Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: {pripbachelor}@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-121 December 14, 2009

Bachelor Thesis at PRIP
Collection of Summer and Winter Term 2009

Yll Haxhimusa, Walter G. Kropatsch,
Robert Sablatnig, Adrian Ion (Ed.)

Preface

This is the second collection in the series of Bachelor Thesis at PRIP. We have collected students best works done in Summer and Winter Term 2009.

First work in the collection is the bachelor thesis of Georg Zankl. He presents an algorithm for computing the eccentricity transform of a polygonal (continuous) shape which can have holes. The algorithm decomposes a polygonal shape, by gradually refining a partitioning. The partitioning is then used to eficiently compute the eccentricity for any point inside the shape.

In the second work, Lukas Fischer, presents a survey on sequential Monte Carlo Methods, also known as Shape Particle Filters, used for the segmentation in medical imagining. This report surveys the robustness of these methods on different medical example images. Results on different data (synthetic rectangles, MRI slices and radiographs) are reported.

The third work done by Timo Kropp, presents a software implementation that allows the user to visualize trajectories of moving object in the Microsoft Bing maps.

We would like to thank all the authors and their supervisors for their support.

Yll Haxhimusa (Ed.)
Walter G. Kropatsch
Robert Sablatnig
Adrian Ion
Vienna University of Technology
Faculty of Informatics
Insititute of Computer Aided Automation
Pattern Recognition and Image Processing Group

Vienna, November 2009

Contents

Eccentricity Transform of Polygonal Shapes

Georg Zankl
Supervisor: Walter Kropatsch and Adrian Ion

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: hru@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

Bachelor Thesis Vienna November 23, 2009

Eccentricity Transform of Polygonal Shapes

Georg Zankl
georg.zankl@gmail.com

Supervised by Prof. Walter Kropatsch, Dr. Adrian Ion

Abstract

The eccentricity of a point of a shape is the maximum of the shortest paths to all other points in the shape (i.e. the shortest path to the point which is furthest away). The eccentricity transform of a shape associates to each point its eccentricity. This thesis presents an algorithm for computing the eccentricity transform of a polygonal (continuous) shape which can have holes. It explains basic principles as well as similar methods and discusses suggestions of further improvement of the presented algorithm. The algorithm decomposes a polygonal shape, by gradually refining a partitioning. The partitioning is then used to efficiently compute the eccentricity for any point inside the shape.

1 Introduction

Eccentricity is a measure from graph theory, associating to each node the maximum of the shortest paths to all other nodes i.e. the length of the shortest path to the node which is furthest away. For the graph $G = \langle V, E \rangle$ and the vertex v it is defined in [Buckley90a, p. 31] as follows:

$$ecc_G(v) = \max\{\lambda(\pi_G(v, w)) \,|\, \forall w \in V\}, \tag{1}$$

where $\pi_G(v, w)$ is any path from v to w and $\lambda(p)$ is the length of path p.

This can also be defined for discrete (as in [Kropatsch06a]) or continuous shapes (as in [Kropatsch07b]) by treating each pixel or point as a node and specifying a neighborhood (e.g. Euclidean, 4- or 8-neighborhood). For a shape $\mathcal{S}$ and a point $p \in \mathcal{S}$ the eccentricity is denoted as $ecc_{\mathcal{S}}(p)$.

Without any obstacle, the shortest path between two points in a 2D convex shape is a straight line. The length of the shortest path between such points is the Euclidean distance. If there are obstacles or the shape is not convex, then the shortest path consists of a sequence of line segments connected at the endpoints. The sum of the length of these line segments is the length of the path and has to be minimal.

The Eccentricity Transform (ECC) associates to each node of a graph its eccentricity. This can be done for discrete shapes (as shown in [Kropatsch06a]) as well. A continuous shape has an infinite number of points and therefore it is not possible to store the eccentricity for each point. Instead the ECC of such a shape becomes a function, that calculates the eccentricity for a specified point.

This work focuses on the computation of such an ECC function in continuous shapes.

1.1 Definitions

A Continuous Polygonal Shape $\mathcal{S}$ in 2D is a connected region in $\mathbb{R}^2$ bounded by a set of line segments.

A Path $\pi_{\mathcal{S}}$ in a continuous shape $\mathcal{S}$ is a curve totally contained in $\mathcal{S}$. A curve with endpoints $p, q \in \mathcal{S}$ is further denoted as path between p and q or $\pi_{\mathcal{S}}(p, q)$.

The Length of a Path $\lambda(\pi_{\mathcal{S}})$ is the length of the curve, i.e. for a parametrized curve $c : [a, b] \rightarrow \mathbb{R}^2$ the length is

$$\lambda(\pi_{\mathcal{S}}) = \oint_c dt = \int_a^b ||c'(t)||_2 dt \tag{2}$$

If the curve is not differentiable in a finite number of points, then the integral has to be split on these points and the sum of all integrals is the length of the path. For example if the path is a sequence line segments, then the length of the path is the sum of the lengths of each line segment.

The Geodesic Distance $d^{\mathcal{S}}$ between two points $p, q \in \mathcal{S}$ is the length of the shortest path between them, i.e.

$$d^{\mathcal{S}}(p, q) = \min\{\lambda(\pi_{\mathcal{S}}(p, q))\} \tag{3}$$

The Single Source Geodesic Distance Transform (SSGDT) also known a geodesic distance function [Soille02a] calculates for every point (or pixel) of a shape the geodesic distance to one source point, which is specified beforehand. Like with the ECC this is a function in a polygonal shape.

An Eccentric Point is a point, which has the biggest geodesic distance to at least one other point.

A Ring names a sequence of points, which form the edges of a hole or the outer boundary of a polygonal shape. The outer boundary will be called "outer ring" in contrast to the holes, which will be called "inner rings".

(Euclidean) Visibility (in a 2D continuous shape): Two points are visible to each other if the line segment formed by these points does not intersect any ring of the shape. Visibility is only defined for points inside the shape. This can be interpreted as a function $vis : \mathcal{S} \times \mathcal{S} \rightarrow \{0, 1\}$ which returns 1 if p and q are visible to each other and 0 otherwise.

The Visibility Graph $G = \langle V, E \rangle$ consists of nodes associated to points within the shape and has edges for each pair of points which are visible to each other, i.e.

$$vis\,(v_1, v_2) = 1 \Rightarrow (v_1, v_2) \in E \qquad v_1, v_2 \in V \tag{4}$$

An Occlusion (in 2D) of a point s is a region of the shape, the points of which are not visible to s. Such a region is also called Occlusion Patch.

An Occlusion Line L of a point s is a simply connected region, the points of which are visible to s and adjacent to an occlusion, i.e. every point $p \in L$ is visible to s and a point q in the ϵ-neighborhood of p exists, which is not visible to s. Each occlusion has at least one occlusion line.

The Occlusion Point o of a point s is the nearest point to s of one occlusion line of s (see Figure 1.1). There is exactly one occlusion point per occlusion line and it is an endpoint of the line. In a polygonal shape this point is a vertex of one of the rings.

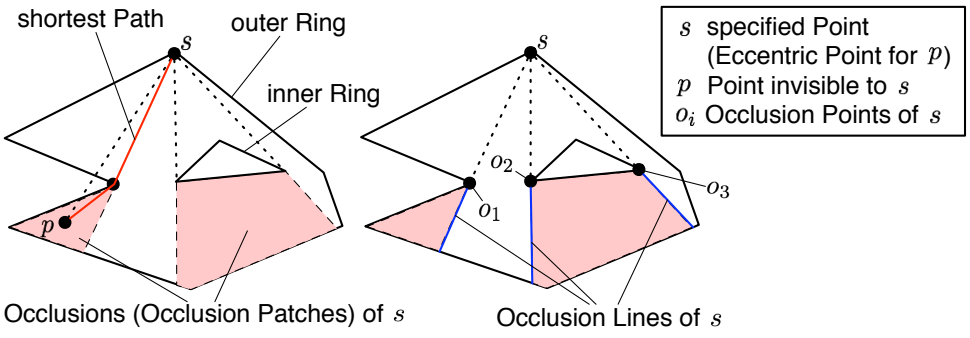


Figure 1.1: Illustration of the Definitions in Section 1.1

1.2 Motivation

As described in [Kropatsch06a] the eccentricity transform can be used in shape matching and image processing and is robust to salt and pepper noise.

Within this bachelor thesis an algorithm is presented which builds on the method described in [Kropatsch07b] (see also Section 2).

It is assumed, that the 2D polygon satisfies the following characteristics:

- ▷ may have polygonal holes
- ▷ all holes lie fully inside the polygon
- ▷ no hole lies within another hole
- ▷ no self intersections of any ring
- ▷ no intersections between two rings
- ▷ all eccentric points lie on the corners of the outer ring

This last item serves only to ensure the accuracy of the computation, as by default corner points are the candidate eccentric points. The algorithm also works when eccentric points are not located on the corners and approximates the exact result in such a case. If all eccentric points are known and added to the list of starting points (see Section 3.3) the accuracy only depends on numerical errors.

1.3 State of the Art

The following papers describe methods for computing the eccentricity transform:

[Kropatsch06a] The Eccentricity Transform (of a Digital Shape):
This paper presents an algorithm, which searches for eccentric points (their number is finite in a discrete shape) and calculates the ECC by accumulating SSGDTs for each eccentric point, always storing the maximum value per pixel.

[Ion08a] Euclidean Eccentricity Transform by Discrete Arc Paving:
In this article the computation of a SSGDT in a digital shape is described and several algorithms are presented, which calculate the ECC similar to [Kropatsch06a].

[Ion07b] Decomposition for Efficient Eccentricity Transform of Convex Shapes:
A method of the ECC is presented in this paper, which uses the properties of a symmetric shape to reduce the complexity of calculating the eccentricity only for one half. It is also discussed for more general shapes, but no algorithm for such cases is given.

[Kropatsch07b] Computing the Eccentricity Transform of a Polygonal Shape:
This paper provides the basis for this thesis, it presents a method for computing the ECC of a polygonal shape $\mathcal{S}$, where the shape is decomposed into smaller patches in such a way, that the eccentric point e for any specified point p can be determined, i.e. the point

$$e = \arg\max_{v \in \mathcal{S}} \{d^{\mathcal{S}}(p, v)\} \tag{5}$$

is found. Then the eccentricity of the point is $ECC(p) = d^{\mathcal{S}}(p, e)$

The first two papers ([Kropatsch06a] and [Ion08a]) consider a discrete shape. Eccentric points are found by calculating the SSGDT from other points inside the shape. It is possible that eccentric points are missed, so these algorithms provide an approximation (except the naive algorithm, which doesn't search for eccentric points). In addition to that, the Euclidean distance function may be approximated as well.

The third method ([Ion07b]) is supposed to increase the efficiency of other algorithms by splitting the shape beforehand. In the associated article it is shown how to do that for some basic symmetric shapes.

There is also a similar method to the one described in [Kropatsch07b], that is the computation of the furthest-site geodesic voronoi diagram presented in [Aranov88], which decomposes the shape into regions, so that all points in the same region have the same eccentric point. The difference is that in [Kropatsch07b] the shape is further divided in such a way, that it is sufficient to work with Euclidean distances. This will be explained in detail in Section 2.

1.4 Organization of the Thesis

This thesis is divided into the following sections:

Section 1 explains important terms, discusses existing methods for computing the eccentricity transform and states the goal of this work.

The following sections are constitute the scientific of the bachelor thesis:

Section 2 summarizes and discusses the method of computing the eccentricity transform of a polygonal shape described in [Kropatsch07b].

Section 3 explains a refined algorithm and shows how it works by providing a simple example.

The following sections report the practical work:

Section 4 discusses problems, which occur due to numerical errors and how they can be considered in the algorithm. Open problems with such errors are covered as well.

Section 5 shows the partitioning and discretization of the ECC function on more complex examples. It is also compared to the results of an algorithm, which computes the ECC for the discretized shape with same starting points as the polygonal computation.

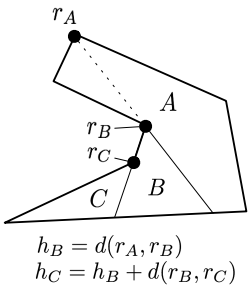
The last part is valid for the overall work:

Section 6 lists open problems and gives hints for further improving the algorithm.

2 Eccentricity Transform in a Polygonal Shape

In the book "12th Iberoamerican Congress on Pattern Recognition (CIARP 2007)", the article [Kropatsch07b] describes a method for the computation of the eccentricity within a polygonal shape.

Starting with a possible eccentric point (e.g. one of the polygon corners) the shape is split into visible and occluded patches. The starting point of the current iteration will be called s. This process is continued from all occlusion points of s inside their associated occlusion patches. Every time a visibility patch is computed, s will be stored as so called reference point. In addition a handicap is calculated for each reference point, consisting of the handicap of the previous point plus the distance between these two points (see Figure 2.1). The result measures the shortest distance to the starting point.



A	Visibility Patch for r_A
$B \cup C$	Occlusion Patch for r_A
B	Visibility Patch for r_B
C	Visibility Patch for r_C, Occlusion Patch for r_B
r_X	Reference Point for $X \in \{A, B, C\}$
h_X	Handicap of r_X

Figure 2.1: Illustration of the Occlusion Propagation starting with r_A

Once this "Occlusion Propagation" is finished (i.e. there aren't any more occlusions for the reference points for each patch), a single source geodesic distance (SSGD) function can be evaluated, which is the basis for computing the eccentricity. This is done by determining the patch in which the query point lies, calculating the distance to its reference point and adding its handicap.

To actually compute the eccentricity, the occlusion propagation has to be done for each point of the known set of eccentric points $\mathcal{E}$, splitting existing patches and adding the reference points accordingly. The result is a partitioning of the shape with multiple reference points per patch.

Calculating the eccentricity is done by again determining the patch for the given point and calculating all the geodesic distances to all starting points,

then taking the maximum (since the eccentricity is the maximum of the length of all shortest paths to this point). Some of the reference points can be eliminated, because they don't have any influence on the result, i.e. they are not part of the shortest path for any point in the patch.

The article also mentions how to deal with holes. Considering the SSGD partitioning a separation hyperbola behind a hole is necessary to be able to determine the relevant reference point (see Figure 2.2).

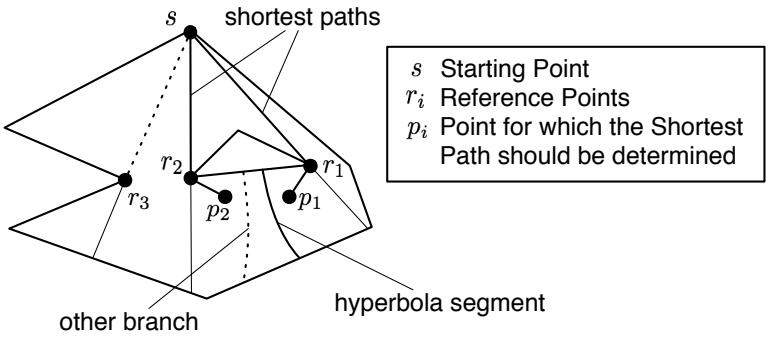


Figure 2.2: Illustration of the SSGD in a continuous shape (the drawn hyperbola is estimated manually)

The reason for this hyperbola is actually, that we have to determine the shorter one of two paths. The decision boundary is defined by equalizing the lengths of both paths.

$$d(\mathbf{p}, r_1) + h_1 = d(\mathbf{p}, r_2) + h_2 \tag{6}$$

The article further explains how to derive the formula of a generic hyperbola. By doing so the information which handicap belongs to which reference point is lost, resulting in the two branches of a generic hyperbola (see Figure 2.2). To determine the shortest path one branch is sufficient, in fact two branches prove to be problematic, since it is not possible to decide which branch is relevant without the lost information.

If the query point p does not lie on a decision boundary, there is exactly one shortest path between the starting point s and p, therefore all other paths, which can be evaluated with the other reference points, are not shortest paths. All the paths which can be evaluated with the reference points of a patch are called shortest path candidates. Since we are looking for the

shortest path, the length of all shortest path candidates can be evaluated and compared. The smallest value is the length of the shortest path and therefore the geodesic distance between s and p:

$$d^{\mathcal{S}}(s,p) = \min_i \{d(r_i,p) + h_i\} \tag{7}$$

So instead of storing any parameters of a hyperbola, the reference points with their handicaps are sufficient to decide for the shortest path. When doing so the eccentricity can be calculated by determining the patch for point p and then calculating the minimum of all shortest path candidates for each eccentric point e in the set of known eccentric points $\mathcal{E}$. The maximum of these shortest paths equals the eccentricity of p.

$$ECC(p) = \max_{e \in \mathcal{E}} \{\min\{d(r_i,p) + h_i | eccPoint(r_i) = e\}\} \tag{8}$$

An alternative is to compute additive weighted Voronoi diagrams (where the weight is the handicap) for each set of reference points with the same starting point (such a set can have a cardinality greater than 2). Then compute the smallest common partition of Voronoi cells and distributing the reference points accordingly. Each remaining patch with more than one reference point can be further divided to gain a partition with one reference point per patch. To do this, in each of the computed patches (in the above case in each of the Voronoi cells) a furthest point additive weighted Voronoi diagram is necessary.

The method described in [Kropatsch07b] does not consider patches with more than two reference points, which have the same starting point. Figure 2.3 is an example of a patch where this is the case and Figure 2.4 shows the associated hyperbola segments as well as the additive weighted Voronoi diagram of the reference points, which gives exactly one Voronoi cell (or subpatch) per reference point.

The process of actually determining the relevant reference point for a specified point is similar in both methods, since finding the Voronoi cell of point is done by testing which reference point is closer/further considering its handicap. This is the same as determining the shortest/longest of the path candidates.

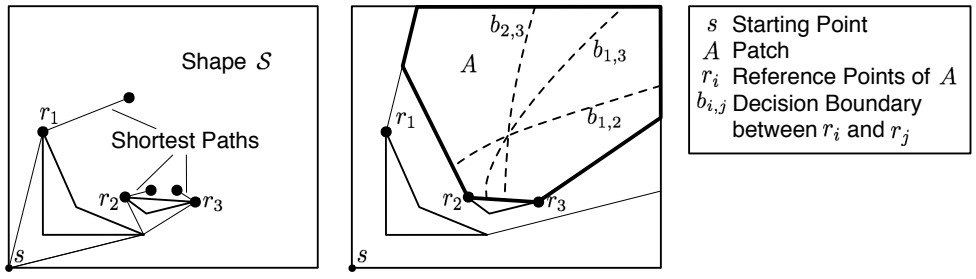


Figure 2.3: Example of a shape with a patch A with multiple reference points for one starting point

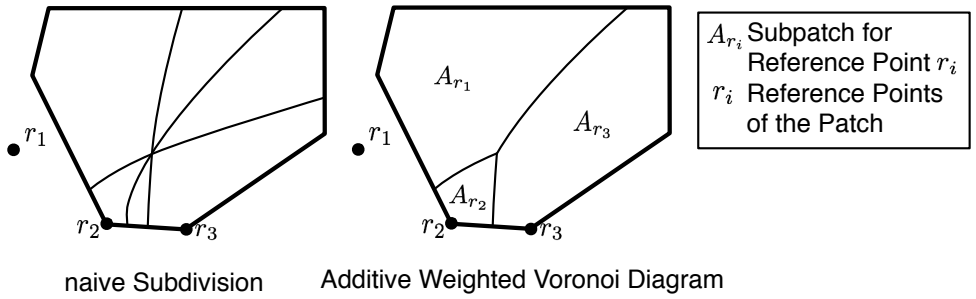


Figure 2.4: Subdivision of patch A from Figure 2.3 (the drawn hyperbola segments are estimated manually)

3 Refined Algorithm

This section presents a refined algorithm for the method discussed in Section 2 and provides pseudo code for the most important operations.

The algorithm creates a hierarchical partition of the shape (i.e. it is an inclusion hierarchy), in such a way that in the leaf nodes the shortest path candidates to the starting points can be determined by calculating the Euclidean distances to the reference points and adding their handicap.

The data structure for all patches is a sequence of vertices per ring, i.e. each two consecutive vertices represent an edge of one ring. The vertices are not stored directly, instead an index for a vertex array is stored. This vertex

array is contained in the data structure of the root node of the hierarchical partition $\mathcal{P}$, which has the following attributes:

vertices is a list of all relevant vertices (of the shape and additional vertices, e.g. starting points), i.e. one element of $\mathcal{P}.indices$ contains a sequence of indices, where two consecutive indices represent an edge of the ring

startPoints is a list of indices to the vertex array, indicating the starting points.

indices contains a sequence of indices to the vertices array for each ring of the shape

children is a list of patches, which is used to create the parent-child relationship in the hierarchy

refPoints is a list of reference points r_i together with their handicaps h_i

All child nodes have the same structure, except that the vertices array and the list of starting points is not needed.

An occlusion line will be further denoted as *occLine* and represents a 2-tuple containing the endpoints of the line. Reference points are stored as a 2-tuple $\langle r_i, h_i \rangle$ containing the point r_i itself and the associated handicap h_i.

The hierarchy represents the accumulation of the SSGD partitions and is created by recursively splitting the partition along the current occlusion lines. The following example explains how the hierarchy is created for two starting points in the corners of the shape:

Example 3.1 *Figure 3.1 shows a simple shape with the first finished occlusion propagation and its hierarchy. From point s one occlusion point is found and its occlusion line calculated. The hierarchy (previously only containing the root node S) is split along this occlusion path, resulting in the patches A and $B \cup C$. The propagation is continued for the second patch, starting from the occlusion point. Another occlusion line is calculated, resulting in the patches B and C.*

The next propagation starts with point s' in Figure 3.2. Two occlusion points are found and their occlusion lines computed. Because of the chosen data structure these lines have to be ordered in such a way, that a sequence of vertices, where consecutive vertices represent an edge of the new patch, can be computed. For example, when the visibility patch is defined clockwise,

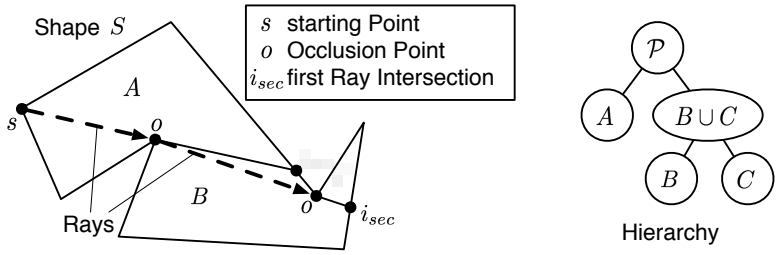


Figure 3.1: Occlusion propagation starting with s

the right occlusion line will be processed first. Such an ordered sequence of occlusion lines will be further called occlusion path.

The root node $\mathcal{P}$ is again split along this path, resulting in the three patches A', B' and C'. The split is done for each subnode of the previous hierarchy as well. The previous patch A is not intersected by the occlusion path, so it is checked, which of the 1st level patches (A', B' and C') contains A and it will be added to the children of this patch (so A becomes a childnode of A'). In the next step the previous patch $B \cup C$ is intersected with the occlusion path, this results in three patches $(B \cup C) \cap A'$, $(B \cup C) \cap B'$ and $(B \cup C) \cap C'$ which will be added as child nodes of the associated 1st level patches (see hierarchy in Figure 3.2).

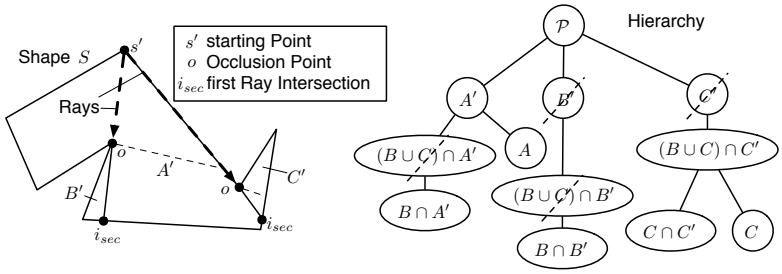


Figure 3.2: Following Figure 3.1, occlusion propagation starting with s'

After the same is done for B and C, the partition is complete and can be optimized, by removing each node, which have only one child, connecting its child to its parent. The optimized hierarchy is shown in Figure 3.3.

Further propagations can be done the same way.

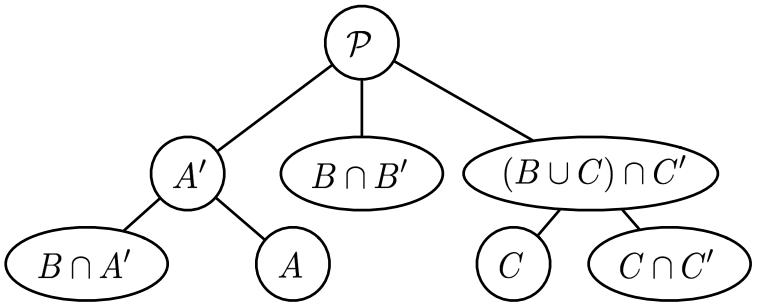


Figure 3.3: Optimized hierarchy

3.1 Occlusion Propagation

The occlusion propagation basically adds one SSGDT to the accumulated partition (see Algorithm 2).

Step 1: From the given starting point s all occlusion points are determined. Algorithm 1 retrieves all vertices in the visibility graph, which are visible from s and checks for each of them if it is an occlusion point, by determining the orientation of the adjacent edges relative to the vector between s and the occlusion candidate c (see Figure 3.4). This method is similar to [Cardenes03], where the occlusion points are determined to calculate the shortest path between two points inside a discrete shape.

Then a ray is cast from each occlusion point o in the direction of of the vector $\overrightarrow{so} = o - s$ (i.e. in the opposite direction of s, see Figure 3.5). The first intersection is stored together with the occlusion point as an "occlusion data" element.

Step 2: Then a call to "calculateOccPath" calculates the occlusion path, a sequence of modified occlusion data elements. This sequence is ordered in such a way, that the calculation of visibility and occlusion patches can be done by following this path.

Since there may already be a partition, which has to be refined, a simple calculation of these patches is not sufficient, instead the partition is split into several parts by the occlusion path. This is done by the function splitTree, which splits each patch of the hierarchy along the occlusion path, creating subtrees for each occlusion patch and the visibility patch in the process.

When the occlusion patches are calculated, the occlusion points of the

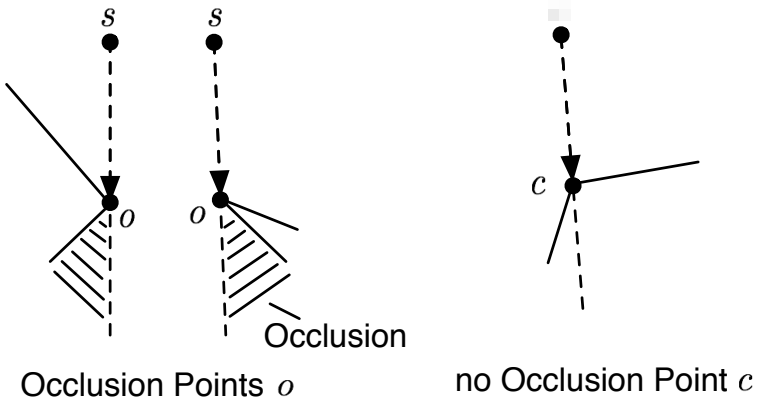


Figure 3.4: Example of occlusion point candidates

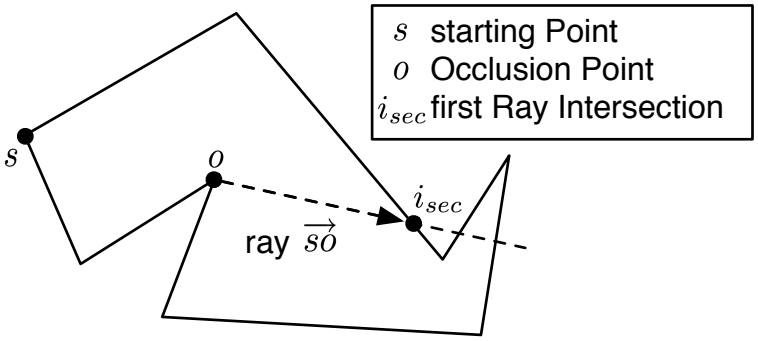


Figure 3.5: Casting a ray from o

Algorithm 1 calculateOcclusions(P, s, G)

Input: polygonal Shape P, Starting Point s, Visibility Graph $G = \langle V, E \rangle$

1: $occData \leftarrow \{\}$
2: $candidates \leftarrow \{(u_i, v_i) \in E : u_i = s \wedge v_i \neq s \wedge v_i \in \mathrm{P}.vertices\}$
3: **for all** $o \in candidates$ **do**
4: **if** o is an Occlusion Point **then**
5: $i_{sec} \leftarrow$ castRay(P, o, $o - s$)
6: $occData \leftarrow occData \cup \langle o, i_{sec} \rangle$
7: **end if**
8: **end for**

Output: Occlusion Data $occData$

Algorithm 2 propagateOcclusion(P, s, G)

Input: current Node P, Starting Point s, Visibility Graph $G = \langle V, E \rangle$

1: $occData \leftarrow$ calculateOcclusions(P, s, G) /* see Algorithm 1 */
2: **if** $|occData| > 0$ **then**
3: $occPath \leftarrow$ calculateOccPath(P, $occData$)
4: $\langle visibilityPatch, occlusionPatches \rangle \leftarrow$ splitTree(P, s, $occPath$)
 /* see Algorithm 3 */
5: **for all p** $\in occlusionPatches$ **do**
6: **for all r** $\in$ **p**$.refPoints$ **do**
7: **p**$.children \leftarrow$ propagateOcclusion(**p**, **r**, G)
8: **end for**
9: **end for**
10: $P.children \leftarrow \{visibilityPatch\} \cup occlusionPatches$
11: **end if**

Output: new Children of the Patch $P.children$

corresponding occlusion data elements are stored as reference points. From these points on, the occlusion has to be propagated, which is done recursively in Algorithm 2. This propagation uses the associated occlusion patch as a shape, therefore no occlusion point is used multiple times by the same path.

3.2 Splitting the Partition

For each call to splitTree (Algorithm 3) first the visibility patch and all occlusion patches of the shape are calculated. When calculating the occlusion patches, the occlusion points stored in $occPath$ are stored with the associated patches to be able to continue the occlusion propagation (see Section 3.1).

In the next step all of the patches in the hierarchy are cut along $occPath$ and then assigned to the visibility or one of the occlusion patches (1st level patches), building a subtree for each of these 1st level patches. This is done by the call to the recursive function "splitSubTree". Which splits each the specified node along $occPath$, creating visibility and occlusion subpatches and repeats this process for each child nodes, attaching the result to the associated subpatches. It also adds s as a reference point with handicap 0 to all visibility patches. If a patch does not intersect $occPath$, it has to be calculated which of the 1st level patches contains it. The result of this algorithm is the same as by intersecting the 1st level patches with each patch of the hierarchy.

3.3 Calculating the Partition

To compute the partition for the ECC, an occlusion propagation (as explained in Section 2) is done for each starting point. Algorithm 4 accumulates multiple SSGD partitions to compute an ECC partition. Instead of manually specifying starting points, they can also be automatically determined (e.g. by using the corners of the polygon, which have a high probability of being eccentric points).

In this algorithm the visibility graph is computed for all vertices and starting points of the shape. This needs to be done only once and will be useful for determining occlusion points (see Section 3.1).

A naive algorithm for this is to check the visibility for each two vertices of the graph and add an edge if the vertices are visible to each other.

Algorithm 3 splitTree(P, s, $occPath$)

Input: polygonal Shape P, Starting Point s, Occlusion Path $occPath$

$visPatch \leftarrow$ calculateVisibilityPatch(P, $occPath$)
$occPatches \leftarrow$ calculateOcclusionPatches(P, $occPath$)
if $|P.children| > 0$ **then**
  for all c $\in P.children$ **do**
    $\langle visSub, occSub \rangle \leftarrow$ splitSubTree(**c**, s, $occPath$)
    /* splitSubTree recursively splits a Subtree along $occPath$ */
    $visPatch.children \leftarrow visPatch.children \cup visSub$
    for $i = 1$ to $|occPatches|$ **do**
      $occPatches[i].children \leftarrow occPatches[i].children \cup occSub[i]$
    end for
  end for
else /* $|P.children|$ is 0 */
  $visPatch.refPoints \leftarrow visPatch.refPoints \cup \langle s, 0 \rangle$
end if

Output: visibility Patch $visPatch$, occlusion Patches $occPatches$

Algorithm 4 createPartitions($\mathcal{S}$, $startPoints$)

Input: polygonal Shape $\mathcal{S}$, Starting Points $startPoints$

$G \leftarrow$ calculateVisibilityGraph($\mathcal{S}$, $\mathcal{S}.vertices \cup startPoints$)
initialize $\mathcal{P}$ as a data structure with the following elements:
$\mathcal{P}.vertices \leftarrow \mathcal{S}.vertices$
$\mathcal{P}.indices \leftarrow \mathcal{S}.indices$
$\mathcal{P}.children \leftarrow \{\}$
$\mathcal{P}.refPoints \leftarrow \{\}$
$\mathcal{P}.startPoints \leftarrow startPoints$
for all s $\in startPoints$ **do**
  $\mathcal{P}.children \leftarrow$ propagateOcclusion($\mathcal{P}$, **s**, G) /* see Algorithm 2 */
end for

Output: Partition $\mathcal{P}$

3.4 Optimizing the Hierarchy

After each occlusion propagation, the hierarchy can be modified without causing inconsistencies. For example nodes, which only have one child, can be eliminated as well as reference points, which cannot be part of the shortest path for a starting point.

This can be done by calculating the path lengths from each starting point to each of the patch vertices $\mathcal{V}$ using each of the reference points $\mathcal{R}$, i.e.

$$d_v(r_i) = d(v, r_i) + h_i \qquad v \in \mathcal{V}, r_i \in \mathcal{R}, \tag{9}$$

where h_i represents the handicap of the reference point r_i.

For each starting point, if there is a reference point, which has a path length bigger than the path lengths of at least one other reference point with the same starting point for each vertex, it can be removed. That means it can be removed if the following holds:

$$\forall v \in \mathcal{V} : \exists r_j \in \mathcal{R}_s : d_v(r_i) < d_v(r_j) \qquad r_j \in \mathcal{R}_s, s \in \mathcal{E} \tag{10}$$

where $\mathcal{E}$ is the set of known eccentric points (starting points) and $\mathcal{R}_s$ is the set of reference points, which are derived from the starting point s.

Further removal of reference points can be done when the patches are subdivided along the decision boundaries defined by every two reference points with the same starting point.

3.5 Computing the Eccentricity

To compute the eccentricity, first the leaf node in the hierarchy, which contains the specified point, has to be determined (see Algorithm 5) and then the reference points of this node have to be used as described in Section 2 (see Algorithm 6).

Algorithm 5 getPatch($\mathcal{P}$, p)

Input: Partition $\mathcal{P}$, query Point p

$Patch \leftarrow \mathcal{P}$
$i \leftarrow 1$
while $i \leq |Patch.children|$ **do**
  if $p \in Patch.children[i]$ **then**
    $Patch \leftarrow c$
    $i \leftarrow 1$
  end if
end while

Output: patch $Patch$

Algorithm 6 getEccentricity($\mathcal{P}$, p)

Input: Partition $\mathcal{P}$, query Point p

$patch \leftarrow$ getPatch($\mathcal{P}$, p) /* see Algorithm 5 */
$ecc \leftarrow 0$
for all $\mathbf{s} \in \mathcal{P}.startPoints$ **do**
  if $\exists r : r \in patch.refPoints \wedge r.startPoint == \mathbf{s}$ **then**
    $dist \leftarrow \infty$
    for all $\{\mathbf{r} \in patch.refPoints | \mathbf{r}.startPoint == \mathbf{s}\}$ **do**
      $newdist \leftarrow d(\mathbf{r}, p) + \mathbf{r}.handicap$
      if $newdist < dist$ **then**
        $dist \leftarrow newdist$
      end if
    end for
    $ecc \leftarrow max\{ecc, dist\}$
  end if
end for

Output: eccentricity ecc

4 Numerical Issues

When splitting the hierarchy along an occlusion path, each patch can either intersect one or more occlusion lines or it can be fully inside the visibility or one of the occlusion patches. If it intersects with occlusion lines it has to be split, which is done by calculating the intersections of the rings of the patch with the occlusion lines. Because the computer is limited to a certain accuracy when calculating these intersections, a numerical error can appear, resulting in inconsistencies in the hierarchy. Using another data structure, such inconsistencies can be avoided. For example this can be done by replacing the edge $\langle v_1, v_2 \rangle$, which has to be intersected at the point i_{sec} with two edges $\langle v_1, i_{sec} \rangle$ and $\langle i_{sec}, v_2 \rangle$.

In the following sections two problems due to numerical errors are discussed, which occur when using the originally defined data structure.

4.1 Intersections inside the Parent Patch

The following example illustrates this case:

Example 4.1 *Patch P is to be split into two patches A and B. In this example P, A and B are interpreted as regions (set of points). The following holds, if the calculation is exact:*

$$P = A \cup B \tag{11}$$

$$A \cap B = \emptyset \tag{12}$$

Because of the numerical error, this may not be the case. It can happen, that a point is neither in A nor in B (see also Figure 4.1):

$$\exists p : (p \in P) \wedge (p \notin A \cup B) \tag{13}$$

When determining the patch for a point, this can be troublesome, since it may be inside one patch in the hierarchy but not in any of its children. A solution to this is to detect if the point is in none of the subpatches and calculate the closest patch to the point. This can be done by calculating the distance to all line segments of the polygonal patch and determining the minimum (see Algorithm 7).

Algorithm 7 findClosestPatch($Children$, P)

Input: subpatches $Children$, specified point P

```
1: i ← 0 /* index of the closest patch */
2: j ← 1 /* counter */
3: while (i == 0) ∧ j ≤ |Children| do
4:    if P ∈ Children[j] then
5:       i ← j /* closest patch found */
6:    end if
7: end while
8: if i == 0 then
9:    d ← ∞ /* smallest distance */
10:   for j = 1 to |Children| do
11:      d′ ← getDistance(P, Children[j])
12:      if d′ < d then /* patch is closer than the stored one */
13:         i ← j
14:         d ← d′
15:      end if
16:   end for
17: end if
```

Output: closest patch $Patches[i]$

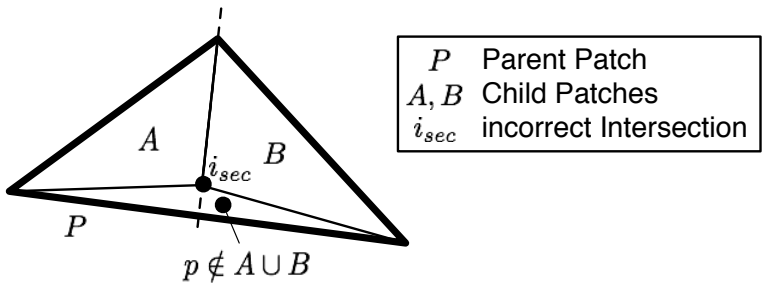


Figure 4.1: Subpatches, which don't fill the parent patch out

4.2 Intersections outside the Parent Patch

Another issue due to numerical errors is, that a subpatch might not be fully within the parent patch.

Example 4.2 *Consider a patch P and one of its child patches A.*
The following holds, if the calculation is exact:

$$A \setminus P = \emptyset \tag{14}$$

Because of the numerical error, this may not be the case. It can happen, that a point is in A but not in P (see also Figure 4.2):

$$\exists p : (p \in A) \wedge (p \notin P) \tag{15}$$

This can be problematic when determining whether a patch A is contained in a patch B or not (as necessary in Section 3.2). The naive algorithm for this would be to check for each vertex of A if it is inside B. Because of the problem mentioned above it can be that one vertex of A is not inside B. Then there should be an intersection, while there was no intersection calculated (since this only works by cutting the patch along an occlusion path). It is sufficient to calculate any point inside A (preferably with a big distance to the border) and check if this point is in B. Algorithm 8 shows how this can be done. It works by first triangulating the polygonal shape and then calculating the incircles of each triangle. The center of the biggest incircle is returned as a point inside the patch. This way the point is guaranteed to be inside the patch and it is not too close to the border, if the patch is not very flat.

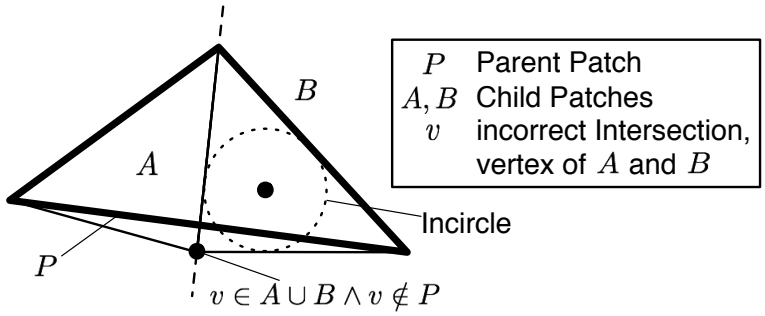


Figure 4.2: Subpatches, which are bigger than the parent patch

Algorithm 8 getPointInPatch(P)

Input: specified patch P

```
1: radius ← 0
2: T ← triangulate(P)
3: for all t ∈ T do
4:    ⟨c, r⟩ ← calculateInCircle(t)
      /* c represents the center and r the radius */
5:    if r > radius then
6:       radius ← r
7:       point ← c
8:    end if
9: end for
```

Output: a point inside the patch *point*

4.3 Other Numerical Issues

There can still be additional inconsistencies when one of the requirements to the shape (as defined in Section 1.2) is broken for any of the subpatches, because of numerical errors. Since these requirements are assumed for the whole program, cases where this happens can be detected and reported.

To keep such issues to a minimum, before calculating anything the whole shape is translated, so that the center of its bounding box is located at $(0, 0)$ (see [Thorne06]). This way, it is prevented, that the program has to use high values which are close together for its calculation in cases where all vertices have high values, e.g. the shape is defined by longitude and latitude as

coordinates. Without the alignment in such a case, the numerical error would be much higher, since it depends on the range of the value. After everything is calculated, the shape is relocated to it's original position, preserving the range of the input vertices.

In addition an error tolerance is used for computations, which are sensitive to numerical errors, e.g. when determining if a point lies on a line segment.

5 Evaluation

This section shows the polygonal ECC on more complex examples and compares the results with a discrete ECC. Figure 5.1 shows a simple manually defined example of a non-convex shape, its partitioning and a sampling of the associated ECC function (illustrated as gray-scale image). The image of the figure shows the modulo of the sampled function to illustrate contour lines as well as the direction of descent. When the pixels along a line are getting darker, the ECC function is decreasing along this line.

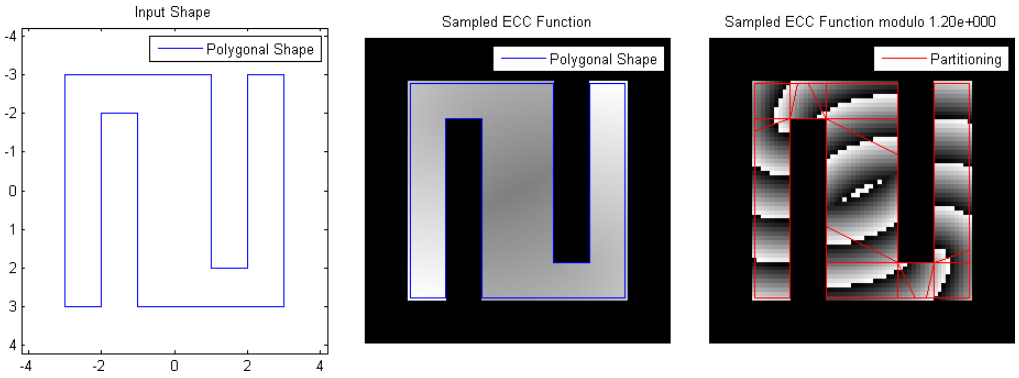


Figure 5.1: Partitioning and sampling of the ECC function of a simple shape

5.1 Comparison to the discrete ECC

The reference program implements the wave propagation using Fast Marching [Sethian99a] and a first order approximation of the differential equation used to model the wave front.

Figure 5.2 compares the sampled polygonal ECC function with the discrete ECC and shows slight differences, which occur due to the approximation of the wave front in the reference program. This shape was especially designed to prevent discretization errors, which occur when a shape, with line segments, which are not axis oriented is discretized (which is necessary for the reference program to work, since it uses a digital input shape).

For this shape, the minimal difference between the discrete and the polygonal ECC is 0 and the maximal difference is 0.25. Figure 5.3 shows the comparison of the histogram of the sampled ECC.

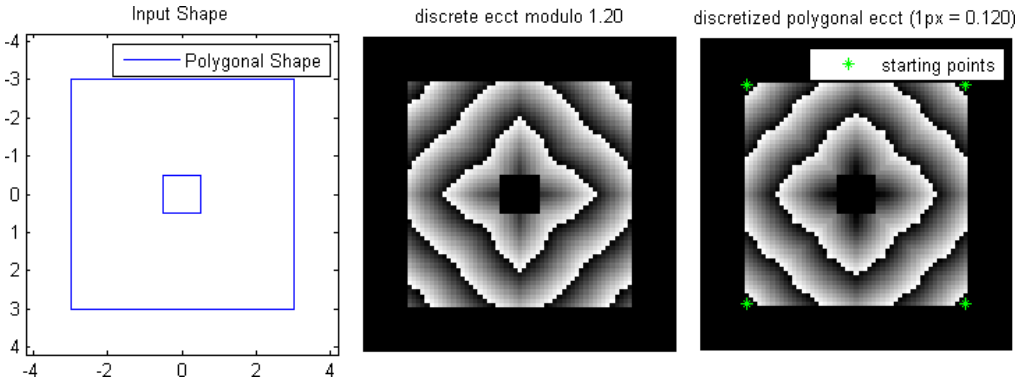


Figure 5.2: comparison of the discrete ECC with the sampled polygonal ECC function

5.2 Selection of Starting Points

Usually the implementation calculates each corner of the outer ring of the shape, in which the shape is convex and uses them as starting points. For shapes without holes or with only few small holes (as in the previous examples), this is an exact solution, however if the shape has bigger holes, eccentric points may lie on the line segments of the outer ring, instead of its corners, in some cases it is even possible, that eccentric points lie inside the shape (see also [IonPhD09, p.33]). Figure 5.4 shows an example for a shape with eccentric points, which do not lie on the corner. The reference program was configured to use all border points as starting points and therefore has a result, which is much more accurate.

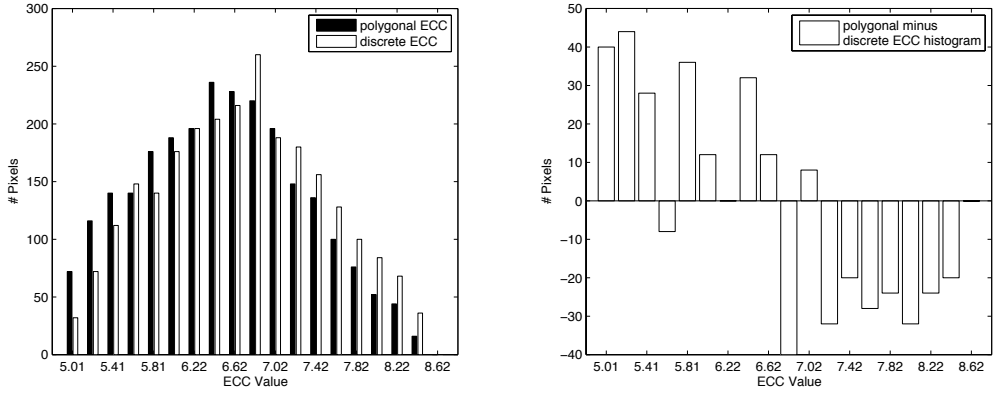


Figure 5.3: Comparison of the histogram of the discrete and sampled polygonal ECC function

The minimal difference in this case is 0 and the maximal difference is 8.73. Figure 5.5 shows the comparison of the histogram of the sampled ECC.

Since the program can use additionally defined starting points (for example in a fixed interval along the outer ring), the result can be improved, which is done in Figure 5.6. Additional starting points are added on the bottom edge, making the result in the top half more accurate.

The minimal and maximal differences are the same as without the additional starting points, because the errors in the bottom half of the shape are still as big as before.

Note that because of inconsistencies due to numerical errors the program may return an error and terminate.

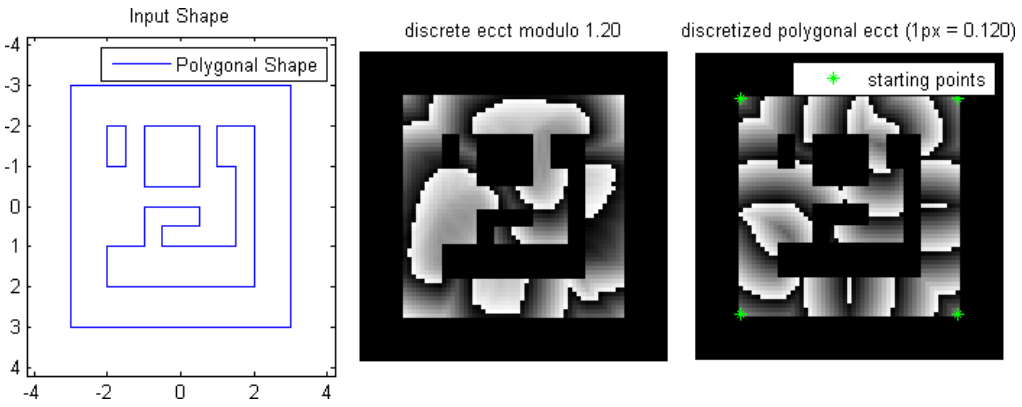


Figure 5.4: Comparison of a shape with eccentric points, which are not in the corners

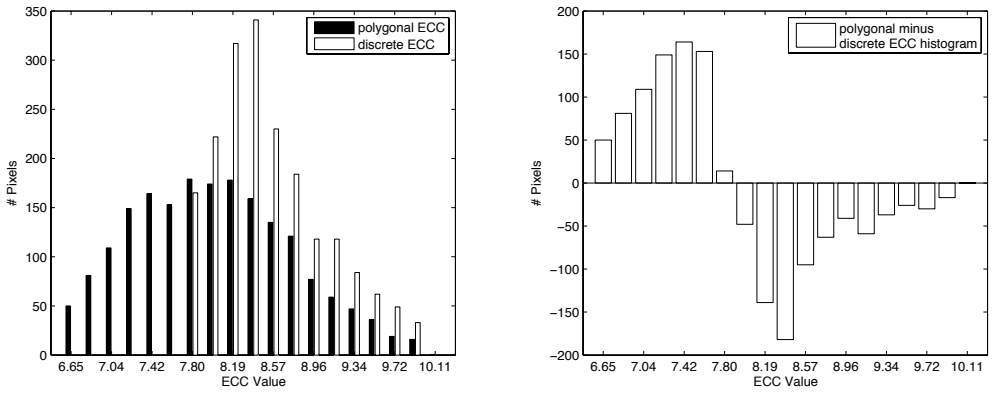


Figure 5.5: Comparison of the histogram of the discrete and sampled polygonal ECC function of Figure 5.4

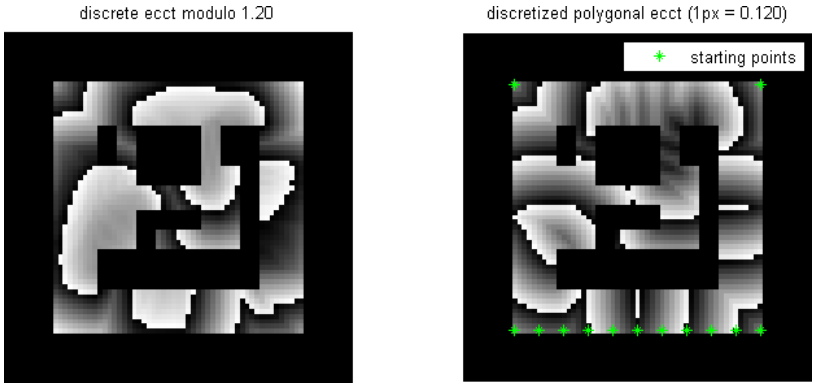


Figure 5.6: Comparison of a shape with eccentric points, which are not in the corners, using additional starting points. A better solution is obtained but as not all eccentric points were included, errors still exist.

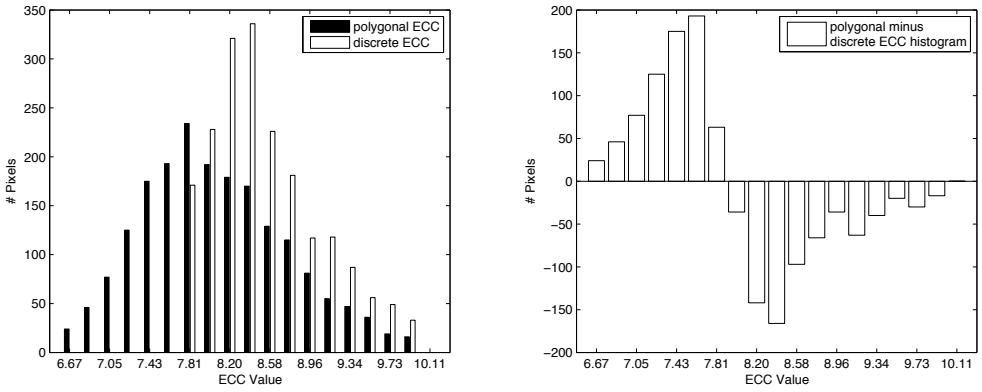


Figure 5.7: Comparison of the histogram of the discrete and sampled polygonal ECC function of Figure 5.6

5.3 Error Tolerance

In the current implementation the value of the error tolerance mentioned in Section 4 is determined after the translation of the shape to the origin. The tolerance t equals four times the machine accuracy of the maximal vertex coordinate:

$$t = 4 * eps(\max[(\max_i x_i), (\max_i y_i)]), \tag{16}$$

where $eps(x)$ returns the machine accuracy of the provided value x. The program is designed so that the error tolerance can be changed from the outside (by providing an additional parameter).

The tolerance can also be set to 0, which means that two values (e.g. two vertices) will only be considered equal, if the difference is exactly 0.

This works with some simple cases, but creates inconsistencies in more complex shapes, because with each intersection the slope of a splitted edge changes (since the calculated intersection has a numerical error) the total error increases. In a bigger hierarchy, edges are splitted more often than in a smaller hierarchy, therefore the total error is further increased causing inconsistencies, which are not compensated due to a tolerance of 0.

The upper limit of the tolerance in the tested shapes is about $x * 10^{-2}$ ($x \in [1, 10)$) in comparison to about $3 * 10^{-15}$) which is calculated like above.

5.4 Complexity

The complexity depends on the number of vertices n_v of the input shape, the number of holes n_h in the input shape and the number of starting points n_s. The number of reference points n_r per patch has an upper limit of $n_s 2^{n_h}$, so the complexity of finding the shortest path using the reference points is $O(n_s 2^{n_h})$. Note that this is the worst case scenario, where each hole lies fully inside the occlusion patch of another hole (except the first) for all starting and previous reference points. The number of edges in any patch of the hierarchy is bounded by $O(n_v)$.

The hierarchy can be degenerated in such a way, that each patch in the hierarchy has to be checked in order to find the leaf node for a given point. The number of patches in the hierarchy is bounded by $O(n_s n_v)$, since each occlusion point creates a patch. In order to check wether a point is inside a patch or not, data has to be calculated for each edge of the patch (i.e. if the edge intersects a ray cast from the point into a certain direction). So

the worst case complexity for finding the patch to a given point is $O(n_s n_v^2)$. Note that the worst case complexity is the same if the partition is stored as a set of patches instead of a hierarchy, but this is a very special case for the hierarchy. In general cases there is improvement. The best case complexity for finding the patch in the hierarchy is bounded by $\Omega(n_v)$, since the best case is a convex polygon, where no occlusions occur and the hierarchy only contains the root node.

The function getEccentricity first finds the patch for a given point and then finds the shortest path inside the patch, so the worst case complexity for this function is bounded by $O(n_s n_v^2 + n_s 2^{n_h})$. The best case complexity is $\Omega(n_v + n_s)$ since n_s is the number of reference points for the whole shape.

With a call to splitTree the whole hierarchy is split along an occlusion path and therefore the complexity is bounded by the number of patches in the hierarchy multiplied by the number of edges per patch. So the function has the worst case complexity of $O(n_s n_v^2)$.

The function calculateOcclusions casts a ray for each occlusion point candidate (which number is bounded by $O(n_v)$, i.e. it is determined if a ray intersects any of the edges of the patch. So it has a complexity bounded by $O(n_v^2)$.

The complexity of propagateOcclusions depends on the total number of calls in the recursion, which is bounded by the number of occlusion points in the shape, since the propagation is only done once for each reference point. Therefore the number of calls in the recursion is bounded by $O(n_v)$. Each call to this function also involves calculateOcclusions and splitTree, resulting in a complexity bounded by $O(n_v(n_s n_v^2 + n_v^2)) = O(n_v^3(1 + n_s))$.

The function createPartitions calls propagateOcclusions for each starting point and therefore has a complexity of $O(n_v^3 n_s(1 + n_s))$.

6 Conclusion

This thesis presents an algorithm to compute the ECC of a polygonal shape based on the method described in [Kropatsch07b]. Holes can be present in the shape. During the process of determining a general formula of a hyperbola, the information about the branch relevant for the algorithm is lost. To cope with this a refinement of the algorithm is presented, which calculates the ECC for a polygonal shape without dealing with hyperbolas.

The output of the algorithm is a hierarchy of patches, whose leaf nodes represent the patches of the full partition. In this hierarchy reference points are stored along with a computed handicap such that the ECC of any point can be computed by computing an Euclidean distance and adding a handicap.

Using the computed decomposition the exact ECC values (to the limit of computer precision) for any point of the shape can be computed efficiently if all eccentric points are used as starting points for the algorithm. If not all eccentric points are given as starting points, the program calculates an approximation of the values.

This thesis also discusses the numerical issues, which occur along with the computation of the patches, and presents solutions for two specified problems associated with the numerical errors.

In the following sections open problems and ideas of further improvement of the algorithm are discussed and the personal experience gained during the work is described.

6.1 Number of Eccentric Points

As mentioned in Section 1.2, if all eccentric points are known (and their number is finite), the eccentricity transform can be calculated accurately, only depending on numerical errors of the distance calculations. This error sums up for each distance calculation, i.e. for each line segment in the shortest path from the eccentric point to the query point. The maximal error is nd where n is the maximal number of line segments of the shortest path (this depends on the shape) and d is the error of the distance calculation. The euclidean distance between two points $a = (a_x, a_y)$ and $b = (b_x, b_y)$ is calculated as follows:

$$d(a,b) = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{17}$$

Assuming the approximation of the square root is accurate enough, the accuracy of the distance calculation depends on the machine accuracy of the provided numbers.

The probability of an infinite number of eccentric points is higher in a shape with big holes. An open question is if and how it is possible to modify the algorithm in this paper to use "eccentric lines" (an infinite number of eccentric points, which lie on the same line segment) instead of just eccentric points.

6.2 Further Improvements of the Algorithm

Occlusion Propagation Stop: Since the propagation is fully done behind holes as well, this may unnecessarily split patches to add reference points, which have no influence on the result (because their handicap is already to big to be part of a shortest path). So the propagation may be stopped, for example when a reference point is reached twice. This presumes a processing order of occlusion points by their handicap. When doing this the occlusion propagation becomes much more complex, since the reference points have to be stored in a list for each level of the propagation and for each occlusion patch, the propagation has to be done step by step ordered by the handicap of the new reference points.

Simple Hierarchical Subdivision of Patches: Patches which have more than one reference point can be further divided as explained in Section 2. Instead of a Voronoi diagram a simple hierarchical subdivision is possible by first determining reference points with the same starting point and 'splitting' the patch along the hyperbola defined by two of these points. The patch does not really need to be split, instead it is sufficient to determine on which side the point lies, by calculating the length of the paths. If this is done using a balanced tree, only $O(\log n)$ instead of $O(n)$ paths have to be calculated [Atallah98].

Sampling of the ECC Function: If the function is sampled as in Section 5, there are additional improvements, which can make the overall calculation faster by a factor $i > 1, i \in \mathbb{N}$. Instead of determining the patch for every point, a scanline algorithm can be used to just compute the ECC for each point without needing to care for patches. if this is

done using the voronoi cells described in Section 2, each patch only has one reference point left and therefore. The hierarchy is not needed in such an implementation, since the determination of the patch for each point is not explicitly necessary anymore. This method calculates the eccentricity with the same accuracy as the unmodified algorithm in this paper. The calculation can also be done using methods of computer graphics (see [Hoff99]), literally rendering the result (and use common graphics hardware in the process), because all the types of voronoi diagrams presented in Section 2 can be rendered by using cones for each reference point r_i, which apex lies on the point $(r_{i,x}, r_{i,y}, -h_i)$ and the base circle is located at $(r_{i,x}, r_{i,y}, \infty)$ with an infinity radius. To render this, it is sufficient, that the patch is fully inside of all these base circles. To render the additive weighted or furthest point additive weighted voronoi diagram this scene needs to be rendered from above or below, looking in the direction of the cones. Looking at it from below results in an additive weighted voronoi diagram.

6.3 Personal Experience

The complexity of such a geometric focused program as well as the mistakes made while thinking of seemingly simple problems were underestimated. Most of these pseudo-simple problems were first solved intuitively, directly tested with cases which seemed representative, but it happened quite often that there are cases which are not considered, especially when looking at it globally.

For example the determination of occlusion points in Section 3.1 was first locally solved, but there is one case where we do not want an occlusion candidate to be an occlusion point even if it satisfies the conditions of the orientation of its adjacent edges: if there are two occlusion points on the same ray (i.e. s, o_1 and o_2 lie on a straight line) and the adjacent edges of o_1 point in the other direction as the edges of o_2 then the second occlusion (the one which is further away from s) should not be an occlusion point, because the ray intersection of the first occlusion will be the second occlusion point, creating an occlusion patch for both occlusions at the same time (see also Figure 6.1)

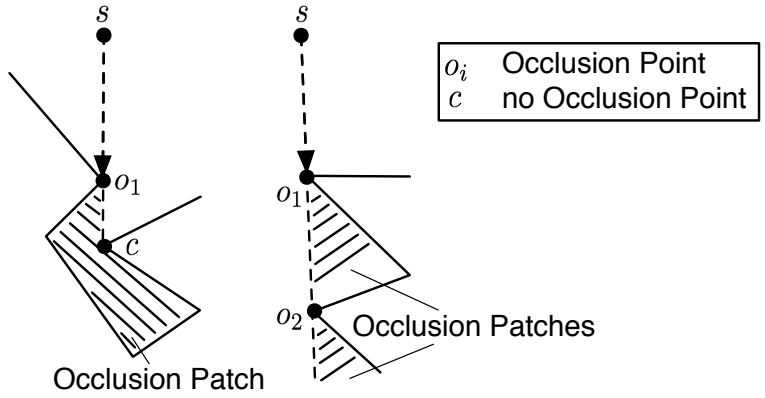


Figure 6.1: Example of an occlusion candidate, which would be considered an occlusion point if the check is made locally

Literature

Aranov88. B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic voronoi diagram. In *Proceedings of the fourth annual symposium on Computational geometry*, pages 229–240, New York, NY, USA, November 1988. ACM.

Atallah98. M. J. Atallah and S. Fox, editors. *Algorithms and theory of computation handbook*, volume 1st edition. CRC Press, Inc., 1998.

Buckley90a. F. Buckley and F. Harary. *Distances in Graphs.* Addison-Wesley Publishing Company, 1990.

Cardenes03. R. Cardenes, S. Watfield, E. Macias, and J. Ruiz Alzolar. Occlusion points propagation geodesic distance transformation. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1, pages 361–364, September 2003.

Hoff99. K. E. Hoff, III, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized voronoi diagrams using graphics hardware, 1999.

IonPhD09. A. Ion. *The Eccentricity Transform of n-Dimensional Shapes with and without Boundary*. PhD thesis, Vienna University of Technology, Faculty of Informatics, 2009.

Ion08a. A. Ion, W. G. Kropatsch, and E. Andres. Euclidean eccentricity transform by discrete arc paving. In D. Coeurjolly, I. Sivignon, L. Tougne, and F. Dupont, editors, *14th IAPR International Conference on Discrete Geometry for Computer Imagery (DGCI)*, volume LNCS 4992 of *Lecture Notes in Computer Science*, pages 213–224, Lyon, France, April 2008. Springer.

Ion07b. A. Ion, S. Peltier, Y. Haxhimusa, and W. G. Kropatsch. Decomposition for efficient eccentricity transform of convex shapes. In W. G. Kropatsch, M. Kampel, and A. Hanbury, editors, *The 12th International Conference on Computer Analysis of Images and Patterns (CAIP)*, volume 4673 of *Lecture Notes in Computer Science*, pages 653–660, Vienna, Austria, August 2007. Springer.

Kropatsch06a. W. G. Kropatsch, A. Ion, Y. Haxhimusa, and T. Flanitzer. The eccentricity transform (of a digital shape). In *13th International Conference on Discrete Geometry for Computer Imagery (DGCI)*, pages 437–448, Szeged, Hungary, 25–27, October 2006. Springer.

Kropatsch07b. W. G. Kropatsch, A. Ion, and S. Peltier. Computing the eccentricity transform of a polygonal shape. In L. Rueda, D. Mery, and J. Kittler, editors, *12th Iberoamerican Congress on Pattern Recognition (CIARP 2007)*, volume LNCS 4756 of *Lecture Notes in Computer Science*, pages 291–300, Viña del Mar / Valparaiso, Chile, November 2007. Springer.

Sethian99a. J. A. Sethian. *Level Sets Methods and Fast Marching Methods*. Cambridge Univ. Press, 2nd edition, 1999.

Soille02a. P. Soille. *Morphological Image Analysis*. Springer, 2nd edition, 2002.

Thorne06. C. Thorne. Error minimising pipeline for hi-fidelity, scalable geospatial simulation. In *CW '06: Proceedings of the 2006 International Conference on Cyberworlds*, Washington, DC, USA, 2006. IEEE Computer Society.

Using Shape Particle Filters for Robust Medical Image Segmentation

Lukas Fischer
Supervisor: Robert Sablatnig

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: lukas.fischer@student.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

Scientific Report Vienna June 9, 2009

Using Shape Particle Filters for Robust Medical Image Segmentation

Lukas Fischer

Supervised by Prof. Sablatnig

Abstract

In recent years segmentation approaches based on sequential Monte Carlo Methods delivered promising results for the localization and delineation of anatomical structures in medical images. Also known as Shape Particle Filters, they were used for the segmentation of human vertebrae, lungs and hearts, being especially well suited to cope with the high levels of noise encountered in MR data and difficult overlaps in radiographs. This report surveys the robustness of these methods on different medical example images. A Differential Evolution approach on Shape Particle Filtering is applied for image segmentation. The goal of this work is to analyze the behavior, e.g. the robustness of the implemented Shape Particle Filter. Results on different data (synthetic rectangles, MRI slices and radiographs) are reported.

1 Introduction

In this report the efforts to implement shape particle filters proposed by Marleen de Bruijne [11, 10] for the detection and segmentation of objects of interest in medical images are documented. The data sources were metacarpal bone radiographs, MRI slices of the heart and lateral spine radiographs provided within the scope of the COBAQUO Project (Austrian National Bank Fond project Computer Based Quantification of Osteoporosis and Bone Alignment, MU Vienna, TU Graz).

1.1 Motivation

The particle filter was introduced by Gordon et al. in 1993 with the intention to implement recursive Bayesian filters [13]. It is also known as Sampling Importance Resampling (SIR) or Bayesian bootstrap filter. Particle filters are also known as *sequential Monte Carlo Methods*.

Compared to other filters that use Monte Carlo Methods to get estimates of the mean and covariance of the posterior, particle filters approximate the complete posterior. Particle filters approximate posterior densities using swarms of points (so called *particles*) in a sample space. By assigning a weight to each particle and using a discrete distribution of the particles the posterior distribution can be approximated. This results in particle probabilities which are proportional to the particle weights. Several algorithms exist differing mainly by the way how the particle swarms evolve and adapt to input data [12].

Shape Particle Filter Marleen de Bruijne used a modification of the particle filters mentioned above for the segmentation of medical images e.g. vertebræ, lungs and hearts [11, 10]. Her proposed *Shape Particle Filter* implementation contains the following steps. First a global shape and a local appearance model are derived from hand annotated example images. Then a unique labeling of the image into two or more classes is applied to each shape. For classification a pixel classifier is trained using local image descriptors (e.g. gaussian derivative filters). The applied pixel classifier is a modification of the k-NN classifier, proposed as *moderate k-NN*. To classify a new image, the label probability maps for the test image are computed. In the final step the

optimal solution is obtained using particle filtering. This is done by generating a random set of shapes, the particles, from the prior shape model. The image labeling associated to the particles is compared with the label probability maps obtained in the classification step and a weight is applied to each particle considering their likelihood. New particles are randomly generated from the current set of particles. While this importance resampling process is repeated the initial sparse particles evolve into a distribution with high density around the most likely shapes. By clustering the particle distribution using the mean shift algorithm and selecting the strongest local mode the optimal fit is obtained.

Medical Background This paragraph briefly describes the medical background of this technical report on the example of the lumbar vertebræ. By manually examining radiographs of the human spine, radiologists can draw conclusions about degenerative vertebræ deformations or pathological altered vertebræ. Such pathological conditions could be exostosis, vertebræ fractures in consequence of osteoporosis, kyphosis or excessive lordosis. This technical report and the implemented image segmentation algorithm concentrates on the five lower vertebræ, the lumbar vertebræ (shown in Figure 1).

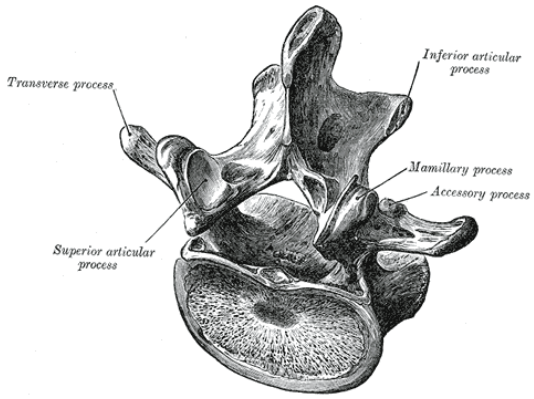


Figure 1: Lumbar vertebra (Henry Gray (1825-1861). Anatomy of the Human Body. 1918.)

Computerized medical imaging is a ever growing field of research providing discriminative techniques for image segmentation and shape detection.

Worth mentioning hereunto are Active Shape Models (ASMs)[7] and their direct successor Active Appearance Models (AAMs)[8]. In contrast to particle filter methods these approaches find a single solution in the shape / appearance model space using heuristic optimization steps or gradient decent methods.

2 Methods

This section covers the fundamental methods used in the scope of this technical report. A brief overview of the overall context and the order in which these methods were applied is given, before detailing the specific methods. A flowchart of the proposed Shape Particle Filter approaches outline is shown in Figure 2.

First the object of interest in the medical image is manually annotated by a human, forming the outline of the object e.g. a lumbar vertebra, a heart or a metacarpal bone. A shape is defined as a finite set of landmarks on this outline. In order to obtain a usable shape (with a feasible number of landmarks) of the annotated object, Minimum Description Length (MDL, Section 2.1) is applied. By aligning all the training shapes and reducing their dimensionality by applying Principle Component Analysis (PCA, Section 2.2) a Shape Model is generated (see Section 2.3).

2.1 Minimum Description Length

Jorma Rissanen first introduced the principle of Minimal Description Length (MDL) in 1989. It is based on the work of Solomonoff on the *Kolmogorov* or *algorithmic complexity*. MDL is a method for inductive inference and is related t o *Bayesian Inference* and the *Minimum Message Length* (MML) principle.

The central idea of MDL is that on the one hand by compressing data every regularity in the data can be used and that on the other hand with finding regularities in data learning can be equated. In other words the goal is to learn about the data by compressing it. Formally by applying MLD we try to find the hypothesis or a combination of hypotheses in a given set of hypotheses $\mathcal{H}$ that compress a given data set D most.

MDL implements *Occam's Razor* which main statement is that the best results can be achieved by using simple solutions. Therefore if MDL has to

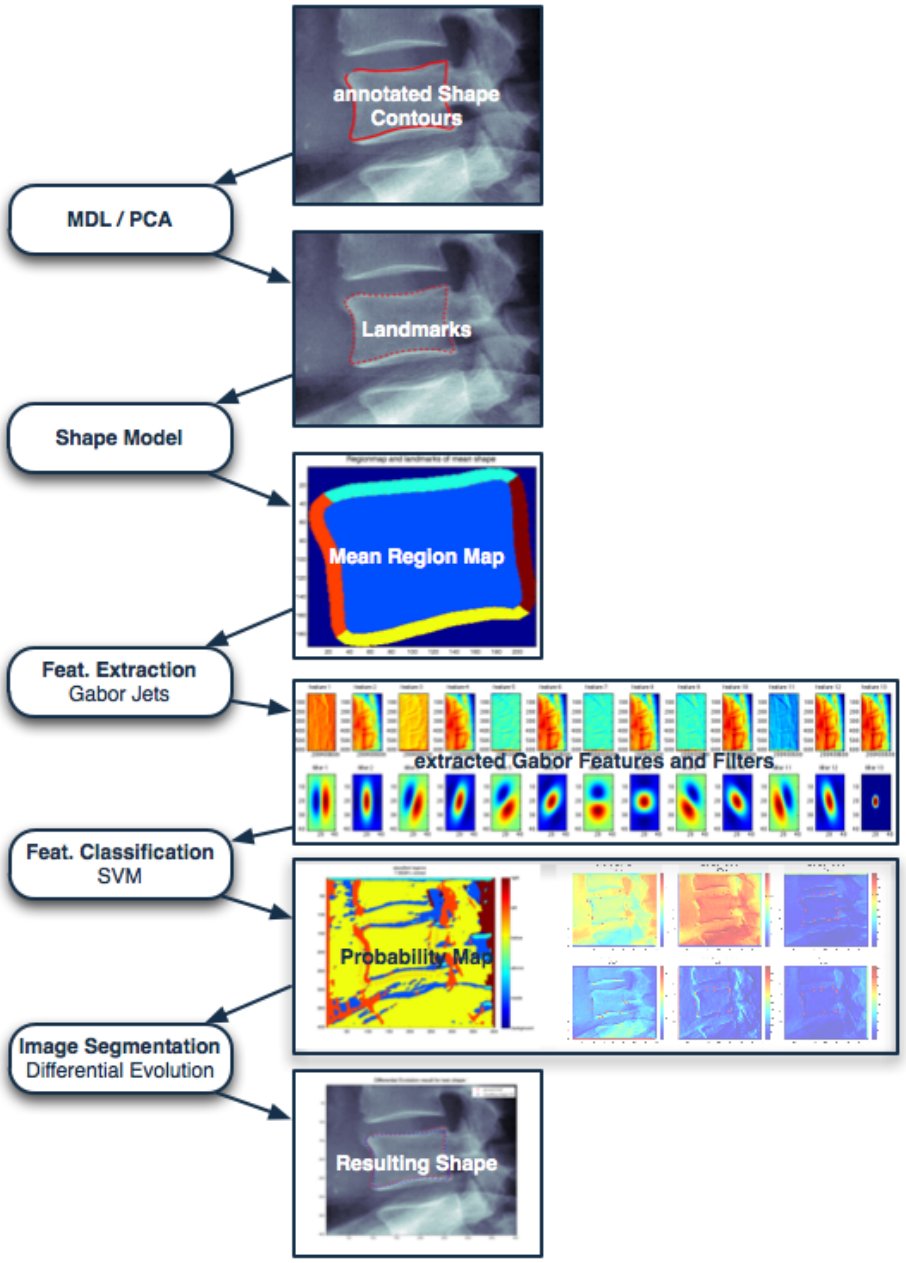


Figure 2: Flowchart of the proposed Shape Particle Filter approach.

choose between two models that fit the data equally well, it will choose the one that allows the shorter description of the data. [14]

To formulate a theoretical ideal MDL that suits the main idea of learning by compressing data, a data sequence and a description method to express the data properties is needed. A general choice for this description method could be a general-purpose computer language. Considering the *Kolmogorov Complexity*, which is the length of the shortest program that maps the data sequence, it can be shown that selection of the computer language does not matter because the lengths of two programs written in different languages differ only on a constant, which in turn does not depend on the length of the data sequence as long as it is sufficient long. This fact is known as *invariance theorem*. Because of the uncomputability, arbitrariness and dependence on the chosen syntax the above described ideal MDL can not be used in practice. [14]

To transform an ideal MDL to a practical MDL, description methods that are less expressive than general-purpose computer languages have to be used. These description methods should fulfill two constraints. First they should always allow to compute the shortest descriptions length and second should be general enough to compress most of the regular sequences. The practical MDL can be formulated using two different approaches, the *crude* or *two-part code* version and the *refined* or *one-part code version*. [14]

The crude version of the MDL principle achieves the best hypothesis H that best describes the data D by minimizing the sum $L(H)+L(D|H)$, where $L(H)$ is the hypothesis' description length and $L(D|H)$ is the length of the description of the data encoded with the hypothesis. The problem with crude MDL is that the description length of a hypothesis can differ largely when using different description methods (codes) and therefore the procedure is in danger to become arbitrary.

To overcome this problem the crude MDL was refined to the refined MDL. In difference to the crude version, where a single Hypothesis $H \in \mathcal{H}$ was intended to find for encoding, the refined MDL aims to find a full model $\mathcal{H}$ of hypotheses for encoding the data. Another difference is that according to its denotation the *one-part code* uses only a single one-part code with lengths $\overline{L}(D|\mathcal{H})$ instead of two parts. This assures that whenever $L(D|H)$ is small ($\exists H \in \mathcal{H}$ that fits the data well), $\overline{L}(D|\mathcal{H})$ will also be small. Refined MDL uses a second concept called *parametric complexity* $\mathbf{COMP}(\mathcal{H})$. It indicates a models ability to fit random data. Despite the improvements from crude to refined MDL, model selection using refined MDL is still a trade-off between

a goodness of fit term $L(D|\hat{H})$ ($\hat{H}$ is the distribution in $\mathcal{H}$, that minimizes the codelength) and the complexity term $\mathbf{COMP}(\mathcal{H})$. [14]

H. Thodberg used the MDL principle for automatic training example generation for ASMs and AAMs [22]. This approach was used to reduce the number of shape pixels (represented as the shapes contour) of the lumbar vertebræ resulting form the manual image annotation and obtain a suitable (< 200 landmarks per shape) amount of shape landmarks.

The n annotated shape contours $\mathbf{c}_i$ ($i = 1, \ldots, s$) are aligned and corresponding points are calculated. These points, so called landmarks are located on the annotated shape contours. With these points the point position vector $\mathbf{v_i}$ (see Eq. 3) for each shape are generated, that are used for the later construction of the Shape Model (see Section 2.3).

2.2 Principal Component Analysis

Principle Component Analysis, short PCA, is a well known and wide spread technique in the field of Pattern Recognition, Image Understanding and Computer Vision [3]. PCA is mainly used for dimensionality reduction, feature extraction, lossy data compression and data visualization [3]. It is also known as the *Karhunen-Loeve* transform, the *Hotelling* transform or proper orthogonal decomposition [3]. The PCA was first introduced by Karl Pearson in 1901 [17]. He defined PCA as the linear projection that minimizes the average projection cost, in other words the mean squared distance between the data points and their projection [17]. Besides this definition from Pearson there exists a second also commonly used definition formulated by Hotelling, defining the PCA as the orthogonal projection of data onto a lower dimensional linear space, which maximizes the variance of the projected data.

Consider an n-dimensional space and the containing data represented as a $m \times n$ Matrix $\mathbf{A}$, whereas m donates the amount of data points in dimension n. In other words each column represents one of the n dimensions containing m points, short $\mathbf{A}_i = (\mathbf{A}_{i1}, \ldots, \mathbf{A}_{im})^T$, $i \in \{1, \ldots, n\}$. To produce a normalized data set (with zero mean), the mean $\overline{\mathbf{A}_i}$ of each dimension or column $\mathbf{A}_i$ is subtracted, short $\mathbf{A}_i = \mathbf{A}_{ij} - \overline{\mathbf{A}_i}$ with $j \in \{1, \ldots, m\}$. Normalization is crucial for PCA to assure that the data is centered around the origin. The next step is to calculate the covariance matrix of the data, which for n-dimensional data is defined as follows [20]:

$$C^{n \times n} = c_{i,j}, \qquad c_{i,j} = cov(\mathbf{A}_i, \mathbf{A}_j) \tag{1}$$

In matrix notation ($i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$):

$$C = \begin{pmatrix} cov(\mathbf{A}_1, \mathbf{A}_1) & \cdots & cov(\mathbf{A}_1, \mathbf{A}_j) \\ \vdots & \ddots & \vdots \\ cov(\mathbf{A}_i, \mathbf{A}_1) & \cdots & cov(\mathbf{A}_i, \mathbf{A}_j) \end{pmatrix} \tag{2}$$

Now the eigenvectors eig and their eigenvalues λ are calculated. The eigenvector with the largest eigenvalue is the data sets *principle component*. To reduce the n-dimensions to p-dimensions, the eigenvectors are sorted by eigenvalue in descending order and the first p eigenvectors are selected forming the feature vector $\mathbf{F} = (eig_1 ... eig_p)$ [20]. So only the p most significant eigenvectors are selected. By ignoring some eigenvectors of lesser significance (e.g. the vectors with eigenvalues not contained in 95% of the point variance) information is lost. If the discarded eigenvectors are small the lost information can be neglected [20]. The most significant or strongest eigenvectors with the largest eigenvalues represent the dimensions with the strongest correlation in the data set and are proportional to the variance. In this way only those data set characteristics, that contribute most to the data set's variance, are kept. A simple result for the reduction of the dimensionality from n-d to 2-d is shown in Figure 3. In this example n-dimensions where reduced to 2-dimensions. The red and green line represent the eigenvectors with the largest eigenvalues, which in turn represent the dimensions with the strongest correlation in the data set. The eigenvalues, that were calculated using PCA during the shape model generation (see Section 2.3) of the lumbar vertebræ are shown in Figure 2.2. The eigenvalues are sorted in descending order. 95 % of the data point variance is covered by the three red bars. The eigenvalues for the other 3 evaluated data sets can be found in Section 6, Figure 27.

PCA is used in several areas of medical image processing including Shape Models [9], ASMs [7], AAMs [8], etc.

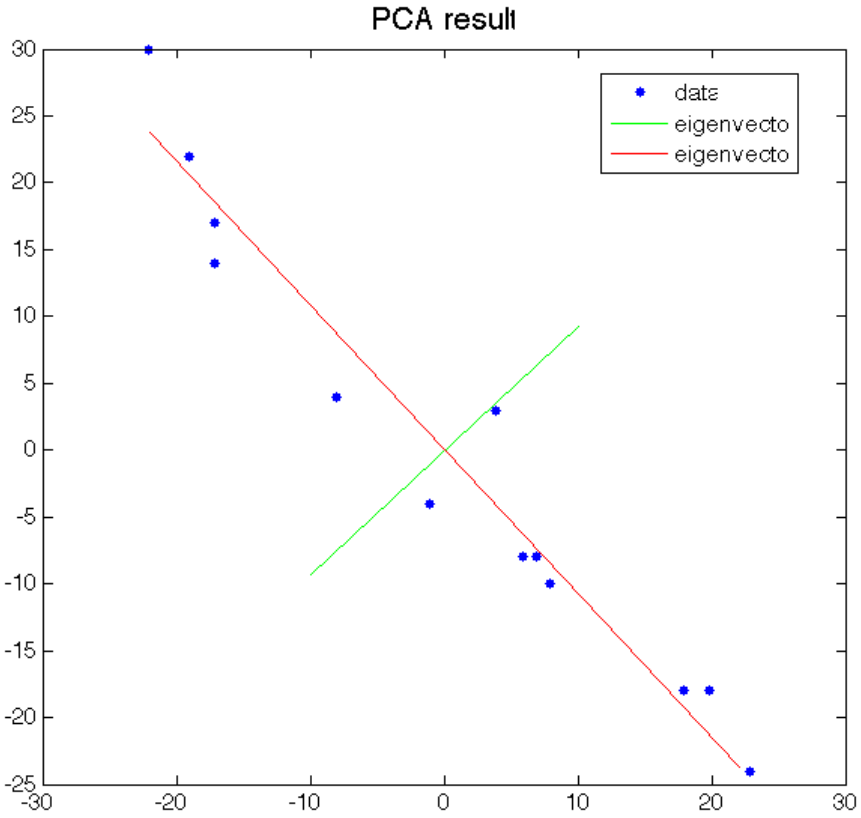


Figure 3: PCA result where n-dimensions where reduced to 2-dimensions. The drawn in eigenvectors (red, green) with the largest eigenvalues represent the dimensions with the strongest correlation in the data set.

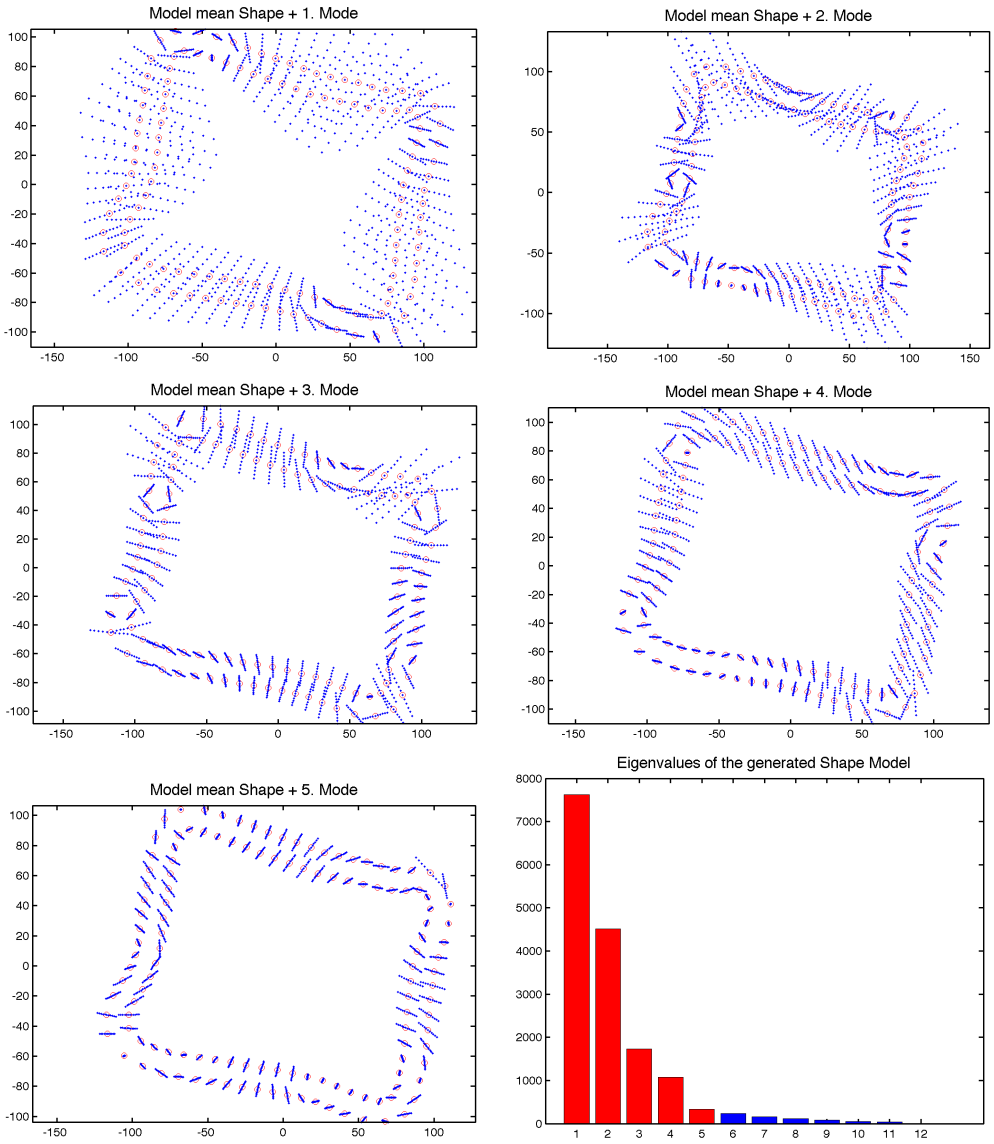


Figure 4: Figures of the calculated eigenvalues and eigenmodes of the lumbar vertebræ. The first five images show the mean shape of shape model (red) and generated modes (blue) of the vertebræ. The sixth image shows the bar diagram of the calculated eigenvalues of the lumbar vertebræ. The red bars represent 95% of the data point variance.

2.3 Shape Models

Objects in images can be represented using certain statistical models of the objects' shape. The Point Distribution Model (PDM) introduced by Cootes et al.[9] is a refinement of Shape Models. These shapes consist of a set of n points or landmarks. The points can have any dimension, but in practice they are usually two or three dimensional. Imaging a simple rectangle whose shape is defined by its four vertices (in 2D or 3D) it is obvious that this shape does not change when it is moved, rotated or scaled. Therefore the shape is invariant to the similarity transformation, in detail translation, rotation and scaling. Considering these facts, shape can be defined as the quality of a set of points that is invariant to several transformations.

Using this shape information the goal is to build models which provide the abilities to analyze new shapes and to create new shapes with similar probabilities of the shapes in a predefined training set.

The first step to create a training set for shape model generation is to define landmarks for several objects. To obtain landmarks for a shape usually a human expert annotates several images containing the corresponding object. Landmarks are derived from this manually annotated image contours by applying MDL (see Section 2.1). Then a vector $\mathbf{v}_i$ for all $i \in 1, ..., s$ annotated shapes is formed by concatenating each dimension of the selected n points. The point position vector $\mathbf{v}_i$ of shape i for nd-dimensional landmarks is defined as

$$\mathbf{v}_i = (d_{1_1}, ..., d_{1_n}, d_{2_1}, ..., d_{2_n}, ..., d_{n_1}, ...d_{n_n}) \tag{3}$$

The training set is then aligned using Procrustes Analysis, which minimizes $\sum |\mathbf{v}_i - \bar{\mathbf{v}}|^2$, where $\mathbf{v}_i$ is the i^{th} point position vector and $\bar{\mathbf{v}}$ is the mean of all vectors.

To be able to generate new shapes out of the training set a parameterized model $\mathbf{v} = M(\mathbf{b})$ of the distribution of the s point position vectors $\mathbf{v}_i$ is defined. $\mathbf{b}$ is a vector containing the model parameters (Eq. 5). With the help of this model it is possible to generate new vectors $\mathbf{v}$ and to estimate the distribution of these new vectors $p(\mathbf{v})$. Due to the fact that the data points used are n-dimensional it is necessary to reduce the dimensionality for simplicity and performance reasons. Furthermore the last 5% of the eigenvectors only represent noise and are therefore useless. Dimensionality reduction can be achieved by applying Principle Component Analysis (PCA see Section 2.2) to the data. Using the resulting p eigenvectors any training set $\mathbf{v}$ can be approximated by applying Eq. 4 [8].

$$\mathbf{v} \approx \bar{\mathbf{v}} + \mathbf{F}\mathbf{b} \tag{4}$$

where $\mathbf{F} = (eig_1, ..., eig_p)$ and $\mathbf{b}$ is a vector with p dimensions which defines the parameters for the deformable model [8]:

$$\mathbf{b} = \mathbf{F}^T(\mathbf{v} - \bar{\mathbf{v}}) \tag{5}$$

The resulting model represents the shape variation of the modeled objects utilizing a single parameter vector $\mathbf{c}$. Each element of $\mathbf{c}$ controls one mode of shape variation, with the first modes being responsible for the highest variation, in descending order.

In addition to these modes for basic operations as translation, scaling and rotation need to be taken into account. Therefore a new linear parameter vector $\mathbf{t} = (s_x, s_y, t_x, t_y)^T$ is introduced, controlling rotation θ, scaling s and translation (t_x, t_y), with $s_x = s\cos\theta - 1$ and $s_y = s\sin\theta$.

Combining the parameter vector of the PCA $\mathbf{b}$ and the parameter vector for translation, rotation and scaling $\mathbf{t}$ results in the combined parameter vector

$$\mathbf{c} = (\mathbf{b}^T, \mathbf{t}^T) \tag{6}$$

.

To ensure that the generated shapes are similar to those in the training set the parameter vectors can be limited using $\pm 3\sqrt{\lambda_i}$, where λ_i is the eigenvalue and also the variance of the i^{th} parameter b_i in the training set.

Finally the number of modes has to be chosen. It is chosen to represent a certain percentage, for example 95%, of the training set variance. This is done because the eigenvectors corresponding to the smallest eigenvalues represent noise. So the modes with the smallest eigenvalues are neglected.

The mean shape of the generated shape model for the lumbar vertebræ and the three resulting eigenmodes are shown in Fig. 2.2. Eigenmodes of the other evaluated data sets can be found in Section 6, Figures 27(e), 29 and 30.

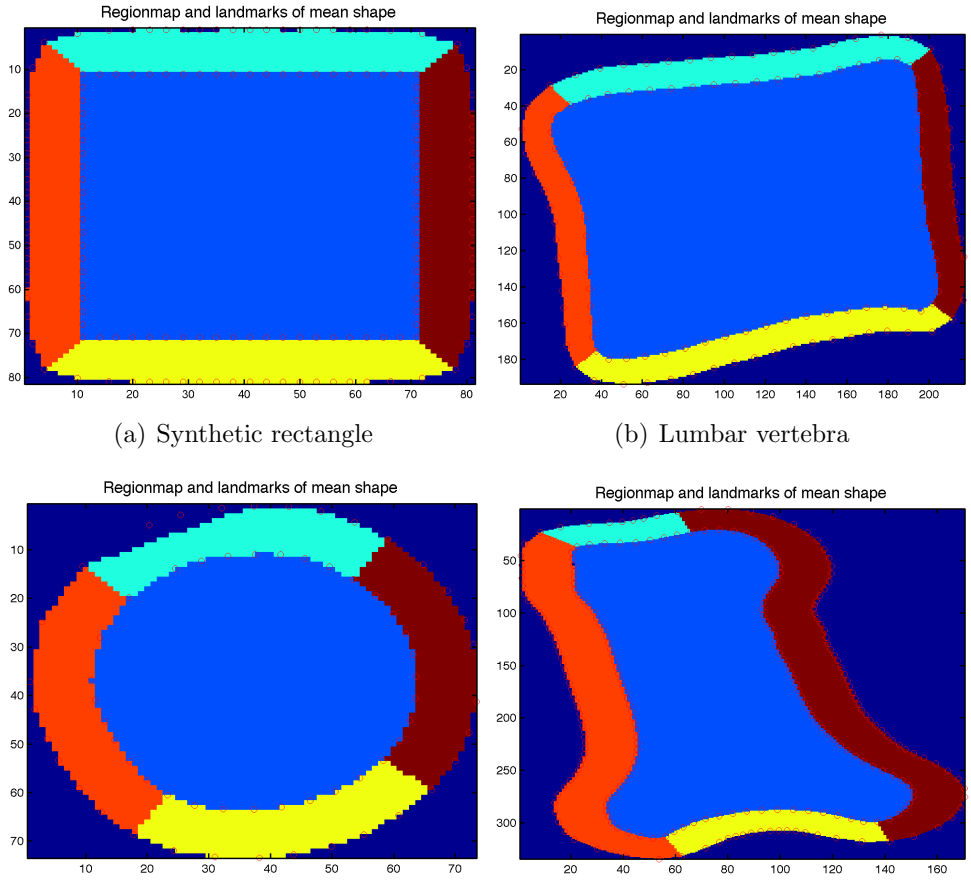
3 Shape Particle Filters

This section documents the modified implementation of Marleen de Bruijne's shape particle filtering approach for image segmentation [11, 10].

A model of the object to segment in the target images was created using a Shape Model similar to those explained in Section 2.3. For the models' mean shape a region map, representing six regions of interest (background, inside, above, below, left, right), was created (see Fig. 5). This mean shape region map was then warped back onto all the model shapes applying Thin Plate Spline Warping (TPS)[4]. The resulting region maps serve as region masks for the feature extraction.

To classify a test image, a pixel classifier (see Section 3.2) is trained using the extracted image features, e.g local image descriptors computed using Gabor Jets (see Section 3.1). This results in a classification probability map, where the probability of belonging to one of the six predefined labels is assigned to each pixel. Finally the optimal solution is computed using a Differential Evolution (DE) approach for Shape Particle Filtering (see Section 3.3). This is done by generating a random set of shapes, the particles, from the prior shape model. The image labeling associated to the particles is compared with the label probability maps obtained in the classification step and a residual is calculated, by summing up all region probabilities. The higher the residual the better is the shape segmentation, leading to the best solution for a given number of iterations (generations).

Section 3.1 covers the feature extraction process using Gabor Jets. The proposed algorithms for feature classification, k-Nearest Neighbors (k-NN), kd-trees in combination with k-NN and Support Vector Machines (SVMs), are described in Section 3.2. In Section 3.3 a Differential Evolution (DE) approach for Shape Particle Filtering, is presented as an alternative to the originally proposed particle filter.



(a) Synthetic rectangle (b) Lumbar vertebra

(c) Heart (d) Metacarpal Bone

Figure 5: Region map of the mean shape for each data set. The six regions are shown in different colors.

3.1 Gabor Jets

Gabor Jets are the outputs from a set of Gabor filters. The advantage of Gabor Jets lies in their phase invariance to edge positions, if the filter responses were converted to amplitude and phase [24].

A Gabor filter consists of two functions, a complex sinusoidal called *carrier* and a Gaussian-shaped function known as the *envelop*

$$g(x, y) = s(x, y)w_r(x, y) \tag{7}$$

where $s(x, y)$ is the carrier and $w_r(x, y)$ is the envelop [16]. The complex sinusoidal carrier function is shown in Eq. 8,

$$s(x, y) = exp\Big(j\Big(2\pi(u_0x + v_0y) + P\Big)\Big) \tag{8}$$

where u_0 and v_0 define the spatial frequency in Cartesian coordinates and P the phase [16]. The Gaussian envelop can be written as

$$w_r = Kexp\Big(-\pi\Big(a^2(x - x_0)^2_{r(\theta)} + b^2(y - y_0)^2_{r(\theta)}\Big)\Big) \tag{9}$$

where K is a scale factor for the envelop's magnitude, (x_0, y_0) is the location of the peak of the Gaussian envelop, a and b are scaling parameters for the two axis of the Gaussian and the $r(\theta)$ stands for a rotation operation, describing a clockwise rotation with the angle θ [16]. In the implementation used in the scope of this report the two scaling parameters are equal (e.g. $a = b$).

The Gabor filters described above were used to extract the image features for each of the six regions predefined by the region map. The amount of extracted features depends on the chosen number of rotation angles and the number of frequencies. The Gabor filters and extracted features are shown in Fig. 6 for the test rectangles and in Fig. 7 for the lumbar vertebræ.

After the extraction the features are normalized to get feature values between 0 and 1.

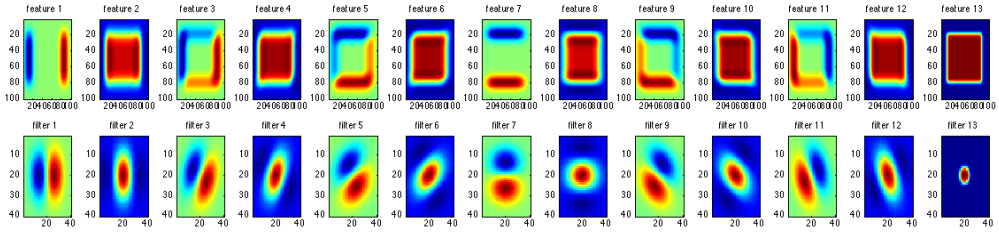


Figure 6: Resulting features extracted of test rectangles using Gabor filters.

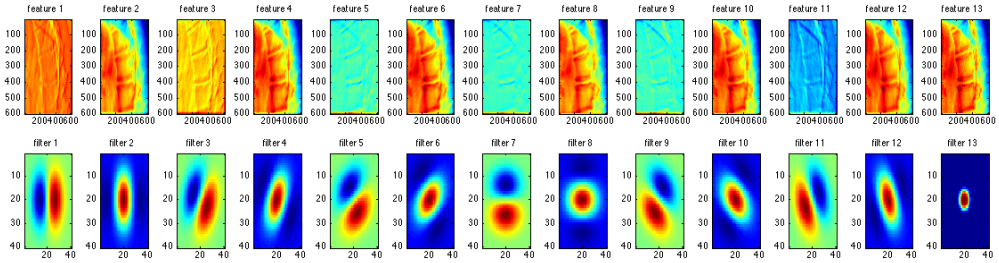


Figure 7: Resulting features extracted of lumbar vertebræ X-ray images using Gabor filters.

3.2 Image Feature Classification

kd-Trees Three methods were implemented to allow the classification of a test image using the information of the extracted training features. Due to the huge amount of features resulting form the feature extraction process the simple k-Nearest Neighbors (k-NN) approach did not meet the requirements in time and computational performance. Therefore a k-dimensional tree (kd-tree) formerly introduced by Bentley in 1975 [2] was used for faster classification.

A kd-Tree is a data structure for storing data in k-dimensional spaces that then can be used for associative searches. It is an extension of a binary tree where every node represents a k-dimensional point. Each node P contains k keys $K_n(P)$ $(n = 0, ..., k-1)$ containing the data and two pointers. These pointers are either null or point to another node in the kd-tree and therefore each pointer can be seen as a subtree forming left and right branches of the node. For keys located on a left branch of node P in depth j the following condition is true

$$K_j(LEFT) < K_j(P). \tag{10}$$

Likewise for keys located on a right branch of node P in depth j

$$K_j(RIGHT) > K_j(P) \tag{11}$$

is true. A special case occurs, if the keys are equal. In this case the decision where the key belongs must be based on the remaining keys. This can be done by tracking back the branch and examining the keys there, or by defining a superkey, that in case of equality returns a predefined value. But actually the decision what to do is arbitrary, depending on the respective area of application.

A 2D tree decomposition is shown in Fig. 8 and its graph representation in Fig. 9.

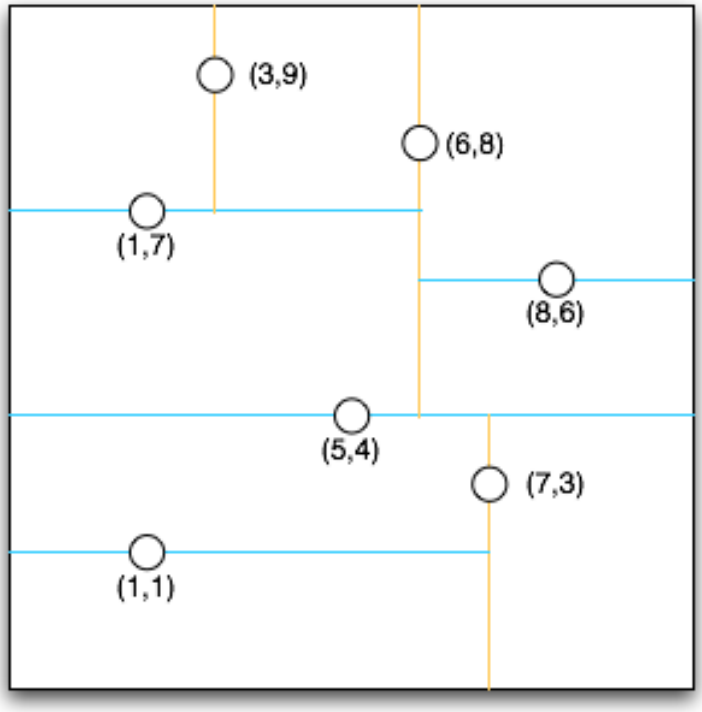


Figure 8: A 2d tree decomposition.

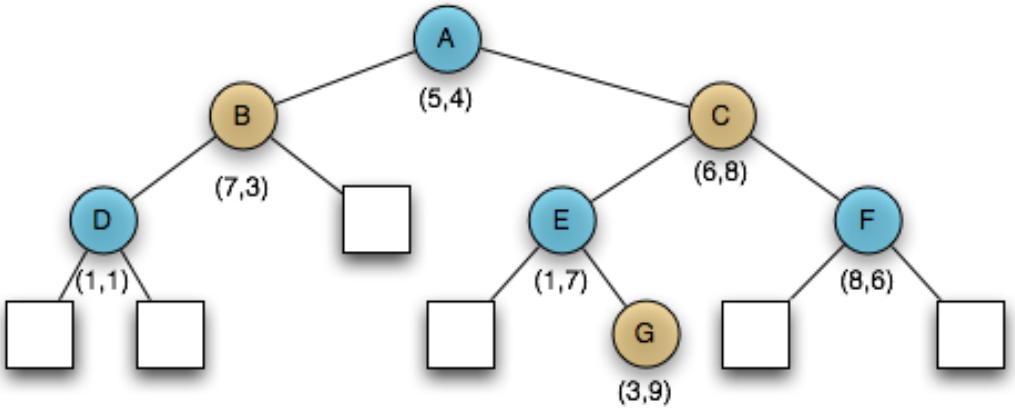


Figure 9: Graph representation of a 2d tree. The squares represent null pointers.

A kd-tree as described above was constructed using the extracted training features. To classify the test feature set a k-Nearest Neighbors approach is employed using the the kd-tree for a first step in which approximately 100 feature vectors are queried around a test vector. With this subset a standard k-NN classifier using euclidean distance is employed.

The k-NN algorithm simply classifies a pixel by assigning it to the class that is most common amongst its k nearest neighbors. k is usually chosen as a small, odd integer. Training and test sets were created using leave-one-out cross-validation. For each pixel of the test image the probability of belonging to a specific region of the region map was calculated using the classification results. This resulted in an image-sized probability map for each region of the region map, e.g. a map of the same size as the test image with the probabilities for each pixel to belong to the background region. Despite the correct classification and the speed improvement compared to the simple k-NN algorithm, the kd-tree implementation was still far to slow for practical application.

Finally Support Vector Machines were used for classification.

Support Vector Machines Based on the statistical learning theory [23] and originally developed for pattern recognition Support Vector Machines (SVM) are supervised learning methods that are used for solving classification and regression problems. By applying an a priori chosen nonlinear mapping

(the *kernel*), an input vector is mapped into a high dimensional feature space and an optimal separating hyperplane is created there. To achieve this, generalization of the SVM in high-dimensional space has to be considered.

SVMs can be divided into linear SVMs and nonlinear SVMs according to the kind of the decision function used. Furthermore two cases have to be distinguished, SVMs used for classification of linear separable data and SVMs used for non-separable data. In this short description of the used SVM the focus lies on linear SVMs. A detailed description on SVMs, especially on the nonlinear form and the therein used kernels can be found in [23, 6, 18, 3].

Consider a linear SVM trained on linear separable data represented by n input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. A label t_i is assigned to each data vector $\mathbf{x}_i$, resulting in targets or labels t, where $t_i \in \{-1, 1\}$. A new data vector $\mathbf{x}$ is classified by the sign of its according label $t(\mathbf{x})$. Supposing a separating hyperplane H, where points which lie on H satisfy $\mathbf{w}\mathbf{x}+b = 0$ ($\mathbf{w}\perp H$, $|b|/\|\mathbf{w}\|$ perpendicular distance from H to origin). SVMs try to find the classification solution with the smallest generalization error. This is done by choosing the decision boundary so that the *margin* is maximized. The margin is defined as the smallest distance between the decision boundary and all the samples. Defining two new hyperplanes $H_1 : \mathbf{x}_i\mathbf{w} + b = 1$ ($\mathbf{w}\perp H$, $|1 - b|/\|\mathbf{w}\|$) and $H_2 : \mathbf{x}_i\mathbf{w} + b = -1$ ($\mathbf{w}\perp H$, $| - 1 - b|/\|\mathbf{w}\|$), data points that satisfy Eq. 12 lie on H_1 and data points satisfying Eq. 13 lie on H_2. H_1 and H_2 have the same normal $\mathbf{w}$ and are therefore parallel.

$$\mathbf{x}_i\mathbf{w} + b \geq +1 \quad \text{for } t_i = +1 \tag{12}$$

$$\mathbf{x}_i\mathbf{w} + b \leq -1 \quad \text{for } t_i = -1 \tag{13}$$

Combining these inequalities yields to the following set of inequalities

$$t_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \quad i = 1, \ldots, n \tag{14}$$

Furthermore no training points fall between them. The training points lying on one of the two hyperplanes H_1 or H_2 and hence satisfying Eq. 14 are called *support vectors*. Removing any of them would change the classification result.

The distances between H and the two other hyperplanes are $d_1 = d_2 = 1/\|\mathbf{w}\|$ and therefore the margin is $2/\|\mathbf{w}\|$.

The solution for a simple 2d classification problem is shown in Fig. 10.

By reformulating the problem into a Lagrangian a later generalization for nonlinear data classification is possible. Therefore unconstrained positive

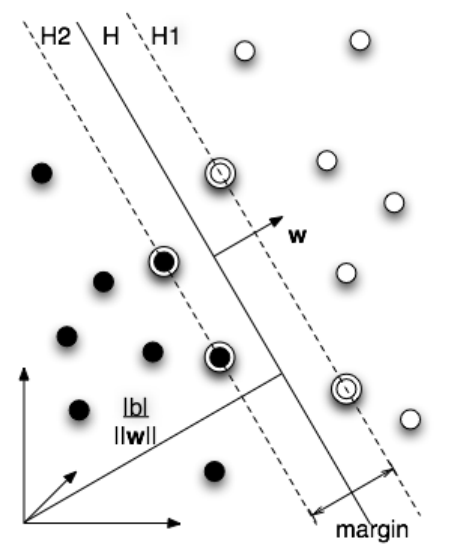


Figure 10: The separating hyperplanes for a linear separable 2d classification problem. The support vectors are marked with circles.

Lagrange multipliers α_i, $i = 1, \ldots, n$ are multiplied with the constraint equations and the result is subtracted from the objective function, leading to the Lagrangian

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n}(\alpha_i t_i(\mathbf{x}_i\mathbf{w} + b)) + \sum_{i=1}^{n}(\alpha_i). \tag{15}$$

Minimizing L_p leads to a convex quadratic programming problem. A dual formulation of the problem called the *Wolfe dual* can be used to get the dual formulation L_D of the Lagrangian L_P, as follows

$$L_D = \sum_i(\alpha_i) - \frac{1}{2}\sum_{i,j}(\alpha_i\alpha_j t_i t_j \mathbf{x}_i\mathbf{x}_j) \tag{16}$$

under the conditions

$$\mathbf{w} = \sum_i(\alpha_i t_i \mathbf{x}_i) \tag{17}$$

$$\sum_i(\alpha_i t_i) = 0 \tag{18}$$

SVM training is then done by maximizing L_D with respect to α_i. The exact description of the derivation of L_D can be found in [6].

Applying this algorithm on non-separable data will lead to no feasible solution (an arbitrary large growing objective function i.e. the dual Lagrangian). This can be overcome by extending the constraints in Eq. 12 and Eq. 13 with further costs in the form of positive slack variables ξ_i, $i = 1, \ldots, n$ to

$$\mathbf{x}_i\mathbf{w} + b \geq +1 - \xi_i \quad \text{for } t_i = +1 \tag{19}$$

$$\mathbf{x}_i\mathbf{w} + b \leq -1 + \xi_i \quad \text{for } t_i = -1 \tag{20}$$

$$\xi_i \geq 0 \quad \forall i \tag{21}$$

The sum over all slack variables ξ_i forms an upper bound on the number of training errors. The dual formulation L_D then becomes

$$L_D \equiv \sum_i (\alpha_i) - \frac{1}{2} \sum_{i,j} (\alpha_i \alpha_j t_i t_j \mathbf{x}_i \mathbf{x}_j) \tag{22}$$

with the conditions

$$0 \leq \alpha_i \leq C \tag{23}$$

$$\sum_i (\alpha_i t_i) = 0 \tag{24}$$

$$\mathbf{w} = \sum_i^{n_s} (\alpha_i t_i \mathbf{x}_i) \tag{25}$$

n_s represents the number of support vectors and C is a manually defined upper bound. By comparing the linear separable and non-separable case it is obvious that the two dual Lagrangians differ only in their constraint conditions, more precisely in the upper bound C of the Lagrangian multipliers in the non-separable case. The separating hyperplanes for the non-separable case are shown in Fig. 11. To transform the primal Lagrangian L_P the Karush-Kuhn-Tucker conditions are needed. Details hereunto can be found in [6].

A linear SVM as described above was trained and used for the classification of the provided test features.

The speed gain for the classification of the data sets achieved by the linear SVM in comparison to the k-NN and the kd-tree is shown in Table 1.

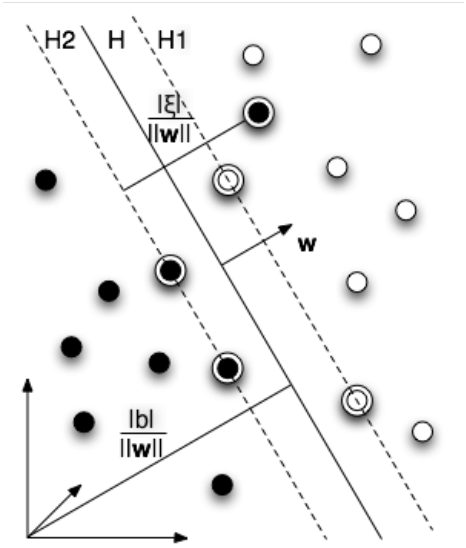


Figure 11: The separating hyperplanes for a linear non-separable 2d classification problem. The support vectors are marked with circles.

	k-NN	kd-tree	linear SVM	speed gain vs. k-NN	speed gain vs. kd-tree
Synth. rectangles	32.6	20.5	4.2	7.8x	4.9x
Lumbar vertebræ	558.0	370.1	77.8	7.2x	4.8x
Hearts	239.8	134.8	19.9	12.1x	6.8x
Metacarpal bones	641.4	363.8	70.5	9.1x	5.2x

Table 1: Mean classification speed over 10 runs for one test image in seconds for the k-NN, the kd-tree and the linear SVM. In the right most column the speed gain achieved by using SVMs compared to the other classification algorithms is shown (SVM times faster than k-NN and kd-tree).

3.3 Differential Evolution (DE)

DE was introduced by Kenneth Price and Rainer Storn in 1997 [21]. It is a genetic algorithm and aims at optimizing functions based on populations in real parameter space.

DE is a parallel direct search method that uses N d-dimensional parameter vectors $\mathbf{x}_i$ $(i = 1, ..., N)$ (e.g. N d-dimensional Markov chains) as members of a population $\mathbf{X}$ for each generation G. New parameter vectors $\mathbf{x}_p$ are then generated by adding the weighted difference vector between two population members to a third member (Eq. 26).

$$\mathbf{x}_p = \mathbf{x}_{r_1} + \gamma(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) \tag{26}$$

where the $\mathbf{x}_{r_j}$ $(j = 1, 2, 3)$ are randomly selected from the population $\mathbf{Y}$ ($\mathbf{Y} = \mathbf{Y}$ ($\mathbf{Y} = \{\mathbf{X} \setminus \mathbf{x}_{r_1}\}$) and γ is a real constant factor that controls the differential variation $(\mathbf{x}_{r_2} - \mathbf{x}_{r_3})$. If the fitness function $\pi(.)$ of $\mathbf{x}_p$ is higher than the one of $\mathbf{x}_i$ $(\pi(\mathbf{x}_p)/\pi(\mathbf{x}_i) > 1)$, $\mathbf{x}_p$ replaces $\mathbf{x}_i$. To keep track of the minimization progress the best parameter vectors is evaluated for each generation G [5].

Using DE for Shape Particle Filtering To apply DE for Shape Particle Filtering, the parameter vector generation previously shown in Eq. 26 has to be modified to assure a detailed balance of proposal and acceptance with respect to the fitness function $\pi(.)$. To do so and to ensure that the whole parameter space is covered, Eq. 26 is modified to

$$\mathbf{x}_p = \mathbf{x}_i + \gamma(\mathbf{x}_{r_2} - \mathbf{x}_{r_3}) + \mathbf{e} \tag{27}$$

where $\mathbf{e}$ has a small variance compared to the target and is drawn from a symmetric distribution, e.g. $\mathbf{e} \sim N(0, b)$ with small b. [5].

Parameter vectors, as described above, representing the spatial information of the target shapes were set up consisting of the shape models eigenmodes, small initial values for scaling and rotation, as well as the processed images size for translation in x/y direction.

Formally this can be denoted in the following way: i Parameter vectors $\mathbf{c}_i = (\mathbf{b}_i^T, \mathbf{t}_i^T)$ (where $\mathbf{b}_i$ is the parameter vector resulting from the PCA containing the shapes modes and $\mathbf{t}_i$ is the parameter vector for translation, rotation and scaling) representing the individuals or population members are generated in each iteration. These parameter vectors are the mathematical

representation of new shapes, and therefore the sum over all parameter vectors $\sum \mathbf{c}_i$ spans our search space. The best parameter vector or shape is selected by optimizing the fitness function $\pi(x_r)$ (Eq. 28).

In each iteration or generation of the DE algorithm several (e.g. 15) shapes were generated using the parameter vectors. Then the region map was warped onto the new shapes. Using the classified probability maps described in Section 3.2 a residual for x_r (e.g. the result of fitness function $\pi(x_r)$ at iteration r) was calculated by summing up all region probabilities and calculating its reciprocal value (Eq. 28).

$$\pi(x_r) = \frac{\sum_{l=1}^{L} \sum_{i=1}^{n} P(i \mid l)}{L} \tag{28}$$

where L is the number of labels (in this case $L = 6$), n is the number of pixels in the region map and $P(i \mid l)$ is the probability of pixel i belonging to label l.

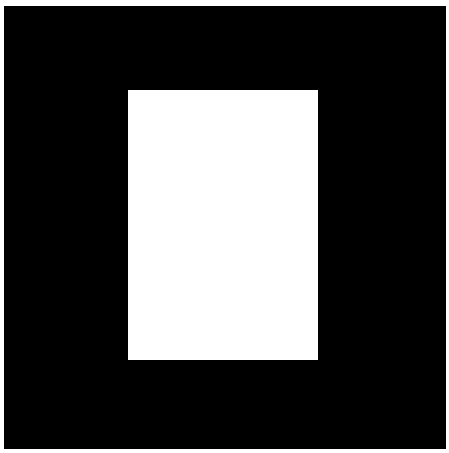
The larger the residual, the higher is the overall probability for a good match for the new shape and therefore the better is the detection result. In each iteration the resulting residual is compared with the previous residual, resetting the best result if better or discarding the current residual. The final results are presented in Section 4, Section 4.2.

4 Experiments

The method described above was evaluated with synthetically generated 2d images, lateral spine radiographs, metacarpal bone radiographs and MRI slices of the human heart. An example of each data set is shown in Fig. 12. The ground truth is represented by hand annotated radiographs and MRI slices.

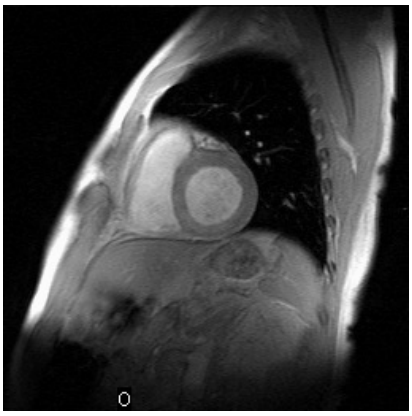
Because of the small amount of training images and therefore training features, Leave One Out Cross Validation (LOOCV) was used for evaluation. In each test run one vertebra was used for validation and the remaining vertebræ were used for training. This was repeated till each vertebra was once used as validation data. The LOOCV runs were repeated 10 times for each input image type.

From each resulting shape of the DE (see Section 3.3) the euclidean distances to the hand annotated ground truth are calculated and visualized using box-and-whisker plots (see Section 4.2, Figure 17).

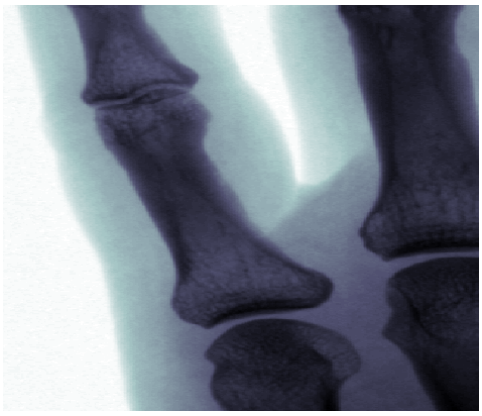


(a) Synthetic Rectangle

(b) Lumbar vertebra



(c) Heart



(d) Metacarpal bone

Figure 12: Images of data sets processed with the proposed particle filter solution.

4.1 Setup

Evaluation was performed with MATLAB on synthetic 2d images, lateral spine radiographs, with focus on the lumbar vertebræ, on metacarpal bone radiographs and on MRI slices of the heart. As mentioned above LOOCV was used for the evaluation of each data set.

Synthetic 2d Images Seven synthetic images with a size 100 x 100 pixels each containing a rectangle were created. The rectangles grow in size by incrementally adding the first mode of shape variation. These rectangles were manly used for testing during development but were also evaluated. An example image is shown in Figure 12(a).

Lateral Spine Radiographs Four radiographs of the human spine were used to evaluate the proposed method. The resolution of these images was about 1500 x 1900 pixels each. Due to the fact that only five lumbar vertebræ were of interest in this project, smaller regions were cut out of the X-ray images. Three of these images were discarded due to the bad visibility of the vertebra and the image quality. The resulting 13 images had a resolution of 400 x 400 pixels each and were centered on the respective lumbar vertebra. The vertebræ were annotated by hand. Afterwards MDL (see Section 2.1) was used to get an equal number of landmarks for each vertebra. An example image is shown in Figure 12(b).

Metacarpal Bone Radiographs The second evaluation data set was a collection of 36 radiographs of human metacarpal bones with the resolution of approximately 500 x 400 pixels each. An example image is shown in Figure 12(d).

MRI Slices of the Heart Finally 14 MRI slices of the human heart with the resolution of 256 x 256 pixels each were used for evaluation. An example image is shown in Figure 12(c).

4.2 Results

In this section the final results of the proposed segmentation algorithm are presented. For each data set an image of the best and worst result is shown (e.g. synthetic rectangles Fig. 13, lumbar vertebræ Fig. 14, hearts Fig. 15 and metacarpal bones Fig. 16). The red points represent the hand annotated ground truth of the test shape. The shape resulting from the DE is drawn in with blue points.

The results of the different input types were generated differing only in two parameter settings. These were the size of the region, that was created around the shape and the settings for the Gabor Jets. The region size for the the four different input images is shown in Table 2.

These differences were conducted to avoid overlapping structures that would lead to misclassifications. The parameters for the Gabor Jets, e.g. the filter size, the frequencies used and the angle of the Gabor filter were equal for the synthetic rectangles, the hearts and the lumbar vertebræ. For the metacarpal bones the filter size was increased and four instead of three frequencies were used. (see Table 3) This was done to avoid to loose structural information especially in the joint area between two bones. The other input images had no comparable structures, where the objects were located this close to each other and therefore the changes were only applied for the metacarpal bones.

	region size in pixel
synthetic rectangles, hearts	10
lumbar vertebræ	15
metacarpal bones	20

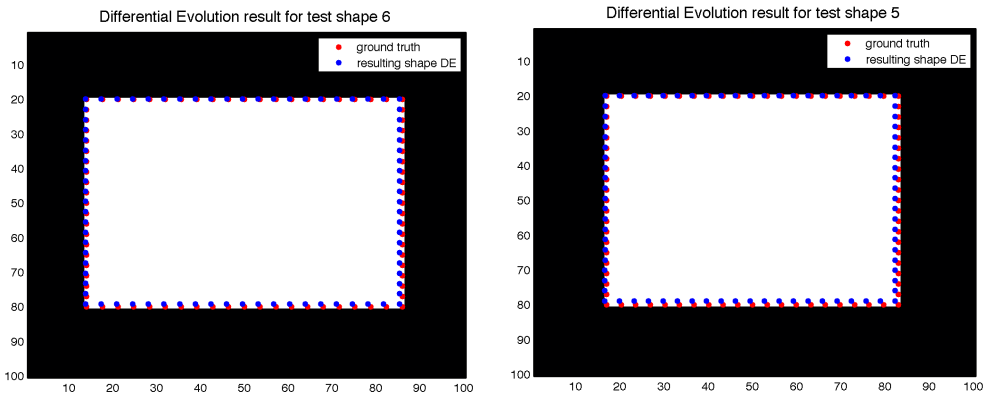
Table 2: Size of the region created around each shape according to the different input images.

Several problems could be observed during the result generation. The extreme differences in the image quality led to discriminative results for all medical input images. Comparing the image quality of the best and the worst result of the hearts (Fig. 15), image 15(b) is highly blurred, whereas the heart in image 15(a) is clearly visible. Another problem resulting from poor image quality accrued with the lumbar vertebræ. In several radiographs it was very hard to precisely determine the vertebra edges, because their upper or

	filter size in pixel	frequencies f	angles α
synthetic rectangles, hearts, lumbar vertebræ	20	1,2,4	0,45,90,135
metacarpal bones	40	1,2,4,6	0,45,90,135

Table 3: Gabor Jet parameters for the different input images including the Gabor filter size, the filter frequencies f and the filter angle α.

lower side was partly visible in consequence of the patients position when the radiograph was taken. This led to overlapping features for different regions of the vertebra and hence to bad classification and segmentation results. Examining the result images for the metacarpal bones (Fig. 16), it can be seen that the hand bones are located very closely to each other. In some cases like in Fig. 16(b) the bones seem to merge, leading to DE results to generally getting stuck several pixels below the bone joint. The outliers in the result boxplot (Fig. 17(d)) are also a manifestation of this fact.

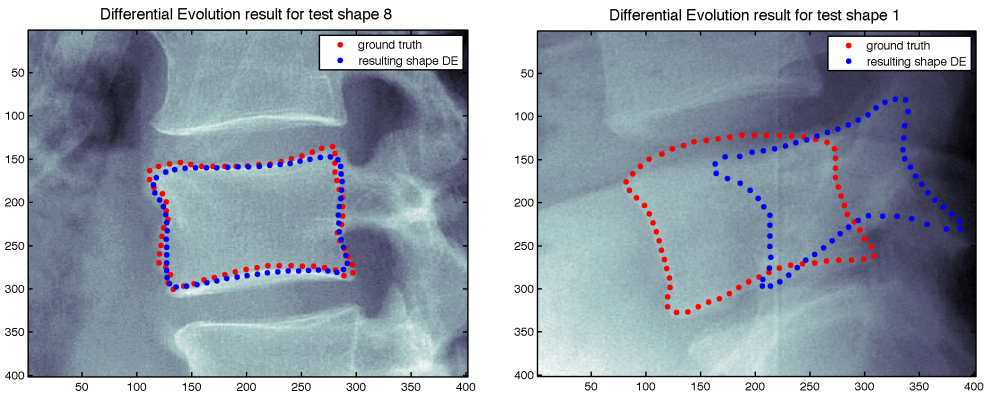


(a) Best (b) Worst

Figure 13: Images of the best and the worst results of the DE for the synthetic rectangles. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.

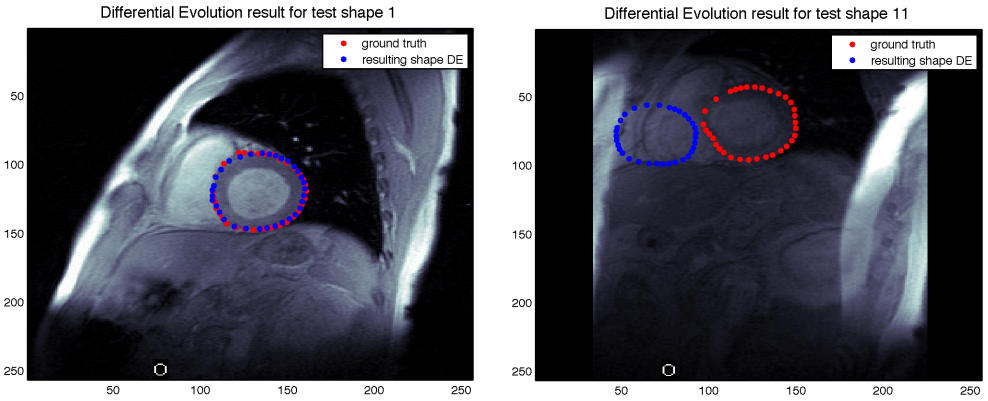
Figure 17 shows Boxplots of the euclidean distances between the hand annotated ground truth and the resulting shape from the DE for each data set for 10 LOOCV runs.

The outliers appearing in the result boxplots of the lumbar vertebræ and



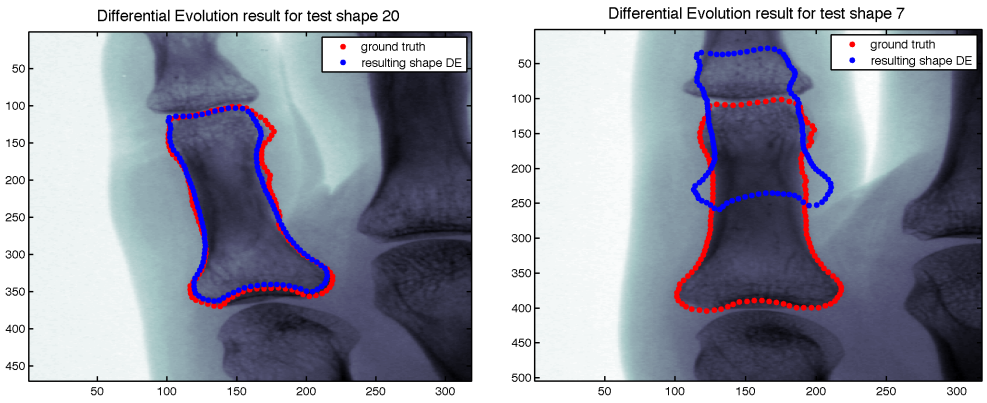
(a) Best (b) Worst

Figure 14: Images of the best and the worst results of the DE for the vertebræ. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.



(a) Best (b) Worst

Figure 15: Images of the best and the worst results of the DE for the hearts. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.



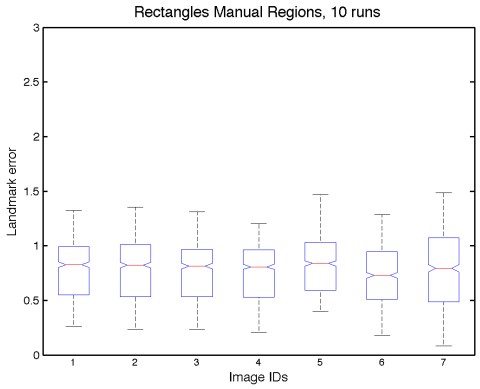
(a) Best (b) Worst

Figure 16: Images of the best and the worst results of the DE for the metacarpal bones. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.

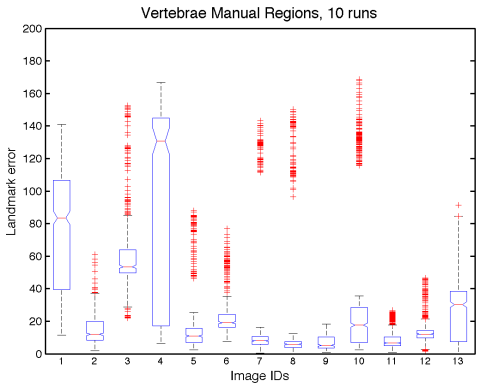
the hearts can be attributed to the stochastic variance resulting from the 10 LOOCV runs. For the metacarpal bones the above mentioned problem of the bone joints add additional weight to the also existing stochastic outliers.

The change of the calculated residual during the DE is shown in Figure 18. The landmark error change over all DE iterations of the best result for each data set is shown in Figure 19.

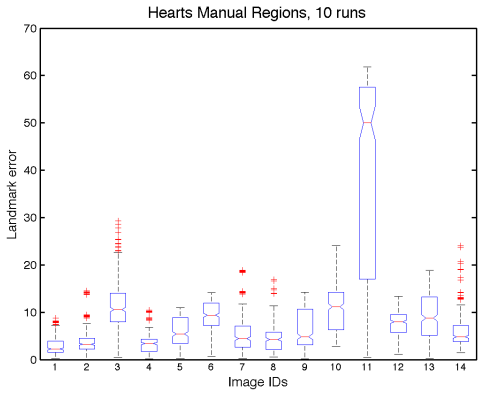
Robustness Experiments To evaluate the robustness of the implemented Particle Filter approach more test runs were set up. A bar with distinctive color values was added centered in the respective test shape. The best result of the normal runs compared to the robustness test result of the same test shape can be seen in Fig. 20 for the synthetic rectangles, in Fig. 21 for the lumbar vertebræ, in Fig. 22 for the hearts and in Fig. 23 for the metacarpal bones. The modification of the test image caused no significant changes in the overall segmentation result for the hearts and metacarpal bones highlighting the predicted robustness. The bad results on the lumbar vertebræ were not surprising due to the general difficulty of segmenting these kind of data. Reasons for this circumstance and possible solutions are given in Section 4.3 and Section 5. The resulting boxplots (Fig. 24) as well as the alteration of



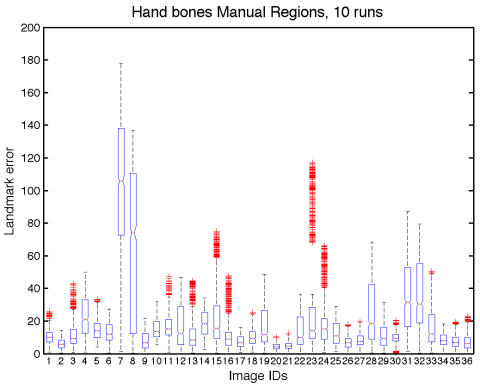
(a) Synthetic rectanges



(b) Lumbar vertebræ

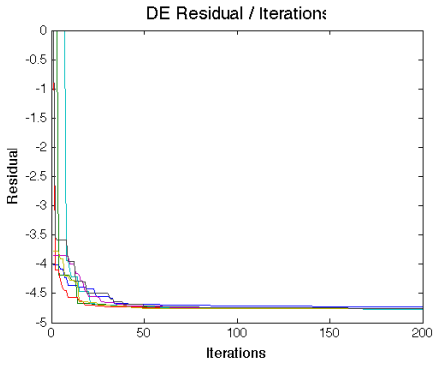


(c) Hearts

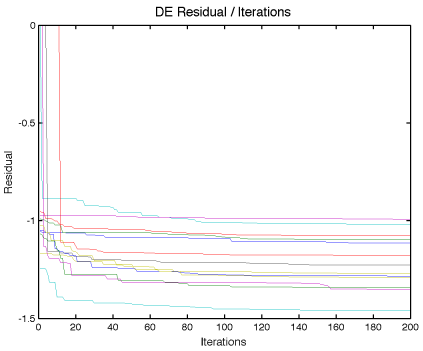


(d) Metacarpal bones

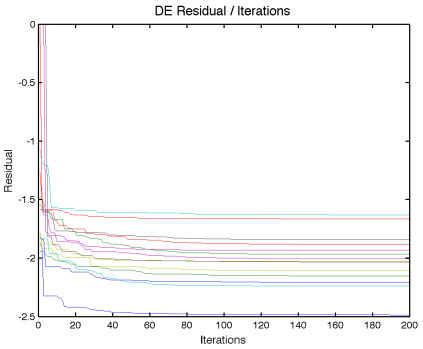
Figure 17: Boxplots of the euclidean distances between the hand annotated ground truth and the shape resulting from the DE for 10 LOOCV runs. The image IDs represent the respective test shapes.



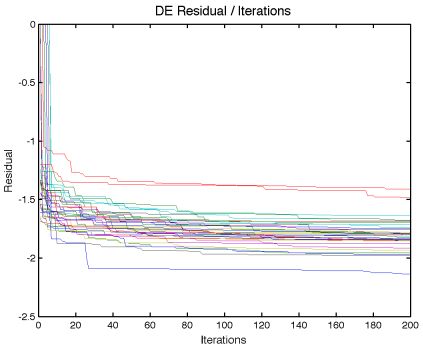
(a) Synthetic rectanges



(b) Lumbar vertebræ

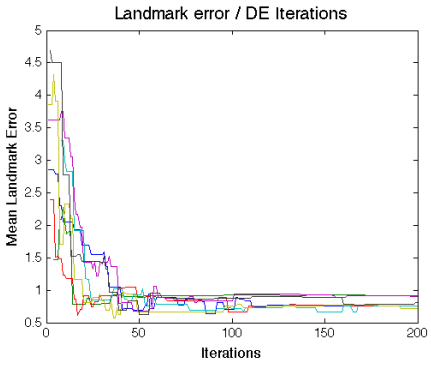


(c) Hearts

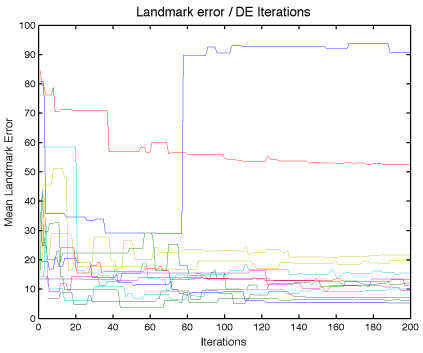


(d) Metacarpal bones

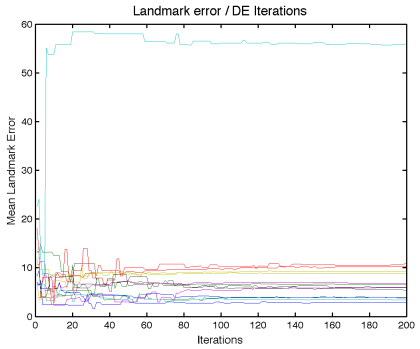
Figure 18: Plots of the DE residual change during the DE iterations.



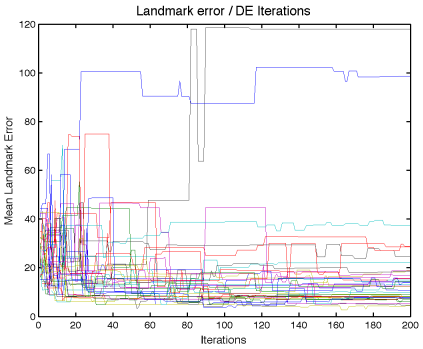
(a) Synthetic rectanges



(b) Lumbar vertebræ



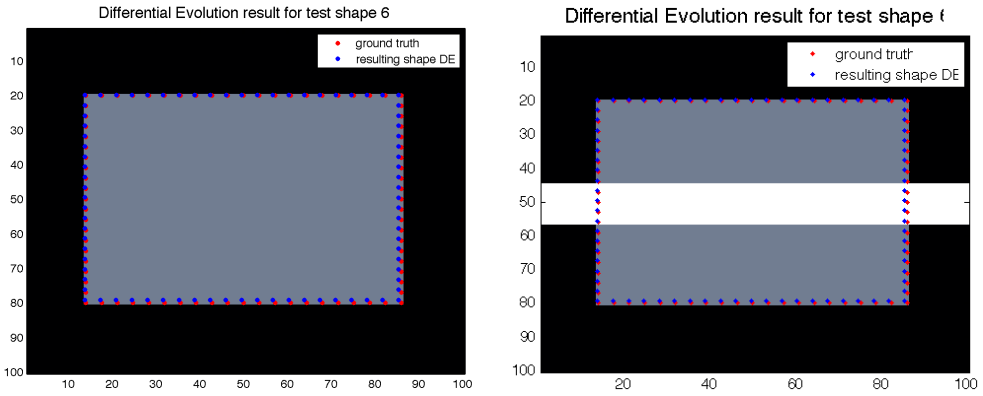
(c) Hearts



(d) Metacarpal bones

Figure 19: Plots of the landmark error change during the DE iterations.

the residual (Fig. 25) and of the landmark error (Fig. 26) per iteration are provided as well.

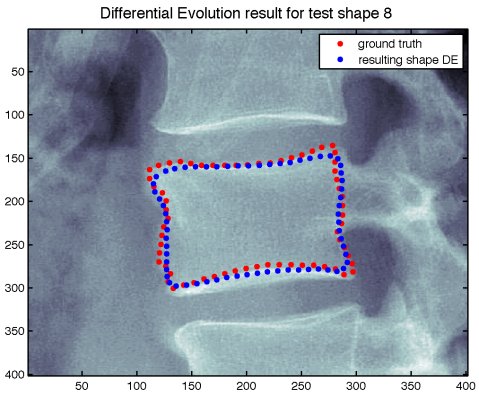


(a) Best Result (b) Result with Bar

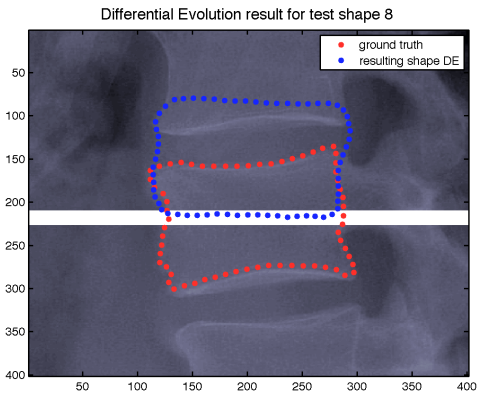
Figure 20: Images of the best result of the DE and the result of the same image with a bar in the center of the shape for the synthetic rectangles. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.

	Mean landmark error (pixels)	
	Particle Filter	robustness test
hearts	8.9	12.7
lumbar vertebræ	31.6	95.1
metacarpal bones	18.2	15.4

Table 4: In this table a comparison of the mean landmark error for the proposed Particle Filter on the hearts, the lumbar vertebræ and the metacarpal bones are presented. As can be seen the results for the lumbar vertebræ are significant worse than those for the other input data.

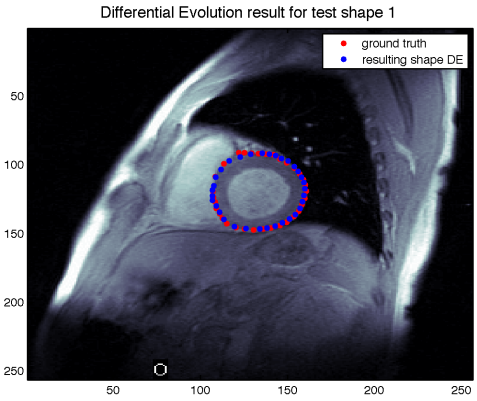


(a) Best Result

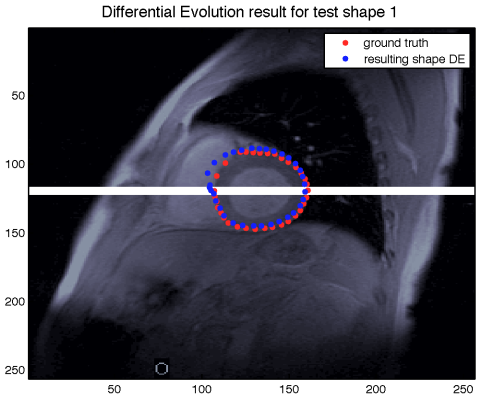


(b) Result with Bar

Figure 21: Images of the best result of the DE and the result of the same image with a bar in the center of the shape for the vertebræ. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.

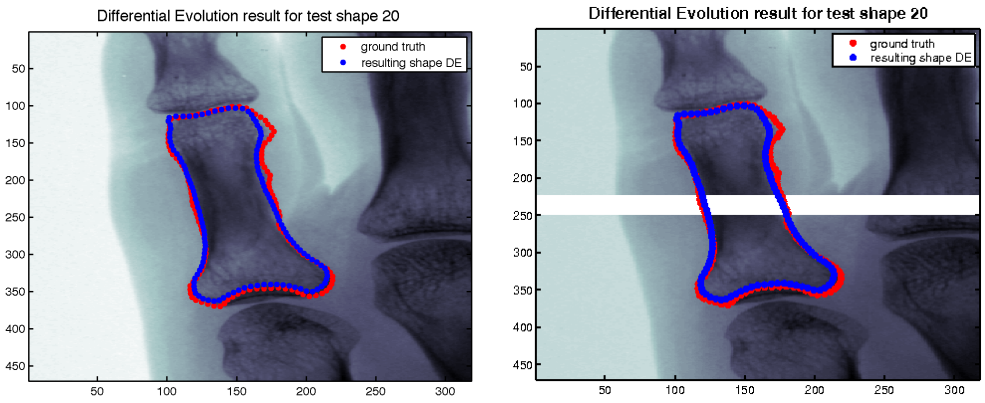


(a) Best Result



(b) Result with Bar

Figure 22: Images of the best result of the DE and the result of the same image with a bar in the center of the shape for the hearts. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.



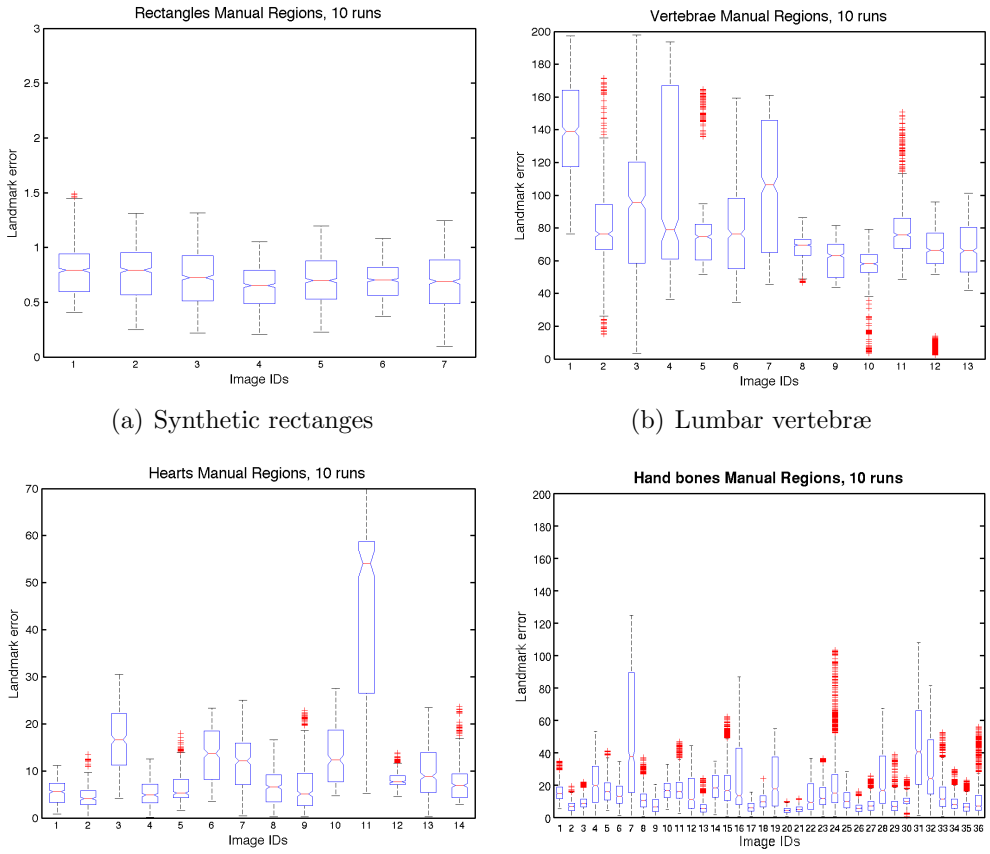
(a) Best Result (b) Result with Bar

Figure 23: Images of the best result of the DE and the result of the same image with a bar in the center of the shape for the metacarpal bones. The red points represent the hand annotated ground truth of the test shape. The shape resulting form the DE is drawn in with blue points.

4.3 Discussion

Taking a closer look on the results it can be observed that the proposed Particle Filter approach performed quite well on the evaluated data. Nevertheless the achieved results are influenced by certain factors. These are on the one hand the manually defined region map, that was used for image features extraction and classification and on the other hand the feature extraction method itself. De Bruijne's Shape Particle Filter [11, 10] implementation used region maps where the predefined regions were drawn in around at least four lumbar vertebræ. So the critical areas between two vertebræ could not lead to wrong results. The solution proposed in the scope of this technical report used only regions around one single vertebra. This was done to sustain the Shape Particle Filter's generality (e.g. to cope with different medical images and anatomical structures like metacarpal bones or hearts). Furthermore the image quality of the evaluated medical data was quite bad due to image artifacts and different radiograph angles. Images with higher resolution and less artifacts could also possibly boost the performance. Due to these circumstances the results on the lumbar vertebræ where not as good as expected.

Focusing on the robustness of the segmentation results, Particle filters



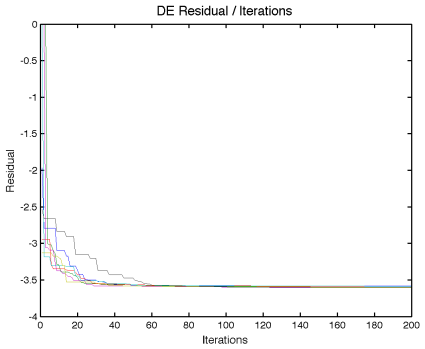
(a) Synthetic rectanges

(b) Lumbar vertebræ

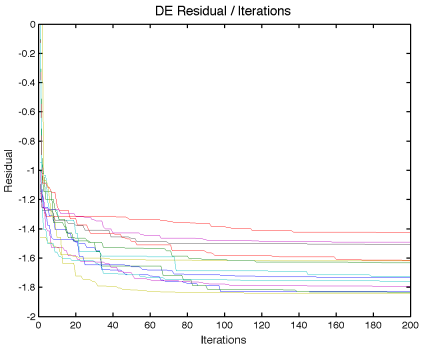
(c) Hearts

(d) Metacarpal bones

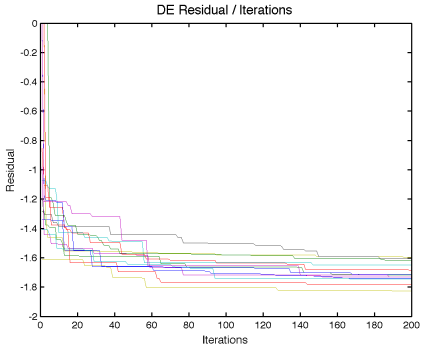
Figure 24: Boxplots of the euclidean distances between the hand annotated ground truth and the shape resulting from the DE for 10 LOOCV runs. For the robustness test a bar was added in the center of the shape. The image IDs represent the respective test shapes.



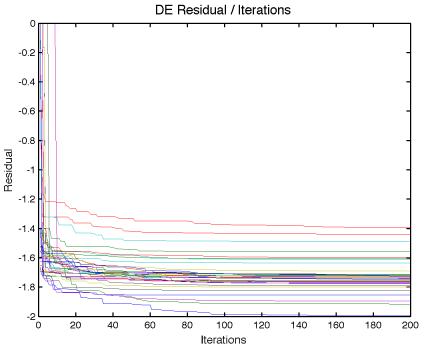
(a) Synthetic rectanges



(b) Lumbar vertebræ

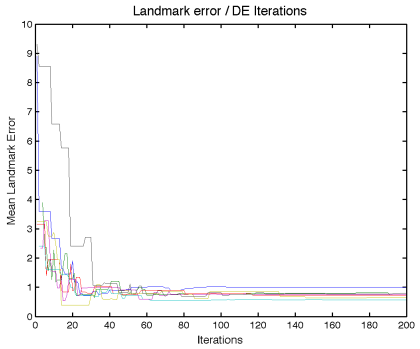


(c) Hearts

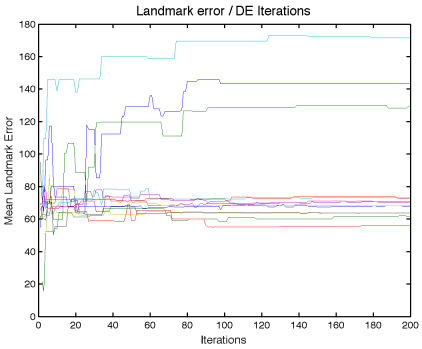


(d) Metacarpal bones

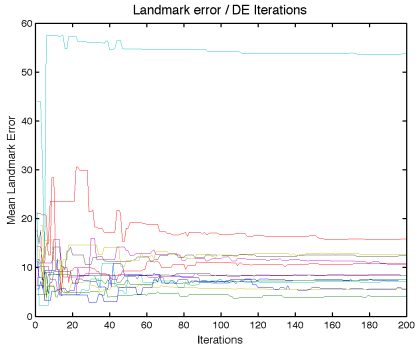
Figure 25: Plots of the DE residual change during the DE iterations for the robustness test.



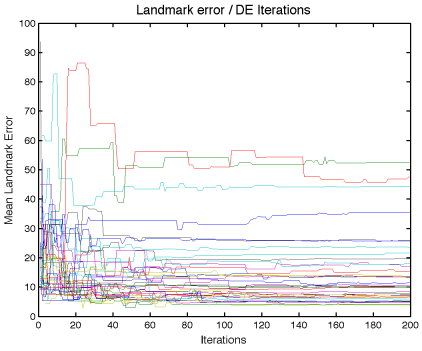
(a) Synthetic rectanges



(b) Lumbar vertebræ



(c) Hearts



(d) Metacarpal bones

Figure 26: Plots of the landmark error change during the DE iterations for the robustness test.

outperform ASMs [15, 19] and AAMs [1, 8, 7]. The results did not significantly change for the hearts and metacarpal bones after adding a bar in the middle of the shape. The computational performance, e.g. the calculation time, of the proposed solution is strongly influenced by the image feature classification done by SVMs. Although classification was first done by simply applying k-NN, later by a combination of kd-tree and k-NN and finally by SVMs the classification step remains the main chokepoint regarding computation time.

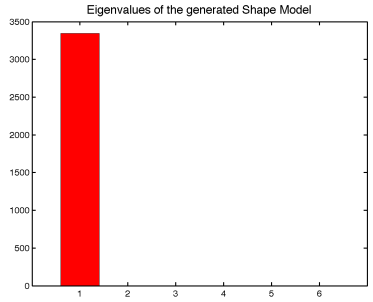
By applying the above mentioned adoptions and enhancements the proposed solution could offer the performance for practical use.

5 Conclusion and Outlook

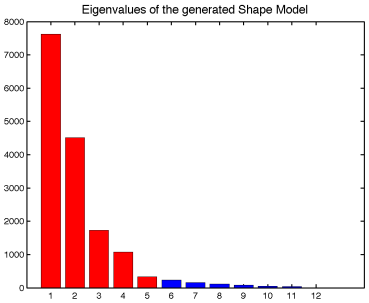
Shape Particle Filters as proposed above provide promising results for image segmentation and shape detection. Unfortunately the obtained results did not achieve the desired accuracy (mean landmark error < 5 pixel for 90% of the tested shapes). This lack of precision on the one hand depends on the created region map, that was used for classification and on the other hand on the extracted features. By replacing the predefined manually set region map consisting only of 6 regions with an automatically computed region map it could be possible to enhance the algorithms accuracy. This automatic region map could for example be generated by clustering the features around the shape of interest. Furthermore a more problem specific selection of features for the later image classification could lead to an additional performance boost. Finally higher quality images with a higher resolution and less image artifacts would significantly enhance the segmentation results.

6 Appendix

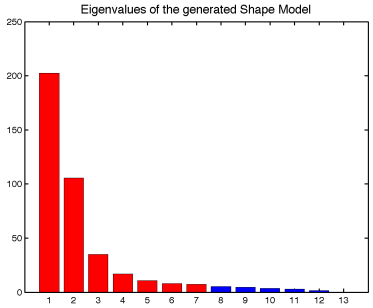
In this section the figures of the eigenvalues and eigenmodes for each data type are presented. Detailed description to each image is given in the respective caption.



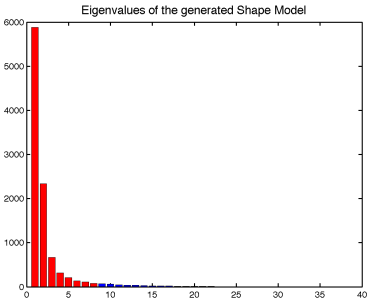
(a) Synthetic rectangles



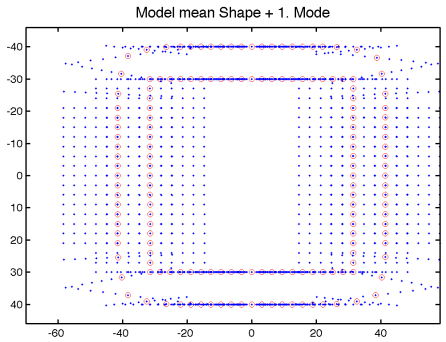
(b) Lumbar vertebræ



(c) Hearts



(d) Metacarpal bones



(e) Mean shape and modes of synthetic rectangle

Figure 27: Bar diagrams of the calculated eigenvalues. The red bars represent 95% of the data point variance. 27(e) shows the mean shape of shape model (red) and generated modes (blue) of the synthetic rectangles.

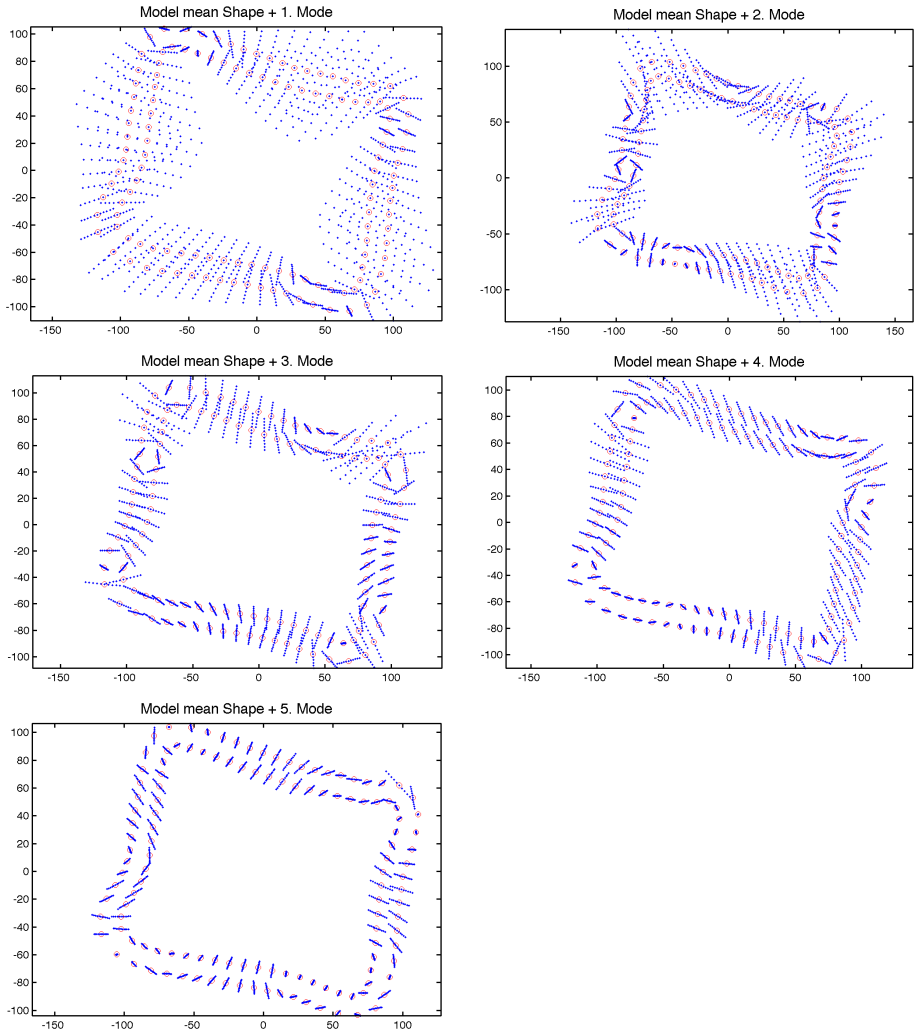


Figure 28: Mean shape of shape model (red) and the generated modes (blue) of the vertebræ. The modes show the data point variance. First they are located equally, but then the distances decrease with decreasing modes.

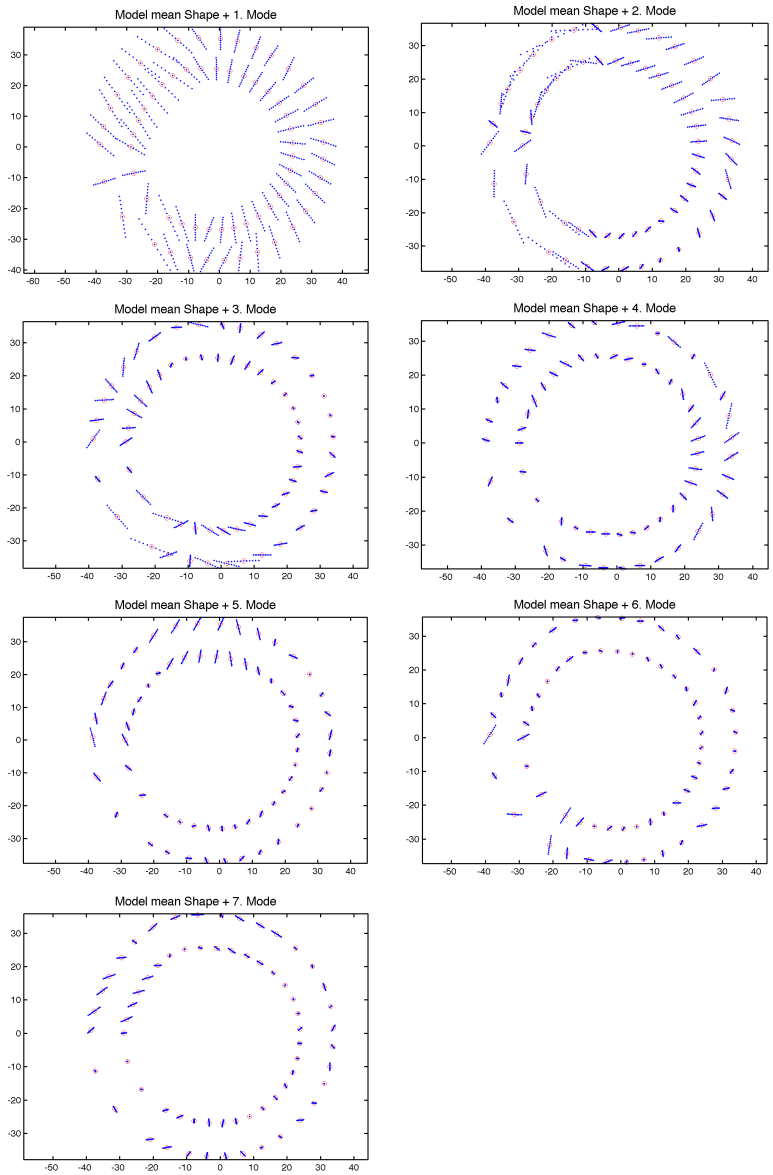


Figure 29: Mean shape of shape model (red) and generated modes (blue) of the hearts. The modes show the data point variance. First they are located equally, but then the distances decrease with decreasing modes.

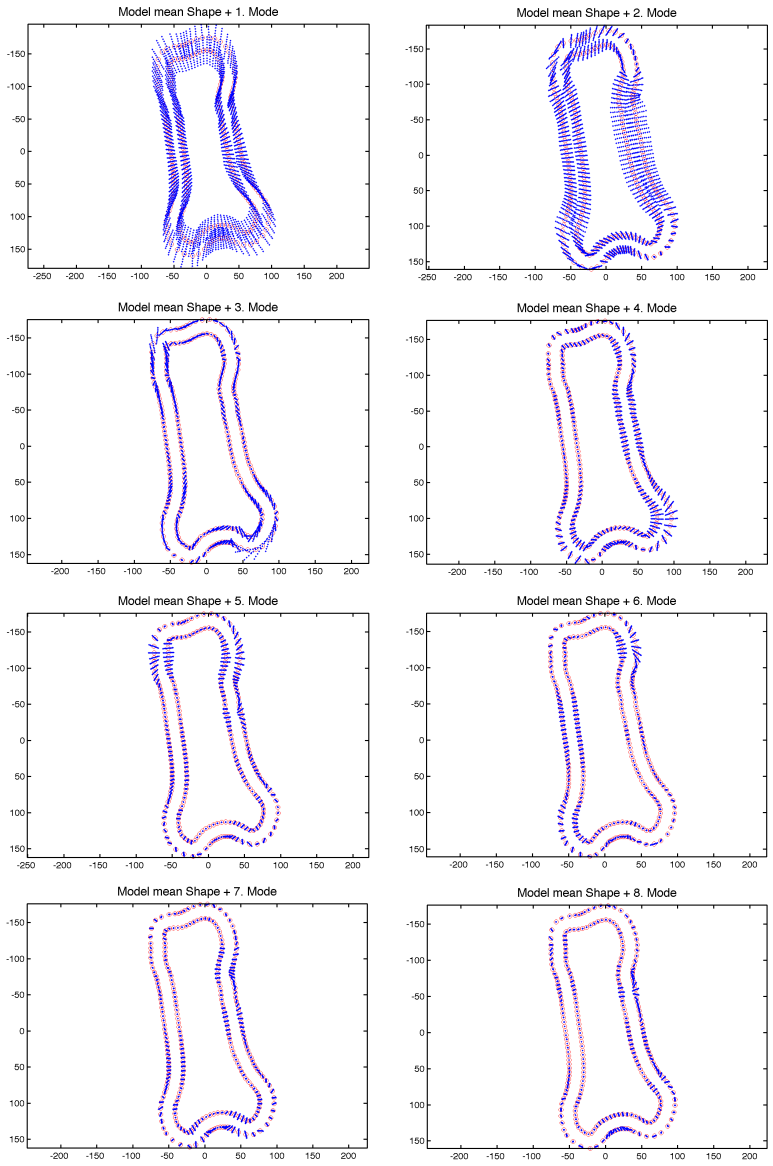


Figure 30: Mean shape of shape model (red) and generated modes (blue) of the metacarpal bones. The modes show the data point variance. First they are located equally, but then the distances decrease with decreasing modes.

References

[1] R. Beichel, H. Bischof, F. Leberl, and M. Sonka. Robust active appearance models and their application to medical image analysis. *IEEE transactions on medical imaging*, 24(9):1151–1169, 2005.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509 – 517, Jan 1975.

[3] C. M. Bishop. Pattern recognition and machine learning. *Published by Springer*, Jan 2006.

[4] F. Bookstein. Principal warps: thin-plate splines and the decomposition of deformations. *Pattern Analysis and Machine Intelligence, IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567 – 585, Jun 1989.

[5] C. Braak. A markov chain monte carlo version of the genetic algorithm differential evolution: easy bayesian computing for real parameter spaces. *Stat Comput*, 16(3):239–249, Jan 2006.

[6] C. Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Jan 1998.

[7] T. Cootes and C. Taylor. Active shape models - 'smart snakes. *In Proc. British Machine Vision Conference*, pages 266–275, Jan 1992.

[8] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. *World Wide Web Publication*, Jan 2001.

[9] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Training models of shape from sets of examples. *In Proc. British Machine Vision Conference*, pages 9–18, Jan 1992.

[10] M. de Bruijne and M. Nielsen. Image segmentation by shape particle filtering. *17th International Conference on Pattern Recognition (ICPR'04)*, 3:722–725, 2004.

[11] M. de Bruijne and M. Nielsen. Shape particle filtering for image segmentation. *Lecture Notes in Computer Science, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2004*, 3216:168–175, Jan 2004.

[12] P. Fearnhead. Sequential monte carlo methods in filter theory. Jun 2008.

[13] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEEE Proceedings for Radar and Signal Processing*, 140(2):107–113, Jan 1993.

[14] P. Grunwald. A tutorial introduction to the minimum description length principle. Jan 2004.

[15] A. Leonardis and H. Bischof. Robust recognition using eigenimages. *COMPUTER VISION AND IMAGE UNDERSTANDING*, 78(1):99–118, 2000.

[16] J. Movellan. Tutorial on gabor filters. *Tutorial paper http://mplab. ucsd. edu/tutorials/pdfs/gabor.pdf*, 1996.

[17] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, Jun 1901.

[18] B. Scholkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. *Neural Computation*, Jan 2000.

[19] D. Skocaj, H. Bischof, and A. Leonardis. A robust pca algorithm for building representations from panoramic images. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 761–775, 2002.

[20] L. Smith. A tutorial on principal components analysis. *Cornell University*, Jan 2002.

[21] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), Jan 1997.

[22] H. H. Thodberg. Minimum description length shape and appearance models. *Proceedings of Information Processing and Medical Imaging*, 2003.

[23] V. N. Vapnik. The nature of statistical learning theory. *Published by Springer*, 2000.

[24] H. Yoshimura, M. Etoh, K. Kondo, and N. Yokoya. Gray-scale character recognition by gabor jets projection. *15th International Conference on Pattern Recognition*, 2:335–338, Jan 2000.

Visualisierung von Trajektorien live in Microsoft Bing Maps

Timo Kropp
Supervisor: Robert Sablatnig

Pattern Recognition and Image Processing Group
Institut für rechnergestützte Automation
Technische Universität Wien
Favoritenstr. 9/183-2
A-1040 Wien Österreich
Tel.: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: e0627880@student.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

Bachelorarbeit Wien, 30. Oktober 2009

Visualisierung von Trajektorien live in Microsoft Bing Maps

Timo Kropp

Unter Anleitung von a.o.Univ.-Prof. Robert Sablatnig

Abstract

Hochauflösende Luftaufnahmen der gesamten Erdoberfläche werden von interaktiven Kartendiensten (z.B. Google Earth, Microsoft Bing Maps) frei im Web angeboten. Diese Webdienste bieten offene Programmierschnittstellen an, die flexible Erweiterungen ermöglichen. Insbesondere für das Verfolgen (engl. Tracking) von bewegten Objekten unter freiem Himmel kann das frei verfügbare Bildmaterial Szeneneinblicke aus der Vogelperspektive bieten, die ansonsten nicht möglich wären. Für diese Arbeit wurde eine Software implementiert, die ein bestehendes Tracking System in Microsoft Bing Maps integriert. Mittels einer Abbildungsfunktion werden Trajektorien in Echtzeit vom Kamerabild auf Microsofts virtueller Erde abgebildet. Es wird gezeigt, dass trotz starken Unterschieden in der Perspektive und der Auflösung von Kamera und Luftbild in mehr als 80 Prozent der untersuchten Szenarien die korrekte Abbildungsfunktion bis auf mindestens 5 Pixel genau bestimmt werden kann. Dies eröffnet neue Möglichkeiten der Szenendarstellung außerhalb des Kamerabildes und zur innovativen Visualisierung von Trajektorien in automatischen Videoüberwachungssystemen.

1 Einleitung

In der Videoüberwachung spielt die Echtzeit-Visualisierung von erkannten Ereignissen und bewegten Objekten, wie z.B. Fahrzeuge und Personen eine wichtige Rolle [3]. Insbesondere das gleichzeitige Tracking mittels einer statischen Kamera von einer so großen Anzahl an bewegten Objekten, so dass die Visualisierung dieser Daten im Kamerabild für einen Menschen nicht mehr intuitiv verständlich ist, erfordert alternative Darstellungsformen. Um solche Anforderungen für Szenarien unter freiem Himmel erfüllen zu können, bietet der Webdienst Bing Maps von Microsoft hochauflösende Satellitenfotos der Erdoberfläche, welche für die Echtzeit-Visualisierung von Tracking Daten einer Überwachungskamera benutzbar sind.

Mit der Visualisierung von bewegten Objekten auf der Erdoberfläche, haben sich ebenfalls Brown und Dunn [2] beschäftigt. Sie haben die Tracking Daten von Flugzeugen und anderen bewegten Objekten mittels des Global Positioning Systems (GPS) bestimmt und diese auf Landkarten und Bildern der Erdoberfläche dargestellt. Im Unterschied dazu werden in dieser Arbeit die Tracking Daten mittels einer statischen Überwachungskamera bestimmt.

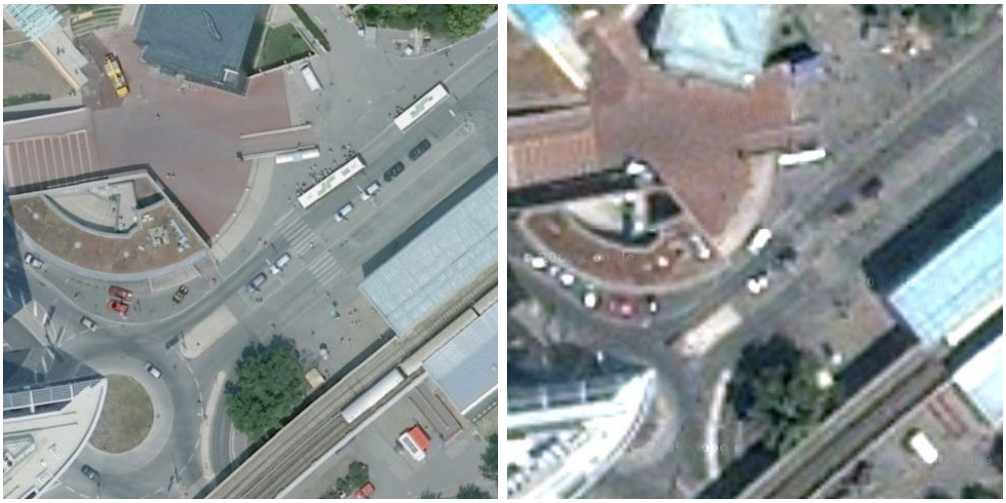
Es wurde eine Anwendung in C++ entwickelt, die ein vorhandenes Tracking System (siehe Kapitel 4.1) in Microsoft Bing Maps integriert. Im Kern der Applikation wird zuerst über mehrere Punktkorrespondenzen im Kamera- und Satellitenbild eine Abbildung zwischen zwei Ebenen ermittelt, die als Homographie bezeichnet wird. Anhand dieser werden anschließend die Tracking Daten in Echtzeit auf die Erdoberflächendarstellung von Bing Maps transformiert. In dieser Arbeit wird mit Hilfe einer Monte Carlo Simulation untersucht, wie genau die Homographie in Hinsicht auf die Auflösung der Satellitenfotos bestimmt werden kann. Des Weiteren wird geprüft, von welchen Faktoren die Genauigkeit der Abbildungsfunktion abhängig ist.

Alle Untersuchungen beziehen sich ausschließlich auf Outdoor-Szenen, da Überwachungskameras innerhalb von Gebäuden keine notwendigen Bezugspunkte zu den Satellitenfotos von Bing Maps liefern können. Außerdem wird vorausgesetzt, dass sich die getrackten Objekte auf einer Ebene der Erdoberfläche bewegen. Anderenfalls ist die Homographie nicht zu bestimmen.

Im Folgenden wird zunächst auf die Funktionsweise des Tracking Systems eingegangen. Anschließend wird erläutert (Kapitel 3), wie die Homographie bestimmt werden kann. Mit diesen Voraussetzungen wird in Kapitel 4 die Implementierung und Funktionsweise der dieser Arbeit zugrunden liegenden Applikation beschrieben. Den Hauptteil dieser Arbeit bildet Kapitel 5, in dem

im ersten Abschnitt die manuelle Homographiebestimmung qualitativ untersucht wird und im zweiten Abschnitt die Abbildungsgenauigkeit mittels der Monte Carlo Simulation evaluiert wird. Abschließend werden die Ergebnisse zusammengefasst und ein Ausblick auf mögliche weiterführende Themen geboten.

1.1 Motivation



(a) Mircosoft Bing Maps (b) Google Earth

Abbildung 1: Screenshot der gleichen Szene in maximaler Zoomstufe (Stand Oktober 2009)

Öffentliche Kartendienste im Internet (z.B. Google Earth [4], Microsoft Bing Maps [8]) bieten neue Möglichkeiten der Visualisierung von Tracking Daten aus Videoüberwachungssystemen. Ihre Satellitenfotos decken die gesamte bewohnte Erdoberfläche ab, und es existieren Aufnahmen, deren Auflösung bis auf einen Meter genau ist [6]. Mit diesen Daten als Grundlage können über offene Programmierschnittstellen (siehe Kapitel 1.2), wie sie z.B. von Google Earth und Microsoft Bing Maps angeboten werden, beliebige Erweiterungen entwickelt werden. Für diese Arbeit wurde Bing Maps als Grundlage für die Visualisierung gewählt, da die Luftaufnahmen beim Standort der verwendeten Überwachungskamera eine bessere Qualität aufweisen als die in Google Earth (siehe Abbildung 1).

Um die Dienste für Visualisierungen nutzen zu können, bietet sich die Integration dieser in ein bestehendes Videoüberwachungssystem an.

1.2 Bing Maps

Microsoft Bing Maps beinhaltet eine 2D- und 3D-Darstellung der Erdoberfläche, auf der interaktiv navigiert werden kann. Im 2D-Modus können sowohl die Satellitenfotos mit oder ohne überlagerten Straßennamen als auch das Kartenmaterial wie in einem Stadtplan angezeigt werden. Es gibt 19 Zoomstufen, wobei jeweils für die Stufen 1 bis 9, 10 bis 14 und 15 bis 19 unterschiedliche Fotos verwendet werden. Für die Visualisierung von Trajektorien sind die Stufen 18 und 19 besonders geeignet, da hier die Auflösung so hoch ist, dass z.B. Autos erkannt werden können (siehe Abbildung 8 in Kapitel 5).

Bing Maps bietet für Entwickler ein „Asynchronous JavaScript and XML Application Programming Interface“ (AJAX API) und ein „Simple Object Access Protocol Application Programming Interface“ (SOAP API) an [9]. Letzteres wurde hauptsächlich für Enterprise Anwendungen entwickelt. Bing Maps wurde mittels der AJAX API in das zu dieser Arbeit gehörende Programm integriert. Diese bietet sich aufgrund der unmittelbaren Schnittstelle zum Frontend von Bing Maps für die Visualisierung von Trajektorien besonders an.

2 Tracking

Das Tracking System, welches für diese Arbeit verwendet wurde, basiert auf einem Point-Tracking Algorithmus [1]. Bei diesem Verfahren werden Regionen verfolgt, die über die Zeit ihre Position in einer Folge von Frames ändern, um ihre Bewegung zu erkennen. Die besonderen Anforderungen an das Verfahren und deren Implementierung sind durch die Ressourcen, wie die Taktrate und den verfügbaren Speicher eines Digitalen Signal Prozessors (DSP) bestimmt. Dieser ist in ein eigenständiges Hardware-Modul „Advanced Video Codec“ (AVC) integriert. In Abbildung 2 befindet sich der AVC zwischen dem Dekstop PC (hinten links) und der Kamera (vorne rechts). Dieser leitet nach erfolgter Berechnung die Tracking-Daten über ein Netzwerk an den Computer weiter.

Abbildung 2: Hardware des Trackingsystems: Desktop Computer, AVC und Kamera

Der Algorithmus besteht aus folgenden Schritten:

1. Vorverarbeitung
2. Segmentierung
3. Tracking

Die Vorverarbeitung beinhaltet die Bestimmung eines Hintergrundbildes. Die Eingabedaten hierfür sind das unbearbeitete Kamerabild im YUV-Farbraum. Das Resultat ist ein Grauwertbild, dessen normierte Werte die

Wahrscheinlichkeit angeben, ob es sich jeweils um ein Vordergrund- oder Hintergrundpixel handelt. Für die Hintergrundmodellierung wird der gleitende Mittelwert verwendet, welcher eine abgewandelte Form des Color Mean and Variance Algorithmus ist [11]. Anstatt die Varianz für jeden Pixel über eine bestimmte Zeit zu berechnen, wird ein fester Schwellwert verwendet.

Angewendet wird dieser im zweiten Verarbeitungsschritt der Vorder- und Hintergrund Segmentierung. Die Annahme konstanter Varianz für jeden Pixel liefert zwar etwas schlechtere Ergebnisse in der Hintergrundmodellierung [10], demgegenüber steht jedoch eine deutliche Verringerung der Berechnungszeit auf dem AVC, welche für die geforderte Verarbeitung von 25 Frames pro Sekunde notwendig ist. Eine weitere Einschränkung ergibt sich für die Integrationszeit des Hintergrundbildes, die festlegt wie schnell ein Objekt vom Zustand „bewegt“ in den Zustand „unbewegt“ in den Hintergrund integriert wird. Wegen der Begrenzung des AVC nur 16 Bit Arithmetik verwenden zu können, ist die Integrationszeit auf den Bereich von 20 bis 60 Sekunden beschränkt. Das Hintergrundbild wird durch folgende Update-Regel adaptiert:

$$\mu(x, y, t) = (1 - \alpha)\mu(x, y, t - 1) + \alpha p(x, y, t), \tag{1}$$

wobei $\mu(x, y, t)$ den berechneten Mittelwert des Pixels an der Stelle x,y zum Zeitpunkt t darstellt und $p(x, y, t)$ der Intensität des Pixels an der Stelle x,y entspricht. $\alpha \in [0, 1]$ wird als Lernrate bezeichnet. Je kleiner der Wert von α ist, desto größer wird die Integrationszeit der Hintergrundadaption.

Der zweite Verarbeitungsschritt, der die Vorder- und Hintergrund Segmentierung beinhaltet, führt ein sogenanntes Labeling auf dem Binärbild durch, welches zuvor aus der Differenz vom Hintergrundbild und dem aktuellen Frame mittels eines Schwellwertes gebildet wurde. In Abbildung 3 ist links oben ein Hintergrundbild dargestellt, welches aufgrund der Mittelwertberechnung jedes Pixels keine bewegten Objekte wie z.B. Autos enthält. Das Differenzbild von diesem zu einem bestimmten Frame ist in der Mitte von Abbildung 3 abgebildet. Dieses enthält nur die Unterschiede der beiden Bilder, welche in diesem Fall Autos und Personen sind. Nach Anwendung des Schwellwertes erhält man das dargestellte Binärbild. Der Schwellwert entspricht der Sensitivität bezüglich der Erkennung des Vordergrundes und ist durch den AVC festgelegt. Beim Labeling werden zusammenhängende Vordergrundregionen (Blobs) eine eindeutige Zahl (Label) zugeordnet. Anschließend werden für jeden Blob im Bild die Bounding Box, der Schwerpunkt und die Anzahl der enthaltenen Pixel ermittelt.

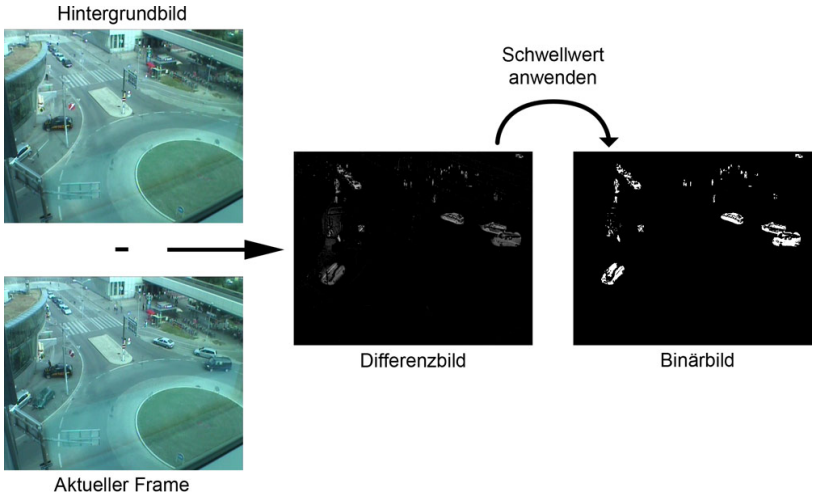


Abbildung 3: Berechnung des Binärbildest mittels der Differenz aus Hintergrundbild und aktuellem Frame mit anschließender Anwendung des Schwellwertes.

Der letzte Verarbeitungsschritt des Algorithmus ist für die Erkennung von Positionsänderungen der erkannten Objekte (Blobs) über die Zeit zuständig. Hierfür werden jeweils zwei aufeinander folgende Frames i und $i+1$ miteinander verglichen. Unter Verwendung des Kalman-Filters [7] wird für jeden Blob in Frame i eine Vorhersage für seine Position in Frame $i+1$ getroffen. Diese Berechnung erfolgt auf Grundlage der zuvor ermittelten Positionsänderungen in den Frames $< i$. Die vorhergesagte Position in Frame $i+1$ wird als Startpunkt für die nachfolgende Nearest Neighbor-Suche verwendet. Dabei wird der Blob, dessen Entfernung der vorhergesagten Positionen am nächsten ist und dessen Größe (Anzahl der Pixel) ungefähr mit der des gesuchten Blobs übereinstimmt, als neue Objektposition angenommen. Für den Fall, dass kein Blob zugeordnet werden kann, wird angenommen, dass das Objekt entweder verdeckt ist oder das Sichtfeld der Kamera verlassen hat.

Die Tracking-Daten, die an das Netzwerk übermittelt werden, beinhalten für jeden Frame alle erkannten Objekte, ihre zugehörige Position und ihre Bounding-Box.

3 Manuelle Bestimmung der 2D Homographie

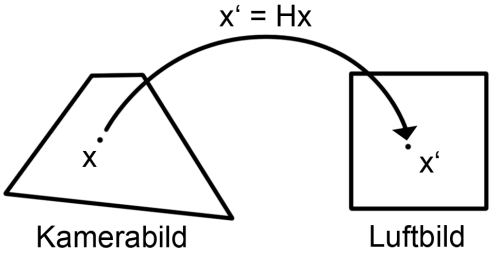


Abbildung 4: Skizze der projektiven Abbildung zwischen den beiden Ebenen des Kamerabildes und des Luftbildes

Um die detektierten Objekte im Kamerabild an der korrekten Position auf dem Satellitenbild darzustellen, ist eine Abbildung notwendig, die jedem Pixel des Kamerabildes einen entsprechenden Punkt im Satellitenbild zuordnet. Im Programmfluss der Visualisierungs-Applikation wird die Bestimmung der Homographie als Kalibrierung bezeichnet. Dieser Schritt muss vom Benutzer manuell durchgeführt werden. Es wird die Annahme getroffen, dass alle getrackten Objekte sich in einer Ebene, auf der Erdoberfläche bewegen. Daraus folgt, dass die Abbildung einer projektiven Transformation (Homographie) $\mathbf{x}' \cong H\mathbf{x}$ entspricht (Abbildung 4). Wobei $\mathbf{x}$ und $\mathbf{x}'$ homogene 3-Vektoren des projektiven Raumes $\mathbb{P}^2$ sind und H eine 3×3 Matrix (2) ist. Alle Vektoren $\mathbf{x} = (x, y, w) \in \mathbb{P}^2$ mit $w \neq 0$ können mit $(x/w, y/w)$ auf ein Element des Vektorraums $\mathbb{R}^2$ zurückgeführt werden. Daraus folgt für $\mathbf{x} \in \mathbb{P}^2$, $t\mathbf{x} \equiv \mathbf{x}$ mit $t \in \mathbb{R} \setminus \{0\}$, d.h. der Vektor $\mathbf{x}$ ist in seiner homogenen Darstellung nur bis auf ein Vielfaches eindeutig bestimmt. Mit der homogenen Darstellung der Vektoren wird die Bestimmung der 2D Homographie zu einem linearen Problem (siehe Kapitel 3.1).

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \tag{2}$$

Es stellt sich nun die Frage, wie viele Punktkorrespondenzen $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ benötigt werden, um die Matrix H zu berechnen. Da aufgrund der homogenen Darstellung h_9 mit einem beliebigen $t \in \mathbb{R} \setminus \{0\}$ multipliziert werden kann, ist H nur auf ein Vielfaches eindeutig bestimmt. Daraus ergibt sich, dass

die Matrix genau 8 Freiheitsgrade besitzt. In der Ebene hat jeder Punkt 2 unabhängige Koordinaten und deswegen müssen vier Punktepaare für die eindeutige Berechnung von H gefunden werden. Diese müssen so gewählt werden, dass das im folgendem aufgestellte Gleichungssystem keine linear abhängigen Gleichungen enthält (siehe Kapitel 3.1).

Mit genau vier Punktepaaren in allgemeiner Lage kann H eindeutig bestimmt werden. Bei der manuellen Wahl der Punktkorrespondenzen können jedoch kleine Abweichungen der korrekten Position, durch Störungen oder Diskretisierungsfehler, auftreten. Wenn ein konstanter Fehler vorhanden ist, wird dieser mit der Wahl von z.B. 5, 6, 7 oder mehr Punktepaaren algebraisch minimiert. [5].

3.1 Direct Linear Transformation (DLT) Algorithmus

Gegeben sind $n \geq 4$ homogene 2D Punktkorrespondenzen $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$. Gesucht ist die Matrix H, so dass $\mathbf{x}'_i \cong H\mathbf{x}_i$ für alle i gilt. Da die Vektoren $\mathbf{x}'_i$ und $H\mathbf{x}_i$ aufgrund der homogenen Darstellung nur bis auf einen Skalierungsfaktor gleich sind, d.h. in dieselbe Richtung zeigen, jedoch in der Länge variieren, gilt $\mathbf{x}'_i \times H\mathbf{x}_i = 0$. Daraus kann folgendes lineares Gleichungssystem hergeleitet werden [5] mit $\mathbf{x}'_i = (x'_i, y'_i, w'_i)^T$:

$$A_i\mathbf{h} = \mathbf{0} \tag{3}$$

$$\Rightarrow \begin{bmatrix} \mathbf{0}^T & -w'_i\mathbf{x}_i^T & y'_i\mathbf{x}_i^T \\ w'_i\mathbf{x}_i^T & \mathbf{0}^T & -x'_i\mathbf{x}_i^T \\ -y'_i\mathbf{x}_i^T & x'_i\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \mathbf{h} = \mathbf{0} \tag{4}$$

A_i ist eine 3×9 Matrix und es gilt $\mathbf{h} = (h_1, ..., h_9)^T$.

Der DLT-Algorithmus zur Bestimmung von H besteht nun aus folgenden Schritten [5]:

1. Für jedes Punktepaar $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ ist die Matrix A_i, siehe (3) zu bestimmen. Dabei kann die dritte Zeile von A_i ausgelassen werden, da diese aufgrund der homogenen Koordinaten bis auf einen Skalierungsfaktor von den ersten beiden Zeilen linear abhängig ist.

2. Die n 2×9 Matrizen A_i sind zu einer einzigen $2n$×9 Matrix A zusammen zu setzen.

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix} \tag{5}$$

3. Von der Matrix A sind die Singulärwerte und -vektoren mittels SVD (Singulärwertzerlegung) zu bestimmen. Der singuläre Einheitsvektor, der zum kleinsten Singulärwert gehört, ist die Lösung $\mathbf{h}$. Es gilt $A = UDV^T$ mit der positiven Diagonalmatrix D. Sind dessen Werte in absteigender Größe angeordnet, dann ist $\mathbf{h}$ die letzte Spalte von V.

4. Die Matrix H ist aus $\mathbf{h} = (h_1, ..., h_9)^T$ wie in (2) zu bestimmen.

Der DLT-Algorithmus minimiert mit mehr als vier Punktkorrespondenzen die Norm $\|A\mathbf{h}\|$, da das lineare Gleichungssystem (4) in diesem Fall überbestimmt ist. $A\mathbf{h}$ wird auch als „residual vector“ bezeichnet.

3.2 Normalisierter DLT Algorithmus

Der DLT-Algorithmus hat den Nachteil, dass die resultierende 2D Homographie von den jeweils gewählten Koordinatensystemen in dem die Punkte gemessen werden, abhängig ist. Das bedeutet, dass der algebraische Fehler bei mehr als vier Punktkorrespondenzen nicht invariant bzgl. einer Transformation des Bildes ist [5].

Aus diesem Grund wird die Genauigkeit der Ergebnisse, mittels einer vorherigen Normalisierung der Datenpunkte, erheblich verbessert [5]. Der normalisierte DLT-Algorithmus besteht aus folgenden zusätzlichen Schritten:

1. Normalisierung der Punkte $\mathbf{x_1}, ..., \mathbf{x_k}$ mittels einer Transformation T, die aus einer Translation und einer Skalierung besteht. Der Mittelpunkt alle Punkte ist anschließend $(0, 0)^T$ und der durchschnittliche Abstand zum Mittelpunkt beträgt dann $\sqrt{2}$.

2. Normalisierung der Punkte $\mathbf{x_1}', ..., \mathbf{x_k}$ entsprechend des 1. Schrittes mittels einer Transformation T'.

3. Durchführung des DLT-Algorithmus um die Homographie $\tilde{H}$ zu bestimmen (siehe Kapitel 3.1).

4. Denormalisierung mit $H = T'^{-1}\tilde{H}T$.

4 Implementierung

Für diese Arbeit wurde eine Applikation TraVis implementiert, die eine Schnittstelle zwischen Bing Maps und dem Tracking System bildet.

4.1 Systemumgebung des AIT und Integration

Der hardwareseitige Systemablauf besteht aus den Komponenten: Kamera, AVC, Netzwerk und Client-PC (siehe Abbildung 5). Es werden die aufgezeichneten Frames des CCD oder CMOS Sensors einer Überwachungskamera in Echtzeit auf dem AVC verarbeitet. Nach erfolgtem Tracking pro Frame werden die Ergebnisse inklusive der Bilddaten in ein 100 MBit Ethernet Netzwerk mittels des Real-Time Transport Protocol (RTP) übertragen. Ein beliebiger Desktop Rechner, der in dem Netzwerk integriert ist empfängt die Tracking-Daten und visualisiert diese mit der Applikation TraVis.

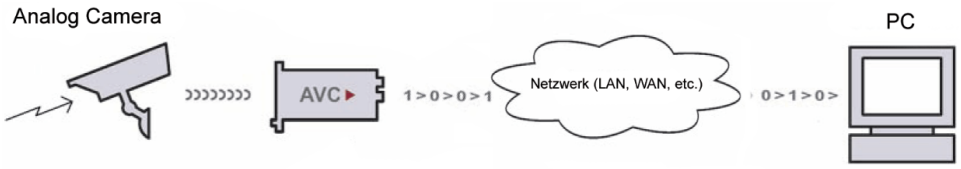


Abbildung 5: Datenfluss des Video Capturing

Durch die kompakte Schnittstelle über das Netzwerk kann die Applikation zur Visualisierung der Trajektorien ohne Änderungen des Tracking Systems unmittelbar integriert werden. Für die Einbindung müssen lediglich die entsprechenden IP-Adressen für die Übertragung festgelegt werden und die eingehenden Daten des AVCs müssen korrekt interpretiert werden.

4.2 TraVis

Die Anwendung ist in 2 Dialogfenster (Kamera Bild und Bing Maps) und einen Menübereich unterteilt. Für dessen Implementierung wurden die Microsoft Foundation Classes (MFC) verwendet, welche eine Sammlung objektorientierter Klassenbibliotheken darstellen.

In dem Dialogfenster, welches das Live Bild der Kamera enthält, werden die Bounding Boxes von erkannten Objekten zusammen mit dessen Trajektorie angezeigt. Ihr Verlauf wird durch den Schwerpunkt der getrackten

Objekte bestimmt. Sobald ein Objekt aus dem Bild verschwindet oder nicht mehr wieder erkannt wird, werden die eingezeichneten Grafikelemente ebenfalls gelöscht. Nach erfolgreicher Kalibrierung werden in der Luftaufnahme der entsprechende Szene, die erkannten Objekte als Punkte und dessen Trajektorien synchron zum Tracking eingezeichnet (siehe Abbildung 6).

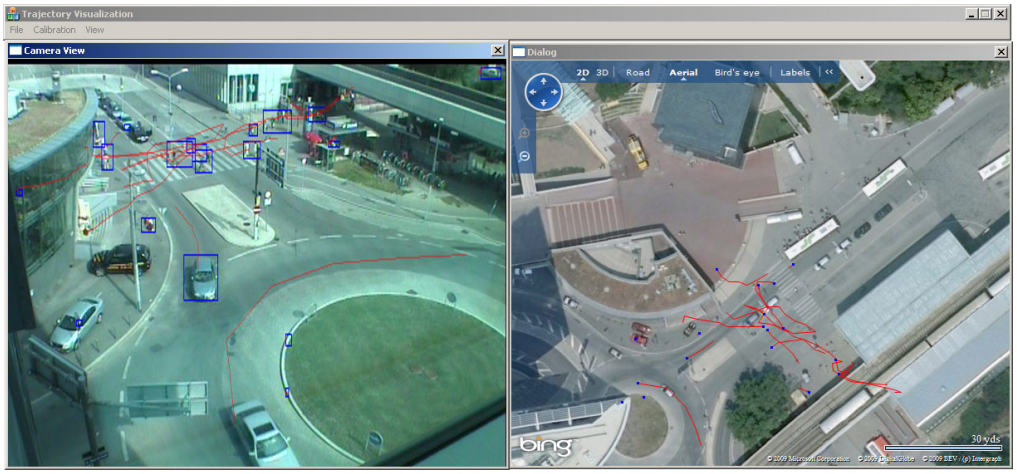


Abbildung 6: Applikation TraVis zur Visualisierung von Trajektorien

Den Kern der Softwarearchitektur von TraVis bilden 8 Klassen (siehe Abbildung 7). Davon stellt die Klasse `CTraVisDlg` den Unterbau dar, über den die Kommunikation zwischen den beiden Dialogen (`CCameraDlg`, `CVirtualEarthDlg`) gesteuert wird. Desweiteren gibt es die Klassen `CTrajectoryManager`, welche für das Verwalten aller Trajektorien und deren Koordinaten in beiden Darstellungen zuständig ist und `CMarkerManager`, der von `CTraVisDlg` im Kalibrierungsmodus verwendet wird. Den mathematischen Teil, die Berechnung der Homographie übernimmt die Klasse `CHomography`. Dessen Methode `caclH()` wird von `CMarkerManager` aufgerufen, sobald eine korrekte Konstellation an Markern vorhanden ist oder diese geändert wurde. Bing Maps wird ausschließlich über ein von Microsoft definiertes AJAX Interface angesprochen. Aus diesem Grund ist `CVirtualEarthDlg` von der MFC Klasse `CDHtmlDialog` abgeleitet. Diese bietet einen Html und JavaScript Interpreter an. Über die Methode `CallJScript()` werden die JavaScript Methoden aufgerufen, die als Schnittstelle zu Bing Maps agieren.

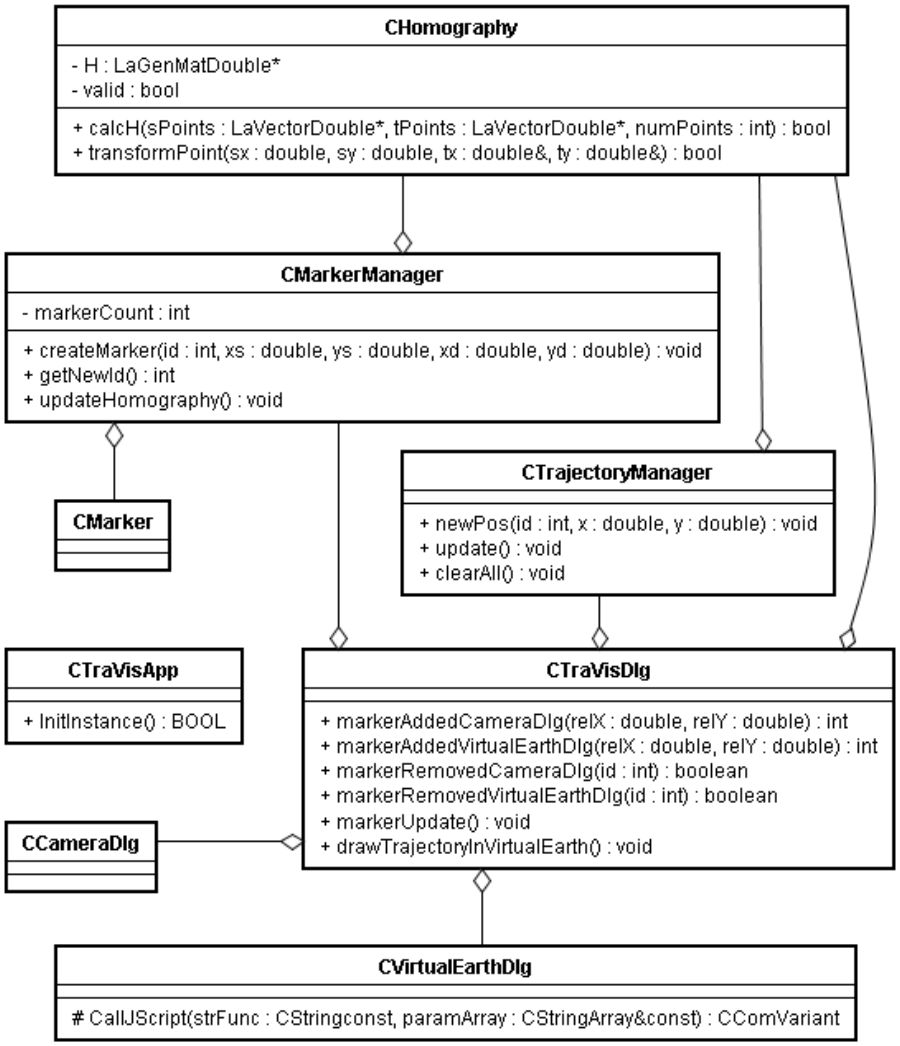


Abbildung 7: Klassendiagramm

5 Evaluierung

Im Folgenden soll untersucht werden, (I) wie genau die manuelle Kalibrierung auf Basis qualitativer Vergleiche durchgeführt werden kann, und (II) mit Hilfe einer Monte Carlo Simulation eine quantitative Aussage über die Fehleranfälligkeit der manuellen Kalibrierung gemacht werden.

Es wird angenommen, dass auf die Pixelgröße des Kamerabildes ein kleinerer oder gleich großer Bereich abgebildet wird, als auf die Größe der Pixel in der Luftaufnahme. Mit den aktuell vorhandenen Luftbildern von Bing Maps trifft diese Annahme im Allgemeinen zu, solange die statische Kamera in einem nicht zu flachen Winkel auf die Erdoberfläche schaut. Im Folgenden wird der Winkel zwischen der Geraden von der Kamera senkrecht zur Erdoberfläche und der Geraden in Blickrichtung der Kamera als flach oder groß bezeichnet, wenn dieser größer als 50 Grad ist. Im Fall von weniger als 50 Grad wird er als steil oder klein bezeichnet. Unter diesen Vorrausetzungen ist die Positionsgenauigkeit der Marker für die Kalibrierung durch die Auflösung der Luftaufnahmen von Bing Maps beschränkt, da dessen Pixel in jedem Fall einen kleineren Bereich abbilden.

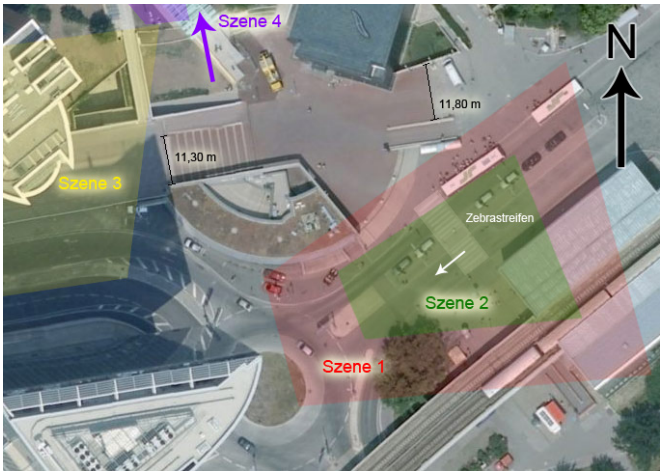


Abbildung 8: Luftbild aus Bing Maps mit gekennzeichneten Szenen

Der Standort der Überwachungskamera, die für die Evaluierung verwendet wurde, befindet sich im Tech Gate Tower in Wien. In Abbildung 8 sind die untersuchten Sichtfelder der verschiedenen Szenen dargestellt. Zu beachten ist, dass die Luftbilder schon mindestens drei Jahre alt sind und nicht

mehr vollständig mit der aktuellen Situation übereinstimmen. So ist z.B. der Zebrastreifen auf der Straße heute ein paar Meter südwestlicher als auf dem Luftbild (Abbildung 8). Alle Luftaufnahmen sind wie Abbildung 8 nach Norden ausgerichtet. Um die tatsächliche Auflösung in Bing Maps von dieser Szene in der maximalen Zoomstufe zu bestimmen, haben zwei unabhängige Messungen von 11,30 m bzw. 11,80 m eine abgebildete Pixelgröße von 0,24 m bzw. 0,21 m ergeben. Gemessen wurden die beiden Strecken, welche in Abbildung 8 jeweils durch eine schwarze Strecke eingezeichnet sind. Mit Hilfe der Pixeldifferenzen in horizontaler und vertikaler Richtung des End- und Startpunktes wurde die Länge (Anzahl der Pixel) der Strecke bestimmt. Die mittlere Pixelgröße entspricht nun dem Quotient aus gemessener Länge in Metern zu der Streckenlänge in Pixeln. Für das Kamerabild kann auf Grund der perspektivischen Projektion keine konstante Pixelgröße angegeben werden.

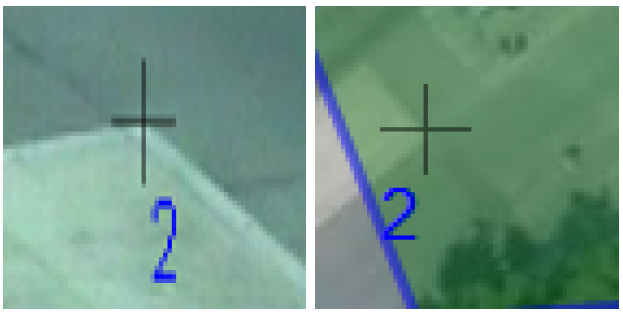
5.1 Setup / Experimentieraufbau

Abbildung 9: Kamera und Szenenausschnitt

Alle Aufnahmen, die für die Evaluierung verwendet wurden, stammen von einer statischen Kamera, die sich im 4. Stock befindet und auf die umliegenden Straße und Plätze blickt (Abbildung 9). Mit diesem Setup ist es möglich sowohl Szenen mit einem großen also auch welche mit einem flachen Kamerawinkel zu konstruieren.

5.2 Qualitative Studie

Für die qualitative Evaluierung der Kalibrierung wurden vier Testfälle mit jeweils unterschiedlichen Szenen erstellt, drei von ihnen mit einem großen Kamerawinkel (Abbildung 11, 12 und 13) und eine mit einem flachen Winkel (>50 Grad). Um zu prüfen wie sich die Anzahl der Marker auf die Kalibrierung auswirkt, werden die abgebildeten Sichtfelder der Kamera im Luftbild miteinander verglichen. Dessen Kanten können für einen anschaulichen Vergleich mit dem Kamerabild herangezogen werden. Der Testfall vier stellt mit seinem sehr großen Sichtfeld eine besondere Konfiguration dar. Es soll ermittelt werden, welchen Einfluss dieses Setup auf die Kalibrierung hat.



(a) Kamerabild (b) Luftbild von Bing Maps

Abbildung 10: Positionsvergleich eines Markerpaares mit Bildausschnitten von 70 x 70 Pixeln. Im Luftbild ist die Gesamtgröße 14,0 x 14,0 Metern mit durchschnittlicher Pixelgröße von 0,2 Metern. Das Kamerabild entspricht einem deutlich kleineren Ausschnitt.

In Abbildung 10 wird der Anfangs erwähnte Auflösungsunterschied deutlich. In Abhängigkeit der Schärfe von markanten Merkmalen im Bild wie z.B. der Ecke in dieser Abbildung können die Marker bis auf ein oder zwei Pixel genau platziert werden. In den drei nachfolgenden Abbildungen 11, 12 und 13 zeigt die erste Spalte jeweils ein Kamerabild von der zugehörigen Szene und in der zweiten Spalte befindet sich der entsprechende Ausschnitt von Bing Maps. Alle Bilder sind im Kalibrationsmodus der Software TraVis entstanden. Dabei nimmt die Anzahl der Marker von oben nach unten zu. Begonnen wird immer mit vier Markern.

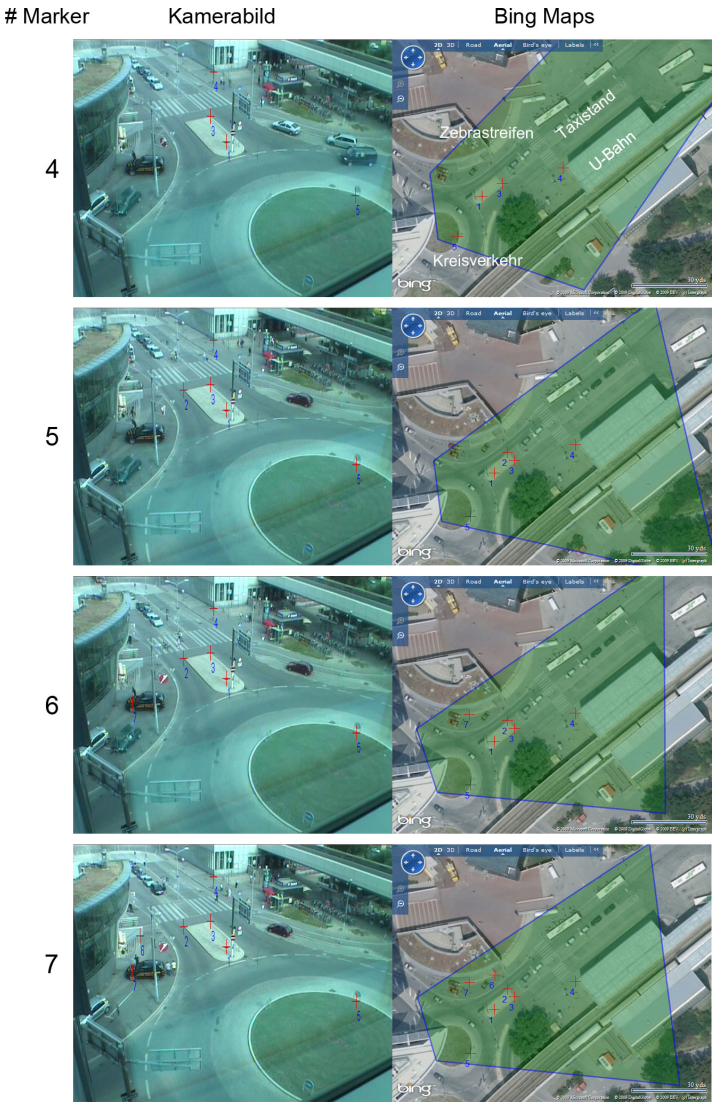


Abbildung 11: Szene 1, Kalibrierung mit unterschiedlicher Anzahl der Markern (rote Kreuze). In der rechten Spalte ist das projizierte Field of View abgebildet. Dessen Grenzen entsprechen mit steigender Anzahl an Markern immer besser der exakten Position, wie beim Vergleich mit den Rändern des Kamerabildes deutlich wird.

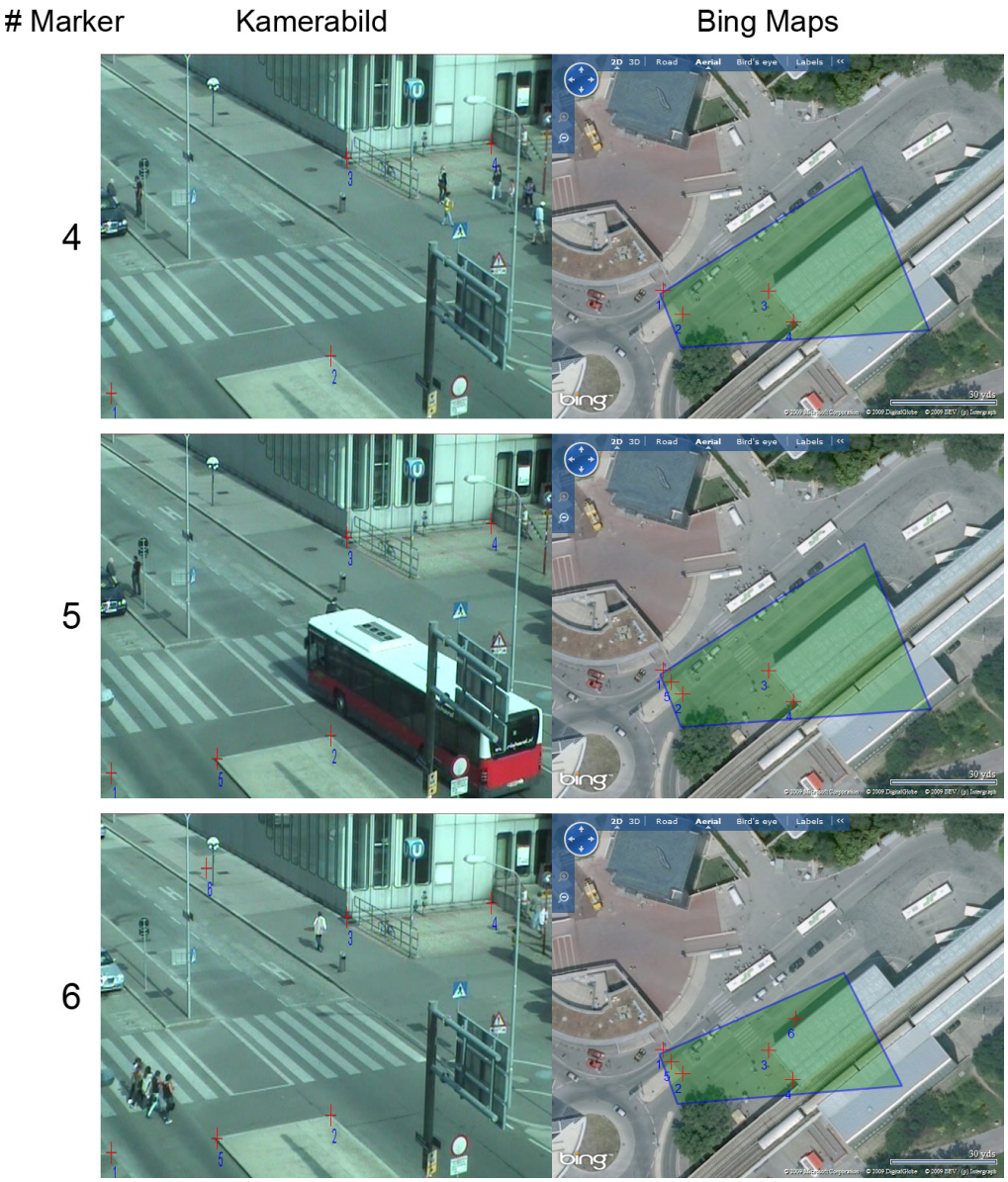


Abbildung 12: Szene 2, Kalibrierung mit unterschiedlicher Anzahl der Markern (rote Kreuze). In der rechten Spalte ist das projizierte Field of View abgebildet. Dessen Grenzen entsprechen mit steigender Anzahl an Markern immer besser der exakten Position, wie beim Vergleich mit den Rändern des Kamerabildes deutlich wird.

Testfall 1 (Abbildung 11) stellt eine typische Konfiguration mit einem großen Sichtfeld von ca. 20 Metern im vorderen Bereich und 40 Metern im hinteren Bereich der Kamera dar. Es ist ein Kreisverkehr mit einer abzweigenden Straße, über die ein Zebrastreifen verläuft, zu sehen. Angrenzend, oben rechts befindet sich eine U-Bahn Station aus der periodisch große Mengen (20 bis 80 Personen) an Menschen heraus strömen und zum Teil die Straße überqueren. Etwas weiter oben die Straße entlang befindet sich ein Taxistand (siehe Abbildung 11 erste Zeile, zweite Spalte). Aufgrund der Kameraperspektive und der nicht konsistenten Position des Zebrastreifens, gibt es nur relativ wenige Features im Bild die als präzise Markerpositionen genutzt werden können. Drei der vier Marker in der ersten Zeile liegen mit kleinen Abweichungen auf einer Gerade. Dadurch wird die Berechnung der Homographie numerisch instabil und das dargestellte Field of View weist entsprechend deutliche Fehler auf. Durch die Hinzunahme eines weiteren Markers wird die Transformation deutlich verbessert. Der sechste Marker schließlich beeinflusst den rechten oberen Bereich des Kamerabildes und bildet diese Ecke auf einen Bereich hinter der U-Bahn Station ab. Durch die Verdeckung der U-Bahnstation Kamerabild kann nicht die Korrektheit bestimmt werden, aber aufgrund des rechten Bildrandes der Kamera scheint diese Position schon auf wenige Meter genau zu sein. In der letzten Zeile wurden sieben Marker für die Kalibrierung verwendet. Im Vergleich zu der vorherigen Konfiguration mit sechs Marker gibt es nur noch minimale Änderungen des Field of Views. Am stärksten wird die linke obere Ecke des projizierten Kamerabildes verändert.

Folgender Testfall 2 (Abbildung 12) vermittelt einen ähnlichen Eindruck der Ergebnisse wie der Erste. Hier wurde die Kameraposition nicht geändert, jedoch mit Hilfe eines optischen Zooms ein deutlich kleinerer Bildbereich gewählt. Die größten Änderungen bilden hier die Positionen der Marker im Kamerabild. Diese haben im Mittel einen geringen Abstand zum Rand des Kamerabildes als die Markerpositionen in der ersten Szene. Dadurch konvergiert das projizierte Field of View schon nach 6 Markern auf einen stabilen Zustand hin.

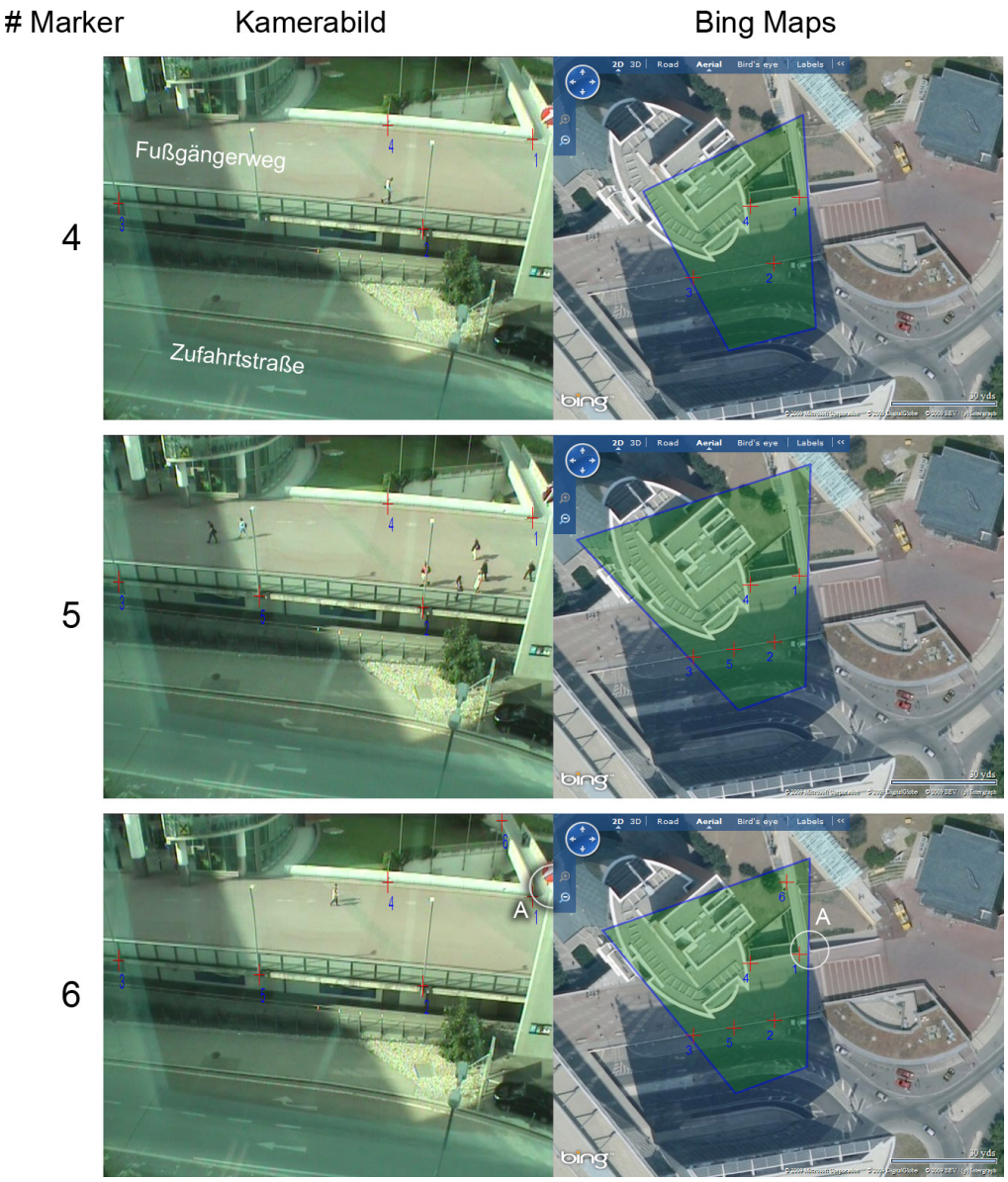


Abbildung 13: Szene 3, Kalibrierung mit unterschiedlicher Anzahl der Markern (rote Kreuze). In der rechten Spalte ist das projizierte Field of View abgebildet. Dessen Grenzen entsprechen mit steigender Anzahl an Markern immer besser der exakten Position, wie beim Vergleich mit den Rändern des Kamerabildes deutlich wird.

Die Homographie transformiert alle Punkte einer Ebene in eine zweite Ebene. Aus diesem Grund wurde als dritten Testfall (Abbildung 13) ein Schauplatz gewählt der nicht nur aus einer planaren Fläche besteht. Es befindet sich im unteren Teil des Bildes eine Zufahrtstraße und weiter oben eine breiter Fußgängerweg (Abb. 13, erste Zeile, erste Spalte). Die Marker wurden in diesem Fall auf dem Fußgängerüberweg platziert, um Trajektorien in diesem Bereich abbilden zu können. Objekte die sich auf der Zufahrtstraße befinden, können mit dieser Kalibrierung nicht korrekt transformiert werden, da diese sich nicht in der gleichen Ebene befinden. Auch in dieser Szene stabilisiert sich die Homographie von vier bis sechs Markern ähnlich wie in Szene 2. Zum Beispiel schneidet der rechte Rand des Kamerabildes eine kleine Brücke (siehe Abb. 13, 3. Zeile, Markierung A). Auch im Luftbild schneiden die Grenzen des projizierten Sichtbereiches diesen Punkt bis auf 2 Pixel genau.

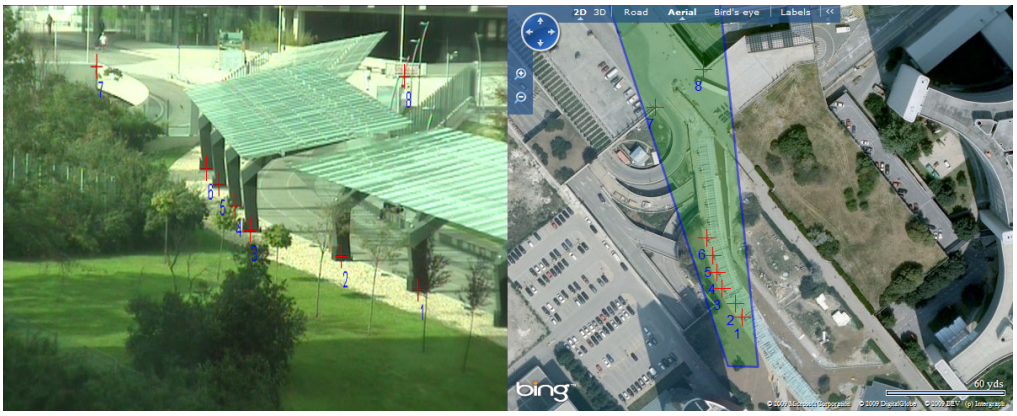


Abbildung 14: Szene 4, Kalibrierung mittels 8 Markern. In der rechten Spalte ist aufgrund des flachen Kamerawinkels der projizierte Field of View sehr schmal und in die Länge gezogen. Das Luftbild stammt aus der zweit höchsten Zoomstufe aus Bing Maps und deshalb entspricht jeder Pixel einem größeren Bereich als 0,2 Meter.

Die vierte Szene (Abbildung 14) stellt einen Grenzfall bzgl. der Homographie Bestimmung dar. Der am weitesten entfernte Bildbereich des Kamerabildes wird von einem Gebäude verdeckt, wodurch dort keine Marker für die Kalibrierung platziert werden können. Um für diese Szene die bestmöglichste projektive Transformation zu bestimmen wurden 8 Marker platziert. Das

Field Of View vermittelt in dieser Abbildung den Eindruck, dass zumindest im vorderen Bereich die Kalibrierung ähnlich genau wie in den Szenen zuvor bestimmt werden konnte. Ausschließlich der hintere Bereich verläuft um die gleiche Größe wie der sichtbare projizierte Field of View im Luftbild aus dem Bildausschnitt heraus.

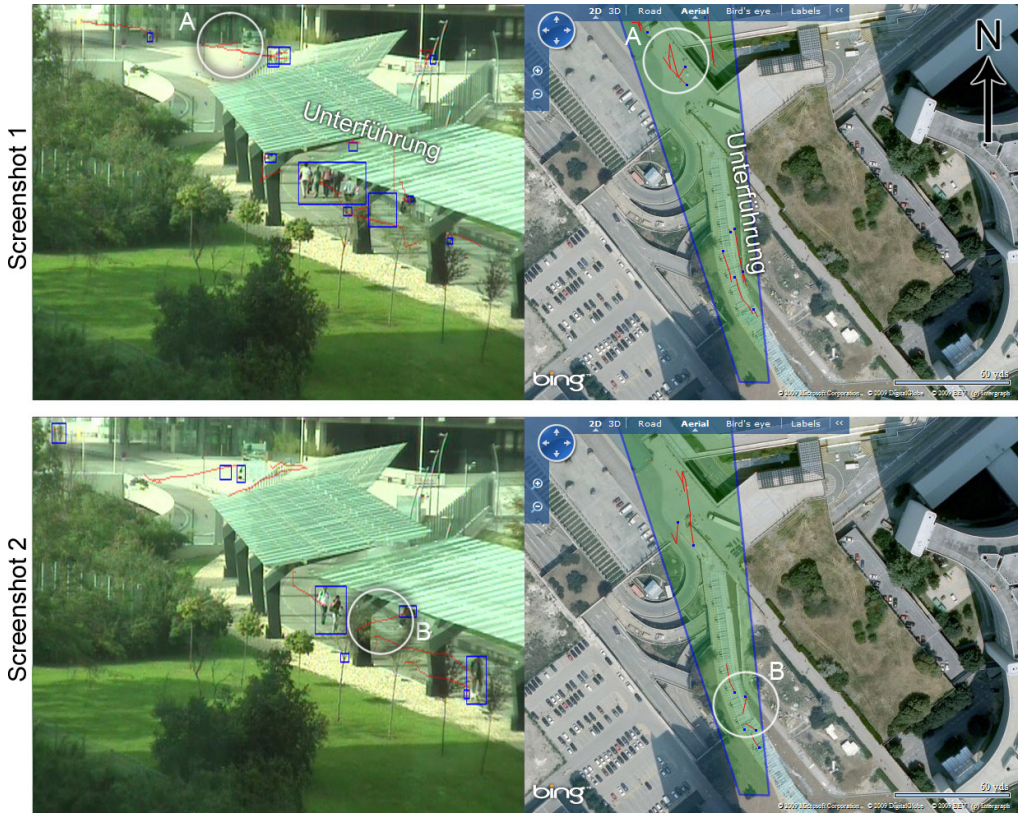


Abbildung 15: Szene 4, Screenshots aus TraVis für einen qualitativen Vergleich der abgebildeten Trajektorien. Bei den Markierungen A und B sind deutliche Fehler in der transformierten Position vorhanden.

Um die Kalibration der Szene 4 auf der Grundlage von transfomierten Trajektorien zu evaluieren, werden zwei Screenshots (Abbildung 15) aus der Applikation TraVis verglichen, denen die Kalibration aus Abbildung 14 zugrunde liegen. In Screenshot 1, Abbildung 15 werden die Trajektorien im Bereich der Unterführung bis auf ca. 6 Pixel genau im Luftbild abgebildet. Insbesondere Personen die sich in vertikaler Richtung auf dem Luftbild bewe-

gen, erfahren einen kleineren Fehler als welche, die sich in anderen Richtungen bewegen. Der zweite Screenshot enthält im vorderen Bereich eine Trajektorie, die kürzer als die benachbarten Trajektorien ist (Abbildung 15, B). Diese verläuft senkrecht zur Unterführung. Im Satellitenbild wird sie jedoch mit einer Bewegung Richtung Norden, statt einer Richtung Osten, abgebildet. Weitere Fehler treten im hinteren Bereich des Sichtfeldes der Kamera auf. Im ersten Screenshot zum Beispiel, befindet sich dort eine anormale Trajektorie (Abbildung 15, A). Für die transformierte Trajektorie ergibt dies eine Abweichung von ca. 30 Pixel Richtung Norden, was einer Rückwärtsbewegung des Objektes entspricht. Tatsächlich hat sich die getrackte Person in diesem Bereiche ausschließlich in gerader Richtung bewegt. Ähnliche Resultate können bei den benachbarten Trajektorien beobachtet werden.

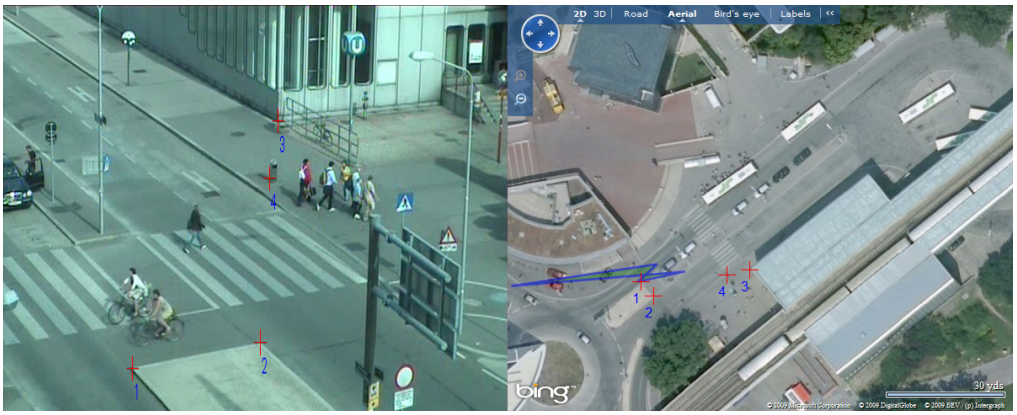


Abbildung 16: Fehlerhafte Kalibrierung aufgrund von mangelhaften Markerpositionen. Marker 2,3 und 4 liegen fast auf einer Linie. Das projizierte Field of View ist vollständig deformiert.

Weitere fehlerhafte Transformationen resultieren aus numerisch instabilen Lösungen des Gleichungssytems zur Bestimmung der Homographie. Dieser Fall liegt vor, wenn zu wenige Gleichungen numerisch stabil, linear unabhängig sind. Für die Marker entspricht dies einer Anordnung auf einer Geraden, d.h. mehrere Marker sind kollinear (Abbildung 16).

5.3 Quantitative Evaluierung

Die Qualitative Evaluierung (Kapitel 5.2) hat gezeigt, dass unter der Vorrausetzung eines Kamerawinkel von weniger als 50 Grad die Kalibrierung mit bis zu 7 Markern zur korrekten Lösung konvergiert. Im Folgenden werden auf Grundlage der bestimmten Kalibrierungen die Streuung transformierter Punkte innerhalb des Field of Views und somit die Robustheit der projektiven Transformation untersucht. Hierfür wird jeweils für die vier Szenen (siehe 5.2) eine Monte Carlo Simulation durchgeführt um quantitative Ergebnisse zu erhalten.

In einem regelmäßigen rechteckigen Gitter auf dem Kamerabild werden Punkte ausgewählt, die mit Gaussian noise gestört und ins Luftbild transformiert werden. Die Varianz der zweidimensionalen Gauß Verteilung beträgt in jedem Fall, sowohl für die x- also auch für die y-Komponente 3 Pixel. Um stabile Schätzungen zu erhalten, werden mindestens 10.000 Samples pro Pixel generiert. Die Ergebnisse werden auf zwei Arten visualisiert. In den folgenden Abbildungen befindet sich auf dem linken Bild die Darstellung der Fehlerellipsen pro gestörten Pixel. Hierbei handelt es sich um den Bereich im Luftbild auf den mehr als 99 Prozent der Gauß-Verteilung transformiert wird. Damit stellt sich heraus, wie störempfindlich die Homographie ist und auf welchen Bereich im Luftbild besonders kleine bzw. große Fehler gemacht werden. Die zweite Darstellungsform bildet eine interpolierte Farbcodierung der gemittelten Varianzen der Fehlerellipsen ab. Normiert ist diese auf den maximalen Wert. Hier wird die relative Störempfindlichkeit innerhalb des projizierten Field of Views deutlich.

Alle vier untersuchten Szenen spiegeln ein ähnliches Ergebnisse in Bezug auf die Änderungen der Standardabweichungen innerhalb des projizierten Field of Views wieder. Aufgrund der perspektivischen Projektion nimmt die Varianz von Vorne nach Hinten im Kamerabild zu. Dies ist durch die tatsächliche Pixelgröße im Bild bestimmt. Absolute Unterschiede ergeben sich durch die tatsächliche Größe des Bildbereiches und durch den Kamerawinkel.

Die erste untersuchte Szene beinhaltet einen sehr großen Field of View, d.h. die Kamera bildet einen Bereich von 20 bis 40 Metern in der Horizontalen ab. Die Fehlerellipsen in Abbildung 17 sind im vorderen Bereich noch Kreisrund und mit zunehmender Entfernung vom Projektionszentrum steigt ihre Exzentrizität und ihre Größe. Daraus resultiert, dass ein getracktes Objekt, welches sich von der Kamera weg bewegt, mit einem monoton wachsenden

Fehler abgebildet wird und eines, welches sich parallel zur Kamera bewegt einen konstanten Fehler erfährt. In diesem Fall ist der Fehler ebenfalls von der Entfernung des Objektes zur Kamera abhängig. In der farbcodierten Darstellung stellt sich heraus, dass die mittlere Standardabweichung in der vorderen Bildhälfte (Abbildung 17, A) maximal 2 Pixel beträgt und bis zur linken oberen Bildecke (Abbildung 17, B) auf über 8 Pixel ansteigt.

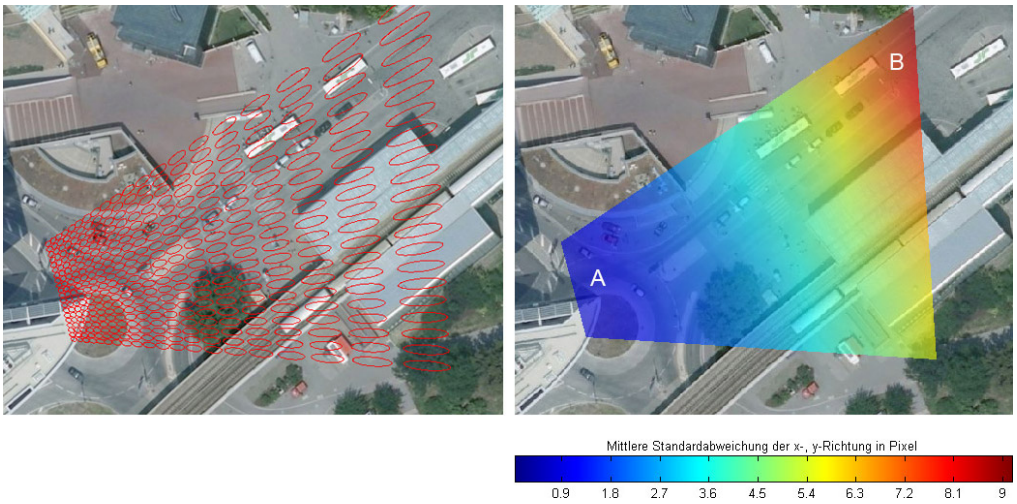


Abbildung 17: Szene 1, linkes Bild zeigt Fehlerellipsen von abgebildeten Punkten, welche mit Gauss-Noise ($\sigma = 3.0$) gestört wurden. Rechtes Bild zeigt die mittlere Standardabweichung für die x- und y-Richtung in einer farbkodierten Darstellung des projizierten Sichtbereiches der Kamera.

Ein maximal halb so großen projizierten Sichtbereich der Kamera als in der ersten Szene, hat das zweite Szenario (Abbildung 18). Die Änderungen der Exzentrizitäten der Fehlerellipsen entsprechen hier dem gleichen Verhalten wie der in Szene 1. Sie unterscheiden sich jedoch in ihrer Größe. In Szene 2 haben die Fehlerellipsen im Mittel nur einen halb so großen Radius wie in der ersten Szene. Ihre zugehörigen Gauß-Verteilungen haben eine mittlere Standardabweichung von anfangs 0,6 bis 1,2 Pixel (blau, Abbildung 18, A) bis maximal knapp über 2,2 Pixel (rot, Abbildung 18, B)). Auch hier ist ein zunehmendes Wachstum von vorne nach hinten zu beobachten. Aufgrund des kleinen projizierten Field of Views können die Trajektorien im blauen Bereich bis auf ca. 0,2 m (1 Pixel) genau dargestellt werden.

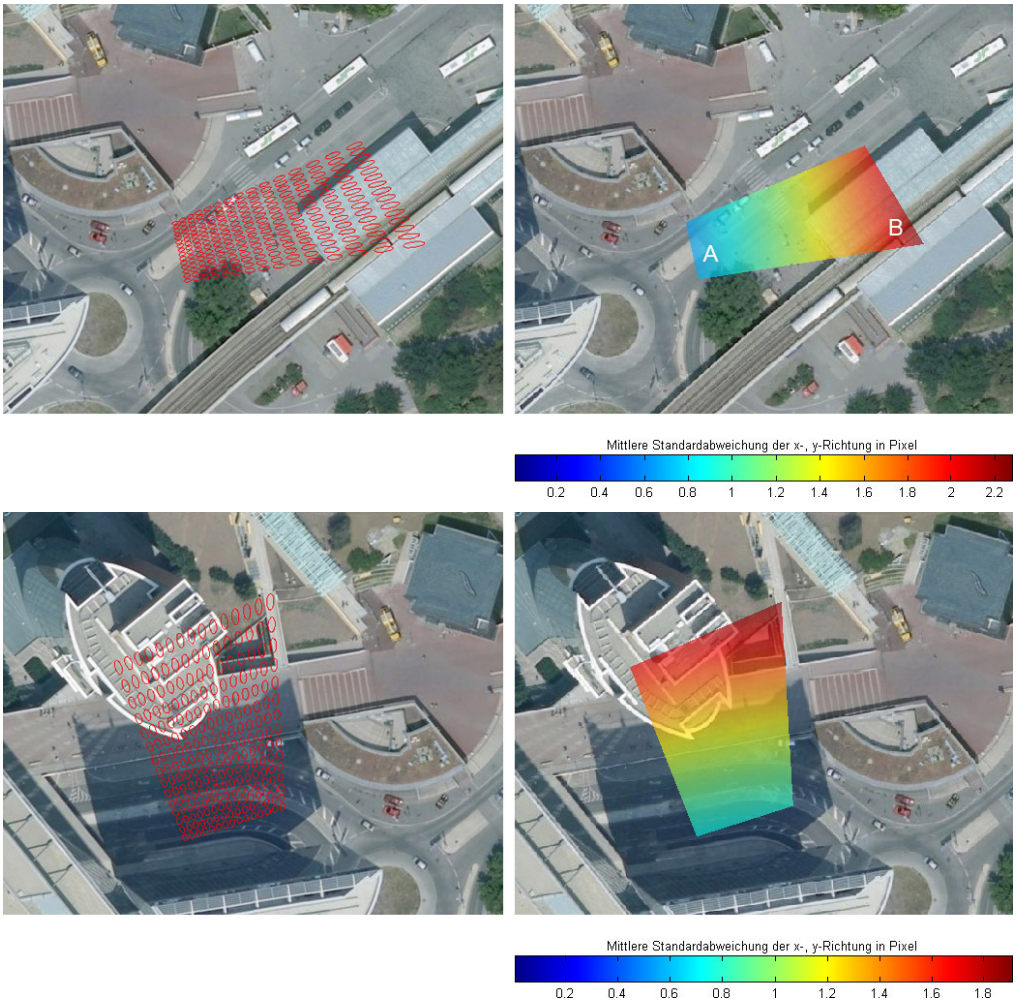


Abbildung 18: Szene 2 (oben) und 3 (unten), linkes Bild zeigt Fehlerellipsen von den abgebildeten Punkten, welche mit Gauss-Noise ($\sigma = 3.0$) gestört wurden. Rechtes Bild zeigt die mittlere Standardabweichung für die x- und y-Richtung in einer farbkodierten Darstellung des projizierten Sichtbereiches der Kamera.

In Szene 3 kann die maximale mittlere Standardabweichung auf weniger als 2 Pixel reduziert werden (Abbildung 18), jedoch ist der gesamte vordere Bildbereich aufgrund der tiefer gelegenen Straße nicht korrekt kalibriert. Der relevante Bereich dieser Szene, auf den die Kalibrierung angepasst wurde (siehe Kapitel 5.2, hat eine Abweichung von 1 bis 1,5 Pixel. Dieses Ergebnis kann auf den steilen Kamerawinkel (ca. 35 Grad) zurück geführt werden.

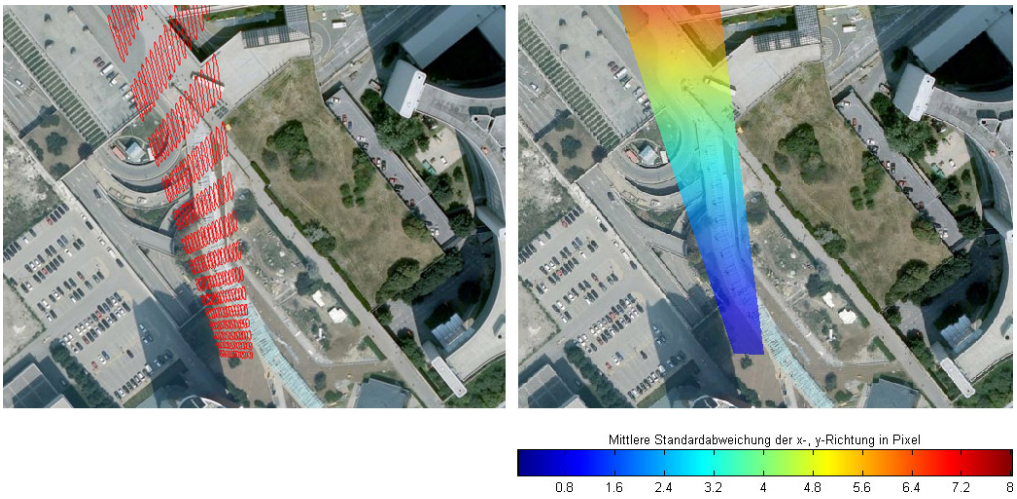


Abbildung 19: Szene 4, linkes Bild zeigt Fehlerellipsen von abgebildeten Punkten, welche mit Gauss-Noise ($\sigma = 3.0$) gestört wurden. Rechtes Bild zeigt die mittlere Standardabweichung für die x- und y-Richtung in einer farbkodierten Darstellung des projizierten Sichtbereiches der Kamera.

Das letzte der vier untersuchten Szenarien zeigt die Grenzen dieses Verfahrens auf (Abbildung 19). Der sehr flache Kamerawinkel (>50 Grad) und der länglich verzerrte projizierte Sichtbereich resultieren in größere Standardabweichungen als in den Szenen 1 bis 3 und betragen über 8 Pixel im hinteren Bildbereich. Unter Verwendung der zweitgrößten Zoomstufe von Bing Maps ist hier die Angabe von 0,2 m Pixelseitenlänge nicht mehr gültig. Die Fehlerellipsen im hinteren Bereich weisen eine größere Exzentrizität auf als die maximale Exzentrizität in Szene 1,2 und 3. Dies erklärt den Fehler in vertikaler Richtung in Kapitel 5.2. Die Ergebnisse aus Kapitel 5.2 im blauen Bereich werden mit den Werten der mittleren Standardabweichung von 2,4 Pixeln bestätigt.

5.4 Ergebnisse

In den nachfolgenden Abbildungen 20 und 21 sind Screenshots der Software TraVis dargestellt. Sie beinhalten Livebilder der ersten drei Szenen mit jeweils der optimalsten Kalibrierung.

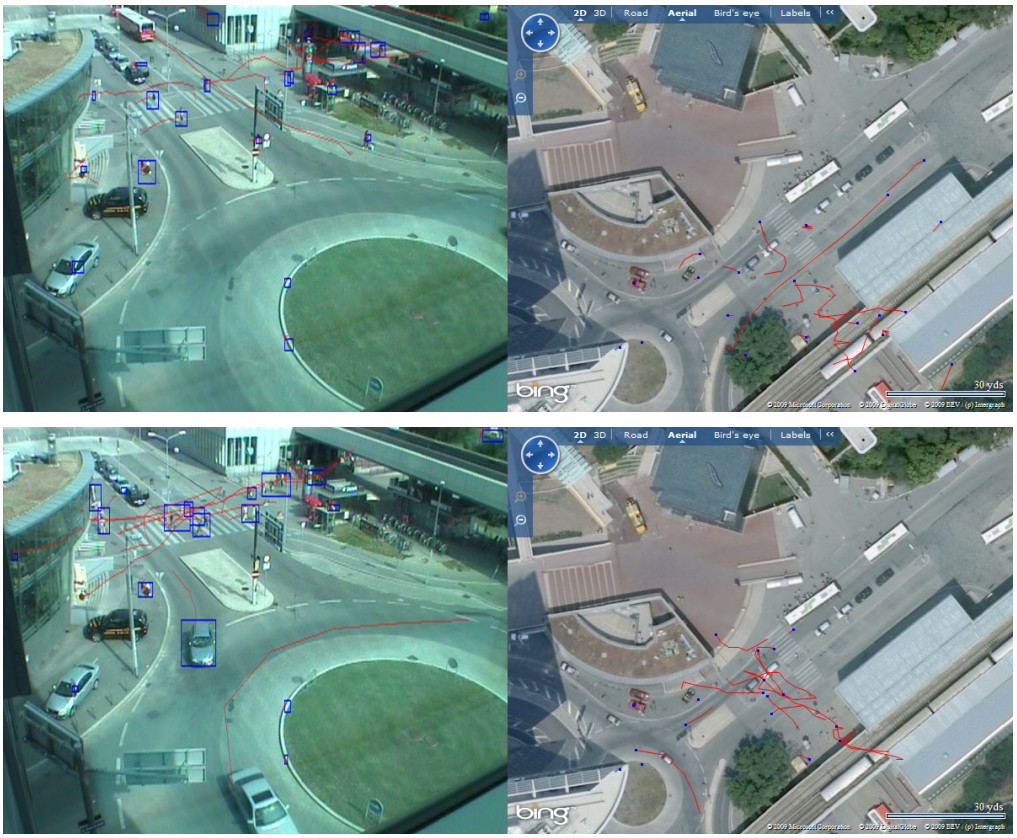


Abbildung 20: Szene 1, Screenshots aus TraVis mit live Darstellung der verfolgten Objekte und dessen Trajektorien im Kamerabild und in Bing Maps.

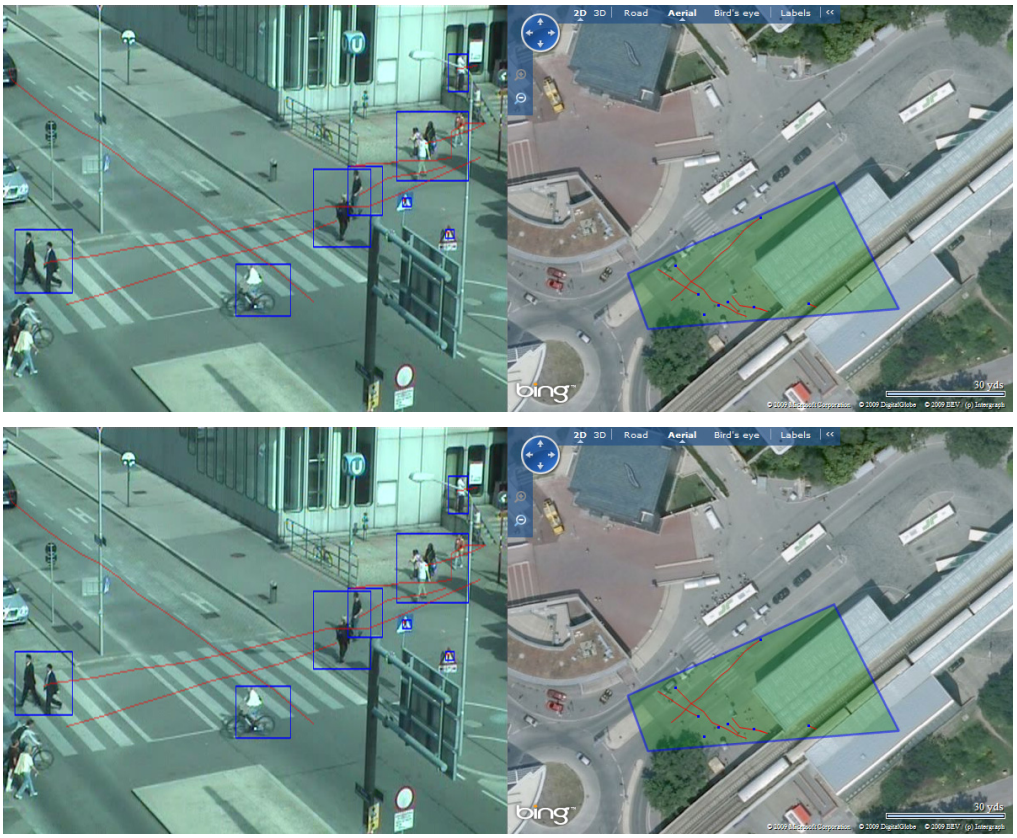


Abbildung 21: Szene 2, Screenshots aus TraVis mit live Darstellung der verfolgten Objekte und dessen Trajektorien im Kamerabild und in Bing Maps.

6 Ergebnisse und Ausblick

Der Online-Kartendienst Bing Maps von Microsoft wurde für die Entwicklung einer Applikation zur Echtzeitvisualisierung von Trajektorien verwendet. Die Auflösung der Luftbilder, die Microsoft zu Verfügung stellt, hat eine Grenze überschritten, die es ermöglicht zu Kamerabildern adäquate Ansichten einer Szene von oben zu bieten. Um diese Möglichkeit effektiv nutzen zu können, muss zunächst eine Kalibrierung zwischen Kamerabild und der Luftaufnahme durchgeführt werden. Hierfür müssen mindestens 4 Punktkorrespondenzen gefunden werden, um die notwendige projektive Transformation bestimmen zu können. Aufgrund der Auflösungsdifferenzen und teilweise älteren Satellitenfotos ist das Finden von gleichen Bildpunkten nicht trivial. Aus diesem Grund wird dieser Schritt in der entwickelten Applikation manuell durchgeführt.

Mit dieser Arbeit konnte gezeigt werden, in welchen Fällen die Kalibrierung von Kamera und Luftbild bis auf eine Abweichung von durchschnittlich 5 Pixel bestimmt werden kann. Hierbei sind im Mittel 7 Punktkorrespondenzen notwendig. Wenn die Kamera sehr steil von schräg oben auf die Szene gerichtet ist (Kamerawinkel < 50 Grad), kann mittels der manuell bestimmten Kalibrierung eine Transformation von über 50 Prozent des Kamerabildes bis auf einen Pixel genau durchgeführt werden. Dies entspricht einer Genauigkeit von ca. 0,2 m im Luftbild. Ebenfalls konnten die Grenzen des Verfahrens aufgezeigt werden. Diese sind durch den Kamerawinkel auf die Szene bestimmt. Sobald die Kamera mit einem größeren Winkel als 50 Grad ausgerichtet ist, steigt der Fehler auf über 8 Pixel im Luftbild an, was einem mittleren Abbildungsfehler von mehr als 1,6 m entspricht.

Sollten zukünftig Luftbilder mit höheren Auflösungen, als zu dem Zeitpunkt der Erstellung dieser Arbeit verfügbar waren angeboten werden, könnte die Kalibrierung automatisch, z.B. mittels Extraktion von Features aus den Bildern, durchgeführt werden. Des Weiteren hat Microsoft Bing Maps einen 3D-Modus integriert, dessen Programmierschnittstelle jedoch noch sehr unflexibel ist. Sollte seine API erweitert werden, könnte der Webdienst für noch komplexere Visualisierungen genutzt werden. Es wäre dann möglich, die Szene nicht nur von oben zu betrachten, sondern aus jeder beliebigen Perspektive oberhalb der Erdoberfläche. Der 3D-Modus bietet zudem 3D-Modelle von Gebäude einiger Großstädte an. Denkbar wäre diese Information in ein Kamerabild zu transformieren um damit auf einen Teil der drei dimensionalen Szeneninformation im Bild zu schließen.

7 Danksagung

Ganz besonders möchte ich mich bei Roman Pflugfelder bedanken! Seine fachliche Unterstützung und die vielen anregenden Ideen haben mir während dieser Arbeit sehr geholfen, mich inspiriert und mich auch in meinem Studium weitergebracht.

Ich bedanke mich bei Herrn Markus Clabian, dem Leiter des Geschäftsfeldes „Video and Security Technology“ am AIT, für den Vorschlag und die angebotenen Möglichkeiten dieses Thema bearbeiten und umsetzen zu können. Ich freue mich ebenfalls über die kompetente Unterstützung des gesamten Teams des Geschäftsfeldes und bedanke mich dafür, dass wann immer ein scheinbar unüberwindbares Problem auftrat, ich hilfreiche Ratschläge bekomme habe. Namentlich möchte ich Michael Dittrich erwähnen, mit dessen Hilfe ich viele Hürden während der Implementierung überwinden konnten.

Bedanken möchte ich mich bei meinen Eltern, die mich stets motivieren meine Ziele zu erreichen! Ein sehr großer Dank geht an meine Schwester Jana Kropp für ihre kurzfristige und fachlich kompetente Unterstützung beim Korrekturlesen.

Ein besonderer Dank geht an Natascha Husar, die mir während des Verfassens dieser Arbeit immer zur Seite stand und mich hilfreich Unterstützt hat!

Diese Arbeit wurde durch das Projekt CAT Calibration and Tracking, ICT-2008-30, des WWTF Wiener Wissenschafts-, Forschungs-, und Technologiefonds unterstützt.

Literatur

[1] T. J. Broida and R. Chellappa. Estimation of object motion parameters from noisy images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(1):90–99, 1986.

[2] D. Brown, D.R. Dunn. Trajectory visualization by using global positioning systems (gps). In *SoutheastCon, 2005. Proceedings. IEEE*, pages 183–186, Piscataway, NJ, USA, 2005. IEEE.

[3] S. Fleck, C. Vollrath, F. Walter, and W. Strasser. An integrated visualization of a smart camera based distributed surveillance system. In *ACST'07: Proceedings of the third conference on IASTED International Conference*, pages 234–242, Anaheim, CA, USA, 2007. ACTA Press.

[4] Google. Google earth, August 2009. `http://earth.google.com/`.

[5] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[6] heise online. Microsofts virtuelle erde mit realen details, Mai 2007. `http://www.heise.de/newsticker/Microsofts-virtuelle-Erde-mit-realen-Details--/meldung/90238`.

[7] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82:35–45, 1960.

[8] Microsoft. Bing maps, August 2009. `http://www.bing.com/maps/`.

[9] Microsoft. Bing maps for enterprise from microsoft - developer resources, online map apis, sdks, and map web services, August 2009. `http://www.microsoft.com/maps/developers/`.

[10] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, pages 255–261 vol.1. IEEE, 1999.

[11] S. Veigl. Visuelle Fahrzeugverfolgung mittels Embedded Systems. Diplomarbeit, Technische Universität Wien, Austria, März 2008.