Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Project

Automatic recognition of texture-less objects from a single image

Václav Tyle

Supervisor: Prof. Ing. Jiří Matas, Ph.D.

Study programme: Cybernetics and robotics

Study branch: Systems and control

May 24, 2013

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Václav Tyle**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **Automatic recognition of texture-less objects from a single image**

Guidelines:

1. Get yourself familiar with the method for automatic recognition of texture-less objects presented in [1] and re-implement it.
2. Evaluate the method on a range of experimental data, discuss possible improvements and implement them.
3. Use the method for real-time detection of objects from a video input.

Bibliography/Sources:

[1] D. Damen et al.: Real-time Learning and Detection of 3D Texture-less Objects: A Scalable Approach. London, 2012.
[2] D. A. Forsyth, J. Ponce: Computer Vision: A Modern Approach. Prentice Hall, 2011.

Bachelor Project Supervisor: Prof.Ing. Jiří Matas, Ph.D.

Valid until the winter semester 2013/2014

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, December 20, 2012

Aknowledgements

I take this opportunity to thank my supervisor Prof. Jiří Matas for his guidance, help and many invaluable advices concerning this project.

I am really thankful also to my family and friends for support and patience.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 24.5.2013

Podpis

Abstract

In the scope of this bachelor thesis, a program PENTAGRAM_VT capable of identifying known objects in an image or in a video was created. It is based on the method published recently by D. Damen et al. at British Machine Vision Conference. The detection is almost real time and the learning may be performed on a very small set of training images.

The observations gained during implementation may be used to further improve the method. Further, a few originally not emphasized or overlooked weak points, worth of attention, have been discovered.

Abstrakt

V rámci této bakalářské práce byl vytvořen program, PENTAGRAM_VT, schopný identifikovat známé objekty z obrázku či videa. Založený je na metodě, nedávno publikované D. Damen a kol. na British Machine Vision Conference. Detekce probíhá téměř v reálném čase a k naučení postačí velmi malá množina trénovacích obrázků.

Poznatky nabyté během implementace lze zužitkovat k dalšímu vylepšení metody. Dále byly objeveny některé původně nezdůrazněné či přehlédnuté slabiny, hodné další pozornosti.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is focused on implementing a program able to learn, to localize and to recognize an object from an image or from a video, i. e. to identify its borders in a scene and to assign it a correct name. To succeed, the program must have became familiar with the object before already, which is done by sanding it other images or a video, in both cases labelled with the object name. Since we will concern with three-dimensional objects, which may look absolutely different when viewed from different angles, it is usually not sufficient to learn an object based on a single image only.

We have decided to use a single camera as the input, since it is cheap, easy to manipulate with and though potentially very powerful for classification since it provides a similar information as our vision (when using one eye only).

From the possible methods we have been choosing among the ones discriminating objects based on their shape or more precisely edges. By an edge we will mean continuous regions where the object has its borders or where the object surface changes its orientation in space. In an image, these regions imply a fast change of color or brightness.

Further, we will avoid images with a significant texture present, that is such, where the change corresponds much more often to the edges of surface symbols and patterns than to the edges of objects.

Images will be considered as two dimensional arrays of pixels only. That means we won't use any information about the depth as the methods based on stereopsis or on shading do it, since in both cases deriving the information about depth is a matter of some considerable computation, which makes it time consuming and difficult to implement. Also the realisation is more complicated and expensive, see [10] and [9] for more details.

1.1 Motivation

Shape is undoubtedly the most important distinguishing factor for a vast amount of miscellaneous objects. The ability not only to recognise but also to localize the object and possibly determine its orientation may be used on assembly lines let's say to target and screw a nut or to turn and correctly join two parts to be welded. We can imagine applications in food industry, automatic waste sorting and more.

If we localise the area of an object it also allows us to segment and further analyse it. This may be used for example in supermarket to estimate the ripeness and quality of fruits placed on a weighing machine based on their color, after we have determined its sort based on the shape, so that we can state its price for a customer. This interesting idea comes from [15].

An other example is an intelligent searching engine in digital libraries, so called content-based image retrieval (CBIR) [8].

While there are quite a plenty of various public available and powerful texture-based recognition applications, like face recognition or most of the already mentioned CBIR engines (see the list at [19]), the shape-based recognition seems to be an opened topic.

1.2 Selecting the method

Three basic approaches may be tracked down among the methods for texture-less recognition. The first approach is based on dividing an image into small cells and computing so called histograms of oriented gradients for each of them. The gradients stand for brightness gradients and the histograms for their distributions in individual cells. These are then used to compare the cells of two different images to detect the presence of same objects. This idea was first introduced in [4]. The key disadvantage is computational complexity, although significantly reduced by some of the derived methods, for example [12].

A few methods have been also presented that try to create a three dimensional models of objects. The comparison is then made by comparing the models. This is useful in robot navigation where the information about depth or distance is crucial. An example of such method is [13].

Another group of methods are based on an old technique, so called chamfer matching. It was introduced in [1] in 1977. Here, we first build edge maps of the two images and then compare them by computing the distances between pairs of closest edge segments. The average distance is than used as a similarity metric. With an advantage, distance transform [11] may be used for fast computation. This technique has been further improved in [16] by taking the orientation of the edges into consideration.

A key disadvantage of this approach is again the computation complexity making it useless if we want to compare an image with hundreds of templates. To solve this, some of the recent works use a two stage classification. The first stage reduces the amount of templates to be compared to just a few potential candidates and the second stage uses the oriented chamfer matching with various modifications and improvements to choose among the candidates.

As an example, we will mention [2]. Here, the first stage is implemented as follows: A thirty six uniformly distributed points forming a filled rectangle are "put" on a test image and a distance to the closest edge point is measured for each of them. The resulting values are descriptors which may be compared with the descriptors of the training views. If a match is found, it's taken for a candidate detection. To make the method robust to geometric transformations, descriptor measure is repeated many times for a single image while the orientation, scale and position of the rectangle changes (sliding window technique). The method is almost real-time and reaches state-of-the-art success rate.

An other example of a two stage method with oriented chamfer matching is the one published by D. Damen et al. in [5]. Although contemporary outperformed by [2], this is also the one, we have chosen for our implementation. It is almost real-time but shows somewhat poorer success rate than [2]. In addition to other state-of-the-art methods, it introduces scale, rotation and translation invariant descriptors and uses a geometric transformation of edges to partially preserve the invariances even to the chamfer-matching-based verification stage. We also believe the method has some potential to be yet uncovered. This is why we have decided to make it a subject of our deeper study.

It is convenient to assign our implementation a name and since the authors of [5] don't use any in their report, we had to choose one on our own. For a reason explained later we decided for Pentagram_vt.

The C++ language has been chosen for the implementation as a compromise between speed and easy readability of the code. The openCV [18] library is strongly used throughout the program.

Chapter 2

The method

2.1 Overlook

This section gives a brief overlook of the individual steps which the algorithm carries out in the learning and in the test phase.

Learning

The algorithm becomes a set of labelled images of the objects to be learned. There shouldn't be multiple objects in a single image but there may be more images of a single object. Each of them undergoes the following procedure:

1. **Edge detection.** An image is handed to the edge detector. A binary image of white edges on a black background is the output.

2. **Polygonal approximation.** The edges are approximated with a polygon or a polygonal chain. Lengths of it's segments are multipliers of a fixed minimal length.

3. **Representation with edgelets.** The polygonal chain is further described with *edgelets*. An edgelet is an oriented line segment of the same fixed length. If a segment was longer than this minimal length after the polygonal approximation, it constitutes now of a few equally oriented edgelets. All the edgelets of an image build a *view* or an *edgelet representation*.

4. **Description.** *Constellations* of edgelets are traced out following a particular rule described later. For every constellation a *descriptor* is created. All descriptors together build a *descriptor representation* of the image being processed.

5. **Storage.** The view gets inserted to the *library*. The descriptors of the descriptor representation get quantised, hashed and used as keys to the respective view in the library.

Classification

The input to the classification is a set of test images. They are further processed in this way:

1. **Search for descriptors.** One after the other, descriptors are being traced out in an incoming test image following the same algorithm as in the learning phase.

2. **Search for correspondences.** As soon as a descriptor has been created, PENTAGRAM_VT tries to find a corresponding one among the known objects.

3. **Overlapping.** If any such is found, the edgelets of the respective training image are mapped "onto" the of the test image, so that they lied as close to each other as possible. If the mapped training edgelets satisfyingly "overlap" the test ones, the classification is finished. If not, the procedure is repeated for a next descriptor.

The rest of the chapter gives a closer look to the individual steps.

2.2 Edge detection

Every Image, PENTAGRAM_VT works with, is first pre-processed by the program APPROX written in 1994 and now kindly provided by Prof. Matas. Originally, it was designed to use files in PGM format for input and output, which was not very convenient for our purpose. Therefore, I have modified it to communicate directly with PENTAGRAM_VT.

The role of APPROX is to detect edges and to *approximate* them with line segments. The first step is done based on the Deriche's modification[6] of Canny Edge Detector [3]. These algorithms both operate in four steps. The key difference is in the first one:

1. **Blur.** The image is convolved with a blurring operator. This is to reduce the sensitivity to noise. In the case of Canny, the operator is a Gaussian filter while Deriche uses an even more effective one - one satisfying the criteria, Canny has stated for an optimal edge detector.

2. **Calculation of gradients.** Every pixel of the blurred image is assigned a gradient. There are again operators, like the Sobel, of which application makes the work for us. The gradient orientation is rounded so that it points to one of the eight neighbouring pixels.

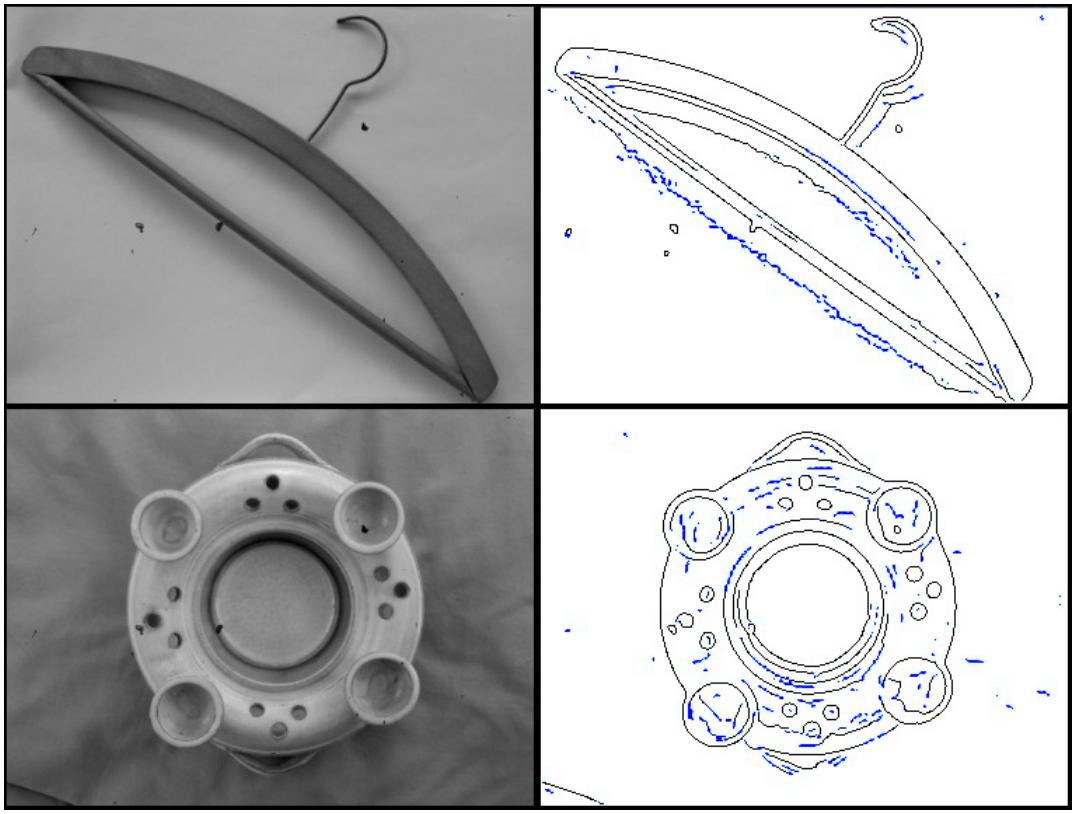


Figure 2.1: A possible result of Deriche's edge detection with the parameters set to discard highly curved places (marked blue).

3. **Non-maximum suppression.** Only the pixels with gradient magnitude being a local maximum are further considered.

4. **Search for continuous curves.** The pixels are double thresholded according to their gradient magnitude. From the pixels satisfying the stricter threshold the program starts to search for solid curves including also the pixels with vaguer threshold into this search. The orientation of a curve in a given point can be estimated since it is perpendicular to the respective gradient. We may therefore reduce the time complexity by searching only in the estimated directions. Only the curves long *enough* are preserved. The demanded length is an adjustable parameter.

A weakness of Pentagram_vt is a sensitivity on highly "frayed" lines. These lines are therefore discarded using the mentioned parameter. As a positive result we usually get rid of shadows in the picture. Nevertheless this issue should be subjected to a deeper study in the future. A shape smoothing may be a possible solution, for example.

2.3 Polygonal approximation

The output of the edge detector is a set of curves; each constituted by a finite nmber of its points, i. e. pixels (because image is a discrete space). In the next step APPROX proceeds to approximate these with polygonal chains following the method described by U. Ramer in [14]: A polygonal chain $\mathcal{P}$ is an ordered set of line segments. To approximate a curve $\mathcal{C}$ let $\mathcal{P} = \emptyset$ and start with the line segment $\overline{P_1P_2}$, where $P_1, P_2 \in \mathcal{C}$ are the boundary points of the curve. Look for an $P_3 \in \mathcal{C}$ whose distance from $\overline{P_1P_2}$ is maximal. If that distance is less than a particular value ϵ, add $\overline{P_1P_2}$ to $\mathcal{P}$ and stop. If not, repeat the algorithm recursively for $\overline{P_1P_3}$ and $\overline{P_3P_2}$.

As soon as the approximation has been finished, the output is handed to PENTAGRAM_VT in the form of a set of line segments of a fixed minimal length. These are further described with edgelets, in a manner obvious from the section 2.1, giving a *view* of the image.

At this moment, if we wanted to compare two objects, we could take their views and proceed like this: For every edgelet e_i in the view $\mathcal{E}_1$ find the closest edgelet e_j from the view $\mathcal{E}_2$. Measure the avarage distance $d(e_i, e_j)$ over all those respective pairs using a particular metric d. If the avarage is under a certain threshold, classify $\mathcal{E}_1$ as being $\mathcal{E}_2$.

This is what we will actually do in the end; however this approach standing alone wouldn't be very effective since it is not rotation-, scale- and even shift-invariant. We will therefore first invent a description of the object that will allow us to perform a candidate classification and to estimate a mapping of $\mathcal{E}_1$ (training image) "onto" $\mathcal{E}_2$ (test image) and to compare the object as described above to verify or refuse the classification.

2.4 Description

Basic terms

Let us start with a few definitions. We have already explained the term *edgelet*. Mathematically, it can be treated as an ordered triple

$$e_i = [\mathbf{m}, \mathbf{v}, l] \tag{2.1}$$

where $\mathbf{m}$-middle, $\mathbf{v}$-direction vector, l-fixed length (same for all edgelets in an image).

Under *orientation of an edgelet* $o(e_i)$ we will understand the smaller of the two base angles it forms with the horizontal axis of a picture. A *relative orientation of two edgelets*, denoted as $\angle e_ie_j$, is the angle $\angle e_ie_j = |o(e_i) - o(e_j)|$. A *distance of two edgelets* $d(e_i, e_j)$ is

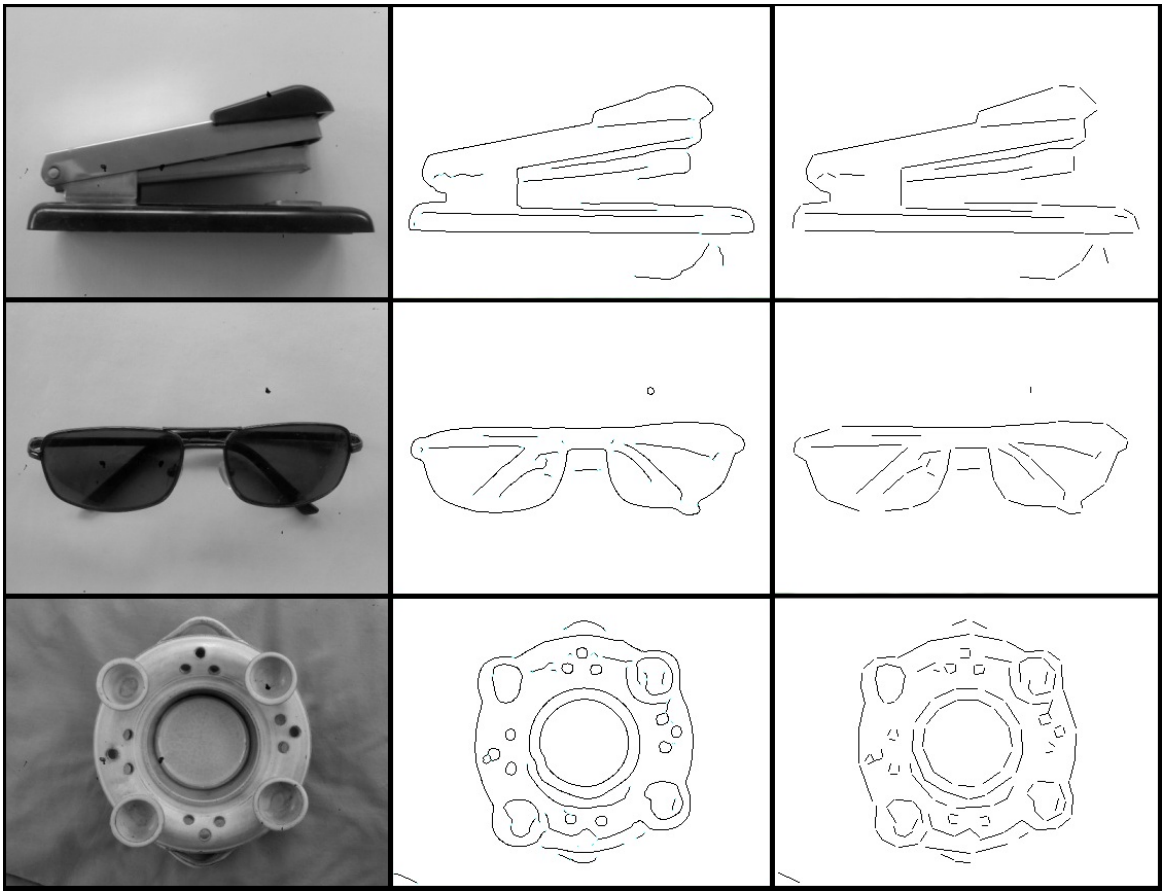


Figure 2.2: Visualisation of the polygonal approximation. On the left the original image, in the middle the detected edges and on the right the resulting edgelets.

the euclidean distance of their middles. Notice that:

$$o([\mathbf{m}, \mathbf{v}, l]) = o([\mathbf{m}, -\mathbf{v}, l]) \in \langle 0, \pi) \tag{2.2a}$$

$$\angle e_i e_j = \angle \in \langle 0, \frac{\pi}{2} \rangle \tag{2.2b}$$

This follows the fact that, when we imagine, for example, a horizontal line in an image, we cannot distinguish if its oriented from the south to the west or reversely.

A *constellation* c may be viewed as an ordered n-tuple of edgelets

$$c = (e_1, e_2, \ldots, e_n) \tag{2.3}$$

A *path* Θ is an ordered tuple of $n-1$ angles:

$$\theta_i : \Theta = (\theta_1, \theta_2, \ldots, \theta_{n-1}) \tag{2.4}$$

The Algorithm

We have already described how to get an edgelet representation or a view. The next step is to build descriptor representation. To achieve this, every edgelet undergoes the following procedure (for the code see the section 4.1)

1. A number i is set to one.
2. The edgelet is considered to be the first edgelet e_1 of a potential constellation c.
3. An imaginary straight line l_{12} intersecting the middle of the edgelet is created, so that the edgelet forms the angle θ_1 with the line l_{12}, i. e. $\angle e_1 l_{12} = \theta_1$.
4. The number i is increased by one.
5. The algorithm looks for all edgelets e_{2j}, having middles on the line l_{12} or very close to it (the width of the tolerance band equals the fixed edgelet length). Let's assume there were N such edgelets, i. e. $j \in N$.
6. For each of them a new potential constellation c_j is created, so that it contains both e_1 and e_{2j} at the first and at the second position respectively.
7. For every e_{2j} a straight half-line l_{23j} is created, so that it starts in the middle of e_{2j} and forms the angle θ_2 with the previous line l_{12}, this time NOT with the edgelet e_{2j}: $\angle l_{12} l_{23j} = \theta_2$.

8. The steps 4-7 are repeated except half-lines are used instead of lines and new edgelets are added at the i-th position of potential constellation leaving the already present ones unchanged.

9. The process stops as soon as i equals the number n of edgelets in constellations.

10. For every constellation $c = (e_1, e_2, \ldots, e_n)$ a descriptor $D(c)$ is created.

The descriptors have the following form:

$$\mathbf{D}(c) = (\phi_2, \ldots, \phi_n, \delta_3, \ldots, \phi_n) \tag{2.5a}$$

$$\phi_i = \angle e_{i-1} e_i, \tag{2.5b}$$

$$\delta_i = \frac{d(e_{i-1}, e_i)}{d(e_{i-2}, e_{i-1})} \tag{2.5c}$$

Such descriptors are rotation, scale and shift invariant.

The angular measures "$\angle$" in the algorithm above have nothing in common with the relative edgelet orientation, despite they are denoted the same. It is obvious, how these are measured, from the picture 2.3.

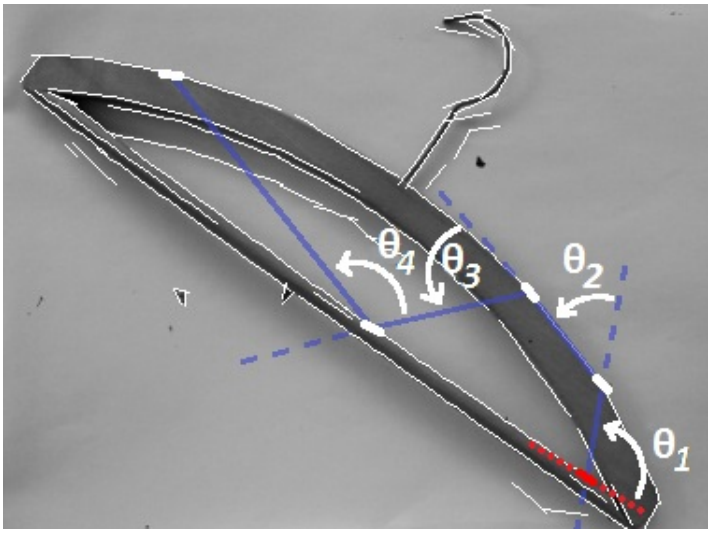
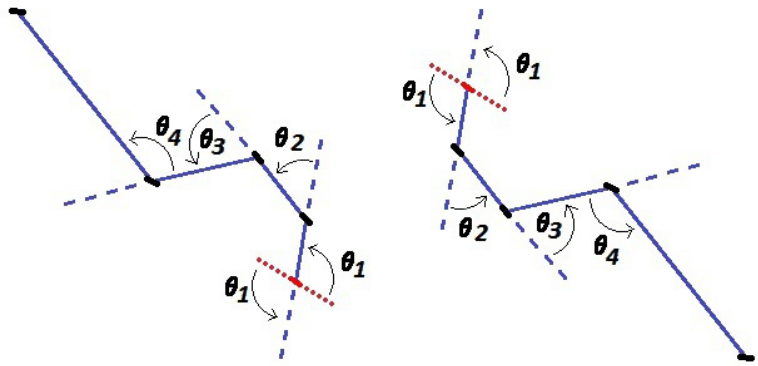


Figure 2.3: The image illustrates, how to trace out a constellation, following a path given by the angles θ_i. The starting edgelet of the constellation is marked red, the others white. The edgelets, not being a part of the constellation, are marked thin white. Notice that θ_1 is formed by the first edgelet and the first line while the others θ by two lines. Here, we will mention, why the program has been given the name PENTAGRAM_VT. It is since the polygonal chain, in our experiments created by five segments, associates us this shape drawn similarly with five straight strokes.

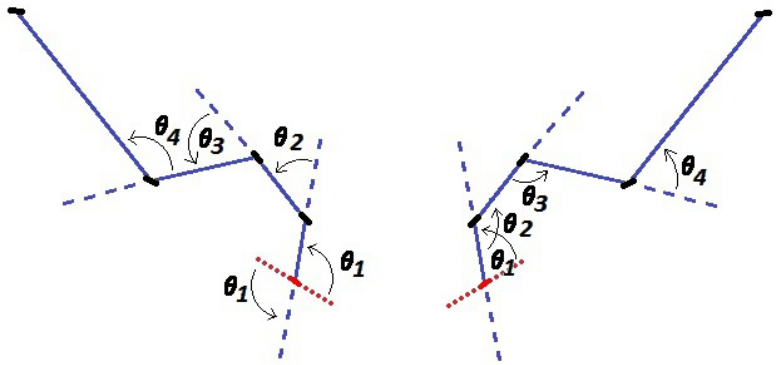
Rotational invariance

After explaining the process of tracing out the constellations, it should be more obvious, why we have emphasized (2.2a) and (2.2b). If we on the contrary defined $o(e_i) \in \langle 0, 2\pi)$ or $\angle e_i e_j \in \langle 0, 2\pi)$, the method wouldn't remain rotation invariant.

Two other issues had to be thought out carefully to preserve this property: First, in the algorithm above l_{12} is really a *line*, while the others are *half-lines*. It means that at the beginning we search for tolerated edgelets in both directions while further just in one direction. Second, the angles θ are measured either always counter-clockwise or always clockwise. See the figure 2.4.



(a) There is a constellation traced out using the path $\Theta = (\theta_1, \ldots, \theta_4)$ on the left. The constellation on the right was derived from the other by rotation. Though, it can be traced out using exactly the same path.



(b) The thetas on the right don't equal the thetas on the left. I. e. the mirrored constellation wouldn't be traced out using the same path.

Figure 2.4: Illustration of the reflection variance and rotational invariance of the descriptors. The solid blue lines are lengthened with dashed lines in the "tolerated" directions, i. e. to both sides, when starting from the first (red) edgelet of the constellation and to single side, when starting from any other edgelet.

2.5 Storage

Introduction

The candidate classification is based on creating the descriptor representation of a test image and comparing it with the known representations of training images. As obvious from (2.5a), a descriptor consist of $2n-3$ numbers, where $n \geq 2$ stays for the number of edgelets in constellation. If we represent these with the standard 4 Byte float data type, we become 2^{64n-96} possible values a descriptor can have. Therefore, when we trace out a constellation in a test image and a respective one in a training image, the arisen descriptors won't most probably be precisely the same (provided the images are not precisely the same). Even if we slightly reduced the precision of the floats, so that it wasn't beyond absurdity, the problem would remain.

We can of course imagine the descriptors as points in a $(2n-3)$-dimensional feature space. Than, the classification could be carried out as a search for the training descriptor representation nearest to the test descriptor representation. However, the time complexity is unbearable.

We will therefore need to organise the data to a structure, a *library*, that will make the classification process fast and effective. For this purpose, D. Damen et. al. use a "quantised hierarchical hash-table." Nevertheless, the term isn't exhaustingly described here and our solution may therefore differ. It is based on quantising the feature space and hashing of the quantised descriptors.

Implementation

The equations (2.2b) and (2.5b) imply, the relative angles ϕ_i in the descriptors are from the interval $\langle 0, \frac{1}{2}\pi \rangle$. We will divide this interval to N_ϕ subintervals, each marked with a number from 1 to N_ϕ. Further we replace every ϕ in the descriptor with the number of the subinterval it belongs to. In other words: we have chosen N_ϕ uniformly distributed numbers from the original interval and rounded the thetas to them.

Similarly, if we decide to add an edgelet e_{i+1} to a potential constellation, only if it satisfies the condition

$$d(e_{i+1}, e_i) \geq d_{min}, \tag{2.6}$$

where e_i stands for the previous edgelet in the constellation, d is the metric stated in the section 2.4 and d_{min} is a particular real parameter, we can say that

$$\delta \in \left\langle \frac{d_{min}}{diag}, 1 \right\rangle \bigcup \left\langle 1, \frac{diag}{d_{min}} \right\rangle \tag{2.7}$$

Here δ is the relative distance from the descriptor and *diag* is the diagonal of the image.

Both of the intervals are then divided into $\frac{N_\delta}{2}$ subintervals, again each subinterval with a number from 1 to N_δ. Every δ is then substituted with one of those numbers in the same manner as we did it with thetas. It is yet to emphasize that, in praxis, the number *diag* is actually not derived from the diagonal of an image. Instead, if a δ occurs, such that $\delta > \delta_{max}$, it is saturated to $\delta = \delta_{max}$. Here δ_{max} is a particular parameter to be tuned.

We will refer to the process described in the previous two paragraphs as mapping $h(\mathbf{D})$:

$$\begin{aligned} &h(\mathbf{D}) = h\big(\,(\phi_2, \ldots, \phi_n, \delta_3, \ldots, \phi_n)\,\big), \\ &h : (\phi_2, \ldots, \phi_n, \delta_3, \ldots, \delta_n) \longrightarrow (\alpha_0, \ldots, \alpha_{n-3}, \beta_0, \ldots, \beta_{n-4}), \end{aligned} \tag{2.8}$$

where $\alpha_i \in \{1, \ldots, N_\phi\}, \beta_i \in \{1, \ldots, N_\delta\}$. The rest of the notations have the same meaning as in (2.5).

The condition (2.6) means, we have decided not to describe very local features of the object being classified. But this has been actually introduced already by the fact, that we use edgelets of a fixed (non-zero) length l_e. Trying to trace out constellations or their parts close to the scale given by l_e would be strongly affected by inaccuracy of approximation and the resulting descriptors wouldn't be characteristic or descriptive for given objects.

We have implemented the hash-table in the form of the UNORDERED_MAP container in C++, where descriptors D are used as keys. The hash value is assigned to them by the composed function $H(D)$:

$$H(D) = f(h(D)), \tag{2.9a}$$

$$f : (\alpha_0, \ldots, \alpha_{n-3}, \beta_0, \ldots, \beta_{n-4}) \longrightarrow \sum_{i=0}^{N-3} \alpha_i N^i + \sum_{i=0}^{N-4} \beta_i N^i \tag{2.9b}$$

As already described, the function h ensures that similar descriptors, i. e. such that their respective deltas and thetas lie in the same subintervals, will be assigned the same hash code. The assignment itself does the function f, which was designed to provide minimal perfect hashing, i. e. it reduces the number of bits needed to store its output while still assigning an unique hash code $f\big(h(D)\big)$ to every possible $h(D)$.

The value, respective to a key (descriptor), is first a constellation, which the descriptor was derived from, and second a pointer to the view, the constellation comes from.

Summary

In the memory, an object is represented with its name and with the views arisen from the training dataset. Further, each object is represented by its entries in the hash-table:

if we use a descriptor, which has been traced out from a particular view, as a key to the hash-table, we become the constellation, from which the descriptor comes, a pointer to *that* view and, of course, also to the name of the object, the view belongs to. We will strongly use this in the classification phase.

2.6 The Classification

Candidate detection

When an image comes to be classified, we get an edgelet representation exactly the same way as in the learning phase. Further the program searches for the constellations and descriptors following the same path $\Theta = (\theta_1, \ldots, \theta_{n-1})$ and the same algorithm. As soon as a constellation c_t is found and the respective descriptor $D(c_t)$ calculated, $D(c_t)$ is used as a key to the hash-table. In return we receive pointers to all the training views (if there were any such) which contained a descriptor $D(c_v)$ satisfying

$$H(D(c_t)) = H(D(c_v)) \tag{2.10}$$

These views are our candidate detections. We further receive the constellations c_v, which those training descriptors came from.

Estimating transformation

To confirm or to refuse a detection, we will estimate a transform $\mathbf{T}$ from the candidate view $\mathcal{E}_v$ to the test view $\mathcal{E}_t$, based on the comparison between constellations c_t, c_v. To perform this we use the following metric:

$$\Delta(e_i, e_j) = d(e_i, e_j) + \lambda\angle(e_i, e_j), \tag{2.11}$$

where d is the euclidean distance between the middles of the edgelets, $\angle$ is the relative orientation of edgelets as defined in 2.2b, λ is a weighting term, e_i, e_j are edgelets. Obviously, this metric stands for the distance between two edgelets, where their orientation is considered. The mapping $\mathbf{T}$ will have the form:

$$\mathbf{T} : e_v \longrightarrow e'_v, \tag{2.12}$$

where e_v, e'_v are edgelets, i. e.:

$$\mathbf{T} : [\mathbf{m}, \mathbf{v}, l] \longrightarrow [T(\mathbf{m}), T(\mathbf{v}), l], \tag{2.13}$$

where we used the notation from 2.1. T is a composed rotation, translation and scaling. We will further demand the T to "roughly" minimize the term

$$\sum_{i=1}^{n} \Delta(\mathbf{T}(e_{v\_i}), e_{t\_i}) \tag{2.14}$$

where e_{v_i}, e_{t_i} are edgelets of the constellations c_v, c_t and n is their size. Less formally, we minimize the "distance" between the two constellations.

In 2.12 we have defined the transform for a single edgelet only. For the sake of simplicity, we will define it for an edgelet representation (a view) $\mathcal{E}$ also:

$$\mathbf{T}(\mathcal{E}) = \{\mathbf{T}(e) | e \in \mathcal{E}\} \tag{2.15}$$

To summarize the previous paragraph: we look for a $\mathbf{T}$ that will map the constellation c_v of the candidate onto the constellation c_t of the test image. Since a constellation consist of just a few edgelets, the transform can be quite easily found. The rotation can be calculated from the orientation of respective edgelets, the scaling and the translation consequently from their absolute position.

After the transformation $\mathbf{T}$ has been calculated, it is applied on the candidate view $\mathcal{E}_v$. See the figure 2.5 for possible results.

Refining the transformation

However, the mapping $\mathbf{T}$ alone is usually not very precise as it uses only the information from a single constellation. The method [21] describes, how to derive a transformation $\mathbf{T}_r$, able to refine the result $\mathbf{T}(\mathcal{E}_v)$.

The author originally invented it to estimate the shortest trajectory between two subsequent video camera frames. So, if we place the video camera on a vehicle, we may estimate it's actual position from the record, if the starting position is known. But it will serve well for our purpose also.

The first step is to find the closest edgelet $e_t \in \mathcal{E}_t$ (the test view) for every edgelet $e_{Tv} \in \mathbf{T}(\mathcal{E}_v)$ and to build a particular set $\mathcal{P}$ of those pairs (e_t, e_{Tv}). We measure the distance according to 2.11. In [21], it is assumed that the motion between two subsequent frames is small. This corresponds with the fact, that the transform $\mathbf{T}$ has already been applied, in our case. The assumption also means that the two edgelets in the pairs usually refer almost to the same point in the real world. We may therefore proclaim that such pairs, for which $\Delta(e_{Tv}, e_t) > \Delta_{max}$, where Δ_{max} is an adaptively set number, are "false positives" and discard them from the set $\mathcal{P}$.

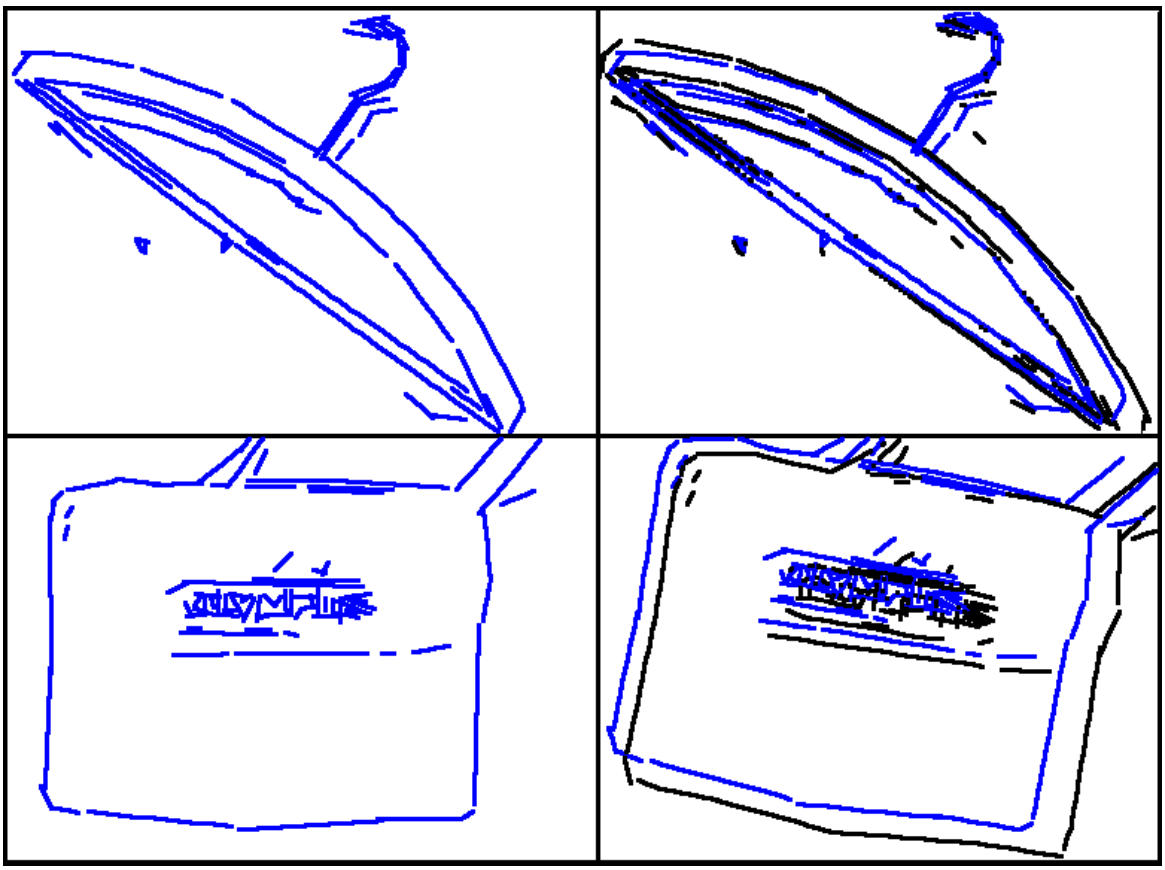


Figure 2.5: Possible results of the mapping under **T**. On the left the original view, on the right the view after mapping (blue) and the edgelets of the test image being classified (black). Obviously, the mapping is not perfect and the refinement using [21] is necessary.

The Δ_{max} is chosen to reduce the size of the set $\mathcal{P}$ to a number $N = |\mathcal{P}|$, reasonably small for further computations. For details see [21] again. The last step is to find a rotation matrix $\mathbf{R}$ and a translation vector $\mathbf{t}$ of a compound transformation $\mathbf{T}'_r(\mathbf{R}, \mathbf{t}, \mathcal{P})$ to minimize the term

$$\sum_{\mathcal{P}} \Delta^2(\mathbf{R}e_{Tv\_i} + \mathbf{t}, e_{t\_i}) \tag{2.16}$$

where $(e_{Tv\_i}, e_{t\_i}) \in \mathcal{P}$. The sum stands for a sum over the set $\mathcal{P}$. By applying the $\mathbf{R}, \mathbf{t}$ on the edgelets e_{Tv_i}, e_{t_i} we mean applying it separately on their middles and direction vectors. Minimizing the term 2.16 actually means to move the edgelets in the pairs as close as possible to each other.

We perform $\mathbf{T}'_r(\mathbf{T}(\mathcal{E}_v))$ and recursively repeat the whole algorithm with the result until convergence of the distances. At that moment, we become the desired refining transformation $\mathbf{T}_r$.

Verification of the candidate detection

The final step is to measure, once again, an average distance 2.11 between the paired edgelets of the classified image and the edgelets of the view that has been mapped onto it using successively the transform T and $\mathbf{T}_r$. For the case, when some edgelets have been paired wrongly and so do not correspond to the same point in the real world, there is a particular saturated value for the distances, so that these cases don't affect too much the resulting average distance. If the average distance is below a threshold, the candidate detection is verified and the test object is classified.

2.7 Modifications

Different classification

While D. Damen et al. use exactly the approach described in the section 2.6, we haven't implemented the refining mapping $\mathbf{T}_r$ yet. Instead we use the following approach:

We first build the whole descriptor representation of the classified image and subsequently use *each* of its element as keys to the hash table to become candidate detections, as described in the beginning of the section 2.6. In the end we count the number of votes for the individual candidates and select the one that occurred most often, i. e., the one with the greatest number of corresponding descriptors. More formally, if we consider the classification as a mapping $\mathcal{C}$ from the set A of all objects being classified to the set B of all

the known objects, we can write:

$$\mathcal{C}(a) = \arg\max_{b \in \mathcal{B}} N_d(a, b), \tag{2.17}$$

where $a \in A$ and $N_d(a, b)$ stands for the number of the descriptors in the object a matching a descriptor in the object b. Under the term *matching* we mean, they are equal in the quantised feature space (see the section 2.5).

In the future, we plan to proceed the same as D. Damen et al. in the classification phase and use the descriptors to get the candidate detections only but we believe, the experience we have acquired in our approach will help us to make the candidate detection and therefore the whole classification more effective. That is actually why we have decided for it.

Branching parameter

We have also introduced a new parameter - a *branching factor*. It serves to reduce the average number of descriptors derived from an image. Normally, a great number of descriptors can be found from a single starting edgelet of a constellation, as apparent from the sections 2.4 or from the pseudo-code in the section 4.1. It is because the process of tracing constellations is implemented as a "branching" recursive function. The branching factor simply states the maximum number of "branches" allowed at each depth of the recursion. Specifically, if the factor is set to one, there will be at most one descriptor for every starting edgelet.

The parameter, if set, reduces both the space and the time complexity but decreases the probability of recognising the object. The program computes the value of the parameter from a *base* given by the user and from the size of the edgelet representation being processed. The greater the representation, the lower the value. In other words, simple objects will be described as detailed as possible, while the complex (and therefore more easily recognizable) ones more vaguely. The base may be set separately for the training and the testing phase, since we usually have different demands for the complexity of the algorithm in each of them.

Elimination of duplicate views and descriptors

In the learning phase, we use an approach which reduces the space complexity, while not harming the later recall rate, and moreover, allows the object to be learned comfortably for the user. It is based on the simple idea, not to store same things twice: if there are many images to learn a single object, we first try if PENTAGRAM_VT will successfully recognize the object based on a newly incoming image. If so, the view of that image will not be stored. However, what *will* be stored, are all *new* descriptors, the image contained.

The intention is that the program can learn an object from a video, where we covered various viewing angles and distances, and it decides independently, which frame to use and which will be discarded.

Similarly, we prevent the hash-table before storing duplicate entries, i. e. the same descriptors, if pointing at the same object.

Expedite classification

During classification, when searching for descriptors, PENTAGRAM_VT doesn't take the potential starting edgelets one after the other, as they are in the view, but pseudo-randomly. It comes especially in hand if we use the descriptors only to verify candidate detections, as described in the section 2.6, because we prevent the algorithm from getting stuck in a local part of an image generating false candidates. This may be for example due to noise or non-characteristics regions.

It is useful even in our approach to the classification, because we allow the classifier to make an expedite decision before all the descriptors of a test image has been found and compared with the library, if the previous ones vote clearly for one object.

Modified deciding rule

If some of the known objects have considerably simpler shape than the others, there will be less constellations satisfying the path and the objects will be represented with respectively low number of descriptors. Since we use the deciding rule 2.17, it follows the algorithm naturally tends to classify everything as the more complex objects. The simplest idea is to solve this issue by redefining the deciding rule:

$$\mathcal{C}(a) = \arg\max_{b \in \mathcal{B}} \frac{N_d(a,b)}{|b|}, \tag{2.18}$$

where we used the same notation instead of $|b|$ which stands for the size of the descriptor representation of the object b. So that we don't classify the object a as being b if b has the greatest number $N_d(a,b)$ among all $b \in \mathcal{B}$ any more but instead if the ratio $\frac{N_d(a,b)}{|b|}$ is the biggest possible.

Chapter 3

The experiments

3.1 Experiments on images

We have performed a series of experiments to test the abilities of the program. In all of them we used a single path $\Theta = (\frac{\pi}{3}, -\frac{\pi}{4}, -\frac{\pi}{3}, \frac{\pi}{3})$ to trace out five-edgelet constellations. The length of edgelets was set to 8px. The base for the branching parameter (see sec. 2.7) was not limited during training. In the classification phase it was set to two. Further, we have chosen the hash-function parameters equal to $N_\phi = 14, N_\delta = 27, \delta_{max} = 21, d_{min} = 25$px. All the parameters were set according to the results of the experiments in the section 4.2.

All the experiments were performed on the training dataset from the figure 3.1. In the first experiment, we used the dataset one from the figure 3.3. The goal was to demonstrate the robustness of the algorithm to scaling, rotation and partial occlusion. Here, we achieved the classification rate of one hundred percent. Mention that although the descriptors 2.5 are scale invariant, this is not fully true for the method as a whole because of the fixed length of the edgelets and scale sensitivity of the edge detector. Nevertheless, a certain robustness has been proven by the experiment.

The second and the more complex experiment was performed on the test dataset two, figure 3.4. Here, we achieved the classification rate of 76%.

See the table 3.1 for the time complexity measured in the both experiments.

3.2 Experiments on video

We have also performed two experiments on a video input. For the sake of simple analysis, they were both performed off-line with the average classification time similar as in the table 3.1. We again used the data from the figure 3.1 for training. The achieved success rate was 42%. This is seemingly a poor result but we have to take into account that there

Figure 3.1: The training dataset. All images are of the size 377 × 233px

Classified as

Data to classify

	bag	car	mobile	stapler	torch	candleholder	hanger	outlet	sunglasses	tuner	not classified	Success rate
bag	5											100%
car		5										100%
mobile			5									100%
stapler				5								100%
torch				2	3							60%
candleholder						4			1			80%
hanger				1			4					80%
outlet		1	1	1				1			1	20%
sunglasses				1					4			80%
tuner		1		2						2		40%
Total success rate:					76%							

Figure 3.2: The confusion matrix achieved on the training dataset and test dataset from the figures 3.1 and 3.4 respectively.

	Average time (s)	Worst time (s)
Training phase	2.8	6.9
Test phase	0.7	1.7

Table 3.1: The time complexity achieved in the both experiments.

Figure 3.3: The test dataset one. The images were all taken by a camera, non was artificially generated from an other in Photoshop or in related software. All images are of the size 377×233px.

Figure 3.4: The test dataset two. All images are of the size 377×233px

Figure 3.5: A few snapshots from the test video. The whole video consist of one hundred sixty 600×384px frames of eight objects from the training set 3.1.

was only a single image for each object in the learning phase. Plus, in the case of a video, it is not necessary to classify all the frames to identify an object. Snapshots from the test video may be found in the figure 3.5.

We also performed an experiment on a part of the dataset which was used in [5] and is public available at [7]. Both the learning and the testing data are a video. However, since our classifier is not capable of multiple detections, these data had to be significantly adjusted. When dealing with six objects only, 56% of all frames were classified correctly and we absolutely failed when the whole dataset was used. This confirms, our approach should not be used standing alone but for candidate detection only (as intended).

Chapter 4

Analysis

4.1 Complexity of the algorithm

Time complexity

The number of descriptors traced out in an image determines the time complexity of the learning phase. It has already been sketched, how the tracing is done, in the section 2.4. The following pseudo code shows it more formally:

```
% Global variables:
Descriptor desc_repre[]; % Empty at the beginning.
Edgelet edg_repre[]; % Edgelet representation. Not empty.
const int EDGS_IN_PATH = 5;

% Main function:
for (int i = 0; i < size(edg_repre); i++) {
    Edgelet toleratedEdgs[] = findTolerated(contEdg, 1);
    for (int j = 0; j < size(toleratedEdgs); j++) {
          describe(new Descriptor, toleratedEdgs[j], edg_repre[i], -1, 2);
    }
}

% Function definitions:
findTolerated(Edgelet contEdg, int depth) {
    Edgelet toleratedEdgs[];
    for (int i = 0; i < size(edg_repre); i++)
      if (tolerated(edg_repre[i])
         toleratedEdgs.add(edg_repre[i])

    return toleratedEdgs;
```

```
}

describe(Descriptor desc, Edgelet contEdg, Edgelet prevEdg,
         double prevDist, int depth) {
   Edgelet toleratedEdgs[] = findTolerated(contEdg, depth);
   for (int i = 0; i < size(toleratedEdgs); i++) {
      Descriptor newDesc = desc;
      if (prevDist != -1)
          newDesc.addDelta(distance(contEdg, toleratedEdgs[i])/prevDist);
      newDesc.addPhi(rel_angle(contEdg, toleratedEdgs[i]));
      if (depth == EDGS_IN_PATH) {
         desc_repre.add(newDesc);
      } else {
         describe(newDesc, toleratedEdgs[i], contEdg,
         distance(contEdg, toleratedEdgs[i]), depth+1);
      }
   }
}
```

The edgelet representation of an image is stored in the array `edg_repre`. The main function hands one after the other the edgelets of the edgelet representation to the function `findTolerated` as potential starting edgelets of constellations. For a given potential starting edgelet, the representation is being searched there again to find the edgelets satisfying the first angle of the path. These are returned.

At this moment the algorithm has traced out the first two edgelets of a few potential constellations. In this example constellations consist of five edgelets. For each of the potential constellations the function `describe` is called. As the argument `depth` increases, It looks for the thirds, fourths and fifths edgelets of the potential constellations in a forked recursive manner.

Notice that at each level given by `depth` the function `describe` calls itself recursively not once but many times depending on the number of potential constellations having been traced out at that moment. This is why we refer to it as a "branching" or "forked" recursion in the text.

Using the famous big O notation, this gives the following time complexity:

$$O(\frac{E-1}{p}N^E) = O(N^E) \tag{4.1}$$

where N-number of edgelets in an edgelet representation, E-number of edgelets in constellations, p-an avarage probability that an edgelet lies in a tolerance band of a previous edgelet of a potential constellation.

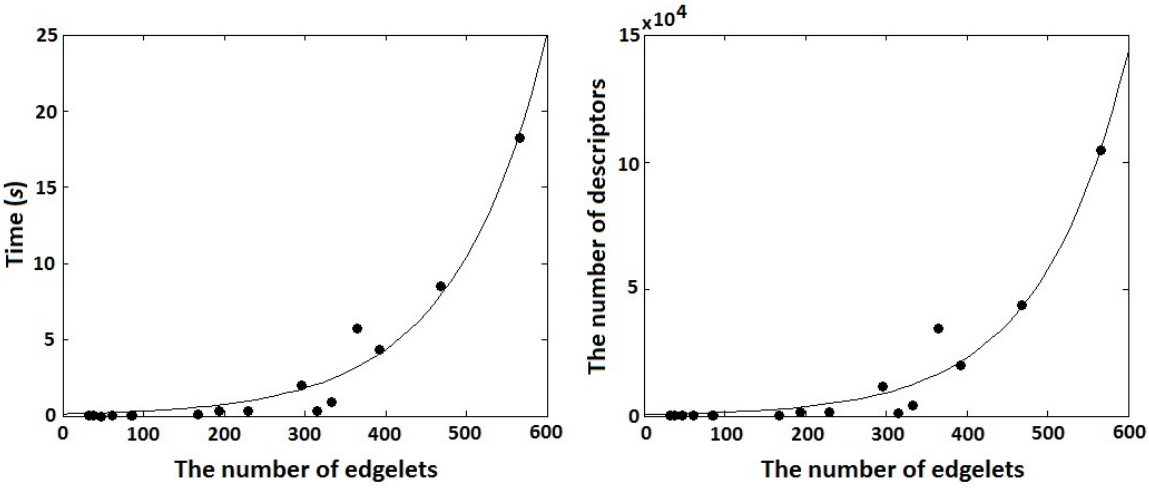


Figure 4.1: The time needed to create the representation (left) and the repective number of descriptors created (right) as functions of the size of edgelet representation. In the experiment, the optimisation with the use of the branching factor (see sec. 2.7) was forbidden.

It is worthwhile emphasizing that, according to [17], the time complexity needed for accessing or reading elements from an UNORDERED_MAP, is constant in the average case. This makes the total asymptotic complexity of the algorithm independent on hashing.

Space complexity

The space complexity also depends mainly on the number of created descriptors and it is a weakness of the method. Possibly, many tens thousands of descriptors may be created for a single view. See the figure 4.1. If we for example use constellations of the size five, i. e. $n = 5$ in the equation 2.5, and if we represent the numbers in descriptor with 4 Byte floats, we typically need about 1.5 MB for a view.

This is a few times more then the original image, though containing much less information. This is an inseparable feature of the method and follows from the way, how we trace out constellations: Most edgelets will be covered much more times then just once.

In the experiment on Bristol dataset in the section 3.2, it was confirmed that this may really cause troubles. We worked with more than twelve hundred training views and had to face the problem, not to run out of the primary memory of the computer.

4.2 Tuning parameters

There is quite a plenty of words introduced with the word *particular* in the text: particular values, thresholds and so on. These correspond to parameters that must be all assigned

some specific values but we don't now which ones at the moment. Setting all the parameters manually at every start of the program is not a very user-friendly solution. A question therefore arises, if some of them may be automatically computed or immutably fixed at a "best" value.

Our program has been written so that all the important parameters can be set directly from the command line. At the end of each run it also outputs a confusion matrix and some more statistics about the success rate; everything in a an easily processable format (in Microsoft Excel for example). We can therefore let Pentagram_vt being automatically repeatedly started by an external simple program, written in the batch language, store the outputs and analyse them in bulk.

We have used this approach together with the training and the testing dataset from the figures 3.1 and 3.4 respectively to randomly search the parameter space to answer the question from the first paragraph. More specifically, we looked for convenient hash-function parameters and paths, where we have stated, the paths will be of the length four, i. e. $n = 5$ in the equation 2.4. See the tables 4.1b, 4.1a for the results.

$\mathbf{d_{min}}$	$\delta_{\mathbf{max}}$	$\mathbf{N}_\phi$	$\mathbf{N}_\delta$	**Correctly classified**	**Incorrectly classified**
25	12	14	27	**38**	**12**
25	21	14	27	**38**	**12**
8	16	17	12	**38**	**12**
8	18	17	12	**38**	**12**
25	5	14	12	**37**	**13**
8	27	17	99	**18**	**32**
8	18	14	99	**18**	**32**
8	27	14	99	**18**	**32**
25	5	30	55	**17**	**33**
8	27	30	99	**17**	**33**

(a) The top five and the worst five choices of the hash-table parameters.

The path $(\theta_1, \theta_2, \theta_3, \theta_4)$	**Correctly classified**	**Incorrectly classified**
$(\frac{\pi}{3}, -\frac{\pi}{4}, -\frac{\pi}{3}, \frac{\pi}{3})$	**38**	**12**
$(\frac{\pi}{3}, -\frac{\pi}{3}, \frac{\pi}{4}, -\frac{\pi}{4})$	**38**	**12**
$(\frac{\pi}{4}, \frac{\pi}{4}, -\frac{\pi}{3}, -\frac{\pi}{3})$	**37**	**13**
$(\frac{3\pi}{4}, \frac{\pi}{3}, \frac{\pi}{4}, -\frac{\pi}{4})$	**37**	**13**
$(\frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{3}, -\frac{\pi}{4})$	**37**	**13**
$(\frac{\pi}{3}, \frac{3\pi}{4}, \frac{\pi}{3}, -\frac{\pi}{4})$	**18**	**32**
$(\frac{\pi}{3}, \frac{3\pi}{4}, \frac{\pi}{3}, -\frac{\pi}{3})$	**18**	**32**
$(\frac{3\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{3}, -\frac{\pi}{4})$	**17**	**33**
$(\frac{3\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{3}, -\frac{3\pi}{4})$	**16**	**34**
$(\frac{3\pi}{4}, -\frac{\pi}{4}, -\frac{\pi}{4}, -\frac{3\pi}{4})$	**16**	**34**

(b) The top five and the worst five choices of the path.

Table 4.1: Results of the parameter tuning.

Chapter 5

Conclusion

My intention in this thesis was to re-implement, to analyse and, if possible, to improve the method [5] for recognition of texture-less objects. It is based on a two-state classification. In the first part a candidate detection is performed on an image and the second one verifies it. I have focused mainly on the first part. It is crucial for the algorithm to be fast and effective, since the verification phase is time consuming and therefore not capable of coping with many false candidate detections, at least not in real time.

The pre-classification is based on describing particular edge segment constellations from a test image and comparing them with a database. To analyse this phase separately, I have introduced a simple classifier based on the descriptors only, skipping the verification. It has served to discover some possible improvements of the algorithm, see the section 2.7.

In the section 4.2 this approach was used to analyse the sensitivity of the algorithm on using different edge constellations and various feature space quantisation. Both have shown to be crucial. Therefore, a search for "the best" choice has been performed. It has been also shown that this may vary for different objects, although some generally better choices exist.

It is again to emphasize that my classifier wasn't expected to outperform state-of-the-art methods but to vote for candidate detections only. Though, it works well and almost real-time on small sets of training objects on a clear background and reaches 76% success rate on test images and somewhat poorer 41% on a test video.

5.1 Future work

We have discovered a few weak points of the method that hasn't been originally mentioned. These are sensitivity to highly curved places and inconvenience for describing very local features of objects. These problems are both worth deeper study. An edge smoothing or involving a curvature metric in descriptors are possible solutions of the first one.

As expected, it has turned up that different objects may be convenient to describe using different paths or constellations. In the future, it would be therefore smart to use more of them at once. The algorithm could decide itself to use an other paths for description, if the previous ones haven't been able to trace out enough constellations.

The main task is to finish the implementation of the verification phase, which is still in experimental stage only since the refining transform based on [21] hasn't been implemented yet.

Bibliography

[1] BARROW, H. G. et al. Parametric correspondence and chamfer matching: two new techniques for image matching. In *Proceedings of the 5th international joint conference on Artificial intelligence - Volume 2*, IJCAI'77, p. 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc. Available from: <http://dl.acm.org/citation.cfm?id=1622943.1622971>.

[2] CAI, H. – WERNER, T. – MATAS, J. Recognition beyond appearance-based methods. Technical report, Centre for Machine Perception, Czech Technical University in Prague, December 2012.

[3] CANNY, J. F. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Maschine Intelligence*. November 1986, p. 679–698.

[4] DALAL, N. – TRIGGS, B. Histograms of Oriented Gradients for Human Detection. In *CVPR*, p. 886–893, June 2005.

[5] DAMEN, D. et al. Real-time Learning and Detection of 3D Texture-less Objects: A Scalable Approach. In *British Machine Vision Conference*. BMVA, September 2012. Available from: <http://www.cs.bris.ac.uk/Publications/Papers/2001575.pdf>.

[6] DERICHE, R. Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector. *International Journal of Computer Vision*. 1987, p. 167–187.

[7] Dima Damen and Pished Bunnun and Andrew Calway and Walterio Mayol-Cuevas . *Experimental dataset obj30* [online]. 2013. [cit. 22.. 5. 2013]. Available from: <http://http://www.cs.bris.ac.uk/Publications/pub_master.jsp?id=2001575>.

[8] FORSYTH, D. A. – PONCE, J. *Computer Vision: A Modern Approach*, Application: finding in digital libraries. Prentice Hall, august 2002.

[9] FORSYTH, D. A. – PONCE, J. *Computer Vision: A Modern Approach*, Sources, shadows and shading. Prentice Hall, august 2002.

[10] FORSYTH, D. A. – PONCE, J. *Computer Vision: A Modern Approach*, Stereopsis. Prentice Hall, august 2002.

[11] FULZENSZWALB, P. – HUTTENLOCHER, D. Distance Transforms of Sampled Functions. Technical report, Cornell University, September 2004.

[12] HINTERSTOISSER, S. et al. Dominant Orientation Templates for Real-Time Detection of Texture-Less Objects. In *CVPR*, p. 2257–2264, June 2010.

[13] HINTERSTOISSER, S. et al. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *CVPR*, p. 548–562, October 2012.

[14] RAMER, U. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*. 1972, 1, 3, p. 244 – 256. ISSN 0146-664X. doi: 10.1016/S0146-664X(72)80017-0. Available from: <http://www.sciencedirect.com/science/article/pii/S0146664X72800170>.

[15] ROCHA, A. et al. Automatic produce classification from images using color, texture and appearance cues. *Computer graphics and image processing*. October 2008, p. 3–10.

[16] SHOTTON, J. – BLAKE, A. – CIPOLLA, R. Contour-Based Learning for Object Detection. In *ICCV*, p. 503–510, October 2005.

[17] THE CPPREFERENCE TEAM. *C++ std::unordered_map container description* [online]. 2013. [cit. 7. 5. 2013]. Available from: <http://en.cppreference.com/w/cpp/container/unordered_map>.

[18] THE OPENCV TEAM. *Home page of openCV project* [online]. 2013. [cit. 7. 5. 2013]. Available from: <http://http://opencv.org>.

[19] WIKIBOOKS CONTRIBUTORS. *List of CBIR engines* [online]. 2013. [cit. 22.. 5. 2013]. Available from: <http://en.wikibooks.org/wiki/List_of_CBIR_engines>.

[20] WIKIBOOKS CONTRIBUTORS. *LaTeX* [online]. 2013. [cit. 12. 5. 2013]. Available from: <http://en.wikibooks.org/wiki/LaTeX/>.

[21] ZHANG, Z. Iterative Point Matching for Registration of Free-Form Curves and Surfaces. In *IJCV*, 13, p. 119–152, 1994.