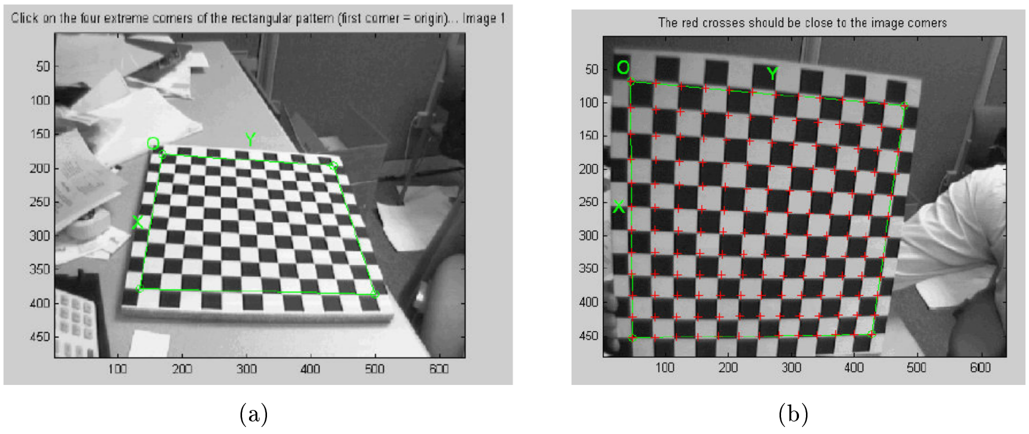

Click on the four extreme corners of the rectangular pattern (first corner = origin)... Image 1
The red crosses should be close to the image corners
O
X
Y
(a)
(b)

Fig. 4.2: Corner selection and pattern definition. (Image taken from [7]).

be a point space of coordinate vector $[X_P, Y_P, Z_P]$ in the grid reference frame showed in Fig. 4.3. Let $C = [X_C, Y_C, Z_C]$ be the coordinate vector of P in the camera frame. Then P and C are related to each other through the following rigid motion equation:

$$C = R_1 \cdot P + T_1$$

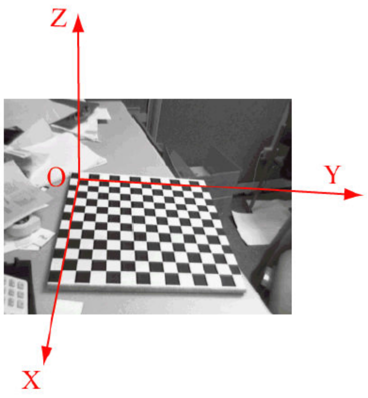


Fig. 4.3: Reference frame (O, X, Y, Z) attached to the calibration grid $grid_i$. (Image taken from [7])

In particular, the translation vector T_1 is the coordinate vector of the origin of the grid pattern (O) in the camera frame, and the third column of the rotation matrix R_1 is the surface normal vector of the plane containing the planar grid in the camera reference frame. The same relations can be found for the images $i = 2, 3, \ldots, n$. Once the coordinates of a point are expressed in the camera reference frame, it may be projected on the image plane using the intrinsic camera parameters.

The points used for the calibration are the corners of the small squares composing the pattern, ones marked with a red cross in Fig. 4.2(b).

Implementation and Its Uses To implement this calibration some functions developed for this calibration, and provided by OpenCV library, have been used.

The calibration has to be made for every camera, does not depend on the model or something else, but the distortion effects are due to camera lens that has some imperfections, therefore they are unique for each camera.

The calibration has to be made once in a while, in order to correct some physical changes that has suffered the camera in its working life, and obviously it is computed off-line.

4.3 Generalized Hough Transform For Object Detection

This method consists of a generalization of the Standard Hough Transform, the method has been extended in order to detect whatever shape. Through an accumulator and some tests, that will modify the accumulator, the algorithm is able to evaluate if a point can be considered as the centre point of the shape, if the accumulator in that point has a higher value than a fixed threshold.

This algorithm has been used as starting point for almost all recent studies about pattern recognition.

4.3.1 The Algorithm in Brief

To generalize the Hough algorithm three parameters are used to describe in a detailed way a generic shape

$$a = (y, s, \theta)$$

where $y = (x_r, y_r)$ is a reference origin for the shape, θ is its orientation and $s = (s_x, s_y)$ describes two orthogonal scale factor along x and y axis.

The reference origin location y is described in terms of a table of possible edge pixel orientations. The computation of the parameters s and θ is then accomplished by straight-forward transformations to this table.

An initial attempt to generalize the Hough Transform consisted of considering that each shape has a specific reference point, then only the boundary points are considered. An analytic description is used to describe these boundary points relative to the reference point of the shape. Moreover, also a two-dimensional accumulator array $A(a)$ is used and initialized to zero. On the input image a gradient operator is applied and then the image will be binarized through a simple binarization function:

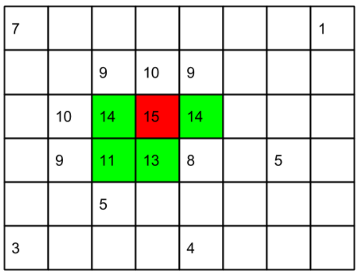
$$B(x) = \begin{cases} 1 & \text{if } x \text{ is an edge pixel} \\ 0 & \text{otherwise} \end{cases}$$

Then the algorithm tests each pixel of the binarized image, considering it as a possible reference point, and for each one looks for boundary points x_B, corresponding to pixels with a value of 1, looking in a perimeter that corresponds to its edges. For every pixel tested is saved in the accumulator the number of pixels that correspond to boundary points relative to reference point tested x.

Last the algorithm looks for local maximas in $A(a)$ which correspond to instances of the searched shape in the image. A simplify illustration of the accumulator is given by Fig. 4.4.

This approach is actually impractical for real image data, in fact an image with a multitude of edge pixels leads to several false instance detections due to coincidence pixel arrangements. Anyway suggests the manner to approach shape detection problem.

The key for the generalization to arbitrary shapes is the use of directional information, that even makes the algorithm faster and more accurate. For example, suppose to have to detect a circle, every significant group of edge points that lies on a circle will be detected as a circle, even if those points do not represent a circle. This happen if we do not use directional information. Considering the case of the circular boundary detector with a



7							1
		9	10	9			
	10	14	15	14			
	9	11	13	8		5	
		5					
3				4			

Fig. 4.4: Simple illustration of the accumulator A in the case of 8x6 image , with no rotations considered and with a step of 1 pixel. Supposing a threshold of 11 the red square corresponds to the most probable istance of the searched object, while green squares are other possible istances.

fixed radius $\boldsymbol{r_0}$, for each gradient point x with a direction ϕ the reference point a is due from $a = x + \boldsymbol{r_0}$.

Extending the idea of the circle detector with a fixed radius to the case of an arbitrary object, for each point x on the boundary with gradient direction ϕ, the accumulator is incremented in the point $a = x + \boldsymbol{r}$. The difference of the circle case is that now $\boldsymbol{r} = a - x$ will vary in magnitude and direction depending on boundary points, as illustrate Fig. 4.5.

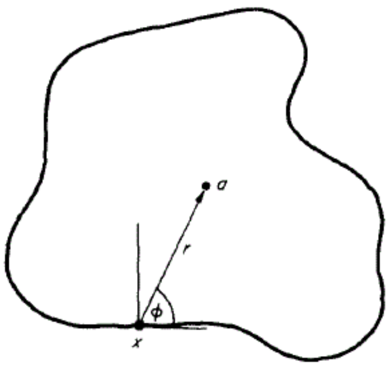


Fig. 4.5: Geometry of an arbitrary shape for Generalized Hough Transform (Image taken from [8])

The fact that $\boldsymbol{r}$ varies in an arbitrary way means that Generalized Hough Transform for an arbitrary shape is best represented by a table called **R-table**, such as Tab. 4.1.

The construction of R-table is made as first step, as soon as the algorithm receives as input the shape model to detect. Its construction is easily constructed by examining the boundary points of that object.

First the algorithm for the construction of R-table chooses a reference point y in an arbitrary way, for each boundary point computes the gradient direction $\phi(x)$ and $\boldsymbol{r} = y - x$ and stores $\boldsymbol{r}$ as function of ϕ . Generally an index ϕ can have many values of the radius, think about a ring that can be thought as a composition of two circles.

To detect the shape in the image each pixel x in the image increments all the corresponding points $x + \boldsymbol{r}$ in the accumulator array A, where $\boldsymbol{r}$ is a table entry indexed by ϕ. The maxima in A correspond to possible instances of the searched object.

The next step is adapting the algorithm developed until now to shapes with different orientations θ and scale s of the model shape given as input. Therefore the accumulator array now has to consider the reference point, scale and rotation, so it consists of three dimension: $A(y, s, \theta)$.

Tab. 4.1: Diagrammatically form of the R-table.

i	ϕ_i	R_{ϕ_i}
0	0	$r \vert a - r = x, x \text{ in } B, \phi(x) = 0$
1	$\Delta\phi$	$r \vert a - r = x, x \text{ in } B, \phi(x) = \Delta\phi$
2	$2\Delta\phi$	$r \vert a - r = x, x \text{ in } B, \phi(x) = 2\Delta\phi$
...	...	...

The R-table can also be used to increment this larger dimensional shape, since different orientations and scales correspond to easily-computed transformations of the table. Therefore a R-table of a shape S with a orientation θ and a scale factor s can be seen as a multiply-vector-valued function and it can be denoted as: $R(S, \theta, s)$.

To appreciate better the transformation of the R-table refer to Fig. 4.6. In the figure, an edge pixel with orientation ϕ may be considered as corresponding to the boundary point x_A, in which case the reference point is y_A. Alternatively, the edge pixel can be considered as x_B on a rotated instance of the shape, in which case the reference point is at y_B which can be specified by translating r_A to x_B and rotating it through $+\Delta\theta$.

Considering a 3D rotation the R-table can be constructed simply applying the rotation matrix of the considered rotation to the R-table.

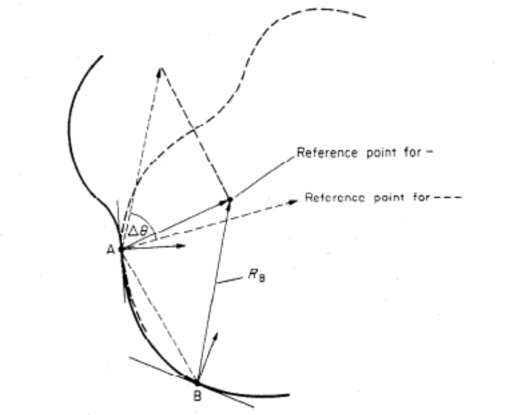


Fig. 4.6: Construction for visualizing the R-table transformation for a rotation by $\Delta\theta$ in a 2D Euclidian space. Point A can be viewed as: (1) on the shape with the continuos line, or (2) as point B on the shape with the discontinuos one, rotated by $\Delta\theta$. (Image taken from [8])

This Generalized Hough Transform, thanks to R-table system, allows to detect also composed objects that are composed of more shapes using the R-tables of the basic shapes of which is composed.

Therefore this object detection method has the following main properties:

- Scale changes, rotations and reference point translation of the shape S can be accounted for by modifications to the R-table, that can be computed off-line.
- Given the boundary of S its R-table can be easily constructed and requires a number of operations proportional to the number of boundary points.
- Its accuracy depends on the rotation step $\Delta\theta$ used to construct R-table and to the translation step.

The concept described above is the basic concept of the Generalized Hough Transform and the paper [8] explains some optimizations and incrementation strategies.

4.3.2 Use of Generalized Hough Transform in the earlier work of Pretto et al.

For the sake of efficiency, they limit the parameters space to scale and rotations around z axis (so-called yaw), and 2D translations along the x (pitch) and y (roll) axis. This is like assume that all objects are disposed on ideal planes, perpendicular to the camera axis, since this assumption is constantly violated in a real world scenario, they take into account rotations around x and y axes along the voting procedure, rotation anyway limited up to $\pm 40°$ due to robot arm movement limitations.

Starting from the CAD model of searched object, they make a rasterization of the template, a set of m points $o_1, \dots, o_m \in \Re^3$ in the object reference frame, along the related discretized direction $dir(o_i)$,$i = 1, \dots, m$.

Assuming that the possible minimum and maximum heights, distance from the camera lens, are given as input parameters, from these are defined discretized distances that represent the first dimension in the parameter space. For each height they define a set of t discretized rotations defined in the space of all the possible rotations around the camera optical axis (object yaw). For each h distance and k rotation (that corresponds to a rotation in the 3D Euclidean space), they define a bi-dimensional accumulator $A(p)$, $p \in \Re^2$, representing the 2D discretized translations along x, and y axis of the defined parameter space. The scale, indicated by the h height considered, and the k rotation, can be represented as a rigid body transformation from the object frame to the camera frame, this consist in the rotation transformation of the R-table of searched object.

The shape's raster boundary, rotated and scaled, will be projected on the image plane, note that now instead to use binarized points of the image like in [8] they use edgelets (straight segments that can be part of a longer, possibly, curved, line) given by the LSD. These will be used to vote in the reduce parameters space represented by the accumulator $A(p)$. For each edgelet given by LSD they collect a set of n edgels $\{e_1, \dots, e_n\} \in \Re^2$ that represent the edgelets pixels, expressed in normalized image coordinates, along with the related edgelet *discretized* directions $dir(e_i) =, i, \dots, n$. This is the main modification they made on the original Generalized Hough Transform and that leads the algorithm to be faster in the detection stage.

For each $\widehat{o}_i^{h,k}$, they select the subset:

$$E_i^{h,k} := \{e_j | dir(e_i) = dir(\widehat{o}_i^{h,k})\}$$

where $\widehat{o}_i^{h,k}$ is a raster point that corresponds to the shape's boundary point at the h height and rotated by a k 3D rotation, projected on the camera frame. The subset selected corresponds to the edgelet with the direction alike to $\widehat{o}_i^{h,k}$.

For each $e_j \in E_i^{h,k}$, the accumulator will be increased by an unity at the point:

$$p = \frac{e_j - \widehat{o}_i^{h,k}}{step}$$

where e_j is an edgel of the subset $E_i^{h,k}$, and *step* is the translation defined for the accumulator A and will affect the accuracy of the object phase detection.

In other words, if a raster point o_i is projected on the image plane we obtain a point where the image coordinates represent that displacement from the origin of the object. Only the edgels e_j with the same direction of $\widehat{o}_i^{h,k}$ can be matched with this point, so they vote for an object 2D translation given by $e_j - \widehat{o}_i^{h,k}$. To improve the efficiency all the possible sets of rasterization (R-table) are computed off-line.

In order to make more robust the algorithm they have employed an optimization procedure that estimates the position in the 3D space of searched object, by minimizing an image-based cost function in a similar way to the one employed in this project and commented in subsection 4.4.3.

4.4 Chamfer Matching For Objects Detection

Present section explains Standard Chamfer Matching [21] and its use in this project.

4.4.1 Chamfer Distance

Standard Chamfer Matching is a popular technique to find the best alignment between a model rasterized and a query of edge map. Let $U = \{u_j\}$, where $i = 1, 2, \ldots, |U|$, be the set of raster points of the object model, and let $V = \{v_j\}$, where $j = 1, 2, \ldots, |V|$, be the set of edge pixels from the input image in which an edge operator has been applied. The *chamfer distance* between the object model U and the template edge map V, originated by the edge operator, is defined as the average over all pixels $\boldsymbol{u}_i \in U$ of the distance between $\boldsymbol{u}_i$ and its nearest pixel in V:

$$d_{CM}(U, V) = \frac{1}{n} \sum_{\boldsymbol{u}_i \in U} \min_{\boldsymbol{v}_j \in V} ||\boldsymbol{u}_i - \boldsymbol{v}_j|| \qquad (4.4.1)$$

where n is the number of template edge pixels, $n = |U|$.

Let W be a warping function defined on the image plane that projects the template onto the camera frame and that is parametrized by s. For instance, if W is a 2D Euclidean transformation, then $s \in SE(2)$ can be written as $s = (\theta, t_x, t_y)$, where t_x and t_y are translations parallel to the x and y axes, respectively, and θ is the in-plane rotation angle. Its action on each image point $x \in \Re^2$ is given via the transformation

$$W(x; s) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} x + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \qquad (4.4.2)$$

In other words, the object model is subjected to a transformation, through rotation and translation matrices, that projects it onto the camera frame, later the algorithm computes the chamfer distance in order to establish how well the projected object match the shape in the image. Note that this step is applied in a binarized image, later to have suffered the edge operator.

The chamfer matching cost can be computed efficiently using the *distance transform* image

$$DT_V(x) = \min_{v_j \in V} ||x - v_j|| \qquad (4.4.3)$$

which specifies the distance from each pixel x in the distance transform image to the nearest edge pixel in V. In Fig. 4.7 is illustrated the distance transform image of the usual input image.

It is standard practice to truncate the *distance transform* to a value τ:

$$DT_V^{\tau}(x) = \min(DT_V(x), \tau) \qquad (4.4.4)$$

so that missing edgels due to noisy edge detection, or for cluttered object, do not have too a severe effect on chamfer distance calculation.

The distance transform can be computed in two steps over the image using dynamic programming. Using the distance transform the cost function 4.4.1 can be evaluated in linear time $O(n)$ through:

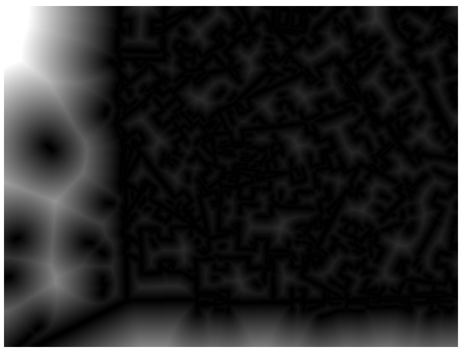


Fig. 4.7: Distance Transform Image - Dark pixels correspond to edges or to points near edges, while the white ones correspond to point far from edges.

$$d_{CM,\tau}(U,V) = \frac{1}{n}\sum_{u_i \in U} DT_V^{\tau}(u_i) \tag{4.4.5}$$

Chamfer matching provides a fairly smooth measure of fitness and can tolerate small rotation, misalignments, occlusions, and deformations. However, it becomes less reliable in the presence of background clutter due to an increase in the proportion of false correspondences. Note that this explanation has no considered rotations around x (pitch) and y (roll) axis, in other words it supposed to have to detect objects on an ideal plane.

To extend the explanation to the 3D Euclidean space case is sufficient to consider $s \in SE(3)$, which can be written as $s = (\gamma, \beta, \alpha, t_x, t_y, t_z)$, where t_x, t_y and t_z are translations parallel to the x, y and z axes respectively, and α,β, γ are the in-space rotation angles around z,y and x axes respectively. Considering a $\bar{x} = (x_0, y_0, z_0)^T$ point in the 3D space, the warping function W 4.4.2 can be rewritten as:

$$W(x;s) = R\bar{x} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \tag{4.4.6}$$

where R is the 3D rotation matrix defined through the yaw, pitch and roll notation:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}, R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}, R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

4.4.2 Use of Line-Based Representation in Chamfer Matching

Exactly as in the Modified Generalized Hough Transform, explained in subsection 4.3.2, a line based representation has been used. This because it is a better representation of the image, since it can take the edge noise off, that would remain in a point-based representation.

The rasterized model will be rotated and translated, projecting it on the camera frame, through the warping function 4.4.6, in order to test all the possible positions that the robot arm can handle. In the following explanation $t = (t_x, t_y, t_z) \in T, T = t_1, t_2, \ldots, t_{m_t}$ refers to a translation in the 3D Euclidean space along a set of possible translations admitted T, while $k = (\alpha, \beta, \gamma) \in K, K = k_1, k_2, \ldots, k_{m_k}$ refers to a rotation in the 3D Euclidean space along a set of possible rotations admitted K. Translations and rotations are taken into account in the alignment parameter s.

On the edge map V, consisting in edge points that make part of line segments, the *distance transform* 4.4.4 is applied, where the eq. 4.4.3 can be written as:

$$DT_V(\widehat{o}_i^{t,k}) = \min_{e_j \in V} ||\widehat{o}_i^{t,k} - e_j|| \tag{4.4.7}$$

It evaluates the distance between the raster point $\widehat{o}_i^{t,k}$ and $\widehat{e}_j$, the nearest point of the line segments map V.

Then the *chamfer distance* can be easily evaluated in the following manner:

$$d_{CM,\tau}(W(U,s),V) = \frac{1}{n} \sum_{\widehat{o}_i^{t,k} \in U} DT_V^{\tau}(\widehat{o}_i^{t,k}) \tag{4.4.8}$$

Note that now V is seen as a set of edge points of line segments representing the input image: $V = e_j$ where $j = 1, 2, \ldots, |V|$.

4.4.3 Robust the chamfer distance

A simple use of the chamfer distance is, as it is explained before about Generalized Hough Transform, using an array that takes into account the chamfer distance between the projected model and the input image, using the line-based representation, and the best matches would be evaluated, with the scoring function, in order to understand if they are false positives or not.

This approach obviously suffers of imprecision since the result depends widely on the yaw, pitch and roll rotations steps as well as the translation steps. In order to make the object detection more robust we find the set of variables, that influence the object model rotations and translations, that minimize the cost function *chamfer distance*. This is actually the most common way to use *chamfer matching* and in computer vision is really common using this approach in order to minimize some objective functions.

Starting from a hypothetical position indicated by the alignment parameter s, that considers a translation $t_{x,y,z}$ and a rotation $k_{y,p,r}$, the algorithm looks into, a neighbourhood of the considered position, for the combination of translations and rotations, that is to say alignment parameter $\widehat{s} = (\widehat{\gamma}, \widehat{\beta}, \widehat{\alpha}, \widehat{t_x}, \widehat{t_y}, \widehat{t_z})$, that minimizes the *chamfer distance*. Therefore $\widehat{s}$ is given by the argument of the minimum *chamfer distance* with a certain rotation and translation:

$$\widehat{s} = \arg \min_{s \in SE(3)} d_{CM,\tau}(W(U,s),V) \tag{4.4.9}$$

$$R_{min} \leq s \leq R_{max}$$

where $W(U,s) = W(u_i, s), i = i, 2, ..., |U|$ is the set of points of the object model rotated by γ, β, α and translated by t_x, t_y, t_z along x, y, z axe respectively, and R_{min} and R_{max} are bounds of the trust region in which looking for $\widehat{s}$.

In the implementation of the algorithm presented in this thesis actually any bound has not been considered for the trust region, but they should be considered in a possible future work, in order to be more precise and even faster.

This is a classic problem of non-linear optimization. To performance the algorithm to minimize the *chamfer distance* the **ceres-solver** library [23] has been used. This is an open source C++ library for modelling and solving large complicated non-linear least squares problems. It is a feature rich, mature and performant library which has been used in production since 2010.

In ceres library, a lot of different ways to resolve non-linear optimization problems are available. In this phase the DENSE SCHUR method has been used. This method is quite fast since has $O(p^3)$ time complexity.

The algorithm that minimizes the *chamfer distance* gives a lot of false positives due to the fact that sometimes the method does not converge so it must be stopped to a certain value. Anyway the scoring function evaluates the resulting alignment parameter $\widehat{s}$ so false detections are easily taken off.

Fig. 4.8 illustrates a simple example of the optimization result and the use of scoring function.

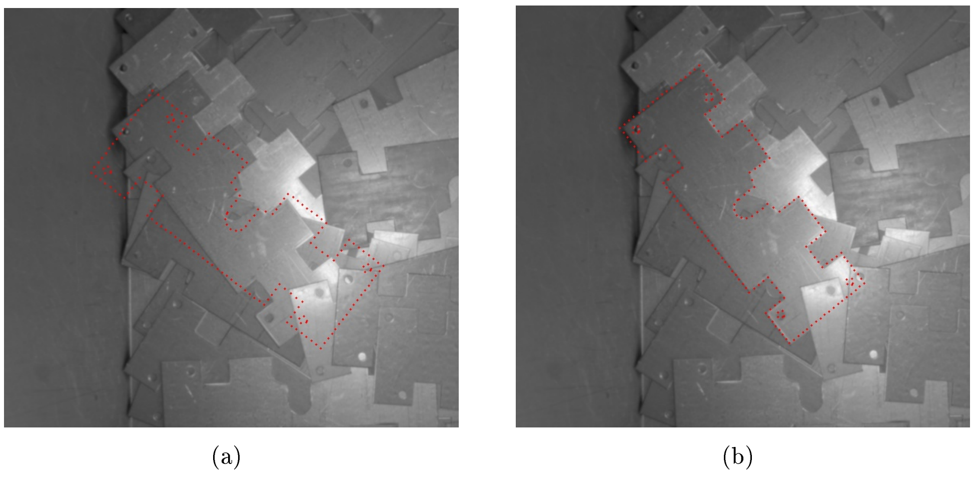


Fig. 4.8: Optimizations and scoring: (a) illustrates the position of the projected template (color red) that is tested, and with some optimizations the algorithm gives as hypothesis (b), that later it is tested with the scoring function: the match showed in (b) has a score of 0,828277 and it is considered as a valid one.

A further optimization, to reduce the execution time of the algorithm, consists of doing the non-linear optimization only with initial hypothesis that have a chamfer distance below to a certain threshold, in our tests a value of 8.0 showed to be the best value for this threshold. It makes no sense making the optimization when the considered hypothesis is quite far to match a real object.

4.5 Scoring

Since some of the selected hypothesis used as initial guess of the optimization may lead to false positives, a scoring function has been used to allow discarding the outliers and selecting the best matches.

The scoring function is actually the same used in [1] and is based on local image gradient directions. Given $\widehat{o}_i^{t,k}$ the n raster points projected on the image plane, translated with a t translation in the 3D Euclidean space and rotated with a k rotation in the 3D Euclidean space, it is easy to compute their normal directions $n_{dir}(\widehat{o}_i^{t,k})$: in the case of a perfect match, these normal directions should correspond to the local gradient directions $I_\theta(\widehat{o}_i^{t,k})$ (up to a rotation of π *radians*), where I_θ is the gradient direction image. The scoring function used is the following:

$$\Psi(W(U,s)) = \frac{1}{n}\sum_{1}^{n} |cos(I_\theta(\widehat{o}_i^{t,k}) - n_{dir}(\widehat{o}_i^{t,k}))|$$

where W is the usual warping function 4.4.6 that projects template points U onto the camera frame.

If there is not an edge point in the gradient image in the coordinates of $\widehat{o}_i^{t,k}$, that is to say that its gradient magnitude and direction is zero, the alignment considered for that point is $\tau = 0.6$, where alignment is $cos(I_\theta(\widehat{o}_i) - n_{dir}(\widehat{o}_i))$.

The scoring function can take values from 0 to 1, where 1 represents a perfect match. In experiments made by Pretto et al, and also in the ones made in this project, all good matches (in-layers) obtain a score greater than 0.8. So all matches with a score minor than this threshold are discarded, since they could be false positives or real objects partially occluded. In the case of real objects partially occluded discarding them will not be a problem since the algorithm will be iterated once all good matches find has been grasped by the robot arm. A simple use of the scoring function is illustrated in Fig. 4.8.

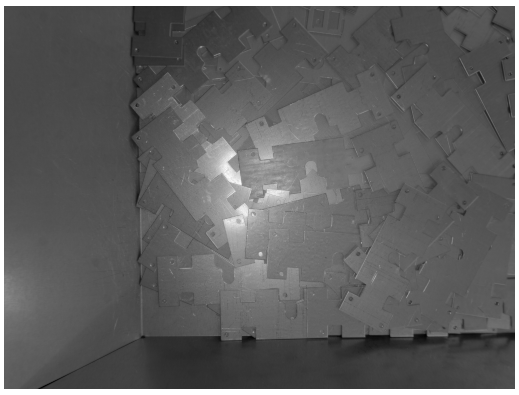
4.6 Tests

Different rotation steps around the camera axis have been tested in order to get the best rotation step accordingly to the refinement procedure. In fact, as the reported tests show, we obtain different results considering different rotation steps. Only the yaw rotation, around the camera axis, has been considered; in fact, the optimization procedure can detect even object rotated around x and y axes. Moreover, with planar objects the sloops that we can encounter are small, and the optimization can find them with no problems. The translation step considered, along x and y axes is equal to 1 centimetre, along the z axis it is equal to 10 centimetre, and the rasterization step of the CAD model is equal to 2 millimetres.

Two different scenes with two different models have been tested. Moreover, more tests have been made with different yaw rotation steps in order to show how detection control changes. Note that some impossible objects are detected, these outliers can be easily discarded checking pitch and roll rotations.

The algorithm gives as output the best matches, therefore different matches (different positions) can correspond to the same object. It can be easily resolved considering the best match in a neighbourhood of each detected match.

Tests The firsts tests have been made on the input image 4.9(a), looking for the object 4.9(b).


(a)

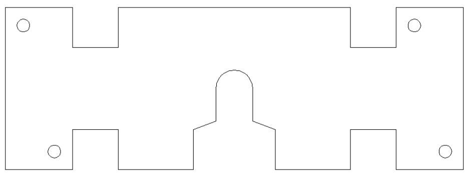
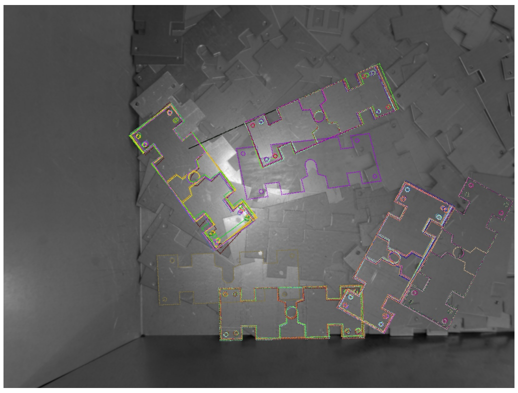
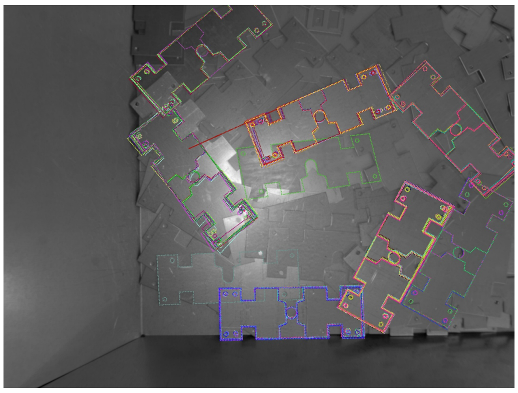

(b)

Fig. 4.9: Inputs of the first test.

The algorithm looks for the objects that are far away from the camera lens, from 1 meter up to 1.2 meters, and it looks for them over all the image with a translation pitch of 10 centimetres along camera axis and with a pitch of 1 centimetre along x and y axes. We have considered different rotations around the camera axis for each test, considering only two rotations around y axis, it means that we look for the object with the initial orientation and later with a rotation of π around y axis. Fig. 4.10 and Tab. 4.2 show the results (the coloured objects are the good matches), with the commented configuration and with different yaw steps considered.


(a) $\Delta yaw = 2\pi/12$


(b) $\Delta yaw = 2\pi/18$

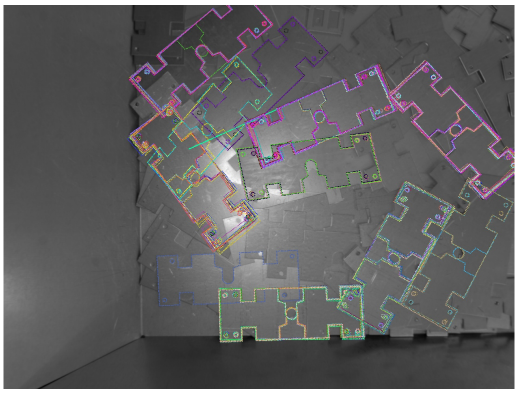

(c) $\Delta yaw = 2\pi/36$

Fig. 4.10: Results of the first test with the inputs of Fig. 4.9.

Tab. 4.2: Results of the tests shown in Fig. 4.10.

Image	**Yaw step**	**Elapsed time**	**Number of matches**
a	$2\pi/12$	88.53	88
b	$2\pi/18$	137.988	121
c	$2\pi/36$	279.375	212

Notice that the algorithm detects the best matches, so more matches can correspond to the same object. In fact, in the test Fig. 4.10(c) 212 matches are detected but actually these correspond to 9 objects.

More tests have been made in the input image Fig. 4.11 looking for the object 4.11(b), with the same parameters setting.

In this case we can notice that testing more yaws doesn't lead to improvements about precision, and it makes the object detection slower.

In order to have a robust and fast object detection, rotations and translation limits have

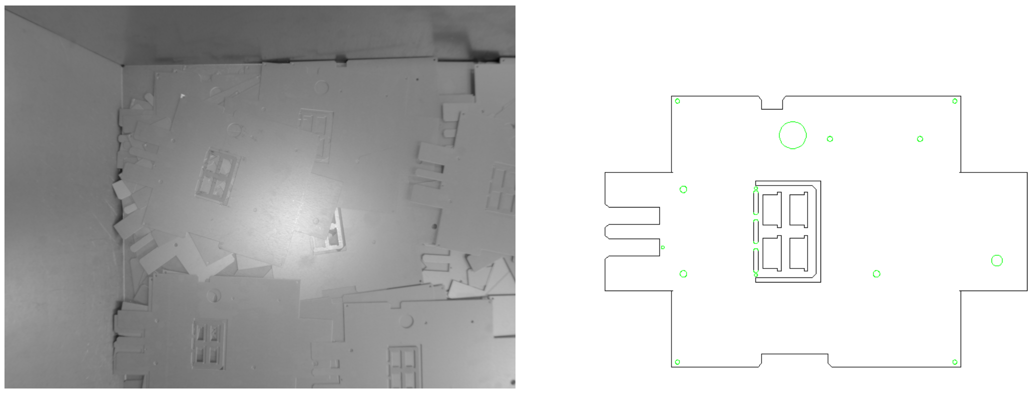
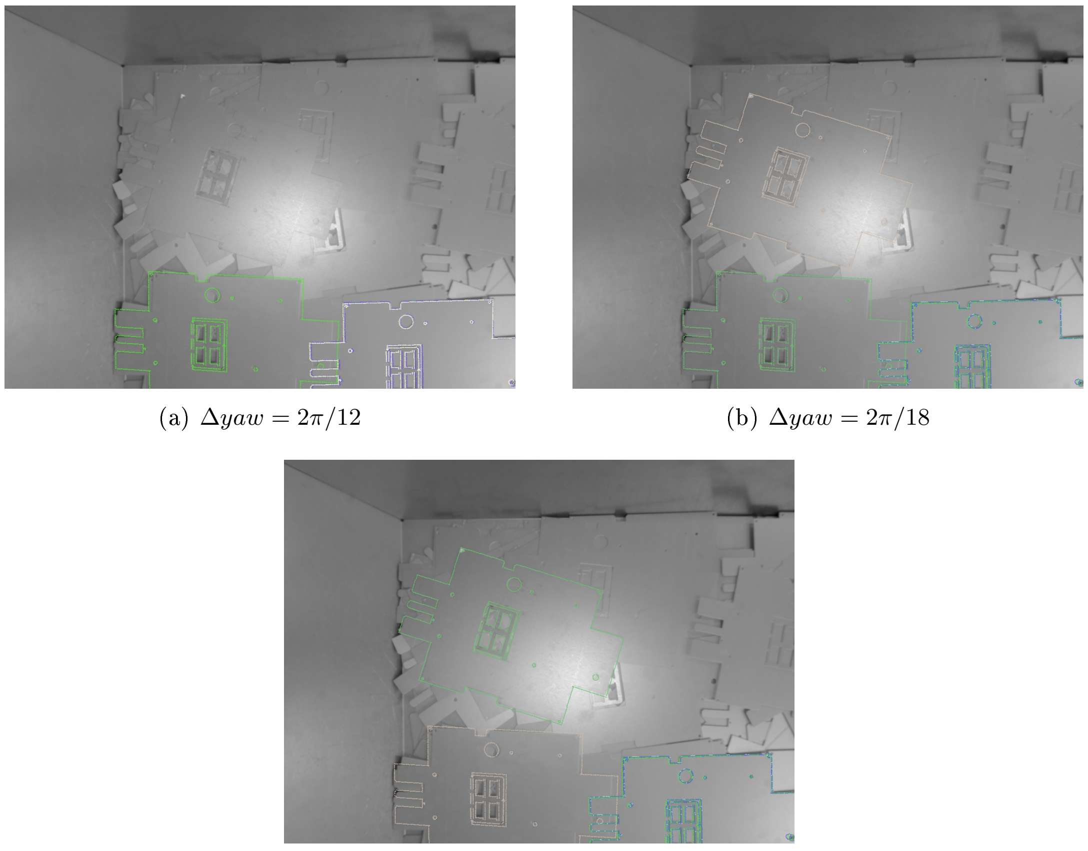

(a) (b)

Fig. 4.11: Inputs of the second test.


(a) $\Delta yaw = 2\pi/12$ (b) $\Delta yaw = 2\pi/18$

(c) $\Delta yaw = 2\pi/36$

Fig. 4.12: Results of the second test with the inputs of Fig. 4.11.

Tab. 4.3: Results of the tests shown in Fig. 4.12.

Image	Yaw step	Elapsed time	Number of matches
a	$2\pi/12$	11.376	5
b	$2\pi/18$	18.38	6
c	$2\pi/36$	36.277	6

to be tuned accordingly with the environment conditions and with the searched objects. In particular, the choice of the limits for the translation along camera axis has to be based on the robot cell in which this algorithm will be used.

It is easy to understand that elapsed times are too long for a practical use because these elapsed times refer to a dual-core machine. For a practical implementation, a multi-core processor with more cores will be used. Today a lot of architectures based on multi-core processors are available with a lot of cores (8-10 cores). These multi-core processors can run different parts of the same program simultaneously, making the execution faster, proportionally to the number of cores. Therefore, supposing to use a processor with 10 cores, the elapsed time in the case showed in Fig. 4.10(b) would be $\approx \frac{137.99}{5} \approx 27.6$ seconds.

Chapter 5

Future Improvements

The use of the standard chamfer distance actually does not lead improvements compared to Generalized Hough Transform, used in [1]. But in the work documented in the paper [9] a new algorithm is explained, it is based on the standard *chamfer distance* with some important improvements. Actually at the beginning of this project the main goal was to perform this algorithm, but for sake of time it could not be possible. Therefore this section will comment the improvements that should be applied to the standard chamfer matching in order to get a more robust and faster algorithm for the object detection.

Oriented Chamfer Matching To improve the robustness of chamfer matching, Shotton et al [2] presented a new method in order to exploit edge orientation information.

In their work chamfer matching is augmented with an additional cost for orientation mismatch, which is given by the average difference in orientations between template edges u_i and their nearest edge point in the query image V. This method is known as *oriented chamfer matching* (OCM). The cost for orientation mismatch is given as:

$$d_{orient}(W(U,s),V) = \frac{2}{\pi|U|}\sum_{u_i \in U} |\phi(u_i) - \phi(ADT_V(u_i))| \tag{5.0.1}$$

where the function $ADT_V(x)$ computes the argument distance transform (ADT) which gives the locations of the closest points to x in V, and is defined as:

$$ADT_V(x) = arg \min_{v_j \in V} ||x - v_j||$$

The function $\phi(x)$ gives the orientation of edgel x, with a modulo π, and $|\phi(x_1)-\phi(x_2)|$ gives the smallest circular difference between $\phi(x_1)$ and $\phi(x_2)$.

Edgels are taken modulo π because, for edgels on the outline of an object the sign of the edgel gradient is not a reliable signal since it depends on the intensity of the background. The normalization by $\frac{\pi}{2}$ ensures that $d_{orient}(W(U,s),V) \in [0,1]$. The OCM distance is graphically illustrated in Fig. 5.1. The improved matching scheme, called *oriented chamfer matching* (OCM), uses a simple linear interpolation between the distance and orientation terms:

$$d_\lambda(W(U,s),V) = (1-\lambda)\cdot d_{CM,\tau}(W(U,s),V) + \lambda \cdot d_{orient}(W(U,s),V) \tag{5.0.2}$$

with *orientation specificity* parameter λ that depends on the contour fragment.

Since each chamfer distance 5.0.1 and 4.4.5 represents an empirical average, one does not need to perform the sums for all discretized points of boundary but over only a fraction of the edgels, adjusting the normalization accordingly. This is an approximation that makes the algorithm quite faster and the precision decreases only a few. With tests Shotton et al. have showed that even matching only 20% of edgels there is no a decrease in detection performance. This actually just affects the rasterization pitch.

Directional Chamfer Matching In the paper [9] Liu et al. have improved OCM in order to get an object detection more robust to clutter, missing edges, and small misalignments, proposing a matching cost function, called *Directional Chamfer Distance*, that

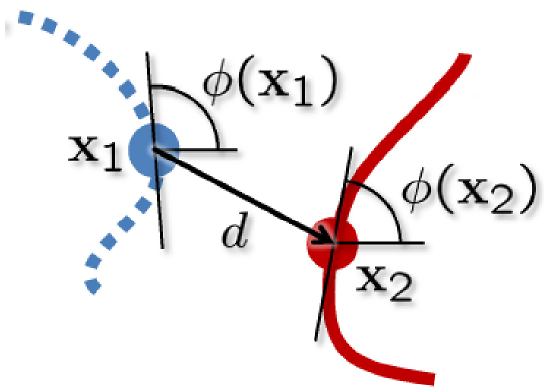


Fig. 5.1: Oriented chamfer matching: For edgel x_1 of the template, the contribution to the OCM distance is determined by the distance d from x_1 to the nearest edgel x_2 in edge map, and the difference between the edgel gradients,$|\phi(x_1) - \phi(x_2)|$. (Image taken from [2])

is a smooth function of both the translation (t_x, t_y) and the rotation (θ) of the template pose.

Each edge pixel x is augmented with a direction term, $\phi(x)$, and the directional chamfer matching score (DCM) is given by

$$d_{DCM}(W(U,s),V) = \frac{1}{n}\sum_{u_i \in U} \min_{v_j \in V}(||u_i - v_j|| + \lambda||\phi(u_i) - \phi(v_j)||_\pi) \qquad (5.0.3)$$

where the parameter λ, is a weighting factor between the location and orientation terms. To compute the direction term, they fit line segments to the edge points, as later will be said, and $\phi(x)$ is the orientation of the line segment associated with point x. Note that the directions are written modulo π: $0 \leq \phi(x) < \pi$, and the orientation error is defined as the minimum circular difference between the two directions:

$$||\phi(x_1) - \phi(x_2)||_\pi = \min\{|\phi(x_1) - \phi(x_2)|, ||\phi(x_1) - \phi(x_2)| - \pi|\} \qquad (5.0.4)$$

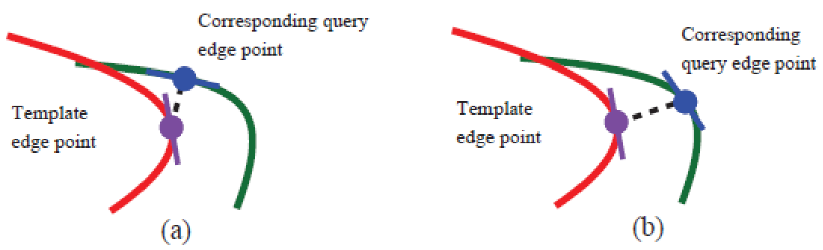

Corresponding query edge point
Template edge point
(a)
Corresponding query edge point
Template edge point
(b)

Fig. 5.2: Matching cost for an edge point. (a) OCM. (b) DCM. Whereas in OCM the location error is augmented with the orientation difference from nearest edge point, DCM jointly minimizes location and orientation errors. (Image taken from [9])

The working philosophies of method OCM and new method DCM are illustrated in Fig. 5.2.

The computational complexity of existing chamfer matching algorithms is linear in the number of template edge points. Although DCM includes an additional direction term it computes the exact DCM score with sub-linear complexity.

Three-Dimensional Distance Transform The matching score given in eq. 5.0.3 requires finding the minimum matching cost over location and orientation terms for each template edge point, therefore, its computational effort is quadratic in the number of template and query image edge points. Liu et al. [9] presented a three-dimensional distance transform representation in order to reduce computing complexity, of the matching cost, to linear time.

This representation is a three dimensional tensor in which the first two dimensions are the locations in the image plane and the third dimension belongs to a discrete set of edge orientations. Moreover the edge orientation is quantized into q discrete channels, $\widehat{\Phi} = \{\widehat{\phi}_i\}, i = 1, 2, \ldots, q$, which divide the range $[0, \pi)$. Each element x of the object template encodes the minimum distance to an edge point in the joint location and orientation space:

$$DT3_V(x, \phi(x)) = \min_{v_j \in V}(||x - v_j|| + \lambda||\widehat{\phi}(x) - \widehat{\phi}(v_j)||_\pi) \tag{5.0.5}$$

where $\widehat{\phi}(x)$ is the nearest quantization level in the orientation space $\widehat{\Phi}$ to the edge orientation $\phi(x)$. In order to compute the $DT3_V$ tensor in $O(q)$ passes, equation 5.0.5 can be rewritten as:

$$DT3_V(x, \phi(x)) = \min_{\widehat{\phi}_i \in \widehat{\Phi}}(DT_{V\{\widehat{\phi}_i\}} + \lambda||\widehat{\phi}(x) - \widehat{\phi}(i)||_\pi) \tag{5.0.6}$$

where $DT_{V\{\widehat{\phi}_i\}}$ is the two dimensional distance transform of the edge points in V that have edge orientation $\widehat{\phi}_i$. First they compute q two-dimensional distance transforms $DT_{V\{\widehat{\phi}_i\}}$, which requires $O(q)$ passes over the image using the standard distance transform algorithm [24].

Subsequently, the $DT3_V$ 5.0.6 tensor is computed by using a second dynamic program for each image pixel separately. The tensor is initialized with the two dimensional distance transform, $DT3_V(x, \widehat{\phi}_i) = DT_{V\{\widehat{\phi}_i\}}(x)$, and is updated with some recursions.

The values of tensor entries continue to be updated in a circular form until the value for a tensor entry is not changed. Theses recursion are executed in linear time and their worst computational time cost is $O(q)$ passes over the image. Using $DT3_V$, the directional chamfer matching score of the template U can be computed as:

$$d_{DCM}(W(U, s), V) = \frac{1}{n}\sum_{u_i \in U} DT3_V(u_i, \widehat{\phi}(u_i)) \tag{5.0.7}$$

where the complexity is linear in n, the number of edge points in U.

Line-Based Representation Using a line-based representation the distance chamfer matching score 5.0.7 can be rewritten in order to use this representation.

Let $l_{[x_1,x_2]}$ represents the line segment in the image plane connecting pixel x_1 and x_2. Let $L_U = \{l_{[s_j,e_j]}\}, j = 1, \ldots, m$, be the line-based representation of template edge points U, where s_j and e_j are the start and end locations of the jth line segment respectively. Assuming that the line segment directions are restricted to q discrete channels $\widehat{\Phi}$, which is enforced in the line-based representation .

Since the edge points in a line segment all have the same orientation, which is the orientation $\widehat{\phi}(l_{[s_j,e_j]})$ of their line segment $l_{[s_j,e_j]}$, the directional chamfer matching score 5.0.7 can be rewritten as:

$$d_{DCM}(W(U, s), V) = \frac{1}{n}\sum_{l_j \in L_U}\sum_{u_i \in l_j} DT3_V(u_i, \widehat{\phi}(l_j)) \tag{5.0.8}$$

where l_j refers to line segment $l_{[s_j,e_j]}$. In this case V refers to all edge pixels that make part of a line segment, and the $DT3_V$ will consider only segments with the same orientation of the u_i raster point considered.

Distance Transform Tensor Moreover Liu et al. even use the integral image concept that can be used for fast calculation of regions sums and linear sums, leading the algorithm to have a complexity linear to the number of line segments. They presented an integral distance transform representation $IDT3_V$ to evaluate the summation of costs over any line segment in $O(1)$ operations.

Let x_0 be the intersection of an image boundary with the line that passes through x and has direction $\widehat{\phi}_i$. Each entry of the $IDT3_V$ tensor is given by

$$IDT3_V(x, \widehat{\phi}_i) = \sum_{x_j \in l_{[x_0,x]}} DT3_V(x_j, \widehat{\phi}_i)$$

The $IDT3_V$ tensor can be computed in one pass over the $DT3_V$ tensor. Using this representation the directional chamfer matching score of any template U can be compute in $O(m)$ operations via

$$d_{DCM}(W(U,s),V) = \frac{1}{n} \sum_{l_{[s_j,e_j]} \in L_U} IDT3_V(e_j, \widehat{\phi}(l_{[s_j,e_j]}) - IDT3_V(s_j, \widehat{\phi}(l_{[s_j,e_j]})$$

Moreover, they presented a search optimization that leads the algorithm to ignore some hypothesis that wouldn't lead to a detection, and therefore their algorithm has a sub-linear complexity.

Evidences of the Work of Lieu et al. The results of their work show how the distance chamfer matching cost function they proposed is a sub-linear function of the number of template points. Moreover only a part of the template line segments can be considered for the matching and not all template line segments. Recapitulating, they present an algorithm that is an evolution of the *oriented chamfer matching* (OCM), it is really reliable and precise for object detection and has a sub-linear complexity. These benefits make this path a good improvement to follow to improve the work presented in this thesis in order to get a more precise and faster Bin-Picking system.

Conclusions

This Project presents an algorithm for the object detection of planar objects for Bin-Picking systems. The algorithm can recognize objects, even if they are sloping, using only a 2D industrial camera.

The algorithm takes as inputs the photo of the bin where it has to look for the objects, and the object's CAD template.

It gives as outputs all the good matches, some of which could correspond to impossible positions. These impossible matches, given as outputs, can be easily discarded checking the roll and the pitch. Moreover, since more matches could correspond to the same object, they can be easily discarded considering the best match in a neighbourhood of the position of each match.

Therefore, the presented algorithm can robustly localize planar objects. But some parameters, such as the translation and rotations step and their limits, have to be tuned accordingly to the robot arm, the searched object and the wanted robustness.

This algorithm is not faster than the algorithm presented by Pretto et al. [1], but anyway it's a good starting point for the implementation of the new improvements presented by Shotton et al. [2] and Liu et al. [9]; improvements that are briefly commented in the chapter 5, which make the algorithm faster and more robust.

Thanksgiving

Desidero ringraziare il prof. Pagello per aver accettato di farmi da relatore, e in particolar modo desidero ringraziare il prof. Alberto Pretto che mi ha proposto la tesi, mi ha aiutato nella implementazione del codice per tutto il corso del lavoro, risolvendomi dubbi concettuali e introducendomi nel mondo della computer vision.

Desidero ringraziare i miei genitori che mi hanno seguito nel percorso dei tre anni aiutandomi e sostenendomi sempre con fermezza e permettendomi di fare esperienze di cui faró sempre tesoro.

I miei fratelli Gloria e Daniele, i miei nonni, mio zio Onorino, e tutti i miei amici che mi hanno sempre accompagnato nel mio percorso di studi fino ad oggi.

Quiero también agradecer a todos mis amigos de Ferrol que he conocido en mi instancia Erasmus, con los cuales viví uno de los meses mas bonitos de mi vida. Y también a mis companeros de piso en Barcellona que en las ultimas dos semanas tuvieron que aguantar a mis preocupaciones y ayudarme con mi inglés.

Bibliography

[1] A. Pretto, S. Tonello, and E. Menegatti, “Flexible 3d localization of planar objects for industrial bin-picking with monocamera vision system,” in *Proc. of: IEEE International Conference on Automation Science and Engineering (CASE)*, 2013, pp. 168 – 175.

[2] J. Shotton, A. Blake, and R. Cipolla, “Robust detection of lines using the progressive probabilistic hough transform,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, 2008.

[3] P. E. Debevec and J. Malik, “Recovering high dynamic range radiance maps from photographs, university of california at berkeley,” in *The 24th International Conference on Computer Graphics and Interactive Techniques*, 1997, pp. 369–378.

[4] Gamma correction, http://en.wikipedia.org/wiki/Gamma_correction, available on 5/09/14.

[5] R. G. von Gioi , J. Jakubowicz, J. M. Morel, and G.Randall, “Lsd: a line segment detector,” *Image Processing On Line*, no. 2, pp. 35–55, 2012.

[6] ELSD (Ellipse and Line Segment Detector), http://ubee.enseeiht.fr/vision/ELSD/, available on 5/09/14.

[7] Camera Calibration Toolbox for Matlab, http://www.vision.caltech.edu/bouguetj/calib_doc, available on 24/08/14.

[8] D. H. Ballard, “Generalizing the hough transform to detect arbitrary shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[9] M.-Y. Liu, O. Tuzel, A. Veeraraghavan, Y. Taguchi, T. K. Marks, and R. Chellappa, “Fast object localization and pose estimation in heavy clutter for robotic bin picking,” *I. J. Robotic Res.*, pp. 951–973, 2012.

[10] OpenCV, http://opencv.org/, available on 31/08/14.

[11] F. Durand and J. Dorsey, “Fast bilateral filtering for the display of high-dynamic-range images,” 2002.

[12] R. G. von Gioi , J. Jakubowicz, J.M. Morel, and G. Randall, “Lsd: A fast line segment detector with a false detection control,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 32, no. 4, 2010.

[13] T. Mertens, J. Kautz and F. Van Reeth, “Exposure fusion,” *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, pp. 382–390, 2007.

[14] P.J. Burt and E. H. Adelson, “The laplacian pyramid as a compact image code,” *IEEE TRANSACTIONS ON COMMUNICATIONS*, vol. 78, no. 14, 1983.

[15] OpenMP, http://openmp.org/wp/, available on 24/08/14.

[16] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[17] A. Etemadi, “Extracting straight lines,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, 1996.

[18] J. B. Burns, A. R. Hanson, and E. M. Riseman, "Robust segmentation of edge data," *Proc. Int'l Conf. Image Processing and Its Applications*, pp. 311–314, 1992.

[19] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 311–314, 2000.

[20] A. Desolneux, L. Moisan, and J.M. Morel, *From Gestalt Theory to Image Analysis, A Probabilistic Approach.* Springer, 2008.

[21] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H.C. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," *SRI International, Menlo Park, California*, 1977.

[22] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1330–1334, 2000.

[23] Ceres-solver, http://ceres-solver.org/, available on 31/08/14.

[24] P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions," *Technical Report TR2004-1963, Cornell Computing and Information Science.*

[25] Sobel Edge Detector, http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm, available on 24/08/14.

Appendix A

Image Gradient

In mathematics, the gradient is a generalization of the usual concept of derivative to the functions of several variables. If $f(x_1, \dots, x_n)$ is a differentiable, scalar-valued function of several variables, its gradient is the vector whose components are the n partial derivatives of f. It is thus a vector-valued function.

The gradient represents the slope of the tangent of the graph of the function. More precisely, the gradient points in the direction of the greatest rate of increase of the function and its magnitude is the slope of the graph in that direction. The components of the gradient in coordinates are the coefficients of the variables in the equation of the tangent space to the graph.

Speaking about 2D images, the function of interest is the discrete function of the intensity of the 2D image $I(x, y)$ that could take different values on different pixels.

The gradient of an image measures how the image intensity is changing, and has two types of information:

- the gradient magnitude: measures how quickly the intensity is changing
- the gradient direction: measures the direction of this change

To illustrate this, think of an image as like a terrain, in which at each point given a height, rather than an intensity. For any point in the terrain, the direction of the gradient would be the direction uphill. The magnitude of the gradient would means how rapidly hill height increases.

To figure out how is possible to calculate the gradient consider a 1D image, when there is a change of intensity between two pixels, one pixel has another value from the pixel before considered, for example for the presence of a edge, the case is, on a quality level, like the figure Fig. A.1.

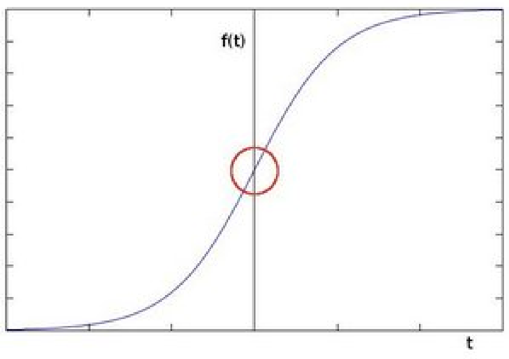


Fig. A.1: Example of pixels intensity on a 1D image. (Image taken from [10])

On the edge there is a changing of the intensity, and correspond to the point where the sloping of the function reaches his maximum value, that is the point where the derivative of image intensity has a maximum, as showed in the figure Fig. A.2.

Therefore for 2D images the gradient could be calculated from partial derivatives of the intensity of pixel along x and y directions. The gradient could be represented by a vector, his length provides the magnitude while his direction provides gradient direction. Since the gradient measures how fast the pixels intensity of an image changes along x and

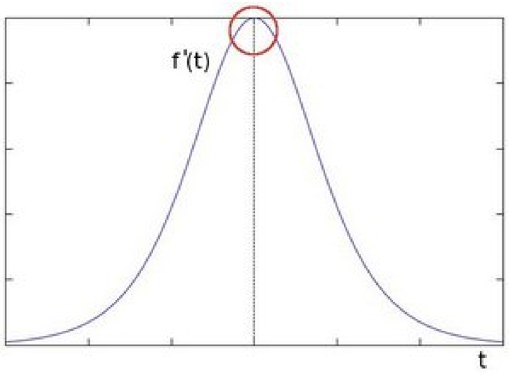


Fig. A.2: Derivate of the intensity function of an 1D image. (Image taken from [10])

y coordinates, and since where are working on 2D images, it can be represented using the derivatives of the pixels values along x and y axis:

$$\nabla I = (\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y}) \tag{A.0.1}$$

The partial derivative $\frac{\delta I}{\delta x}$ along x directions determines how fast the intensity is changing along x direction. In the discrete case, we can only take differences at one pixel intervals. So the partial derivative could be calculated with the difference between $I(x, y)$ and the pixel before it, or the one after it, but there are more efficient methods to calculate partial derivatives in discrete cases, like Sobel derivatives [25], that considers the influence of all pixel nearby the considered one. The Sobel operator performs a 2D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input gray-scale image. The operator consists of a pair of 33 convolution kernels, one for each direction, note that the kernel for the y direction is the same for the x direction but rotated by 90, the convolutions kernels are here reported:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation. Then these can be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Typically, an approximate magnitude is computed using:

$$|G| = |G_x| + |G_y|$$

That is much faster to compute and the approximation is not bad.

The gradient orientation (relative to the pixel grid) is given by:

$$\theta = \arctan(\frac{G_y}{G_x})$$

Sobel operator involves a grater computational effort than others operators, but has the great benefit that its larger convolution kernel smooths the input image to a greater extent and so makes the operator less sensitive to noise.

This gradient has a fundamental importance on edge detection because, since the most human-made objects have flat surfaces and on these surfaces pixels intensity does not change higly.

Image a mono-coloured object for manufacturing, like the one in Fig. A.3, pixels corresponding to edges have a high gradient magnitude and the gradient orientation has an orthogonal orientation in respect to edge, these data can be used to represent a line.

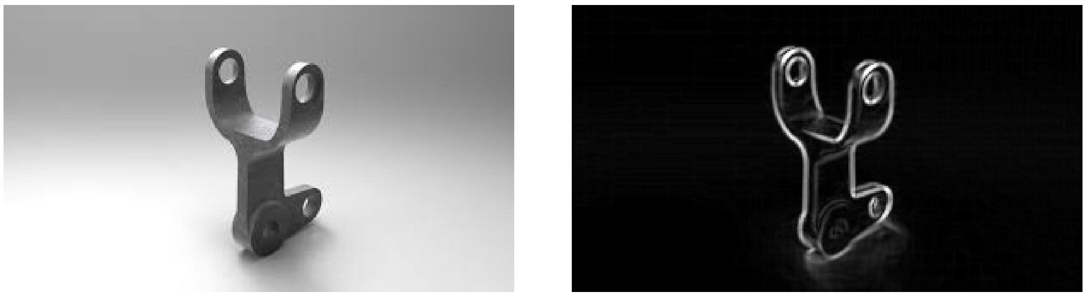


Fig. A.3: Image representing a generic component on the left and the result of Sobel operator on that image on the rigth.

Gradient magnitude, but overall, gradient orientation are fundamental parameters for line segment detection as explained in chapter 3.

The presented Sobel operator has only one of the several gradients operators, but one of the most common, another operator really common that work in a way almost alike Sobel operator but with more complicated kernels is the Scharr operator.