Robust Geometric Matching using Interval Newton Methods

Marc Rochel

October 28, 2004

Abstract

The branch and bound method is commonly used for finding boundaries for global optimal solutions within a specified interval. However, this method spends a lot of time in getting a high accuracy of the solution. The Interval Newton method improves the speed of convergence, but it requires two time continuous differentiable functions to be optimized. The problem of robust matching does not fit into that group of functions. We discuss combining the branch and bound method and the Interval Newton method for robust geometric matching, to achieve a method that does not require the function to be globally twice continuous differentiable while speeding up convergence in the end to get high accuracy.

Contents

Chapter 1

Introduction

Many problems in the field of computer vision have no closed form solutions and require iterative numerical solutions. Numerical methods based on real numbers return a solution, but usually do not guarantee global optimality or numerical accuracy. Interval methods allow us to find globally optimal solutions with known numerical bounds. A well-known method based on interval arithmetic for finding the global maximum for a function is branch and bound optimization. For geometric matching problems, this approach is discussed in [3], [21], [20], [11] and [14] and reviewed in [4].

While such methods have be proven to be usefull in many practical applications, if the parameter space is high-dimensional or high accuracy is needed, branch and bound performs slowly. This thesis describes how we can use the interval Newton method to speed up the convergence. The interval Newton method is the natural extension of the Newton method to interval arithmetic. Van Hentenryck [23] shows that it is applicable for finding solutions of high-dimensional polynomial systems. The interval Newton method requires objective functions that are twice differentiable.

Our problem in applying interval Newton to problems in computer vision is that the problems faced in computer vision are not differentiable. Thus the Newton method alone does not allow us to simply find solutions for these problems. With an appropriate interval arithmetic as described in Chapter 2, the Newton method is still applicable for non-differentiable functions, but

the higher convergence speed gets lost. Since the evaluation of a Newton step is much more expensive than an iteration of the branch and bound method, the Newton method alone may perform even worse on such problems.

The work presented here combines these two methods by introducing a function depending on the problem that decides whether the current interval should be processed by a cheap branch and bound step or by a step of the Interval Newton method. The latter may not always succeed in shrinking the interval. In such a case, we fall back on a branch and bound step. The goal is to make this function as correct as possible so it decides on Newton as often as possible when a Newton step is useful and retains us from spending time on calculating unsuccessful Newton steps. We call this *dynamic switching*.

To keep the method simple, we use the natural interval extension. The method needs the function and its first and second derivative within areas near the solution which can be analytically derived.

Matchlists introduced by Breuel [4] to speed up branch and bound methods especially for point matching can be adapted to work with the Interval Newton method. As we will show later they also work with the Newton method using dynamic switching.

This thesis is organized as follows: In the next section, previous work on the geometric matching problem is discussed. In Chapter 2, the basic concepts needed for the discussed methods are presented. Afterwards, the methods that are used are explained. In Chapter 3 these methods are applied to the problems of finding lines, circles, axes aligned ellipses and rotated ellipses in point datasets. Chapter 4 shows experimental results of the methods for these problems. In Chapter 5 the problems are modified to also use normals of the points in the datasets. The effects of adding this additional information is shown in Chapter 6, presenting experimental results. Chapter 7 concludes this thesis and describes possible future work.

1.1 Previous Work

Previous approaches have used the Hough transform to find objects in datasets of points. Let us begin with a short summary of the Hough transform. (See [9], [7], [13] and [17] for a more detailed discussion). Every object to be found can be represented by parameters in a parameter space P. There are several possible objects running through each point m. By discretization and quantization of P, for every point p in P that represents an object that includes m, a counter for p is increased. The point in P with the highest counter is returned as the object that is the best match.

There are some practical problems with the Hough transform:

- The memory requirements rise with the number of dimensions.

  The grid size of the quantization of P has to be chosen appropriately. A too coarse quantization can lead to false solutions, as several objects get counted as one. A too fine quantization can lead to ignoring the actual solution if small errors in the dataset result in counting for points in P that are only nearby instead of inside the same grid element.

- The Hough transform is vulnerable to noise. It is possible that there are higher counts created in P from noise in the dataset than the counts from the points actually describing the object that should be found.

From a statistical point of view, this can be optimized by adding a Gaussian spot to the grid of P at p each time p describes an object where m is part of. This is better rather than just increasing the counter of p by one. But the problems mentioned above are still present. There are several versions of the Hough transform, like the probabilistic Hough transform [16], the randomized Hough transform [25] and the hierarchical Hough transform [22].

For the geometric matching problems of line circle and ellipse finding, the hough transform gets applied as follows:

A line gets parameterized by an angle of its normal and an offset along that normal. The parameter space is two-dimensional and each point m gets

represented as a line in P. This well-known approach is discussed in [9] and [7].

For the circle, the parameter space is three-dimensional. For example a circle can be parameterized by its center and its radius. To avoid the computational requirements of a three-dimensional Hough transform, Davies [7] proposes to split the problem into two stages. First, find the center of the circle by a two-dimensional Hough transform and then determine the radius by a one-dimensional Hough transform.

The case of the ellipse is more complicated. Since five parameters are needed to describe an ellipse, a direct approach would require a five-dimensional Hough transform. Yuen et al. [10] proposed to split the problem again into two stages: First, find the center of the ellipses by a two-dimensional Hough transform and afterwards find the remaining parameters by a three-dimensional Hough transform.

McLaughlin and Alder further developed in [19] their UpWrite method introduced in [1] and [18] to find lines, circles and ellipses. As in our method discussed later, this approach is more robust against perturbation noise. They split the problems into three stages. In stage one they compute local models of small regions of the input data. In stage two, objects are build from these local models. Stage three determines whether an object is a solution to the matching problem or not, by calculating moments of the objects.

As another approach, Breuel [2] and [4] used the branch and bound approach in combination with a reduction of the number of features by matching on datasets consisting of line segments instead of points.

If robustness against outliers is not needed, least square fitting methods can be applied. For example Fitzgibbon, Pilu and Fisher [8] introduce a method for least square fitting an ellipse using the algebraic distance as a parameterization of the ellipse. By adding an additional constraint the result is guaranteed to be an ellipse. However, all points of the dataset are assumed to be part of the ellipse, thus the problem is simpler than the problem that was accomplished by the previously mentioned methods.

Chapter 2

Basic Concepts

2.1 Interval Arithmetic

Let us begin by defining the real intervals as

$$\mathrm{R_I} = \{[a, b] : a, b \in \mathbb{R} \cup \{-\infty, \infty\} \wedge a \leq b\}. \tag{2.1}$$

Every function and operator gets defined on intervals, such that

$$\forall x \in [a, b] \in \mathrm{R_I} : f(x) \in f_\mathrm{I}([a, b]) \tag{2.2}$$

holds, where $f : \mathbb{R} \to \mathbb{R}$ is a function and $f_\mathrm{I} : \mathrm{R_I} \to \mathrm{R_I}$ is its corresponding interval extension. According to Van Hentenryck [23], we call this the *natural interval extension* of f. If for a function

$$f = f^{(1)} \circ f^{(2)} \circ \ldots f^{(n)} \tag{2.3}$$

the interval extension of the $f^{(i)}$ is known, the interval extension of f can be achieved by combining the interval extensions of the $f^{(i)}$:

$$f_\mathrm{I} = f_\mathrm{I}^{(1)} \circ f_\mathrm{I}^{(2)} \circ \ldots f_\mathrm{I}^{(n)} \tag{2.4}$$

Interval extensions are not unique. From all the possible interval extensions, we want to have one which has tight bounds. For example we do not want

to chose $f[a,b] = [-\infty, \infty]$ for every function f. The bounds given by an interval extension do not have to be optimal. A property we require from the interval extensions of the functions we use later on is

$$\forall\ a < b < c: \quad \lim_{[a,c]\to[b,b]} f_{\mathrm{I}}([a,c]) = f_{\mathrm{I}}([b,b]) = [f(b), f(b)]. \tag{2.5}$$

It is at least guaranteed that the resulting interval value of the function converges towards a single value if the parameter interval converges to a single value.

Addition, subtraction, multiplication and division by intervals not containing zero can be extended in a straight forward way as in Chen [6]:

$$\begin{array}{lcc} [a,b] + [c,d] & = & [a+c, b+d] \\ [a,b] - [c,d] & = & [a-d, b-c] \\ [a,b] \ \cdot\ [c,d] & = & [\min\{ac, ad, bc, bd\}, \max\{ac, ad, bc, bd\}] \\ [a,b] \ /\ [c,d] & = & [a,b] \cdot [1/c, 1/d] \quad \text{if } 0 \notin [c,d] \end{array} \tag{2.6}$$

Special care has to be taken of the division by intervals containing zero. Division of $[a,b]$ by an interval $[c,d]$ where $c \le 0 \le d$ is defined as described by Chen [6] and van Hentenryck [23]:

$$[a,b] \ /\ [c,d] = \begin{cases} [b/c, \infty] & \text{if } b \le 0 \text{ and } d = 0 \\ [-\infty, b/d] \cup [b/c, \infty] & \text{if } b \le 0 \text{ and } c < 0 < d \\ [-\infty, b/d] & \text{if } b \le 0 \text{ and } c = 0 \\ [-\infty, \infty] & \text{if } a < 0 < b \\ [-\infty, a/c] & \text{if } a \ge 0 \text{ and } d = 0 \\ [-\infty, a/c] \cup [b/c, \infty] & \text{if } a \ge 0 \text{ and } c < 0 < d \\ [a/d, \infty] & \text{if } a \ge 0 \text{ and } c = 0 \end{cases} \tag{2.7}$$

We do not want to represent the intervals by union of intervals, because using only one interval is faster and running costs can be guaranteed while the representation of the bound is not as accurate after divisions by intervals containing zero. Therefore, the interval extension of the division that

contains zero in the denominator which will be used later is

$$[a,b] \ / \ [c,d] = \begin{cases} [b/c, \infty] & \text{if } b \leq 0 \text{ and } d = 0 \\ [-\infty, b/d] & \text{if } b \leq 0 \text{ and } c = 0 \\ [-\infty, a/c] & \text{if } a \geq 0 \text{ and } d = 0 \\ [a/d, \infty] & \text{if } a \geq 0 \text{ and } c = 0 \\ [-\infty, \infty] & \text{otherwise} \end{cases} \tag{2.8}$$

The cases where the division returned splitted intervals are combined here to return the union of the two splitted intervals. The idea behind this is in practice to try to avoid divisions by zero so that the first interval extension of the division (2.7) would not return two intervals and the second (2.8) would not return narrow bounds. If division by intervals containing zero are guaranteed to occur rarely, the disadvantage of having non-optimal bounds is negligible.

The functions we evaluate with interval arithmetic are defined on real values in the first place. Therefore the support can be restricted in such a way, that divisions by zero do not occur. Then, assuming we have optimal bounds, switching to interval extension cannot result in divisions by zero. However, since we do not have optimal bounds, but only tight bounds, this case may occur from time to time. We need to handle it accurately to ensure correctness of the described methods. Thus, the case where divisions by zero occur can be regarded as rare.

Real numbers cannot be represented in hardware. Every interval calculation described here can be mapped on a hardware representation by growing the borders to the nearest possible values: $[a, b]$ has to be implemented as $[\text{prevfp}(a), \text{nextfp}(b)]$, where prevfp(x) returns the largest value l representable by the hardware used for the calculation with $l \leq x$ and nextfp(x) returns the smallest value h representable by the hardware with $x \leq h$.

The interval extension can also be applied to vectors and matrices:

$$x \in \mathrm{R_I}^n : x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} \tag{2.9}$$

$$x \in \mathrm{R_I}^{n \times m} : x = \begin{pmatrix} x_{0,0} & x_{0,1} & \cdots & x_{0,m-1} \\ x_{1,0} & x_{1,1} & \cdots & x_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n-1,0} & x_{n-1,1} & \cdots & x_{n-1,m-1} \end{pmatrix} \tag{2.10}$$

Analogously, this can be transferred to functions of vectors and matrices. Let X and Y be $\mathbb{R}$, $\mathbb{R}^n$ or $\mathbb{R}^{n \times m}$ and $\mathrm{X_I}$ and $\mathrm{Y_I}$ be $\mathrm{R_I}$, $\mathrm{R_I}^n$ or $\mathrm{R_I}^{n \times m}$.

$$\forall x \in X \in \mathrm{X_I} : f(x) \in f_\mathrm{I}(X) \tag{2.11}$$

where $f : \mathrm{X} \to \mathrm{Y}$ is a function and $f_\mathrm{I} : \mathrm{X_I} \to \mathrm{Y_I}$ its corresponding interval extension. We will denote f_I by f depending on the signature. Additional definitions used later on are: Let $x \in \mathbb{R}^n, y \in \mathbb{R}^n$, then define

$$[x, y] := \begin{pmatrix} [x_0, y_0] \\ [x_1, y_1] \\ \vdots \\ [x_{n-1}, y_{n-1}] \end{pmatrix} \tag{2.12}$$

and let $x \in \mathbb{R}^{n \times m}, y \in \mathbb{R}^{n \times m}$, then define

$$[x, y] := \begin{pmatrix} [x_{0,0}, y_{0,0}] & [x_{0,1}, y_{0,1}] & \cdots & [x_{0,m-1}, y_{0,m-1}] \\ [x_{1,0}, y_{1,0}] & [x_{1,1}, y_{1,1}] & \cdots & [x_{1,m-1}, y_{1,m-1}] \\ \vdots & \vdots & \ddots & \vdots \\ [x_{n-1,0}, y_{n-1,0}] & [x_{n-1,1}, y_{n-1,1}] & \cdots & [x_{n-1,m-1}, y_{n-1,m-1}] \end{pmatrix} \tag{2.13}$$

and

$$[x, y].lo = x, [x, y].hi = y. \tag{2.14}$$

2.2 The Branch and Bound Method

In this chapter, we will give an overview over the branch and bound method for optimization problems on continuous domains.

Consider the following problem: Given a bounded rectangular domain $X \subset \mathbb{R}^n$ in a parameter space. Let the objective function be $f(x) : X \to \mathbb{R}$. Find the position y of a global maximum of f in the domain X: $y \in X$ such that $f(y) = \max_{x \in X} f(x)$.

Assume there is a function f_u such that

$$\forall y \in Y : f_u(Y) \geq f(y) \tag{2.15}$$

holds, where $Y \subset X$ and

$$\lim_{Y \to [y,y]} f_u(Y) = [f(y), f(y)] \tag{2.16}$$

Then, the branch and bound method works as described by the following pseudo code: Let x be the domain in parameter space.

```
PriorityQueue p
p.Insert (x, f_u(x))
while ( not Terminate(x) )
{
    x = p.RemoveHead ()
    x0 = Split (x, 0)
    x1 = Split (x, 1)
    p.Insert (x0, f_u(x0))
    p.Insert (x1, f_u(x1))
}
return x
```

The meanings of the functions used in the pseudo code are:

- p.Insert(x, v) inserts x into the priority queue p with a priority of v.
- p.RemoveHead() returns the head of the priority queue p and removes it from the priority queue.

- Split $(x,\ i)$ splits the interval box x along the longest edge e. The returned half depends on the i. If $i = 0$, the interval with the smaller lower bound in the direction of e is returned. If $i = 1$, the other interval is returned.

- Terminate(x) describes the termination condition. It is true if the edges of the interval box are smaller than specified accuracies.

A suitable function f_u has to be derived for each problem separately. For example Breuel [4] calculated functions for geometric matching problems.

A more general way is to use the natural interval extension of the function f for optimization as described by Breuel in [5]. Setting $f_u(x) = f(x).hi$, (2.15) is satisfied because of (2.11). Using natural interval extension, (2.16) is also satisfied due to (2.5). Therefore, $f(x).hi$ fulfills the required properties for use in the branch and bound method. But we achieve even more by doing this. The natural interval extension does not only give us an upper bound for the function f on the current interval, but also a lower bound. This can be used to prove optimality as follows: Let the current interval be denoted as x, then it holds

$$\text{if } \forall y \in \text{p.Set}() : f(x).lo > f(y).hi, \text{ then x contains the global maximum,} \tag{2.17}$$

where p.Set() returns the set of intervals in the priority queue p. Note that such additional knowledge can also be achieved by not using interval arithmetic but by manually deriving a lower bound depending on the problem like previously for the upper bound. This condition can be efficiently checked by the use of the priority queue. Thus, (2.17) can be rewritten as

$$\text{if } f(x).lo > \text{p.Head}(), \text{ then x contains the global maximum,} \tag{2.18}$$

where p.Head() returns the head of the priority queue p without removing it. The user may want to change the termination condition of the branch and bound method described earlier by verifying if this condition is true to guarantee that the solution is the global maximum. However, depending on the problem including the given problem data, getting this proof may

take some time. Furthermore, if the problem has multiple global maxima, then this condition cannot be proved and therefore the algorithm will not terminate anymore.

In practice, the termination condition checks the widths of the interval. If they are below predefined thresholds, the termination condition is true and otherwise false. After termination of the algorithm, the optimality condition (2.17) can be checked by evaluating (2.18) to perhaps achieve the additional information that the returned interval is the global maximum.

Let x_{max} be a global maximum. If the optimality condition is not satisfied, then we know at least that for our solution x

$$(f(x_{max}) - f(x)).hi \leq f(x).hi - f(x).lo \tag{2.19}$$

holds, because our solution is taken from the head of the priority queue and therefore $f(x_{max}) \leq f(x).hi$ and $f(x_{max}) \geq f(x).lo$. Thus, the error we get in quality is less or equal to the size of the quality of the solution x.

2.3 Notation for Differentiation

The methods that will be described need the first and second derivative of the quality function. Let us define the notation that will be used:

For a function $h : \mathbb{R} \to \mathbb{R}$ the derivative is written as usual as h'.

The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as a column vector:

$$\nabla f(x) = \begin{pmatrix} \frac{\delta f}{\delta x_0} \\ \frac{\delta f}{\delta x_1} \\ \vdots \\ \frac{\delta f}{\delta x_{n-1}} \end{pmatrix} (x) \tag{2.20}$$

D defines the derivative of the function following it. This means the partial

derivatives are written as a row vector:

$$\mathrm{D}f(x) = \left(\frac{\delta f}{\delta x_0} \quad \frac{\delta f}{\delta x_1} \quad \cdots \quad \frac{\delta f}{\delta x_{n-1}}\right)(x) \tag{2.21}$$

D can also be applied to a function $g : \mathbb{R}^n \to \mathbb{R}^m$ resulting in a matrix:

$$g(x) = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{m-1} \end{pmatrix}(x) \tag{2.22}$$

$$\mathrm{D}g(x) = \begin{pmatrix} \frac{\delta g_0}{\delta x_0} & \frac{\delta g_0}{\delta x_1} & \cdots & \frac{\delta g_0}{\delta x_{n-1}} \\ \frac{\delta g_1}{\delta x_0} & \frac{\delta g_1}{\delta x_1} & \cdots & \frac{\delta g_1}{\delta x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta g_{m-1}}{\delta x_0} & \frac{\delta g_{m-1}}{\delta x_1} & \cdots & \frac{\delta g_{m-1}}{\delta x_{n-1}} \end{pmatrix}(x) \tag{2.23}$$

The operator D only applies to the function directly following it. For example the following is true:

$$\mathrm{D}f(g(x)) = (\mathrm{D}f)(g(x)) \tag{2.24}$$

According to the chain rule, this leads to

$$\begin{aligned} \mathrm{D}(f(g(x))) &= \mathrm{D}f(g(x)) \cdot \mathrm{D}g(x) \\ &= \left(\frac{\delta f}{\delta x_0} \quad \frac{\delta f}{\delta x_1} \quad \cdots \quad \frac{\delta f}{\delta x_{n-1}}\right)(g(x)) \\ &\quad \cdot \begin{pmatrix} \frac{\delta g_0}{\delta x_0} & \frac{\delta g_0}{\delta x_1} & \cdots & \frac{\delta g_0}{\delta x_{n-1}} \\ \frac{\delta g_1}{\delta x_0} & \frac{\delta g_1}{\delta x_1} & \cdots & \frac{\delta g_1}{\delta x_{n-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta g_{m-1}}{\delta x_0} & \frac{\delta g_{m-1}}{\delta x_1} & \cdots & \frac{\delta g_{m-1}}{\delta x_{n-1}} \end{pmatrix}(x), \end{aligned} \tag{2.25}$$

which fits perfectly because $f \circ g : \mathbb{R}^n \to \mathbb{R}$ and $\mathrm{D}(f \circ g) : \mathbb{R}^n \to \mathbb{R}^n$.

Therefore $\mathrm{D}\nabla f(x)$ is the Hessian matrix:

$$\begin{aligned}
\mathrm{D}\nabla f(x) &= \mathrm{D}\begin{pmatrix} \frac{\delta f}{\delta x_0} \\ \frac{\delta f}{\delta x_1} \\ \vdots \\ \frac{\delta f}{\delta x_{n-1}} \end{pmatrix}(x) \\
&= \begin{pmatrix} \frac{\delta^2 f}{\delta x_0 \delta x_0} & \frac{\delta^2 f}{\delta x_1 \delta x_0} & \cdots & \frac{\delta^2 f}{\delta x_{n-1} \delta x_0} \\ \frac{\delta^2 f}{\delta x_0 \delta x_1} & \frac{\delta^2 f}{\delta x_1 \delta x_1} & \cdots & \frac{\delta^2 f}{\delta x_{n-1} \delta x_1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta^2 f}{\delta x_0 \delta x_{n-1}} & \frac{\delta^2 f}{\delta x_1 \delta x_{n-1}} & \cdots & \frac{\delta^2 f}{\delta x_{n-1} \delta x_{n-1}} \end{pmatrix}(x) \qquad (2.26)
\end{aligned}$$

$\mathrm{D}\nabla g(x)$ is still undefined because of the geometric arrangement of the components as vectors and matrices. This limitation can be broken by using the tensors. This will be used in the later chapter of multiple constraint matching. Let the spaces be defined as

$$\begin{aligned}
\mathrm{R}^n &= \mathrm{span}\{e_i : 0 \leq i < n\} \\
\mathrm{R}^m &= \mathrm{span}\{f_j : 0 \leq j < m\} \qquad (2.27)
\end{aligned}$$

where e_i and f_i are bases of the corresponding vector spaces R^n and R^m. Then, $g : \mathrm{R}^n \to \mathrm{R}^m$ can be represented in tensor notation as

$$g(x) = \sum_{j=0}^{m-1} g_i(x)\ f_j \qquad (2.28)$$

We define the first derivative of g, $\mathrm{D}g$ and the second derivative $\mathrm{D}^2 g$ in tensor notation as

$$\begin{aligned}
\mathrm{D}g(x) &= \sum_{j=0}^{m-1}\sum_{i=0}^{n-1} \mathrm{D}g_{j,i}(x)\ f_j \otimes e_i \\
\mathrm{D}^2 g(x) &= \sum_{j=0}^{m-1}\sum_{i=0}^{n-1}\sum_{\tilde{i}=0}^{n-1} \mathrm{D}^2 g_{j,i,\tilde{i}}(x)\ f_j \otimes e_i \otimes e_{\tilde{i}} \qquad (2.29)
\end{aligned}$$

where

$$\begin{aligned} \mathrm{D}g_{j,i} &= \frac{\delta g_j}{\delta x_i} \\ \mathrm{D}^2 g_{j,i,\tilde{i}} &= \frac{\delta^2 g_j}{\delta x_{\tilde{i}} \delta x_i}. \end{aligned} \tag{2.30}$$

2.4 The Interval Newton Method

The Interval Newton method described by Hansen and Greenberg [12] is the extension of the Newton method on real values to the interval arithmetic. It can be used to find zeros of a continuously differentiable function h, given additionally the derivative of h. Since we are interested in global maxima, the optimization problem defined in the previous chapter can basically be solved by finding zeros of the derivate of f. Therefore we ask for twice continuous differentiability of f. Let us reformulate the optimization problem:

Given a bounded rectangular domain $X \subset \mathbb{R}^n$ in a parameter space. Let $f(x) : X \to \mathbb{R}$. Find the position y of a global maximum of a twice continuous differentiable function f in the domain X: for example $y \in X$ such that $f(y) = max_{x \in X} f(x)$.

A step of the Interval Newton method works as follows: Let x be the current interval, $p \in x$ the pivot point, arbitrarily chosen from the current interval and $g = \nabla f$. Then a Newton step is to perform

$$x_{\text{next}} = \big(x - (\mathrm{D}g(x))^{-1} g(p)\big) \cap x. \tag{2.31}$$

In practice, this is calculated by solving

$$\mathrm{D}g(x) x_{\text{sol}} = x - g(p) \tag{2.32}$$

and applying

$$x_{\text{next}} = x_{\text{sol}} \cap x. \tag{2.33}$$

Figure 2.1 illustrates the method. Within the current interval x, a pivot point p is chosen. Using the bound for the derivative of g in the interval, a

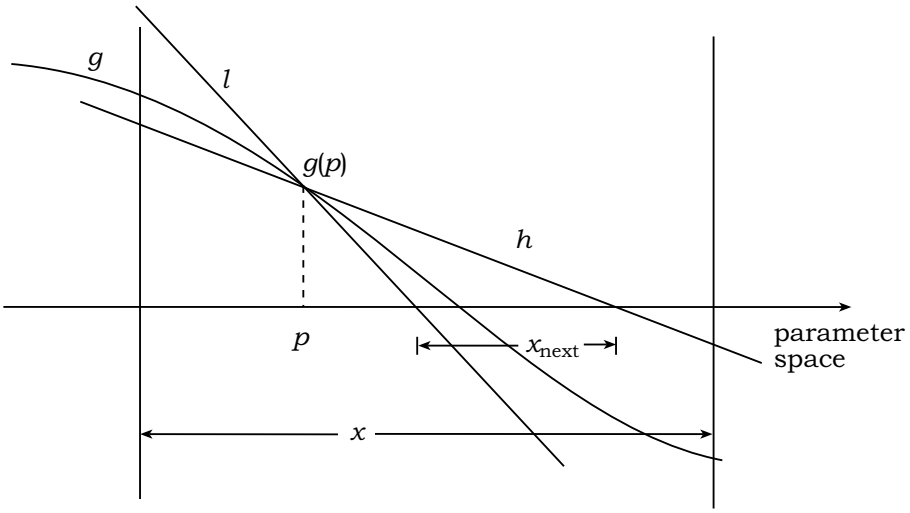

Figure 2.1: Illustration of a Newton step

new bound x_{next} for the solution is calculated. h is the straight line through the point $(p, g(p))$ with the slope equal to the upper bound of the derivative of g over the interval x. l is the straight line through the point $(p, g(p))$ with the slope equal to the lower bound of the derivative g over the interval x. Thus, g has to be between h and l on the interval x. Because we search for a zero of g, we take the position where h and g are zero as the bounds for the next interval, cut with the current interval x since we already know that the solution has to be inside x.

Plugging this into our algorithm leads to a combination of the branch and bound and Interval Newton method as it can be seen in figure 2.2. Let x be the domain in parameter space. x.contains(y) checks if x contains y and x.isempty() checks if x is the empty interval. The other notations are equivalent to the previous notations used in the pseudo code of the branch and bound method.

As we can see, the algorithm falls back to the branch and bound method if the interval cannot be shrunk. An important point of the behavior of this method is that if the interval is reduced by an iteration step, only this interval is considered from then on. The area $x \backslash x_{\text{next}}$ is proved to have no local maximum and therefore no global maximum either. Consequently if x_{next} is empty, it is proved that the global maximum is not inside x.

```
PriorityQueue p
p.Insert (x, f(x).hi)
while ( not Terminate(x) )
{
    x = p.RemoveHead ()
    x_next = (Dg(x))^{-1}(x - g(p))
    if not x_next.contains(x)
    {
        x_next = x_next ∪ x
        if not x_next.isempty()
        {
            p.Insert (x_next, f(x_next).hi)
        }
    }
    else
    {
        x0 = Split (x, 0)
        x1 = Split (x, 1)
        p.Insert (x0, f(x0).hi)
    }
}
return x
```

Figure 2.2: Pseudo code of the interval Newton method.

Note that the solutions of this combined method may be different from the solutions of the branch and bound method. This depends on two different definitions of the problem that has to be solved. The branch and bound method searches for the position x of a global maximum which satisfies

$$\forall y \in X, f(y) \leq f(x). \tag{2.34}$$

The combined method searches for the position x of a largest local maximum in the interior which satisfies

$$\forall y \in X, f(y) \leq f(x) \wedge \nabla f(x) = 0. \tag{2.35}$$

This is caused by the Newton method searching for zeros of ∇f. If the function only has local minima in the prescribed domain, it will not return a maximum. This has to be kept in mind if this method is used. Let us assume

that the f has at least one local maximum. Then, the problem that Newton may converge to a local minimum instead of a maximum is addressed by the priority queue. If an interval contains a local minimum, it will stop to be considered at some stage since the upper bound gets low. The difference of the search criterion causes problems if the global maximum x lies on the boundary of the domain and it does not hold $\nabla f(x) = 0$. In such a case, branch and bound will return that maximum while Newton will prove that there is no local maximum in the sense of Newton in the environment of the global maximum and therefore will return some other position. In practice, this will not be a problem if the domain chosen is large enough.

This combined method can be viewed from two different points of view. It can be seen as a branch and bound method using Newton steps to subdividing the current interval better. Or it can be seen as the Interval Newton method described by Hansen and Greenberg [12] with an additional criterion for choosing the next interval to process.

For solving the resulting linear system (2.32), Hansen and Greenberg [12] proposed to try an LU-decomposition and forward and back substitution. If this fails, they fall back on the Interval Gauss Seidel method discussed in the next chapter. For solving our problems, the LU-decomposition nearly always failed. Therefore the Gauss Seidel method is used directly for the method used during this thesis.

As described by Chen [6] and Hansen and Greenberg [12], not all components of the matrix $\mathrm{D}g(x)$ of the system to be solved (2.32) have to be evaluated over the whole current interval. Instead, parts of the matrix can be evaluated over intervals having some components restricted to the components of the position of the pivot point p, resulting in a matrix $\mathrm{D}g(x, p)$. The advantage is that one gets get better bounds in a single Newton step. The tradeoff is that, if the matrix is expensive to evaluate, it cannot be reused in consecutive iterations because the pivot point most likely lies outside the next interval. Then a new pivot point $\tilde{p}$ has to be chosen and $\mathrm{D}g(x, \tilde{p})$ has to be evaluated with respect to it. Furthermore, $\mathrm{D}g(x, \tilde{p})$ is not symmetric anymore and the cost of evaluation of one matrix multiplies by two. The methods discussed here evaluate over the whole interval because the calcu-

lation of the second derivatives of the discussed problems turn out to be expensive.

The combined branch and bound and Interval Newton method is referred to as the Interval Newton method in the rest of this thesis. This is suitable since we only discuss finding of global maxima from now on.

2.5 Interval Gauss Seidel Method

As mentioned earlier, we want to avoid division by intervals containing zero. Thus, we want to have a method for solving linear systems which does not divide by zero during the calculations.

The Interval Gauss Seidel method is the extension of the Gauss Seidel method defined on real values to interval arithmetic. First of all, let us define the problem of solving a linear system in interval arithmetic: Let $A \in {\mathrm{R_I}}^{n \times n}$, $b \in {\mathrm{R_I}}^{n}$ be given. By asking for a solution x for

$$Ax = b \tag{2.36}$$

we mean, find an $x \in {\mathrm{R_I}}^{n}$ such that

$$\forall a \in A : b \subset ax \tag{2.37}$$

Theoretically, this is a more general definition than we actually need for solving the resulting system from the Newton method. There, we only have a real value $g(p)$ on the right side of the equation. However, the exact $g(p)$ might not be representable in the limited precision of a computer and therefore all we can ask for is that it returns an interval including the correct value.

A simple solution can be found by just taking a very large interval. But the amount of information we get from such a choice is low. Thus, the x should be a as small as possible.

Such an x can be found by LU-decomposition and forward and back

substitution as mentioned earlier. The systems we have to solve later on are low-dimensional and using iterative methods to solve low-dimensional systems is slower. The problem with that approach is that many divisions by different intervals take place. If our interval arithmetic is used that always returns only one interval also if a division by an interval containing zero occurs, it is very likely that the result is a very large interval. In the worst case, bounds of $\pm\infty$ may occur. In the case of using an interval arithmetic that may result in two intervals if a division by an interval containing zero takes place, the resulting bound for x is a lot better. But the number of subintervals may increase. This becomes a problem especially if the result is processed further. The complexity of the algorithm may become larger than needed.

Experiments have shown, that for our problems in most cases, the resulting bound was much worse than the result from the Gauss Seidel method. Furthermore, in many cases, the returned bound had $\pm\infty$ components, if the calculations led to a division by an interval containing zero.

The Interval Gauss Seidel method solves these problems. Let us begin with a definition of how the Gauss Seidel method works: First of all, a starting interval x_0 for which (2.37) holds is needed. Then an Interval Gauss Seidel step is defined as follows:

$$\begin{aligned} for \quad & i = 0 \ to \ n-1 \ do \\ & x_{k+1,i} = \frac{\left(b_i - \sum_{j=0}^{i-1} A_{i,j} x_{k+1,j} - \sum_{j=i+1}^{n-1} A_{i,j} x_{k,j}\right)}{A_{i,i}} \cap x_{k,i} \end{aligned} \tag{2.38}$$

This step is iterated until the difference in size of two concurrent results is less or equal to a predefined epsilon:

$$| \ ||x_{k+1}|| - ||x_k|| \ | \leq \epsilon \tag{2.39}$$

where $||\cdot||$ denotes the Euclidean norm.

The advantage of this method is that only divisions by the intervals on the main diagonal occurs. If it is known that the diagonal only contains

intervals not containing zero, the case of division by an interval containing zero does not happen. In practice, it is hard to show for specific problems that the diagonal fulfills this requirement. If the problem is appropriate, as our problems will be, it is simple to show that the equation that calculates the matrix A on real numbers cannot result in zeros on the main diagonal. But this alone still does not guarantee that evaluating the equation with interval arithmetic also has this property; the resulting bound may be unnecessarily larger. In practice, this has not turned out to be a problem for our purpose. In the Newton method, if the Gauss Seidel method does not converge because of this reason, we fall back on a branch and bound splitting step. In the next iteration, the bound for A is smaller and the parts overlapping zero probably vanish. Therefore, using the simplified division is suitable for our problems.

Instead of iterating until (2.39) is true, we can also choose to terminate if the number of iterations reaches a predefined value λ. If the result is not accurate enough, the iteration is continued in a later step in the Newton method with a recalculated second derivative. If $\lambda = 1$, the second derivative is evaluated every step, what becomes equal to every iteration. If the evaluation of the second derivative is expensive, it may be better to increase λ to a higher value and reuse the calculated matrix with the tradeoff of needing more steps because of the less accurate bounds. A good choice of λ depends on the problem. The value used later on in out implementation is between 1 and 10.

Kearfott et al. [15] reviewed the use of preconditioners for the Interval Gauss Seidel method. Before starting to iterate, A and B get multiplied by another matrix P to speed up convergence. As noted in [12], a common choice is to take the midpoint inverse of A, meaning the matrix consisting of the midpoints of the components of A and inverted afterwards. Kearfott et al. discussed other choices of P, especially for splitting of the interval during the Gauss Seidel method caused by divisions.

Since we limit λ to a rather low limit, the cost of solving an additional linear system on real values did not turn out to be an advantage in out experiments. As mentioned earlier, we also care about avoiding divisions by zero by preparing our second derivative not to produce intervals containing

zero on the main diagonal. This property may get lost if A gets multiplied by some other matrix. It turned out to be more efficient in our case, not to use preconditioners.

2.6 Dynamic Switching

An Interval Newton step is expensive compared to a branch and bound step. If we have to find the maximum of a quality function Q mapping from X to $\mathbb{R}$ where X is n-dimensional, we only have to evaluate Q for one creation of a new interval in the branch and bound method which leads to a complexity of $O(1)$ for the number of function evaluations and a complexity of $O(n)$ for splitting the interval. For Newton, we have to additionally evaluate the first and second derivative and solve an n-dimensional system. Using iterative methods like the Gauss Seidel method from the previous chapter to solve the system, a Newton step results in a complexity of $\Omega(n^2)$ which is undesirably high. Furthermore, the constants are high, too, because computing the derivatives involves many complicated expressions. This also holds for our problems, even though they are low-dimensional. The highest dimension we have is five for the rotated ellipse.

For smooth functions like polynomials, Newton always is the better choice as shown by Van Hentenryck [23]. A problem arises if our functions are not continuously differentiable everywhere. Newton still works in that case if the interval arithmetic is extended to have $\pm\infty$ values as boundaries of the intervals like we have. But the algorithm slows down drastically since the time spent on expensive Newton steps turns out not to help in narrowing down the solution interval.

This issue can be dealt with if the problem has the following property: consider a problem that is twice continuous differentiable near the local maxima. Then, at some point the current interval will be small enough for taking advantage of using a Newton step. If it is known when the interval is small enough and when it is not, we can efficiently decide when a Newton step should be made.

Let Q be the quality function to be maximized. We handle this problem by evaluating an additional function which gives us a hint proposing to try a Newton step or not. This function is called *dynamic switching function*.

$$S : \mathrm{R_I}^{n \times n} \rightarrow \{true, false\} \tag{2.40}$$

We require: if S returns true, our function Q is at least twice continuous differentiable in the tested interval for our problem.

One could ask: why not use a non-interval method after dynamic switching tells us to use Newton? One may think that there is only one local maximum inside the interval if the dynamic switching function is true, which is nice enough to be found by a non-interval method. There are two reasons for continuing to use the Newton method after the dynamic switching function becomes true. The first one is, that the proof for the error boundary of the solution stops exactly when switching to a non-interval method. The only thing we know afterwards is the error boundary we got from the last branch and bound step we made. The second reason is that we also lose the information about whether the solution is optimal. This can happen for example, if the interval contains several local maxima. The non-interval method may run into the wrong maximum. Another possibility is that we do not get a proof that the current interval includes the optimal solution. If that is the case, we will never know which interval contains the optimal solution if we continue with a non-interval method.

The first possibility actually occurs very often in the problems discussed later on. Q is defined as a sum over several functions depending on the different points where a primitive has to be matched. Therefore Q can be thought of as a sum of humps for each point, all slightly shifted. Q is very hilly, especially if we have a lot of points contributing to the value of Q of the current interval, which in general is the case for the optimal and near optimal solutions. Therefore, in general, Newton steps using interval arithmetic should be preferred to continue the calculation.

2.6.1 Definition of Optimality for Dynamic Switching

As we will see in the following chapters, using dynamic switching seems to speed things up. But the question arises, how good dynamic switching is. To be more precise, let us specify what we mean by optimal in this context:

A Newton step is called successful, if it shrinks the interval. Otherwise, it is called unsuccessful. Dynamic switching requires the definition of a dynamic switching function which depends on the current interval. It returns true if a Newton step should be used and false if it should not. A dynamic switching function is optimal, if it returns true if and only if a Newton step will be useful.

Note that this does not mean that a Newton step always shrinks the interval more than a branch and bound step does. It is only assumed that this is the case because of the theoretically higher convergence rate.

Obviously, this optimality can be achieved by a function trying a Newton step and returning if it was successful or not. Since we don't gain speed from this, we should also add some condition on the complexity of the function. A Newton step has a complexity of $\Omega(n^2)$. We demand that the dynamic switching function has a complexity lower than $O(n^2)$, where n is the number of dimensions of the problem. Note that the complexity of a Newton step, a branch and bound step and the dynamic switching function rises linearly with the number of points. But since it is the case for all of them, this factor is left out.

In practice, it is not so easy to guarantee that a Newton step will be successful. A Newton step might also be unsuccessful if the function is twice continuous differentiable when the calculated bound for the derivatives is bad. But it is simple in many problems to know if a function over a specific interval is twice continuous differentiable. If that is the case, a Newton step will probably be successful. It will turn out that requiring this is good enough in practice. This property has to be and will be shown for the dynamic switching function used later on.

Chapter 3

Applications

3.1 The Problem

Let a dataset M be a set of points in $\mathbb{R}^2$. The problem of robust geometric primitive matching in a dataset can be formalized as finding

$$\arg_x \max_{x \in X} Q(x) \tag{3.1}$$

$$Q(x) = \sum_{m \in M} q_m(x) \tag{3.2}$$

$$q_m(x) = \Phi(d_m(x)) \tag{3.3}$$

where x is a vector in the parameter space X as proposed by Breuel [5]. Q is called the quality function since it represents the quality of x with respect to the optimization problem. q defines the quality of one point in M with respect to x. $d_m(x)$ is the distance between the point m and the primitive with the parameters x. This has to be calculated separately for every primitive.

$\Phi(x)$ is a function with its maximum at zero and $|\Phi(x)|$ is monotonically decreasing so that the quality function gets a higher value if the points have

smaller distances to the primitive. For example we use

$$\Phi(x) = \max\left(0, 1 - \frac{x^2}{\epsilon^2}\right) \tag{3.4}$$

See figure 3.1 for a plot of this function.

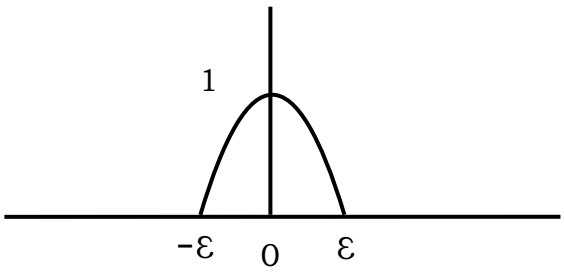

Figure 3.1: Φ

ϵ describes the border between counted, as a part of the primitive, and counted as an outlier, not contributing to the quality function. That way, the solution is robust against outliers, not disturbing the solution, in contrast to the least square method

$$\Phi(x) = -x^2. \tag{3.5}$$

3.2 Statistical Justification of Φ

Defining Φ as (3.4) can also be justified by a statistical model. According to Wells III [24], if we assume a Gaussian error model for the dataset, the probability of having a value x in the dataset while the correct value is X is

$$P_d(x) = ae^{-\frac{(x-X)^2}{b}} \tag{3.6}$$

with constants $a = \frac{1}{\sqrt{2\pi\sigma^2}}$ and $b = \frac{\sigma^2}{2}$ where σ^2 is the variance.

Assuming an equal distribution of the noise in the dataset, the probability density of having a feature somewhere is a constant

$$P_b(x) = P_{b,0} \tag{3.7}$$

Switching to log-likelihood functions, we get

$$\log P_d(x) = m + \frac{(x - X)^2}{b} \tag{3.8}$$

$$\log P_b(x) = \log P_{b,0} \tag{3.9}$$

where $m = \log a$.

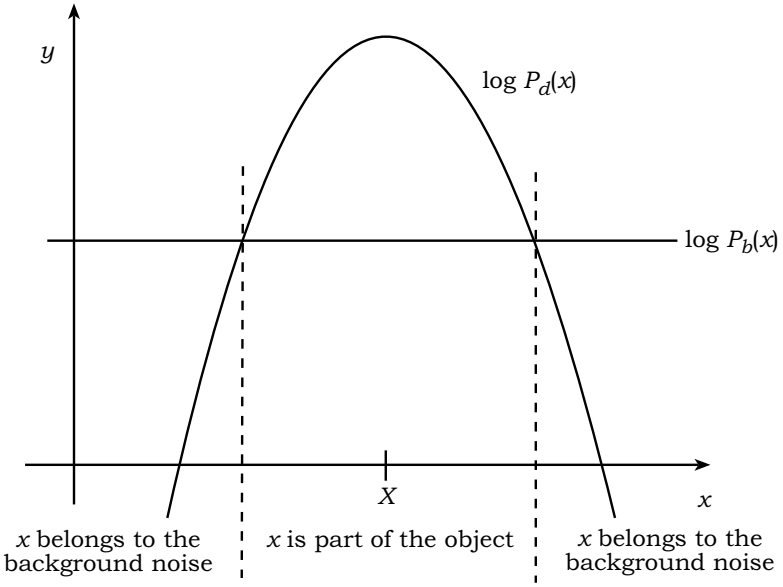

Figure 3.2: Creation of a classifier for distinguishing between the object and the background noise.

Building a classifier from these two distributions, we assign a value x to be a part of the object to be found if $\log P_d(x) \geq \log P_b(x)$ and we assign it to the background noise if $\log P_b(x) > \log P_d(x)$. See Figure 3.2 for a visual explanation. The classifier does not change, if we scale and shift it along the y-axis. If we assume $X = 0$, we get a log-likelihood function telling us if a point in the dataset is a point of the solution or if it is produced by the background noise:

$$\Phi(x) = \max\left(0, 1 - \frac{(x - X)^2}{b}\right) \tag{3.10}$$

That is exactly the Φ defined earlier in (3.4).

3.3 Derivatives for the Newton Method

For the Newton method, the first and second derivative of the quality function is needed. As a part of that, the first and second derivative of Φ is also needed. These can be calculated as

$$\Phi(x) = \max\left(0, 1 - \frac{x^2}{\epsilon^2}\right) \tag{3.11}$$

$$\Phi'(x) = \begin{cases} 0, & if|x| > \epsilon \\ -\frac{2x}{\epsilon^2}, & otherwise \end{cases} \tag{3.12}$$

$$\Phi''(x) = \begin{cases} 0, & if|x| > \epsilon \\ \infty, & if|x| = \epsilon \\ -\frac{2}{\epsilon^2}, & otherwise \end{cases} \tag{3.13}$$

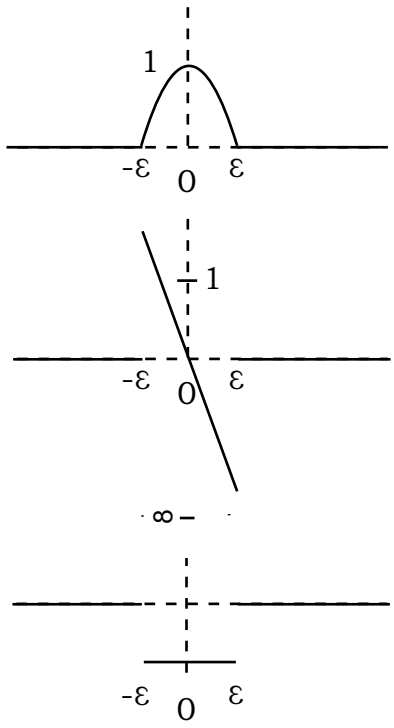

Figure 3.3: Φ and its first and second derivative

These functions are plotted in figure 3.3. Since we need the first and the second derivative and our quality function Q maps only to a one-dimensional space, we transpose the first derivative and get the gradient of Q and denote

from now on the second derivative of Q as the derivative of the gradient of Q:

$$\begin{aligned} Q(x) &= \sum_{m \in M} q_m(x) && (3.14) \\ F = \nabla Q(x) &= (\mathrm{D}Q(x))^T = \sum_{m \in M} \nabla q_m(x) && (3.15) \\ \mathrm{D}F = \mathrm{D}(\nabla Q)(x) &= \sum_{m \in M} \mathrm{D}(\nabla q_m)(x) && (3.16) \end{aligned}$$

As denoted earlier, the problems discussed later on are all designed to find robust solutions. Their q_m are of the form

$$q_m(x) = \Phi(d_m(x)) \tag{3.17}$$

The first and second derivatives are in general

$$\begin{aligned} f(x) &= \nabla q_m(x) = \Phi'(d_m(x)) \nabla d_m(x) \\ \mathrm{D}f(x) &= \Phi''(d_m(x))(\nabla d_m(x))^2 + \Phi'(d_m(x)) \mathrm{D} \nabla d_m(x) && (3.18) \end{aligned}$$

where

$$(\nabla d_m(x))^2 = (\nabla d_m(x))(\nabla d_m(x))^T \tag{3.19}$$

is the outer product instead of the scalar product and therefore an $n \times n$-matrix.

These are the parts of the derivatives that all problems discussed here have in common. In the following chapters, the concrete derivatives for the problems are shown.

3.4 Matchlists

The quality function Q can be directly plugged into the branch and bound method. However there is an efficient way to optimize the evaluation of Q in each step by making use of the coherence between two concurrent steps of related intervals. The idea is to remember if a point contributes a value

larger than zero to the sum of the quality function. When an interval x that was created from a previous interval y by splitting is processed, the points that did not contribute to $Q(y)$ will also not contribute to $Q(x)$ because $x \subset y$. Therefore, the calculations for these points can be dropped and the evaluation speed increases.

The lists of points contributing to the sum are called matchlists as introduced by Breuel [4]. He also used them for speeding up the solving of the correspondence problem in finding matches between two sets of points under linear transformations. For that application, the gain in speed is even higher because considering to have n points, finding the best match of each point with each other results in a complexity of $\Omega(n^2)$ if tried naively or $\Omega(n \log n)$ if a spatial data structure like a R-tree is used. Remembering which points cannot be the optimal match reduces n drastically in the later phase of the calculation.

Matchlists can be easily adapted for use in the Newton method. If in one step a point contributes zero to Q over the whole current interval, then it will also contribute zero to Q evaluated over all subintervals created from the current interval.

For the branch and bound method, the amount of memory needed has to be considered. Branch and bound tends to create a lot of small intervals near the solution at the end. Depending on the desired accuracy, the number of intervals can grow rather large. Breuel [4] describes several ideas to deal with that problem. It is important to think about the storage of the matchlists. One option is to store a list of indices to the points. In the beginning it is better to only store the points that do not contribute to the interval. Later on, the opposite is the case. Another option is to use a bit mask representing the points. Run length encoding the matchlists also helps. Another option is to use depth first instead of breadth first which first continues with the best interval created from one step without looking for intervals with higher quality in the priority queue. That way, the algorithm only processes the best subinterval of the current interval, ignoring all other intervals left in the priority queue, until the desired accuracy is reached. If it is successful, fewer intervals and therefore matchlists are created and less memory is needed.

The tradeoff is that the algorithm may take longer if the assumption that the direct choices are the best turns out to be false. However, Breuel showed that for the problem of geometric matching the loss in speed is negligible with respect to the lower requirements of memory.

Matchlists used in the Newton method turn out to practically not have this memory problem. Considering the higher convergence rate and the ability to prove nonexistence of a maximum in an interval, it is understandable to get fewer intervals and matchlists created.

3.5 The Dynamic Switching Function

As denoted before, for our problems $q_m(x)$ is of the form:

$$q_m(x) = \Phi(d_m(x)) \tag{3.20}$$

where $d_m : \mathbb{R}^n \to \mathbb{R}$ describes a distance that should be small. Let us assume that d_m is twice continuous differentiable on x.

Let us define S as

$$S : \mathrm{R_I}^n \to \{true, false\} : S(x) = (\forall m \in M : \pm\epsilon \notin d_m(x)) \tag{3.21}$$

Then, by definition, S is true if and only if the contributing part $q_m(x)$ of every point m to Q is [0,0] or does not contain zero. This is exactly the case when $d_m(x)$ lies clearly inside or outside the hump of Φ, not overlaying the non-differentiable points.

Theorem. *If d_m is twice continuous differentiable and $S(x) = true$, then Q is twice continuous differentiable on x.*

Proof. From the precondition we know that

$$S(x) = true \Rightarrow \forall m \in M : \pm\epsilon \notin d_m(x). \tag{3.22}$$

If $d_m(x)$ lies outside the hump of Φ, the corresponding point m does not

contribute to the sum of Φ. It holds that

$$Q(x) = \sum_{m \in M} \Phi(d_m(x)) = \sum_{x \in \{x : x \in M \wedge |d_m(x)|.lo \, < \, \epsilon\}} \Phi(d_m(x)). \tag{3.23}$$

Together with (3.22) this leads to

$$S(x) \Rightarrow Q(x) = \sum_{x \in \tilde{M}} \Phi(d_m(x)) \tag{3.24}$$

where

$$\tilde{M} = \{m : m \in M \wedge |d_m(x)|.hi \; < \; \epsilon\}. \tag{3.25}$$

The first and second derivatives of Q are

$$F(x) \;\; = \;\; \sum_{\tilde{M}} \mathrm{D}\Phi(d_m(x)) \cdot \mathrm{D}d_m(x) \tag{3.26}$$

$$\mathrm{D}F(x) \;\; = \;\; \sum_{\tilde{M}} \mathrm{D}^2\Phi(d_m(x)) \cdot (\mathrm{D}d_m(x))^2 + \mathrm{D}\Phi(d_m(x)) \cdot \mathrm{D}^2 d_m(x) \tag{3.27}$$

Because of the definition of $\tilde{M}$ we know that for all $m \in \tilde{M}$, $|d_m(x)| < \epsilon$. According to (3.12),

$$\mathrm{D}\Phi(d_m(x)) = -\frac{2d_m(x)}{\epsilon^2}, \tag{3.28}$$

what is obviously continuous. Since d_m is twice continuous differentiable and the product and sum of continuous functions is again continuous, F is continuous and therefore Q is continuous differentiable with (3.26) as the first derivative.

Analogously, according to (3.13),

$$\mathrm{D}^2\Phi(d_m(x)) = -\frac{2}{\epsilon^2}, \tag{3.29}$$

which is also continuous. Since also $(\cdot)^2$ is continuous, $\mathrm{D}F(x)$ is continuous and, including the previous statement, Q is twice continuous differentiable with (3.27) as the second derivative. □

Now we know that Q is twice continuous differentiable if d_m is twice continuous differentiable and S is true. As mentioned earlier, this does not

guarantee a successful Newton step because the bounds may be too loose. But we will see that a Newton step is successful in most of the cases if S is true. As we will see from the results of the experiments with the example datasets, the optimality condition is really not completely satisfied. But it is close enough to optimal to speed up the solution of the discussed problems.

By the definition of optimality for the dynamic switching function we required the costs for evaluation to be less than $O(n^2)$ with respect to the number of dimensions. The dynamic switching function (3.21) can be computed with a complexity of $O(1)$. Therefore, the complexity condition is satisfied. Since a branch and bound step costs $O(n)$ and the evaluation of our dynamic switching function costs $O(1)$, they together also have a complexity of $O(n)$ and therefore are cheaper than a Newton step with a complexity of $\Omega(n^2)$ because of the Gauss Seidel iteration. We will also observe this speed-up later using dynamic switching with the Newton method instead of the plain Newton method.

The dynamic switching function (3.21) is cheap to evaluate because most of the terms used already get evaluated for the value of Q. It also profits by the matchlists because they already tell us which points clearly lie outside of the neighborhood.

3.6 The Line Finding Problem

A line can be parameterized by an angle w of a normal n and an offset t along that normal. See figure 3.4 for a visualization. For every point $m = (m_x, m_y)$, the distance $d_m(w, t)$ to the line primitive can be calculated by projecting m onto n and calculating the difference to t.

The quality function for the line finding problem can be formalized together with (3.1) and (3.2) as

$$q_m(w, t) = \Phi\left(\cos(w)m_x + \sin(w)m_y - t\right) \tag{3.30}$$

where w is the angle of the normal of the line and t the offset along this

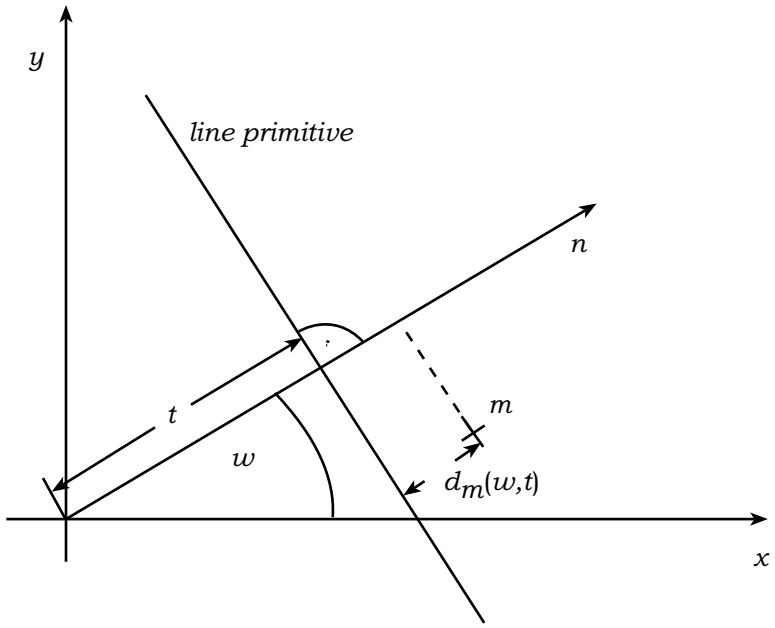

Figure 3.4: Parameterization of the line

normal.

Referring to (3.17) ff, the only term we still need to differentiate is

$$d_m(w,t) = \cos(w)m_x + \sin(w)m_y - t \tag{3.31}$$

The first and second derivatives of d_m are

$$\nabla d_m(w,t) = \begin{pmatrix} -\sin(w)m_x + \cos(w)m_y \\ -1 \end{pmatrix} \tag{3.32}$$

$$\mathrm{D}\nabla d_m(w,t) = \begin{pmatrix} -\cos(w)m_x - \sin(w)m_y & 0 \\ 0 & 0 \end{pmatrix} \tag{3.33}$$

d_m of this problem is obviously twice continuous differentiable. Plugging it all together, the first derivative is

$$f_m(w,t) = \nabla q_m(w,t) = \begin{pmatrix} \Phi'(\Delta t)\ t'_{rot}(w) \\ -\Phi'(\Delta t) \end{pmatrix} \tag{3.34}$$

where

$$\Delta t = t_{rot}(w) - t \tag{3.35}$$

and

$$\begin{aligned} t_{rot}(w) &= \cos(w)m_x + \sin(w)m_y \\ t'_{rot}(w) &= -\sin(w)m_x + \cos(w)m_y. \end{aligned} \tag{3.36}$$

This leads to the second derivative (3.14) with

$$\mathrm{D}f_m(w,t) = \begin{pmatrix} -\Phi'(\Delta t)t_{rot}(w) + \Phi''(\Delta t)t'_{rot}(w)^2 & -\Phi''(\Delta t)t'_{rot}(w) \\ -\Phi''(\Delta t)t'_{rot}(w) & \Phi''(\Delta t) \end{pmatrix} \tag{3.37}$$

Note that $\mathrm{D}f_m$ is symmetric. This is obvious for the areas where Q is twice continuous differentiable. For the other areas where Q is not twice continuous differentiable, $\mathrm{D}f_m$ is also symmetric if the interval arithmetic supports values of $\pm\infty$.

3.7 The Circle Finding Problem

A circle can be parameterized by its center (x, y) and its radius r. See figure 3.5 for a visualization.

For every point $m = (m_x, m_y)$, the distance $d_m(x, y, r)$ to the circle primitive can be calculated by computing the distance between m and the center of the circle (x, y) and subtracting the radius r.

The quality function for the circle finding problem can be formalized together with (3.1) and (3.2) as

$$q_m(x, y, r) = \Phi\left(\sqrt{(x - m_x)^2 + (y - m_y)^2} - r\right) \tag{3.38}$$

where x and y represent the center of the circle and r the radius.

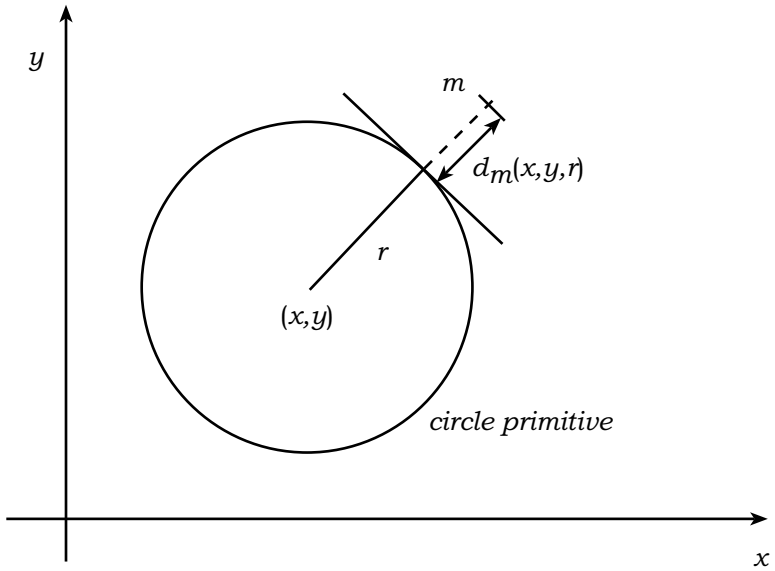

Figure 3.5: Parameterization of the circle

For this problem, d_m can be chosen as

$$d_m(x, y, r) = \sqrt{(x - m_x)^2 + (y - m_y)^2} - r \tag{3.39}$$

Assuming, $t_r \neq 0$, the derivatives are

$$\nabla d_m(x, y, r) = \begin{pmatrix} (x - m_x)/t_r \\ (y - m_y)/t_r \\ -1 \end{pmatrix} \tag{3.40}$$

$$\mathrm{D}\nabla d_m(x, y, r) = \begin{pmatrix} \frac{\Delta y^2}{t_r^3} & \left(\frac{\Delta x \Delta y}{t_r^3}\right) & 0 \\ \cdot & \frac{\Delta x^2}{t_r^3} & 0 \\ \cdot & \cdot & 0 \end{pmatrix} \tag{3.41}$$

where

$$\begin{aligned} t_r &= \sqrt{(x - m_x)^2 + (y - m_y)^2} \\ \Delta t &= \sqrt{(x - m_x)^2 + (y - m_y)^2} - r = t_r - r \end{aligned} \tag{3.42}$$

Again, the matrix of the second derivative is symmetric. For simplicity, only the upper right values are printed.

t_r is the distance from a point m to the center of the circle (x, y). We require the lower bound $r_0.lo$ for the radius of the circle to be larger than the ϵ of the Φ. Then, if the dynamic switching function is true, we know that the distance between the circle and every point m contributing to the sum of Q is smaller or equal to ϵ. Therefore, the distance to the center of the circle is then larger or equal $r_0.lo - \epsilon > 0$ and t_r is always larger than zero. The previous assumption $t_r \neq 0$ holds.

Inserting these equations into (3.17), we get

$$f_m(x, y, r) = \nabla q_m(x, y, r) = \begin{pmatrix} \Phi'(\Delta t)(x - m_x)/t_r \\ \Phi'(\Delta t)(y - m_y)/t_r \\ -\Phi'(\Delta t) \end{pmatrix} \tag{3.43}$$

The second derivative calculates as

$$\mathrm{D}f_m(x, y, r) = \begin{pmatrix} d_{11} & d_{12} & d_{13} \\ d_{12} & d_{22} & d_{23} \\ d_{13} & d_{23} & d_{33} \end{pmatrix}$$

$$\begin{aligned} d_{11} &= \Delta y^2 \frac{\Phi'(\Delta t)}{t_r^3} + \Delta x^2 \frac{\Phi''(\Delta t)}{t_r^2} \\ d_{12} &= \Delta x \Delta y \left(\frac{\Phi''(\Delta t)}{t_r^2} - \frac{\Phi'(\Delta t)}{t_r^3} \right) \\ d_{13} &= \Delta x \frac{\Phi''(\Delta t)}{t_r} \end{aligned}$$

$$\begin{aligned} d_{22} &= \Delta x^2 \frac{\Phi'(\Delta t)}{t_r^3} + \Delta y^2 \frac{\Phi''(\Delta t)}{t_r^2} \\ d_{23} &= \Delta y \frac{\Phi''(\Delta t)}{t_r} \end{aligned}$$

$$d_{33} = \Phi''(\Delta t) \tag{3.44}$$

where

$$\begin{aligned} \Delta x &= x - m_x \\ \Delta y &= y - m_y. \end{aligned} \tag{3.45}$$

There is also another parameterization possible:

$$q_m(x, y, r) = \Phi\left(\sqrt{\left(\frac{x - m_x}{r}\right)^2 + \left(\frac{y - m_y}{r}\right)^2} - 1\right) \tag{3.46}$$

This parameterization normalizes the $(x - m_x, y - m_y)$ to the unit circle and compares it with one instead of directly comparing it with the radius. The difference is the interpretation of ϵ. While in the first parameterization, ϵ directly describes the maximum geometric distance a point is allowed to have to be counted to the circle, in the second it describes the maximum geometric distance a point is allowed to have to be counted to the circle, normalized to the unit circle. The advantage of the first is, that ϵ can be intuitively chosen by the user. The advantage of the second is that the allowed error is related to the size of the found match. For example, a match of a small circle requires higher accuracy than a match of a huge circle.

With respect to efficiency, the first one is more efficient. The equation is simpler than the second and the derivatives are also simpler, resulting in cheaper Newton steps.

3.8 The Ellipse Finding Problem

Multiple parameterizations of an ellipse are thinkable. Let us start with the parameterization that will be used in the experiments later.

3.8.1 Geometrically inspired Match Functions

An ellipse can be parameterized by its center (x, y) and the lengths of its main axes a and b which are assumed to be parallel to the axes of the coordinate system. See figure 3.6 for a visualization.

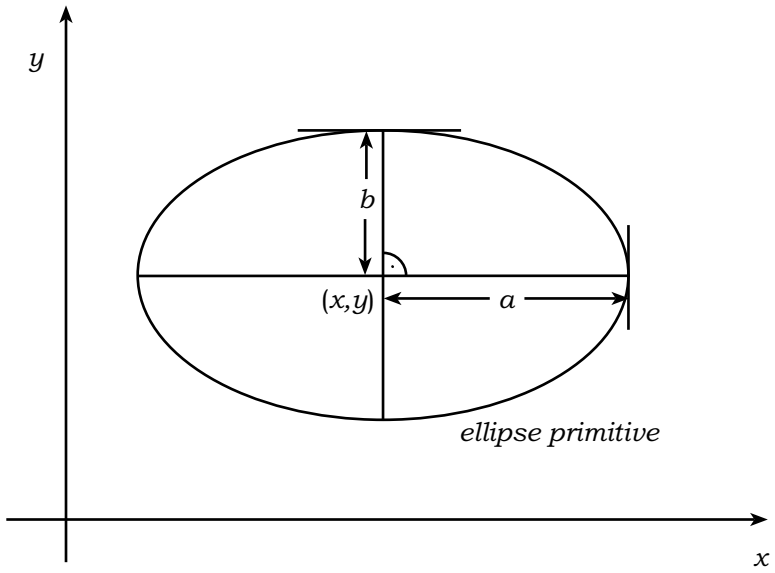

Figure 3.6: Parameterization of the ellipse

For every point $m = (m_x, m_y)$, the distance $d_m(w, t)$ to the ellipse primitive can be calculated by scaling the scene by the lengths of the main axes to transform the ellipse into a circle - into the unit circle in the following calculations. Afterwards, the distance can be achieved as it was done for the circle. Note that this is not the geometric distance which would require to take the length of the line perpendicular to the ellipse through m.

Thus, the quality function for the problem of robust ellipse finding can be formalized as

$$q_m(x, y, a, b) = \Phi\left(\sqrt{\left(\frac{m_x - x}{a}\right)^2 + \left(\frac{m_y - y}{b}\right)^2} - 1\right) \tag{3.47}$$

where x and y describe the center and a and b the length of the main axes of the ellipse, which are assumed to be parallel to the axes of the coordinate system.

Therefore, d_m is defined as

$$d_m(x, y, a, b) = \sqrt{\left(\frac{m_x - x}{a}\right)^2 + \left(\frac{m_y - y}{b}\right)^2} - 1 \tag{3.48}$$

and assuming that $t_r \neq 0$ and $\gamma \neq 0$ the derivatives are

$$\nabla d_m(x, y, a, b) = \begin{pmatrix} \frac{m_x - x}{t_r a^2} \\ \frac{m_y - y}{t_r b^2} \\ \frac{(m_x - x)^2}{t_r a^3} \\ \frac{(m_y - y)^2}{t_r b^3} \end{pmatrix} \tag{3.49}$$

$$\mathrm{D}\nabla d_m(x, y, a, b) = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{12} & d_{22} & d_{23} & d_{24} \\ d_{13} & d_{23} & d_{33} & d_{34} \\ d_{14} & d_{24} & d_{34} & d_{44} \end{pmatrix}$$

$$\begin{aligned}
d_{11} &= \frac{a^2(m_y - y)^2}{a^2 \gamma t_r} \\
d_{12} &= (m_x - x)^2 (m_y - y)^2 \frac{-1/t_r^3}{a^2 b^2} \\
d_{13} &= (m_x - x) \frac{(b^2(m_x - x)^2 + 2a^2(m_y - y)^2)}{(b^2 t_r^3 a^5)} \\
d_{14} &= (m_x - x)(m_y - y)^2 \frac{-1}{t_r^3 a^2 b^3}
\end{aligned}$$

$$\begin{aligned}
d_{22} &= \frac{b^2(m_x - x)^2}{b^2 \gamma t_r} \\
d_{23} &= (m_y - y)(m_x - x)^2 \frac{-1}{t_r^3 b^2 a^3} \\
d_{24} &= (m_y - y) \frac{a^2(m_y - y)^2 + 2b^2(m_x - x)^2}{(a^2 t_r^3 b^5)}
\end{aligned}$$

$$
\begin{aligned}
d_{33} &= (m_x - x)^2 \frac{(2b^2(m_x - x)^2 + 3a^2(m_y - y)^2)}{b^2 t_r^3 a^6} \\
d_{34} &= (m_x - x)^2 (m_y - y)^2 \frac{-1}{t_r^3 a^3 b^3} \\
d_{44} &= (m_y - y)^2 \frac{(2a^2(m_y - y)^2 + 3b^2(m_x - x)^2)}{a^2 t_r^3 b^6}
\end{aligned} \tag{3.50}
$$

where

$$
\begin{aligned}
t_r &= \sqrt{\left(\frac{m_x - x}{a}\right)^2 + \left(\frac{m_y - y}{b}\right)^2} \\
\gamma &= b^2(m_x - x)^2 + a^2(m_y - y)^2
\end{aligned} \tag{3.51}
$$

Note that ϵ has to be smaller than one for this parameterization. Otherwise, points with a distance larger than the length of the axes are regarded as being part of the ellipse. Then, all points inside of the ellipse would be part of the ellipse and therefore, an ellipse is returned which covers many points adding to the sum of Q instead of actually finding an ellipse that fits to the data points. There has to be an area inside the ellipse where points do not contribute to Q.

We need to show that if the dynamic switching function equals true, d_m is twice continuous differentiable on x. It is sufficient to show that t_r and γ are larger than zero. All other functions used in the calculations are two time continuous differentiable.

Theorem. *If the dynamic switching function is true, then $t_r > 0$ and $\gamma > 0$.*

Proof. Because the dynamic switching function is true, we know that $\forall\ m \in M$ contributing to Q :

$$
\Phi\left(\sqrt{\left(\frac{m_x - x}{a}\right)^2 + \left(\frac{m_y - y}{b}\right)^2} - 1\right) = \Phi(t_r - 1) > 0 \tag{3.52}
$$

By the definition of our Φ, this requires

$$
|t_r - 1| < \epsilon \tag{3.53}
$$

what is equivalent to

$$t_r - 1 < \epsilon \quad \wedge \quad 1 - t_r < \epsilon \tag{3.54}$$

$$\Rightarrow \quad 1 - \epsilon < t_r < \epsilon + 1 \tag{3.55}$$

Because $\epsilon < 1$ it is true that

$$t_r > 0. \tag{3.56}$$

Since this is true,

$$m_x - x \neq 0 \quad \vee \quad m_y - y \neq 0. \tag{3.57}$$

Thus, γ is also larger than zero. □

Finally, the first derivative evaluates to:

$$f_m(x, y, a, b) = -\Phi'(\Delta t) \cdot \begin{pmatrix} \frac{m_x - x}{t_r a^2} \\ \frac{m_y - y}{t_r b^2} \\ \frac{(m_x - x)^2}{t_r a^3} \\ \frac{(m_y - y)^2}{t_r b^3} \end{pmatrix} \tag{3.58}$$

And the second derivative to:

$$\mathrm{D}f_m(x, y, a, b) = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{12} & d_{22} & d_{23} & d_{24} \\ d_{13} & d_{23} & d_{33} & d_{34} \\ d_{14} & d_{24} & d_{34} & d_{44} \end{pmatrix}$$

$$
\begin{aligned}
d_{11} &= \frac{a^2(m_y-y)^2\Phi'(\Delta t)+b^2(m_x-y)^2t_r\Phi''(\Delta t)}{a^2\gamma t_r}\\
d_{12} &= (m_x-x)^2(m_y-y)^2\frac{-\Phi'(\Delta t)/t_r^3+\Phi''(\Delta t)/t_r^2}{a^2b^2}\\
d_{13} &= (m_x-x)\left(\frac{(b^2(m_x-x)^2+2a^2(m_y-y)^2)\Phi'(\Delta t)}{(b^2t_r^3a^5)}\right.\\
&\quad \left.+\frac{a^2b^2(m_x-x)^2\Phi''(\Delta t)}{\gamma a^5}\right)\\
d_{14} &= (m_x-x)(m_y-y)^2\frac{-\Phi'(\Delta t)/t_r^3+\Phi''(\Delta t)/t_r^2}{a^2b^3}\\
d_{22} &= \frac{b^2(m_x-x)^2\Phi'(\Delta t)+a^2(m_y-y)^2t_r\Phi''(\Delta t)}{b^2\gamma t_r}\\
d_{23} &= (m_y-y)(m_x-x)^2\frac{-\Phi'(\Delta t)/t_r^3+\Phi''(\Delta t)/t_r^2}{b^2a^3}\\
d_{24} &= (m_y-y)\left(\frac{(a^2(m_y-y)^2+2b^2(m_x-x)^2)\Phi'(\Delta t)}{(a^2t_r^3b^5)}\right.\\
&\quad \left.+\frac{b^2a^2(m_y-y)^2\Phi''(\Delta t)}{\gamma a^5}\right)\\
d_{33} &= (m_x-x)^2\left(\frac{(2b^2(m_x-x)^2+3a^2(m_y-y)^2)\Phi'(\Delta t)}{b^2t_r^3a^6}\right.\\
&\quad \left.+\frac{a^2b^2(m_x-x)^2\Phi''\Delta}{\gamma a^6}\right)\\
d_{34} &= (m_x-x)^2(m_y-y)^2\left(\frac{-\Phi'(\Delta t)}{t_r^3a^3b^3}+\frac{\Phi''(\Delta t)}{t_r^2a^3b^3}\right)\\
d_{44} &= (m_y-y)^2\left(\frac{(2a^2(m_y-y)^2+3b^2(m_x-x)^2)\Phi'(\Delta t)}{a^2t_r^3b^6}\right.\\
&\quad \left.+\frac{b^2a^2(m_y-y)^2\Phi''\Delta}{\gamma b^6}\right)
\end{aligned}
\tag{3.59}
$$

where

$$\Delta t = t_r - 1 \tag{3.60}$$

Repeated terms should be evaluated only once to accelerate the evaluation. This is done in the code corresponding to the later results. These optimizations are left out here to not introduce too many variables.

Analogously to the second parameterization of the circle, this scales the allowed error defined by ϵ with the size of the current match. It also does it

separately for both main axes. This has to be remembered when these formulas are used. For the ellipse, keeping ϵ as the real allowed distance between the shape of the ellipse and a point, results in a much more complicated parameterization. However, if the main axes of the ellipses that should be found are not too degenerated, the following parameterization approximates this behavior.

$$q_m(x, y, a, b) = \Phi\left(\left(\sqrt{\left(\frac{m_x - x}{a}\right)^2 + \left(\frac{m_y - y}{b}\right)^2} - 1\right)\frac{a+b}{2}\right) \tag{3.61}$$

The downside is, the additional factor complicates the derivatives even more. Besides, scaling the allowed error with the size of the match may be desired, if we assume that the error in the data scales with size of the objects that have to be found.

3.8.2 Algebraic Match Functions

Another useful parameterization comes from the algebraic distance:

An ellipse is defined as all points (m_x, m_y) fulfilling

$$\Gamma(m_x, m_y) = Am_x^2 + Bm_xm_y + Cm_y^2 + Dm_x + Em_y + F = 0 \tag{3.62}$$

The algebraic distance of a point (m_x, m_y) is defined by $\Gamma(m_x, m_y)$. Obviously, if it is zero, the point m belongs to the ellipse. Note that this is actually different from the distance from the first paraterization.

Many approaches have been tried to least square fit an ellipse into a set of points minimizing the algebraic distance. Fitzgibbon, Pilu and Fisher [8] discussed methods based on that approach. One problem is that there are six unknowns but only five degrees of freedom for a rotated ellipse and five unknowns but only four degrees of freedom for the axes aligned ellipse respectively. This equation also describes parabola and hyperbola. To guarantee that the result is an ellipse,

$$B^2 - 4AC < 0 \tag{3.63}$$

has to hold. Otherwise, the result may not be an ellipse. Since we are currently dealing with axes aligned ellipses, we can set $B = 0$. Obviously, (3.63) is satisfied if we choose A and C to be positive. Returning to our setup of qualify functions, if q_m is defined as

$$d_m(x, y, a, b) = Am_x^2 + Cm_y^2 + Dm_x + Em_y + F \tag{3.64}$$

an optimal match considering the algebraic distance is found. Note that in contrast to Fitzgibbon, Pilu and Fisher, this describes a robust fit instead of a least square fit. The geometric equation of the ellipse is

$$\frac{(m_x - p_x)^2}{a^2} + \frac{(m_y - p_y)^2}{b^2} = f \tag{3.65}$$

where p is the center of the ellipse, can be transformed into

$$\frac{m_x^2}{a^2} + \frac{m_y^2}{b^2} - 2\frac{p_x m_x}{a^2} - 2\frac{p_y m_y}{b^2} + \frac{p_x^2}{a^2} + \frac{p_y^2}{b^2} - f = 0 \tag{3.66}$$

Comparing coefficients with (3.64), the unknowns get

$$A = \frac{1}{a^2},\ C = \frac{1}{b^2},\ D = -2\frac{p_x}{a^2},\ E = -2\frac{p_y}{b^2},$$
$$F = \frac{p_x^2}{a^2} + \frac{p_y^2}{b^2} - f \tag{3.67}$$

Without loss of generality, we can set $f = 1$, because the scaling of equation (3.65) produced by f can be calculated into a and b. It is still possible to represent every ellipse with $f = 1$ except for the degenerated case where $f = 0$.

$$F = \frac{p_x^2}{a^2} + \frac{p_y^2}{b^2} - 1 = \frac{D^2}{4A} + \frac{E^2}{4C} - 1 \tag{3.68}$$

Switching our parameter spaces, we can calculate the new bounds where a solution has to be found in with respect to the new parameter space by (3.67) and redefine q_m as

$$d_m(A, C, D, E) = Am_x^2 + Cm_y^2 + Dm_x + Em_y + \frac{D^2}{4A} + \frac{E^2}{4C} - 1 \tag{3.69}$$

This parameterization turns out to be far less efficient than the one described at the beginning. The parameter space gets deformed in a nonlinear way and for branch and bound, splitting an interval in the middle is not optimal anymore. A more efficient parameterization than this can be obtained by combining the idea of the geometric and algebraic distance. Setting $D = 0$ and $E = 0$ shifts the center of the ellipse as in the first parameterization, we get

$$d_m(x, y, A, C) = A(x - m_x)^2 + C(y - m_y)^2 - 1 \tag{3.70}$$

The center of the ellipse gets shifted and only the lengths of the ellipse axes are handled the algebraic way. This parameterization still was not as efficient as the geometric parameterization shown before. Therefore, the experimental results that will be discussed later are based on the first geometric parameterization (3.30).

3.9 Finding Ellipses at Arbitrary Orientations

An rotated ellipse can be parameterized by its center (x, y) and the lengths of its main axes a and b and an angle w between the first main axis and one of the axis of the coordinate system. See figure 3.7 for a visualization.

For every point $m = (m_x, m_y)$, the distance $d_m(w, t)$ to the ellipse primitive can be calculated by rotating the point by w around the origin and scaling the dataset by the lengths of the main axes to transform the ellipse into the unit circle. Afterwards, the distance can be achieved like it was done for the circle. Again this is not the same as the geometric distance for the same reason as for the axes aligned ellipse, but it behaves reasonably in practice for commonly used parameters.

Consequently, the problem of robust rotated ellipse finding can be for-

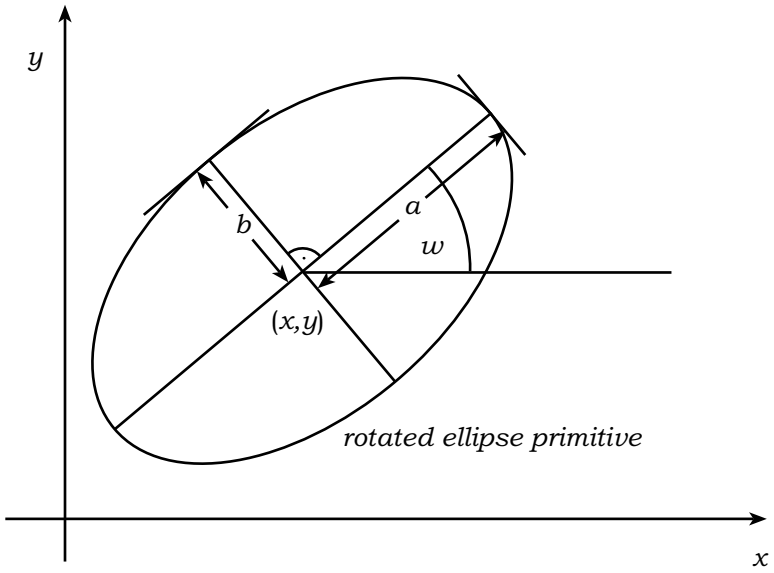

Figure 3.7: Parameterization of the rotated ellipse

malized as

$$q_m(x,y,a,b,w) = \Phi\left(\left(\left(\frac{\cos(w)m_x + \sin(w)m_y - x}{a}\right)^2 + \left(\frac{-\sin(w)m_x + \cos(w)m_y - y}{b}\right)^2\right)^{\frac{1}{2}} - 1\right) \tag{3.71}$$

Therefore, d_m equals to

$$d_m(x,y,a,b,w) = \sqrt{\left(\frac{v_x - x}{a}\right)^2 + \left(\frac{v_y - y}{a}\right)^2} - 1. \tag{3.72}$$

with

$$\begin{aligned} v_x &= \cos(w)m_x + \sin(w)m_y \\ v_y &= -\sin(w)m_x + \cos(w)m_y \end{aligned} \tag{3.73}$$

For the areas where the dynamic switching function is true and if $\epsilon < 1$, the

derivatives of d_m are

$$\nabla d_m(x,y,a,b,w) = \begin{pmatrix} \frac{v_x-x}{t_r a^2} \\ \frac{v_y-y}{t_r b^2} \\ \frac{(v_x-x)^2}{t_r a^3} \\ \frac{(v_y-y)^2}{t_r b^3} \\ -\frac{(v_y-y)v_x}{t_r b^2} + \frac{v_y(v_x-x)}{t_r a^2} \end{pmatrix} \tag{3.74}$$

$$\mathrm{D}\nabla d_m(x,y,a,b) = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{12} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{13} & d_{23} & d_{33} & d_{34} & d_{35} \\ d_{14} & d_{24} & d_{34} & d_{44} & d_{45} \\ d_{15} & d_{25} & d_{35} & d_{45} & d_{55} \end{pmatrix}$$

$$\begin{aligned}
d_{11} &= \frac{1}{a^4}\left(-\frac{(v_x-x)^2}{t_r^3} + \frac{a^2}{t_r}\right) \\
d_{12} &= -\frac{(v_x-x)(v_y-y)}{a^2 b^2 t_r^3} \\
d_{13} &= \frac{v_x-x}{a^5}\left(-\frac{(v_x-x)^2}{t_r^3} + \frac{2a^2}{t_r}\right) \\
d_{14} &= -\frac{(v_y-y)^2(v_x-x)}{a^2 b^3 t_r^3} \\
d_{15} &= \frac{1}{2a^2}\left(\frac{(v_x-x)\eta}{t_r^3} - \frac{2v_y}{t_r}\right) \\
d_{22} &= \frac{1}{b^4}\left(-\frac{(v_y-y)^2}{t_r^3} + \frac{b^2}{t_r}\right) \\
d_{23} &= -\frac{(v_x-x)^2(v_y-y)}{b^2 a^3 t_r^3} \\
d_{24} &= \frac{v_y-y}{b^5}\left(-\frac{(v_y-y)^2}{t_r^3} + \frac{2b^2}{t_r}\right) \\
d_{25} &= \frac{1}{2b^2}\left(\frac{(v_y-y)\eta}{t_r^3} - \frac{2v_x}{t_r}\right)
\end{aligned}$$

$$
\begin{aligned}
d_{33} &= \frac{(v_x - x)^2}{a^6}\left(-\frac{(v_x - x)^2}{t_r^3} + \frac{3a^2}{t_r}\right) \\
d_{34} &= -\frac{(v_x - x)^2(v_y - y)^2}{a^3 b^3 t_r^3} \\
d_{35} &= \frac{v_x - x}{2a^3}\left(\frac{(v_x - x)\eta}{t_r^3} - \frac{4v_y}{t_r}\right) \\
d_{44} &= \frac{(v_y - y)^2}{b^6}\left(-\frac{(v_y - y)^2}{t_r^3} + \frac{3b^2}{t_r}\right) \\
d_{45} &= \frac{v_y - y}{2b^3}\left(\frac{(v_y - y)\eta}{t_r^3} - \frac{4v_x}{t_r}\right) \\
d_{55} &= -\frac{\eta^2}{4t_r^3} + \big((m_x^2 - m_y^2)(a^2 - b^2)\cos(2w) \\
&\quad +(m_y b^2 x - m_x a^2 y)\sin(w) + \cos(w)(m_x b^2 x + m_y a^2 y \\
&\quad +m_x m_y (a^2 - b^2)\sin(w))\big) \cdot \frac{1}{a^2 b^2 t_r}
\end{aligned} \tag{3.75}
$$

where

$$
\begin{aligned}
t_r &= \sqrt{\left(\frac{v_x - x}{a}\right)^2 + \left(\frac{v_y - y}{b}\right)^2} \\
\gamma &= b^2(v_x - x)^2 + a^2(v_y - y)^2 \\
\eta &= 2\left(-\frac{(v_y - y)v_x}{b^2} + \frac{v_y(v_x - x)}{a^2}\right)
\end{aligned} \tag{3.76}
$$

We have to impose the same restriction for the same reason to the ϵ as done for the axes aligned ellipse: $\epsilon < 1$.

We know that quality function of the axes aligned ellipse is twice continuous differentiable if the dynamic switching function is true. The quality function of the rotated ellipse only differs in the previous rotation of the scene. Since rotation is a twice continuous differentiable function, it also holds for the rotated ellipse.

Finally, the first derivative evaluates to:

$$f_m(x,y,a,b,w) = -\Phi'(\Delta t) \begin{pmatrix} \frac{v_x - x}{t_r a^2} \\ \frac{v_y - y}{t_r b^2} \\ \frac{(v_x - x)^2}{t_r a^3} \\ \frac{(v_y - y)^2}{t_r b^3} \\ -\frac{(v_y - y)v_x}{t_r b^2} + \frac{v_y (v_x - x)}{t_r a^2} \end{pmatrix} \tag{3.77}$$

And the second derivative:

$$\mathrm{D}f_m(x,y,a,b,w) = \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{12} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{13} & d_{23} & d_{33} & d_{34} & d_{35} \\ d_{14} & d_{24} & d_{34} & d_{44} & d_{45} \\ d_{15} & d_{25} & d_{35} & d_{45} & d_{55} \end{pmatrix}$$

$$\begin{aligned}
d_{11} &= \frac{1}{a^4}\left(-\frac{(v_x - x)^2\Phi'(\Delta t)}{t_r^3} + \frac{a^2\Phi'(\Delta t)}{t_r} + \frac{(v_x - x)^2\Phi''(\Delta t)}{t_r^2}\right) \\
d_{12} &= \frac{(v_x - x)(v_y - y)}{a^2 b^2}\left(\frac{-\Phi'(\Delta t)}{t_r^3} + \frac{\Phi''(\Delta t)}{t_r^2}\right) \\
d_{13} &= \frac{v_x - x}{a^5}\left(-\frac{(v_x - x)^2\Phi'(\Delta t)}{t_r^3} + \frac{2a^2\Phi'(\Delta t)}{t_r} + \frac{(v_x - x)^2\Phi''(\Delta t)}{t_r^2}\right) \\
d_{14} &= \frac{(v_y - y)^2(v_x - x)}{a^2 b^3}\left(-\frac{\Phi'(\Delta t)}{t_r^3} + \frac{\Phi''(\Delta t)}{t_r^2}\right) \\
d_{15} &= \frac{1}{2a^2}\left(\frac{(v_x - x)\eta\Phi'(\Delta t)}{t_r^3} - \frac{2v_y\Phi'(\Delta t)}{t_r} - \frac{(v_x - x)\eta\Phi''(\Delta t)}{t_r^2}\right) \\
d_{22} &= \frac{1}{b^4}\left(-\frac{(v_y - y)^2\Phi'(\Delta t)}{t_r^3} + \frac{b^2\Phi'(\Delta t)}{t_r} + \frac{(v_y - y)^2\Phi''(\Delta t)}{t_r^2}\right) \\
d_{23} &= \frac{(v_x - x)^2(v_y - y)}{b^2 a^3}\left(-\frac{\Phi'(\Delta t)}{t_r^3} + \frac{\Phi''(\Delta t)}{t_r^2}\right) \\
d_{24} &= \frac{v_y - y}{b^5}\left(-\frac{(v_y - y)^2\Phi'(\Delta t)}{t_r^3} + \frac{2b^2\Phi'(\Delta t)}{t_r} + \frac{(v_y - y)^2\Phi''(\Delta t)}{t_r^2}\right) \\
d_{25} &= \frac{1}{2b^2}\left(\frac{(v_y - y)\eta\Phi'(\Delta t)}{t_r^3} - \frac{2v_x\Phi'(\Delta t)}{t_r} - \frac{(v_y - y)\eta\Phi''(\Delta t)}{t_r^2}\right)
\end{aligned}$$

$$
\begin{aligned}
d_{33} &= \frac{(v_x-x)^2}{a^6}\left(-\frac{(v_x-x)^2\Phi'(\Delta t)}{t_r^3}+\frac{3a^2\Phi'(\Delta t)}{t_r}+\frac{(v_x-x)^2\Phi''(\Delta t)}{t_r^2}\right)\\
d_{34} &= \frac{(v_x-x)^2(v_y-y)^2}{a^3b^3}\left(-\frac{\Phi'(\Delta t)}{t_r^3}+\frac{\Phi''(\Delta t)}{t_r^2}\right)\\
d_{35} &= \frac{v_x-x}{2a^3}\left(\frac{(v_x-x)\eta\Phi'(\Delta t)}{t_r^3}-\frac{4v_y\Phi'(\Delta t)}{t_r}-\frac{(v_x-x)\eta\Phi''(\Delta t)}{t_r^2}\right)\\
d_{44} &= \frac{(v_y-y)^2}{b^6}\left(-\frac{(v_y-y)^2\Phi'(\Delta t)}{t_r^3}+\frac{3b^2\Phi'(\Delta t)}{t_r}+\frac{(v_y-y)^2\Phi''(\Delta t)}{t_r^2}\right)\\
d_{45} &= \frac{v_y-y}{2b^3}\left(\frac{(v_y-y)\eta\Phi'(\Delta t)}{t_r^3}-\frac{4v_x\Phi'(\Delta t)}{t_r}-\frac{(v_y-y)\eta\Phi''(\Delta t)}{t_r^2}\right)\\
d_{55} &= -\frac{\eta^2\Phi'(\Delta t)}{4t_r^3}+\big((m_x^2-m_y^2)(a^2-b^2)\cos(2w)\\
&\quad +(m_yb^2x-m_xa^2y)\sin(w)+\cos(w)(m_xb^2x+m_ya^2y\\
&\quad +m_xm_y(a^2-b^2)\sin(w))\big)\cdot\frac{\Phi'(\Delta t)}{a^2b^2t_r}+\frac{\eta^2\Phi''(\Delta t)}{4t_r^2}
\end{aligned}
\tag{3.78}
$$

where

$$\Delta t = t_r - 1 \tag{3.79}$$

Because the scene is rotated around the origin of the coordinate system, the scene has to be transformed to be around zero. Alternatively, the scene can be rotated around the center of the ellipse:

$$
\begin{aligned}
q_m(x,y,a,b,w) = \Phi\Bigg(\Bigg(\Bigg(&\frac{\cos(w)(x-m_x)+\sin(w)(y-m_y)}{a}\Bigg)^2+\\
&\left(\frac{-\sin(w)(x-m_x)+\cos(w)(y-m_y)}{b}\right)^2\Bigg)^{\frac{1}{2}}-1\Bigg)
\end{aligned}
\tag{3.80}
$$

This parameterization turned out to be less efficient than the previous one in the experiments because of even more terms produced in the derivatives.

Chapter 4

Evaluation

This chapter provides the results of performance tests between the several variations of the branch and bound method and Interval Newton method.

4.1 The Datasets

The algorithms are tested on different geometric matching problems. For each problem equivalence classes are defined and representative datasets are selected. Each dataset consists of a set of points from which the optimal geometry of the particular problem has to be found. The considered problems are:

- line finding
- circle finding
- ellipse finding
- finding ellipses at arbitrary orientations

See figure 4.1 for examples of these problems. The domain of the points is $[-1, 1]^2$. The points get an additional error to the defining primitive. For each problem, the error is chosen randomly for each dataset between 0 and 0.01.

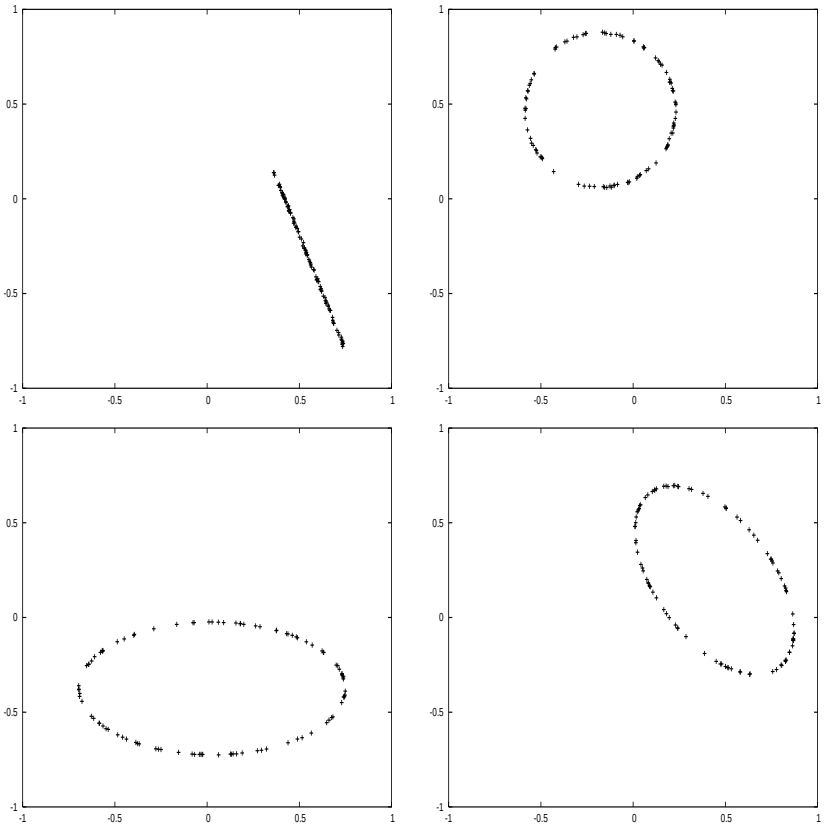

Figure 4.1: Examples of the datasets used in the experiments.

In practice, this means, assuming a picture of 1000^2 pixels, the displacements of the features are allowed to be within five pixels. The equivalence classes for each problem are:

- simple data only consisting of points describing the primitive.
- simple data and additional uniformly distributed random points.
- only uniformly distributed random points.

See Figure 4.2 for examples. The third class represents the worst case because in general there is no favored match to find. Everywhere in parameter space the quality of a match is almost identical. The first class represents the best situation for the algorithm. It shows the lower limit of the time needed. The second class describes a typical situation. An optimal match

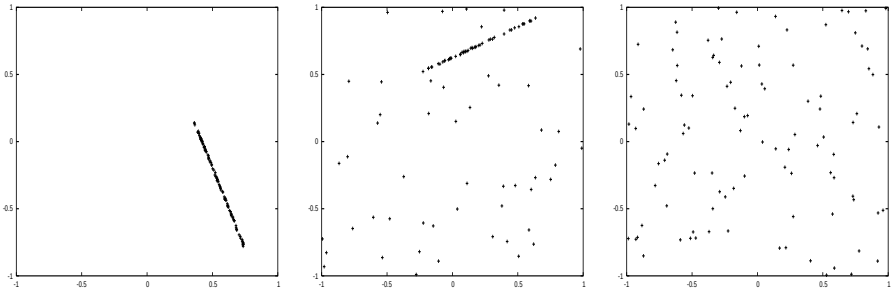

Figure 4.2: Examples of the classes of test cases for line finding. From the left to the right, the pictures show class one to class three.

clearly exists but is harder to find because of additional noise in the given data.

The number of points in the three classes is always 100. For the first class, all points belong to the primitive that has to be found. For the second class, 50 points belong to the primitive and 50 are additional uniformly distributed random points. The third class only consists of random points.

4.2 Evaluation

We applied five algorithms to the datasets of the line and the circle, all slightly modified versions of the branch and bound or Interval Newton method.

- the original branch and bound method
- the branch and bound method using matchlists
- the plain Interval Newton method
- the Interval Newton method using dynamic switching to decide whether to try a Newton step or not
- the Interval Newton method with dynamic switching and matchlists

For the ellipse and rotated ellipse datasets only the best representatives of branch and bound and Newton were tested, namely the branch and bound

method using matchlists and the Interval Newton method with dynamic switching and matchlists, items two and five in the previous list.

4.3 The Empirical Results

In this section the results of the tests are presented and discussed. All the described tests were evaluated on a Pentium 4 with 2.4 GHz and 2 GB of RAM. The gcc compiler version 3.3.3 with optimization for speed -O2 was used.

4.3.1 The Line Finding Problem

Figures 4.3 and 4.4 show the results for the three classes for the line finding problem. There are several observations to make from these plots. First of all, we see that the calculation time needed for the Newton methods nearly stops to raise at some point of accuracy while the branch and bound methods keep rising it. The simple Newton method without dynamic switching is the worst in the beginning. Therefore, by applying dynamic switching, the Newton method behaves like branch and bound in the beginning and at some point, it switches to using Newton steps. Using Newton steps occurs later, the more random noise there is in the datasets. For the plain line, it starts at an accuracy of 0.002. For the line with random points this point is already shifted to an accuracy of 0.003. For the random points, it starts with an accuracy of 0.0001. In the beginning, branch and bound steps are used to separate the noise points from the points belonging to the line and later on, Newton steps are used to achieve a higher accuracy.

Another point to be mentioned is the influence of the matchlists on the speed. In the first case, the matchlist versions are even slower than the methods using matchlists, because all points in the dataset belong to the optimal solution. So the overhead of keeping the matchlist data structures does not turn out to be more efficient in this case since the calculation of the distance test in the non-matchlists version is cheap enough. As mentioned

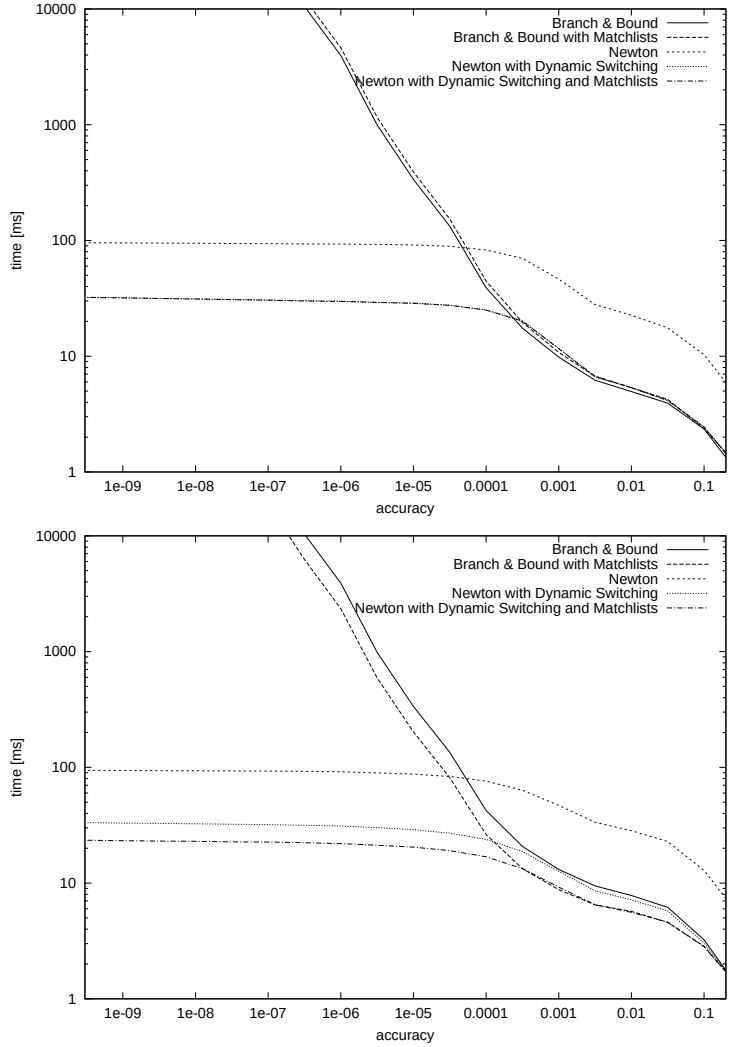

Figure 4.3: Speed comparison between the five methods discussed in the text on the first two classes of datasets for the line finding problem. The first one is for the plain line, the second for the line with additional random points.

earlier, this can be done efficiently by reusing the calculated terms for the evaluation of the function to be optimized. This advantage vanishes as the random noise in our dataset gets larger. In the second plot, the matchlist versions are already faster and in the last plot, where there are only random points, the difference gets even larger. In practical situations we will probably always have some noise, so in general using matchlists is the right choice. But if we know for some reason that the data is nice, choosing not to use matchlists might be better.

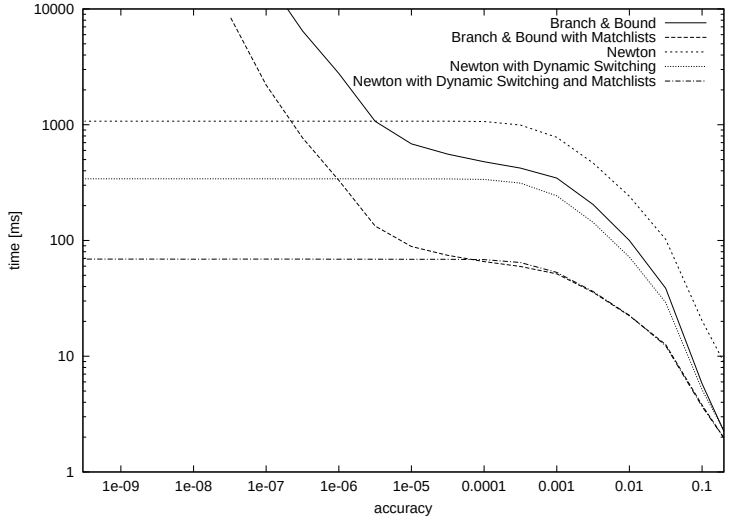

Figure 4.4: Speed comparison between the five methods discussed in the text on the third class, having only random points, for the line finding problem.

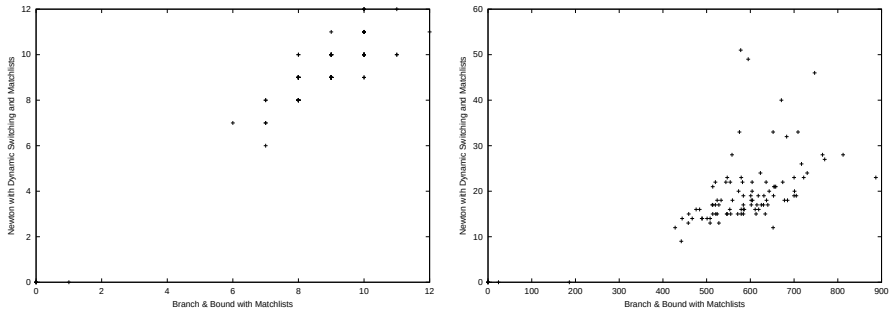

Figure 4.5: The 100 representatives of the class of line dataset with random points that were used. On the left side the times to get an accuracy of $10^{-2.5}$ is plotted and on the right for an accuracy of 10^{-5} per parameter in parameter space. Times are given in milliseconds.

In Figure 4.5 the representatives of the class of the line dataset with random points that were used are shown. The plots are only for the branch and bound method with matchlists and Newton method with dynamic switching and matchlists. Again we see that at low accuracy the methods behave the same while at higher accuracy Newton has an advantage. We also see that not only the calculation time rises for higher accuracy, but the distribution of the single problems rises, too. Figures 4.6 and 4.7 show the maximum and minimum times taken for a specific accuracy over the datasets. As we can see, the behavior discussed above for the averaged plots also applies to the easiest and hardest datasets. Note that for the case of equivalence class

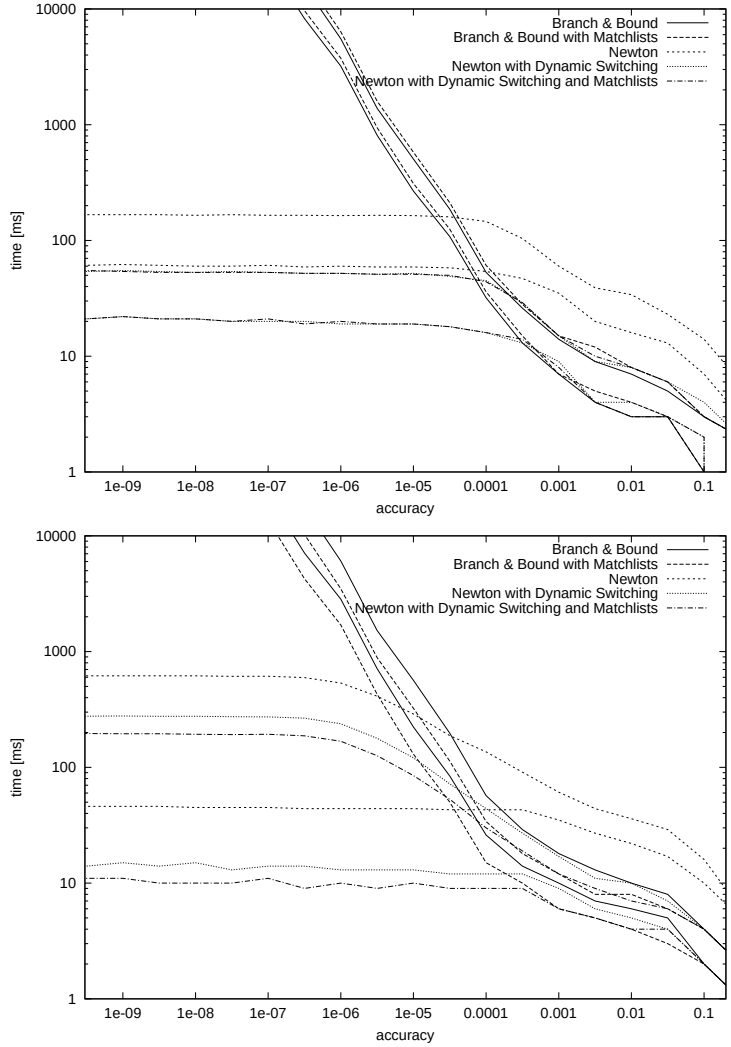

Figure 4.6: Maximum and minimum of the running times over the 100 datasets for a given accuracy for the line finding problem. The first plot corresponds to equivalence class one, the second to class two. Note that the waves in the plots come from measurement accuracies. Since the values are minima and maxima, there no averaging occurred that removes these artifacts.

three, the methods with matchlists on the worst dataset were faster than the corresponding method without matchlists on the simplest dataset. This underlines the efficiency of matchlists on noisy input data. There are very few cases where branch and bound is better. For example, for the given datasets with some noise, Newton is always the better choice.

Figure 4.8 shows the number of iterations for a desired accuracy. The

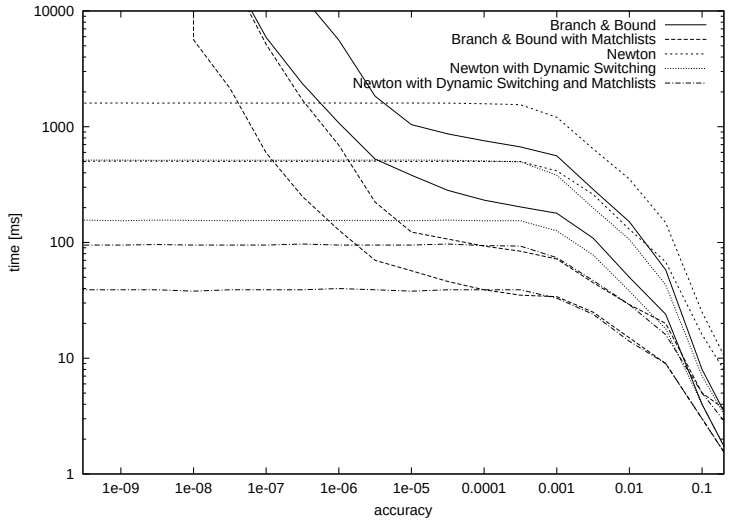

Figure 4.7: Maximum and minimum of the times over the 100 datasets for a given accuracy for the line finding problem. The plot corresponds to the third class of datasets.

values are achieved from the line finding problem with one line and additional random points. Again the values are averaged over 100 datasets. The first plot is related to the Newton method without dynamic switching. The second plot represents the Newton method with dynamic switching. The area from the bottom to the first graph describes the number of iterations spent with branch and bound steps. The uppermost area describes the number of iterations spent with successful Newton steps. The area in between represents the number of unsuccessful Newton steps. Note that in the case of the Newton method without dynamic switching, the number of iterations for the Branch and Bound steps are zero. As we can see, the complete numbers of iterations are nearly equal. The area of unsuccessful Newton steps in the case of plain Newton is substituted with branch and bound steps if we use dynamic switching. There is just a small number of iterations used for unsuccessful Newton steps left. Therefore, our dynamic switching function is well adapted to the line finding problem.

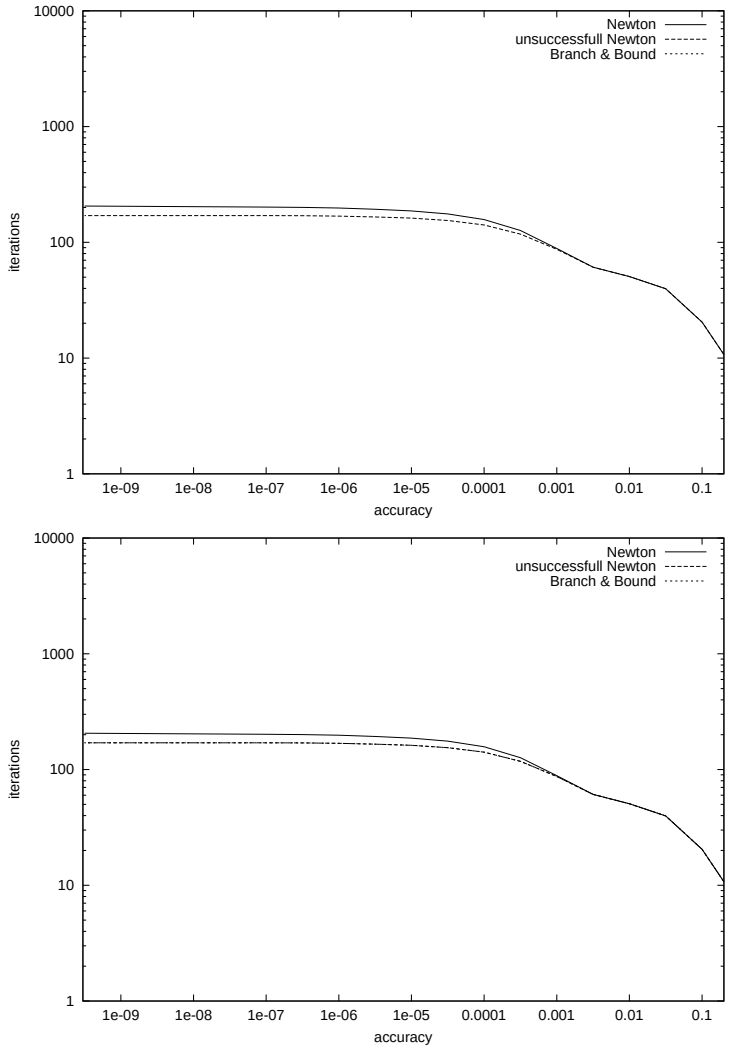

Figure 4.8: The effectiveness of dynamic switching with respect to the line finding problem. The first plot is related to the Newton method without dynamic switching. The second plot represents the Newton method with dynamic switching.

4.3.2 The Circle Finding Problem

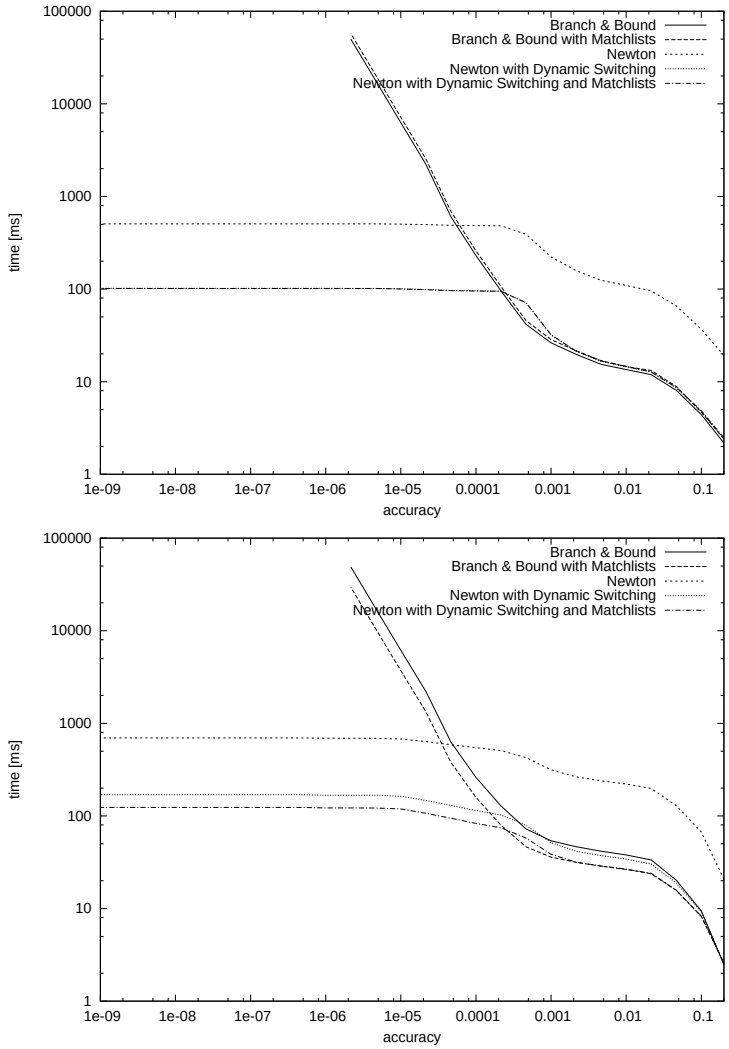

Figure 4.9: Speed comparison between the five methods discussed in the text on the first two classes of datasets for the circle finding problem. The first one is for the plain circle, the second for the circle with additional random points.

The experimental results of the circle finding problem show the same behavior as the results of the line finding problem. Figures 4.9 and 4.10 show the results for the three classes of the circle finding problem. Again we see that the Newton method at some point start to converge extremely fast to the solution, offering higher accuracy at practically no additional cost from that point on, while the time needed by the branch and bound

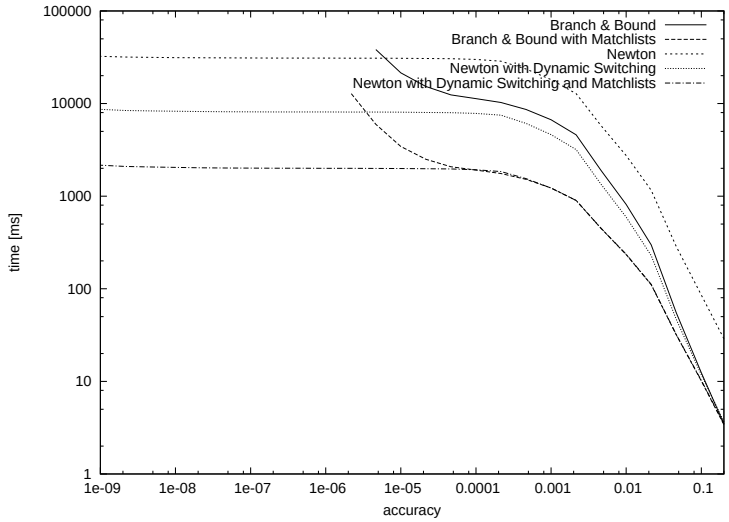

Figure 4.10: Speed comparison between the five methods discussed in the text on the third class, having only random points, for the circle finding problem.

methods increases with the desired accuracy. In the beginning, the plain Newton method is the worst. Once again, it turned out to be efficient to use branch and bound steps in the beginning and to switch to Newton steps later on, decided by the Dynamic Switching Function.

We also see that the use of matchlists accelerates the algorithm depending on the noise in the dataset. For the plain circle dataset, there is practically no difference between the methods not using matchlists and the methods using them. In contrast to the line finding problem, where the overhead produced by the handling of the data structures for the matchlists slowed down the method applied to the plain primitive dataset, the overhead is negligible in the circle case with respect to the higher complexity of calculations per step. For the second dataset with 50% noise, there is already an acceleration and for the third dataset only consisting of noise, the speed increase is even higher. Thus, if a problem is not very simple, using matchlists is the right choice.

Figure 4.11 shows the maximum and minimum times taken for a specific accuracy over the datasets. As we can see, the behavior discussed above for the averaged plots also applies to the easiest and hardest datasets. Therefore there are very few cases where branch and bound is better. For the given

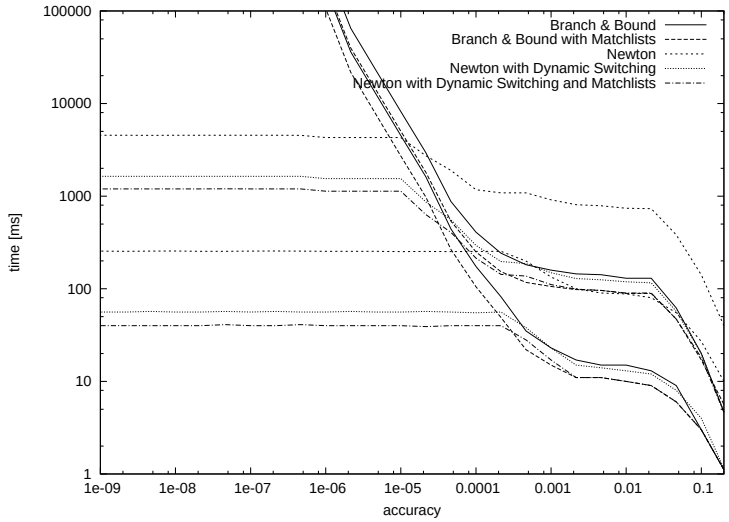

Figure 4.11: Maximum and minimum of the times over the 100 datasets for a given accuracy for the circle finding problem. The second class, containing a circle and additional random points is shown.

datasets, for example, Newton is always the better choice.

Figure 4.12 shows the number of iterations with respect to accuracy. The values are achieved from the circle finding problem with one circle and additional random points. Again the values are averaged over 100 datasets. The first plot is related to the Newton method without dynamic switching. The second plot represents the Newton method with dynamic switching. The graph have to be interpreted as figure 4.8 for the line finding problem. Note that in the case of the Newton method without dynamic switching, the number of iterations for the Branch and Bound steps is zero. As we can see, the complete numbers of iterations are nearly equal. The unsuccessful Newton steps in the case of plain Newton are replaced by branch and bound steps if we use dynamic switching. Just a small number of iterations used for unsuccessful Newton steps is left. Therefore, our dynamic switching function is well suited to the circle finding problem.

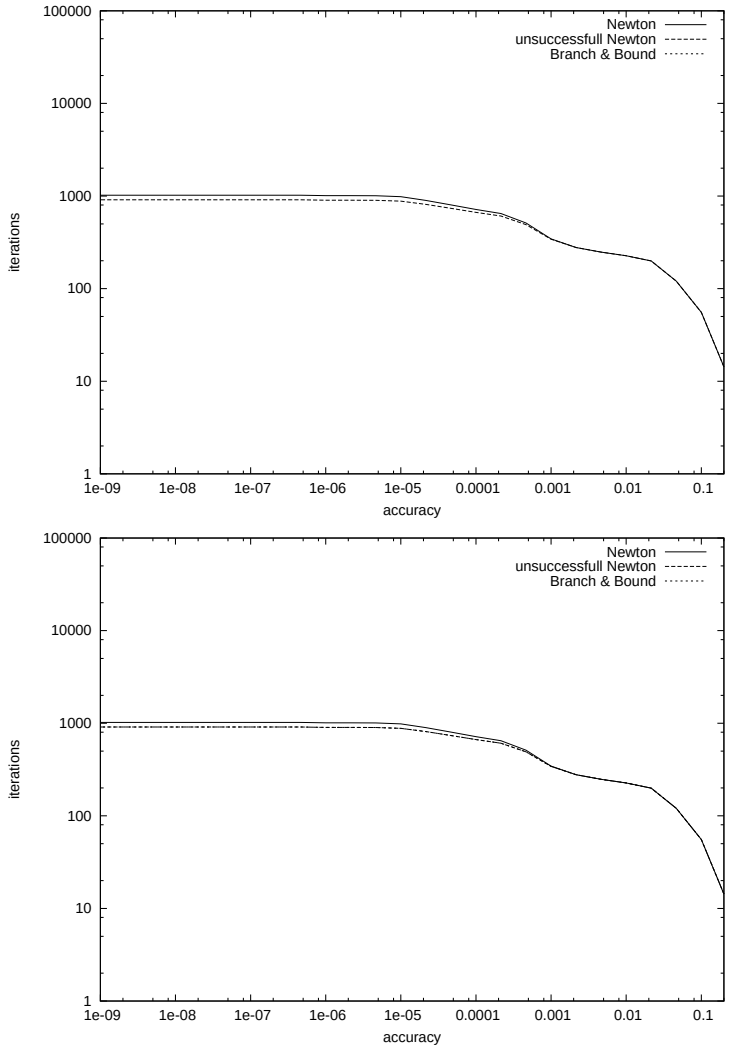

Figure 4.12: The effectiveness of dynamic switching with respect to the circle finding problem. The first plot is related to the Newton method without dynamic switching. The second plot represents the Newton method with dynamic switching.

4.3.3 The Ellipse Finding Problem

This section discusses the results for the ellipse finding problem. Because we already saw the advantage of using matchlists and dynamic switching, the latter only for Newton, we omit measurements for the methods not using them. Therefore, only branch and bound with matchlists and Newton with dynamic switching and matchlists will be compared in the following text.

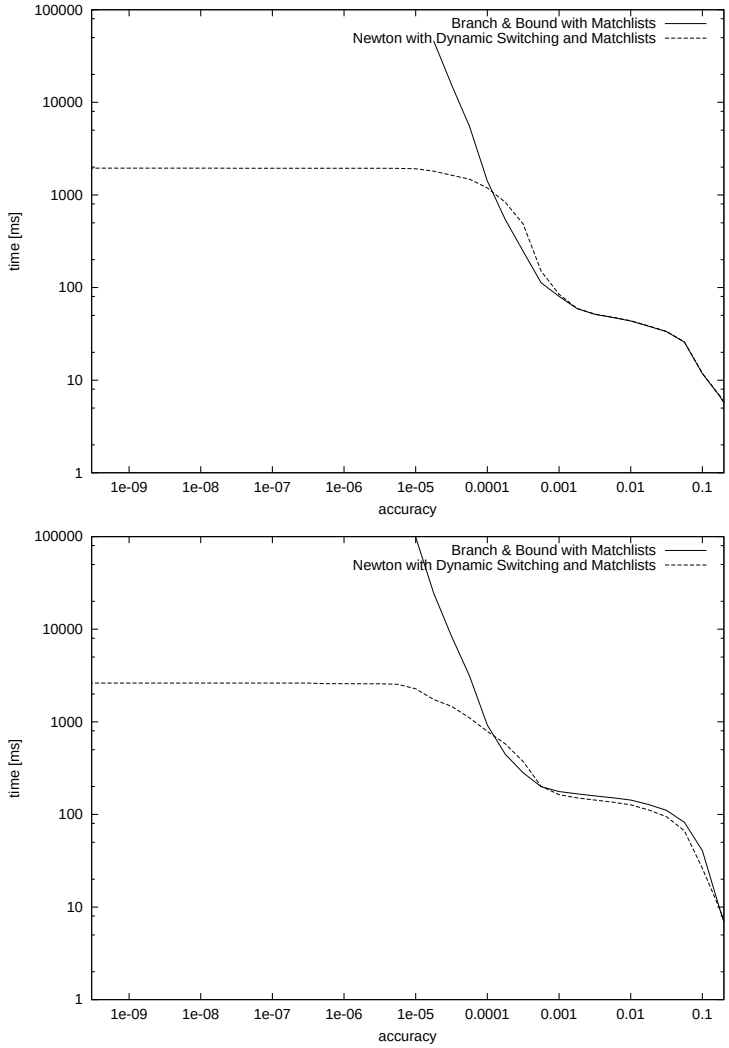

Figure 4.13: Speed comparison between the branch and bound method with matchlists and the Newton method with dynamic switching and matchlists on the first two classes of datasets for the ellipse finding problem. From the top to the bottom: plain ellipse, ellipse with additional random points.

Figure 4.13 and 4.14 show the speed measurements for the two methods

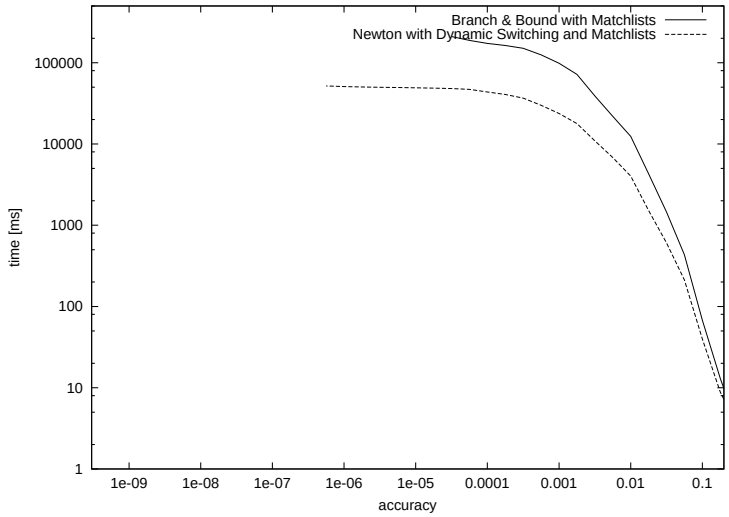

Figure 4.14: Speed comparison on the third class, having only random points, for the ellipse finding problem.

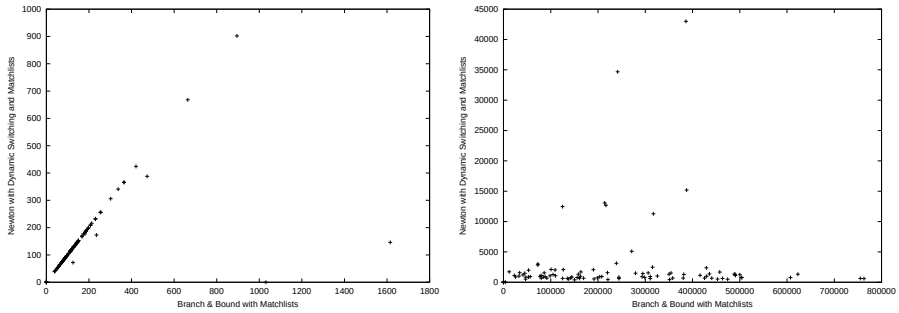

Figure 4.15: The 100 representatives of the ellipse dataset with random points. The times are with respect to an required accuracy of $10^{-2.5}$ and 10^{-5} per parameter in parameter space. The times are given in milliseconds.

on the ellipse finding problem averaged over 100 datasets. The behavior of the methods we already know from the previous problems are also visible in the first two plots. At some point the Newton method gets higher accuracy nearly for free while the branch and bound method needs a lot of time to gain a high accuracy. In comparison to the previous problems, we see that the problem is harder to solve than finding a line or a circle. The point where Newton stops to take time for higher precision is later than for the other problems. We can also see that there is an area of accuracy where branch and bound is faster than Newton. We can explain that by either Newton steps that were not optimal or by Newton steps being optimal, but

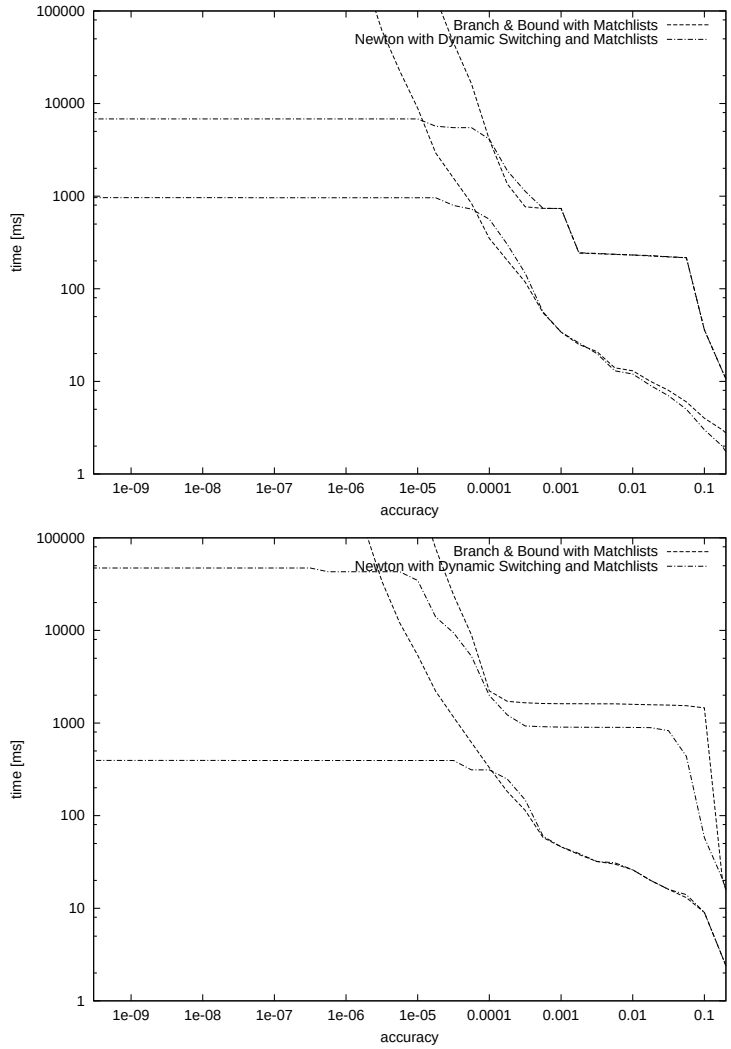

Figure 4.16: Maximum and minimum of the times over the 100 datasets for a given accuracy for the ellipse finding problem. The first corresponds to the plain ellipse datasets. The second to the ellipse datasets with additional uniformly distributed points.

only shrinking the interval by less than one half. If splitting the interval by half is better, branch and bound is faster in that case. In the third plot the behavior might be different. Newton is still faster, but we cannot see if branch and bound might also perhaps behave like Newton. Probably, it does not, but the calculations are not run long enough to ensure this guess from the averaged plot. For estimates of the behavior of the methods for practical applications, the statements we got from the second plot are the most important.

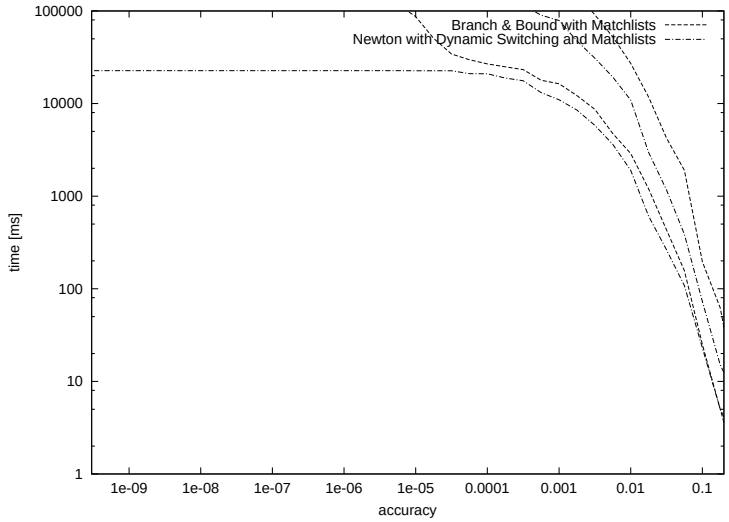

Figure 4.17: Maximum and minimum of the times over the 100 datasets for a given accuracy for the ellipse finding problem. The plot corresponds to the random datasets.

In Figure 4.15, we see again that the Newton method behaves like the branch and bound method in the beginning. At higher accuracies, it switches to Newton steps and gains speed. An interesting observation is that in general, Newton becomes faster for high accuracy, but comparing single datasets we see that most of them are easy to solve, taking less than three seconds for an accuracy of 10^{-5} while there are also a few datasets taking drastically more time. For real time applications, these few datasets destroy a small bound for the needed time for specific datasets. If the method should be used in an area where guaranteed low bounds for time with respect to a specified accuracy is needed, the problem created by so few datasets should be investigated further in detail first.

Looking at the minimum and maximum of the time taken for the sample datasets 4.16, we see that the behavior observed from the averaged plots also applies to the minimum and maximum. Therefore, we can expect this behavior in practical situations. For the random dataset 4.17 we can see that the expected behavior also applies to the minimum. So that is one more reason why our original guess, that the behavior of the methods also applies for the random dataset is right.

Observing the iterations made, 4.18 shows that the dynamic switching

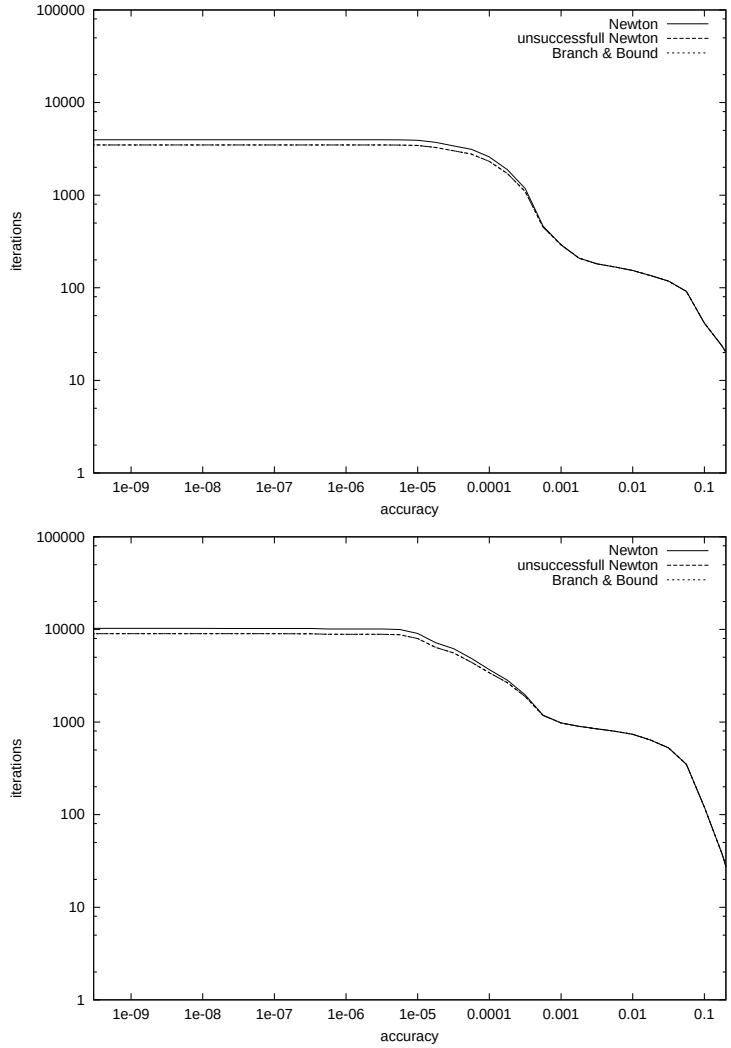

Figure 4.18: The effectiveness of dynamic switching. The plots represent the Newton method with dynamic switching. The first corresponds to the plain ellipse datasets. The second to the ellipse datasets with additional points.

function is at least nearly optimal for the ellipse finding problem. Nearly every Newton step made was successful. For the random dataset 4.19 no Newton steps were made. The method did not reach the area of accuracy where the quality function becomes twice continuous differentiable within the maximum calculation time specified. Newton steps would not have been successful and therefore, no Newton steps were made.

Concluding the measurements of the ellipse finding problem, one can say that the Newton method is faster than the branch and bound method and

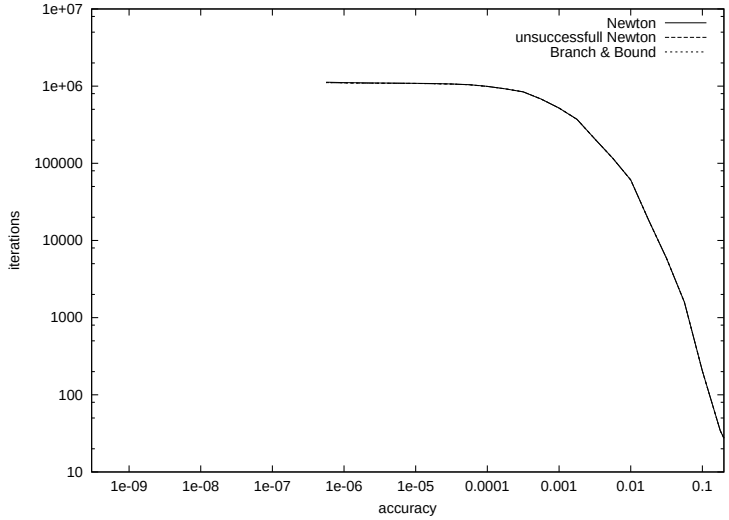

Figure 4.19: The effectiveness of dynamic switching. The plot represents the Newton method with dynamic switching and the plot corresponds to the random datasets.

the behavior of the Newton method, getting accuracy for free at some point, is valid for this problem, too. The disadvantages are, that the worst case where no ellipse can be found, takes much longer than the case where an ellipse is in the dataset. Furthermore, there were some special datasets that took much longer than the majority of datasets. If a system tries to find ellipses in a dataset where there might be no ellipses, this method has a comparatively high bound of runtime.

4.3.4 Finding Ellipses at Arbitrary Orientations

This section discusses the results for the rotated ellipse finding problem. As already seen from the previous problems, the point where the Newton method actually uses Newton steps recedes as the number of dimensions of the problem rises. Figure 4.20 and 4.21 show that this point recedes so much, that the behavior of the branch and bound method and Newton are the same within the measured time interval. The minimum and maximum plots 4.23 show basically the same, but there is at least one dataset of the second class where the Newton method comes to the point with a high speed of convergence.

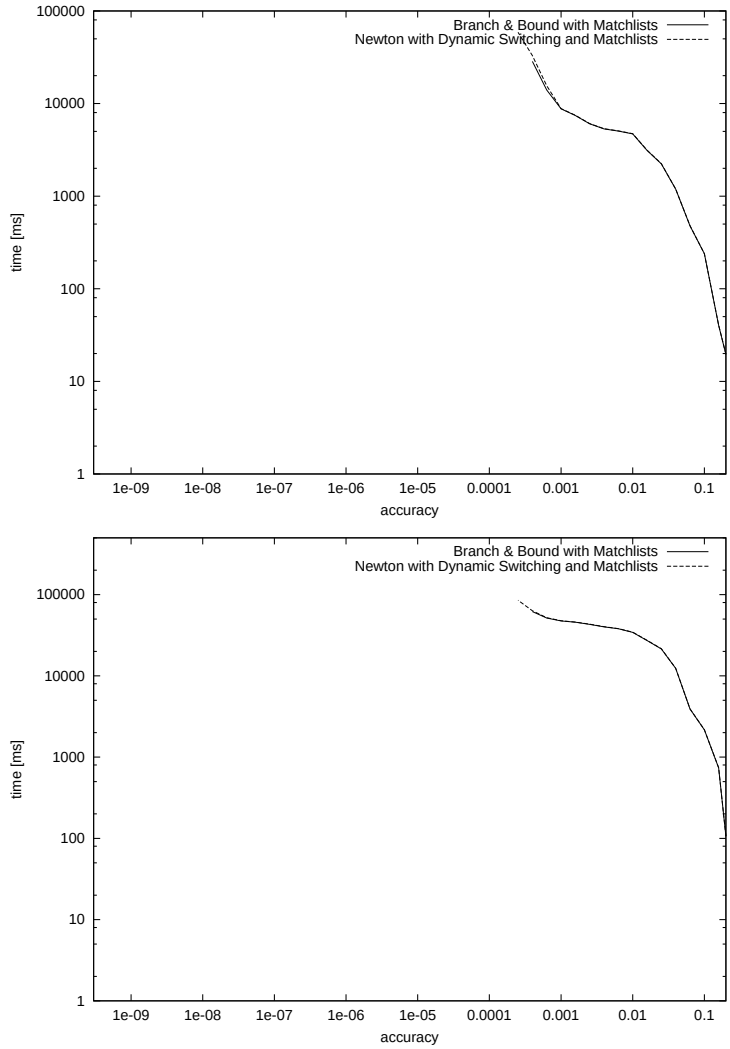

Figure 4.20: Speed comparison between the branch and bound method with matchlists and the Newton method with dynamic switching and matchlists on the first two classes of datasets for the rotated ellipse finding problem. From the top to the bottom: plain rotated ellipse, rotated ellipse with additional random points.

In Figure 4.22 we see again that the behavior of the two methods are the same at the beginning. But later on, for single datasets, the two methods take different steps. Newton actually makes some Newton steps, but it does not get clearly faster than the branch and bound method. Instead, for some datasets, Newton is faster, for others branch and bound is. Averaged over the datasets, both methods are nearly equally fast as already noted. The number of Newton steps that were done is very low. In the iterations plots

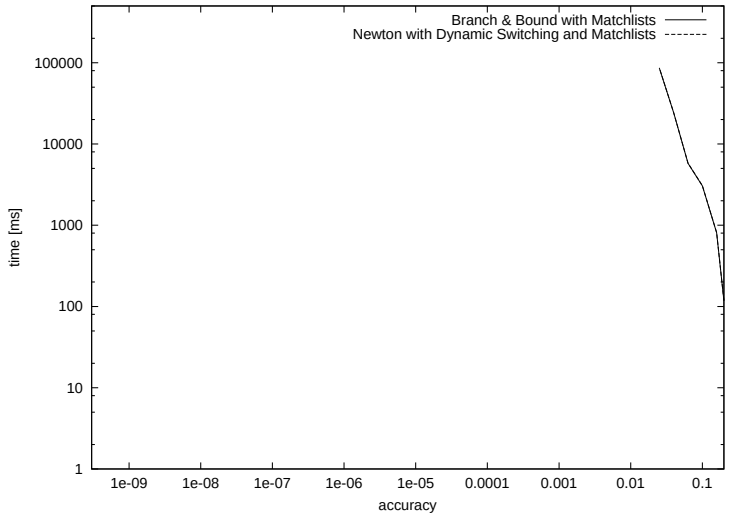

Figure 4.21: Speed comparison on the third class of datasets, having only random points, for the rotated ellipse finding problem.

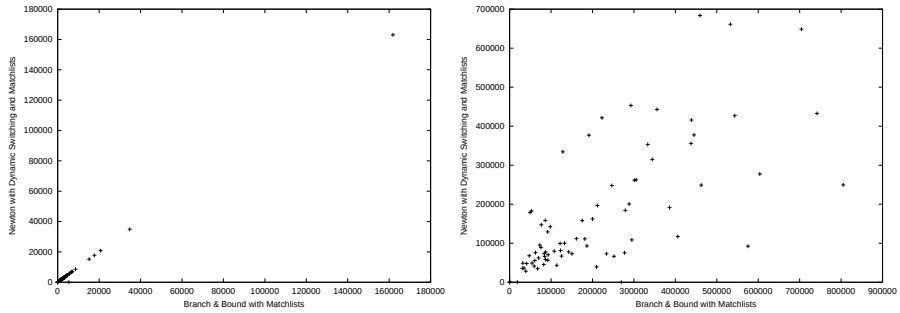

Figure 4.22: The 100 representatives of the rotated ellipse dataset. The times are with respect to an required accuracy of 10^{-2}, 10^{-4} per parameter in parameter space. Times are given in milliseconds.

4.24 they are not even visible.

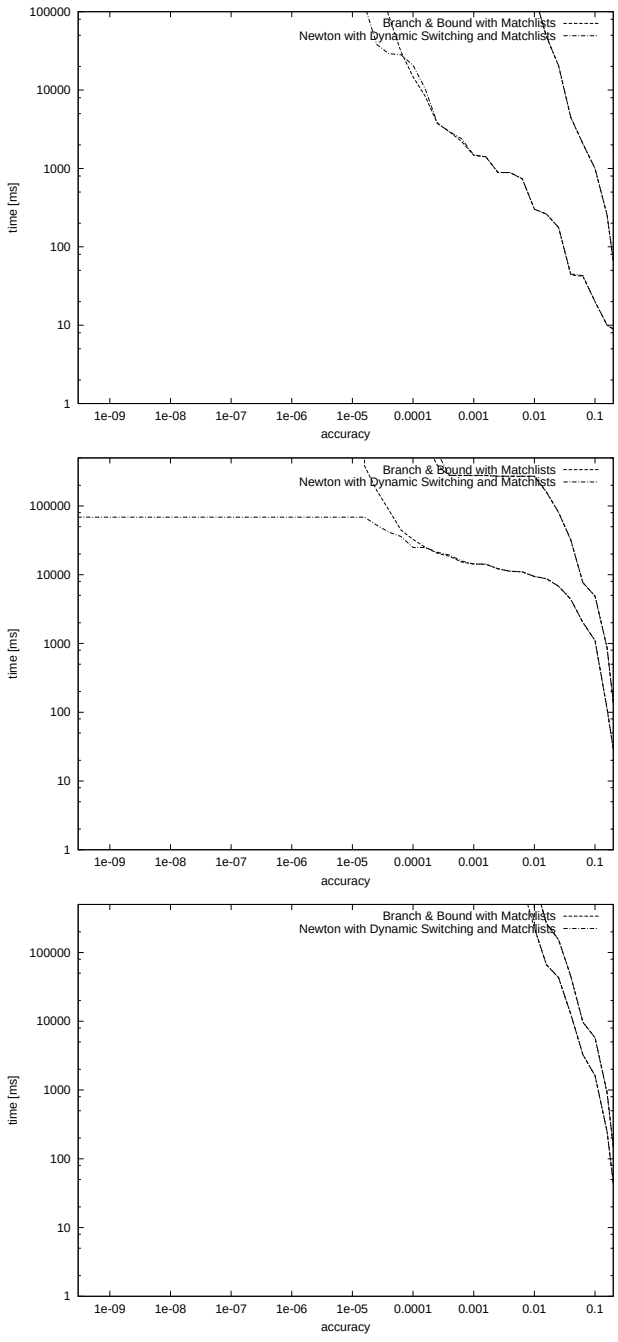

Figure 4.23: Maximum and minimum of the times over the 100 datasets for a given accuracy for the rotated ellipse finding problem. The first corresponds to the plain rotated ellipse datasets. The second to the rotated ellipse datasets with additional points and the third to the random datasets.

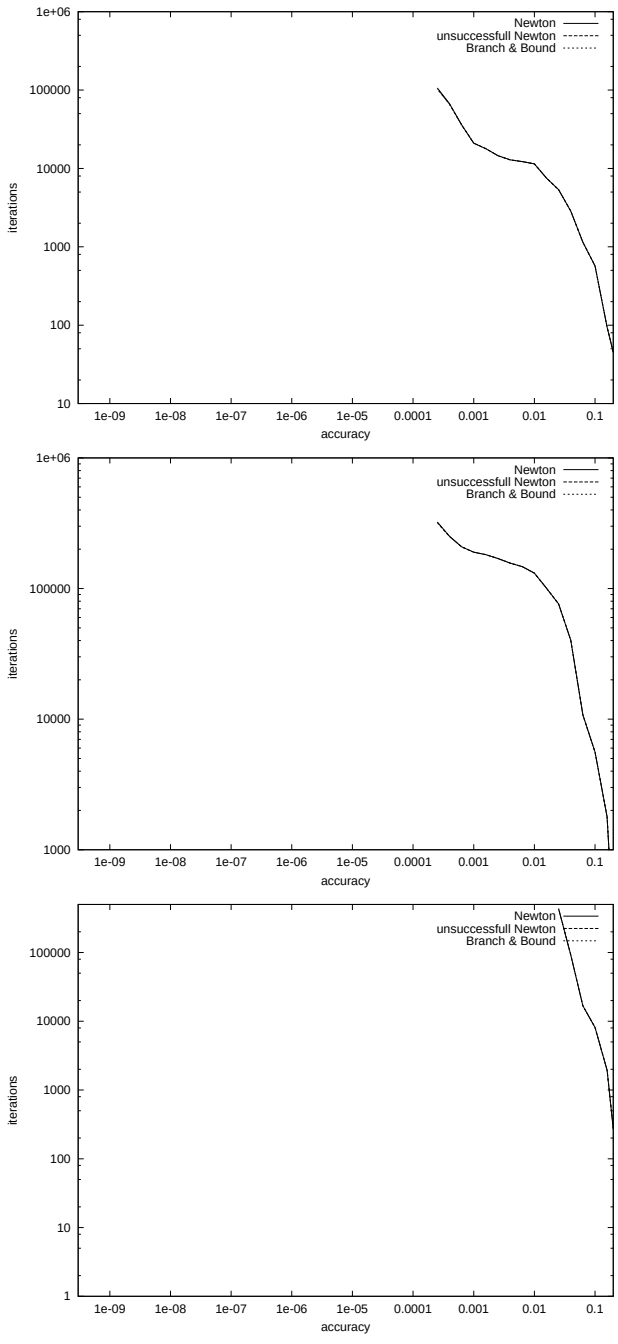

Figure 4.24: The effectiveness of dynamic switching. The plots represent the Newton method with dynamic switching. The first corresponds to the plain rotated ellipse datasets. The second to the rotated ellipse datasets with additional points and the third corresponds to the random datasets

4.4 Analysis of the Results

The Newton method accelerates the problems of robust line finding, robust circle finding and robust ellipse finding problem on point based datasets compared to the branch and bound method. For the robust rotated ellipse finding problem, no acceleration can be duplicated.

Matchlists were experimentally shown to accelerate the methods for most cases, except for comparatively simple noise free datasets.

The purpose of these simulations was to analyze the effects of the different implementation strategies and to support experimentally the notion that dynamic switching and interval Newton methods can speed up matching. In the next chapter, we will request more information as input: the orientation associated with the point features. Such orientation information can be obtained, for example, from image gradients.

Chapter 5

Using Feature Orientations

The methods discussed before only considered one constraint for finding a match: the position of the points in the datasets. But other constraints are imaginable. If additional knowledge is available, this can be integrated in the previous methods. Later on we will use the rotation of the points as a second constraint. But for now, let us discuss multiple constraint matching in general.

5.1 Statistical Examination of Multiple Constraints

Suppose, we have k constraints we want to get fulfilled with equal weighting. Assume a Gaussian error model for all constraints. For every constraint i let the probability, of having a feature x_i while the correct value is X_i be:

$$P_i(x_i) = a_i e^{-\frac{(x_i - X_i)^2}{b_i}} \tag{5.1}$$

with constants $a_i = \frac{1}{\sqrt{2\pi\sigma_i^2}}$ and $b_i = \frac{\sigma_i^2}{2}$ where σ_i^2 is the variance.

Assuming an equal and independent distribution of the noise for the

datasets of each constraint, the probability of having a feature somewhere is

$$N_i(x_i) = N_{i,0} \tag{5.2}$$

Assuming independent Gaussian distributions, the probability of a feature with values $x = (x_i)_{i=0}^{k-1}$ is

$$P(x) = \prod_{i=0}^{k-1} P_i(x_i) = \prod_{i=0}^{k-1} a_i e^{-\frac{(x_i - X_i)^2}{b_i}} \tag{5.3}$$

$$N(x) = \prod_{i=0}^{k-1} N_i(x_i) = N_0 \tag{5.4}$$

Switching to log-likelihood functions, this leads to

$$P_l(x) = m + \sum_{i=0}^{k-1} \frac{(x_i - X_i)^2}{b_i} \tag{5.5}$$

where $m = \log(\prod_{i=0}^{k-1} a_i)$, but constant offsets can be omitted and

$$N_l(x) = \log N_0 \tag{5.6}$$

Building a classifier from these two distributions, we assign a value x to be a part of the object to be found if $P_l(x) \geq N_l(x)$ and we assign it to the background noise if $P_l(x) < N_l(x)$. The classifier does not change, if we scale and shift it along the y-axis. We get a likelihood function telling us if a point in the dataset is a point of the solution or if it is produced from noise:

$$\Phi(x) = \max\left(0, 1 - \frac{1}{k} \sum_{i=0}^{k-1} \frac{(x_i - X_i)^2}{b_i}\right) \tag{5.7}$$

This looks like the Φ defined before but for multiple dimensions.

5.2 The Multiple Constraint Problem

Setting X to the origin in (5.7), we get the new optimization problem:

$$\arg_x \max_{x \in X} Q(x) \tag{5.8}$$

$$Q(x) = \sum_{m \in M} q_m(x) \tag{5.9}$$

$$q_m(x) = \Phi\left(d_m(x)\right) \tag{5.10}$$

$$\Phi(x) = \max\left(0, 1 - \frac{1}{k}\sum_{i=0}^{k-1}\frac{x_i^2}{\epsilon_i^2}\right) \tag{5.11}$$

In comparison to the previous problem definition d_m is now a function mapping from $\mathrm{R}^n \rightarrow \mathrm{R}^k$, where n is number of dimensions in the parameter space as before while k is the number of constraints. The question arises, whether the assumption of independent Gaussian distributions is suitable. Alternatively, one may want to multiply the single constraints instead of adding them:

$$\Phi_{alt}(x) = \prod_{i=0}^{k-1}\left(\max\left(0, 1 - \frac{x_i^2}{\epsilon_i^2}\right)\right) \tag{5.12}$$

At first sight, this looks like a better approach because $\Phi_{alt}(x)$ gets zero if at least one of the constraints is not fulfilled within the corresponding ϵ_i.

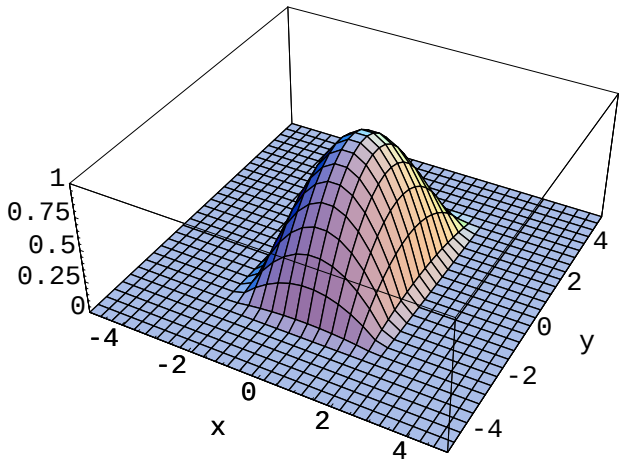

Figure 5.1: Plot of Φ_{alt}

Figure 5.1 shows a plot of Φ_{alt} in two dimensions with $\epsilon_0 = 2$ and $\epsilon_1 = 3$. It can be clearly seen that there is a rectangular area in parameter space $[-\epsilon_0, \epsilon_0] \times [-\epsilon_1, \epsilon_1]$ where Φ is larger than zero. Everywhere else, it is zero. Thus, it is sufficient for a point to be ignored, if one constraint is not fulfilled. Unexpected solutions only fulfilling one constraint perfectly will not be the result.

In contrast to this, a sum does not have this property. Assuming that only one constraint i is not fulfilled in (5.11) within the corresponding ϵ_i, q_m can still be larger than zero.

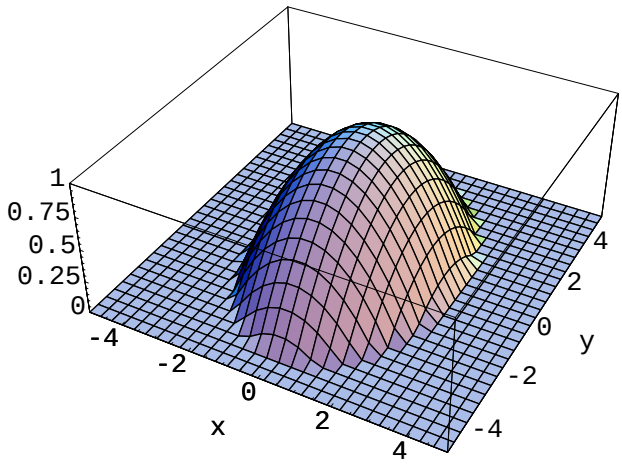

Figure 5.2: Plot of Φ

But looking at 5.2, we see that at some distance from the origin, everything gets zero like for Φ_{alt}. Thus it also cannot happen that a solution is returned where only one constraint is perfectly fulfilled and another is not fulfilled. Instead of a rectangular area where the function is larger than zero, there is an elliptic area including the rectangular area of Φ_{alt}.

The Φ is still local. The behavior is identical to Φ_{alt} for our purpose. The ϵ_i have to be chosen a bit differently for Φ and Φ_{alt}. One advantage of Φ over Φ_{alt} is that there is a statistical explanation, even though the assumption of independent Gaussian distributions may not be correct for every situation. But the more important advantage of Φ is that the derivative of a sum is again a sum, while the derivative of a product leads to applying the chain

rule. Especially the second derivative of Q would produce a lot more terms with Φ_{alt} than with Φ, resulting in longer calculation times. Therefore, Φ is the better choice in combination with the Newton method.

5.3 Derivatives of the Problem

For multiple constraints, we get another dimension in the formulas. The calculations involve tensors of degree three. Hence, we cannot keep on writing the formulas as matrices. Let us define the different spaces and their bases that get involved.

$$\begin{aligned} \mathrm{R}^k &= \mathrm{span}\{e_i : 0 \leq i < k\} \\ \mathrm{R}^n &= \mathrm{span}\{f_j : 0 \leq j < n\} \end{aligned} \tag{5.13}$$

The important functions are

$$\begin{aligned} \Phi &: \mathrm{R}^k \to \mathrm{R} \\ d_m &: \mathrm{R}^n \to \mathrm{R}^k \end{aligned} \tag{5.14}$$

They and their derivatives are

$$\begin{aligned} \Phi(x) &= \Phi(x) \cdot 1 \\ \mathrm{D}\Phi(x) &= \sum_{i=0}^{k-1} \mathrm{D}\Phi_i(x)\ e_i \\ \mathrm{D}^2\Phi(x) &= \sum_{i=0}^{k-1} \sum_{\tilde{i}=0}^{k-1} \mathrm{D}^2\Phi_{i,\tilde{i}}(x)\ e_i \otimes e_{\tilde{i}} \end{aligned} \tag{5.15}$$

$$
\begin{aligned}
d_m(x) &= \sum_{i=0}^{k-1} d_{m,i}(x)\ e_i \\
\mathrm{D}d_m(x) &= \sum_{i=0}^{k-1}\sum_{j=0}^{n-1} \mathrm{D}d_{m,i,j}(x)\ e_i \otimes f_j \\
\mathrm{D}^2 d_m(x) &= \sum_{i=0}^{k-1}\sum_{j=0}^{n-1}\sum_{\tilde{j}=0}^{n-1} \mathrm{D}^2 d_{m,i,j,\tilde{j}}(x)\ e_i \otimes f_j \otimes f_{\tilde{j}}
\end{aligned}
\tag{5.16}
$$

Calculating the first derivative of q_m, we get

$$
\begin{aligned}
q_m(x) &= \Phi(d_m(x)) \\
\mathrm{D}q_m(x) &= \mathrm{D}\Phi(d_m(x))\mathrm{D}d_m(x) \\
&= \left(\sum_{i=0}^{k-1} \mathrm{D}\Phi_i(x)\ e_i\right) \cdot \left(\sum_{i=0}^{k-1}\sum_{j=0}^{n-1} \mathrm{D}d_{m,i,j}(x)\ e_i \otimes f_j\right) \\
&= \sum_{j=0}^{n-1} \left(\sum_{i=0}^{k-1} \mathrm{D}\Phi_i(x)\mathrm{D}d_{m,i,j}(x)\right)\ f_j
\end{aligned}
\tag{5.17}
$$

we handle the second derivative as two terms

$$
\begin{aligned}
\mathrm{D}^2 q_m(x) &= T_1 + T_2 \\
T_1 &= \left(\mathrm{D}^2\Phi(d_m(x)) \cdot \mathrm{D}d_m(x)\right) \cdot \mathrm{D}d_m(x) \\
T_2 &= \mathrm{D}\Phi(d_m(x)) \cdot \mathrm{D}^2 d_m(x)
\end{aligned}
\tag{5.18}
$$

Thus, T_1 equals to

$$\begin{aligned}
T_1 &= \left(\left(\sum_{i=0}^{k-1} \sum_{\tilde{i}=0}^{k-1} \mathrm{D}^2\Phi_{i,\tilde{i}}(d_m(x))\ e_i \otimes e_{\tilde{i}} \right) \right. \\
&\quad \left. \cdot \sum_{\tilde{i}=0}^{k-1} \sum_{j=0}^{n-1} \mathrm{D}d_{m,\tilde{i},j}(x)\ e_{\tilde{i}} \otimes f_j \right) \\
&\quad \cdot \sum_{i=0}^{k-1} \sum_{\tilde{j}=0}^{n-1} \mathrm{D}d_{m,i,\tilde{j}}(x)\ e_i \otimes f_{\tilde{j}} \\
&= \left(\sum_{i=0}^{k-1} \sum_{j=0}^{n-1} \left(\sum_{\tilde{i}=0}^{k-1} \mathrm{D}^2\Phi_{i,\tilde{i}}(d_m(x)) \mathrm{D}d_{m,\tilde{i},j}(x) \right)\ e_i \otimes f_j \right) \\
&\quad \cdot \sum_{i=0}^{k-1} \sum_{\tilde{j}=0}^{n-1} \mathrm{D}d_{m,i,\tilde{j}}(x)\ e_i \otimes f_{\tilde{j}} \\
&= \sum_{j=0}^{n-1} \sum_{\tilde{j}=0}^{n-1} \left(\sum_{\tilde{i}=0}^{k-1} \sum_{i=0}^{k-1} \mathrm{D}^2\Phi_{i,\tilde{i}}(d_m(x)) \mathrm{D}d_{m,\tilde{i},j} \mathrm{D}d_{m,i,\tilde{j}} \right)\ f_j \otimes f_{\tilde{j}} \quad (5.19)
\end{aligned}$$

and T_2 gets

$$\begin{aligned}
T_2 &= \left(\sum_{i=0}^{k-1} \mathrm{D}\Phi_i(d_m(x))\ e_i \right) \cdot \left(\sum_{i=0}^{k-1} \sum_{j=0}^{n-1} \sum_{\tilde{j}=0}^{n-1} \mathrm{D}^2 d_{m,i,j,\tilde{j}}(x)\ e_i \otimes f_j \otimes f_{\tilde{j}} \right) \\
&= \sum_{j=0}^{n-1} \sum_{\tilde{j}=0}^{n-1} \left(\sum_{i=0}^{k-1} \mathrm{D}\Phi_i(d_m(x)) \mathrm{D}^2 d_{m,i,j,\tilde{j}}(x) \right)\ f_j \otimes f_{\tilde{j}} \quad (5.20)
\end{aligned}$$

Switching to our specific Φ (5.11) simplifies these equations a little bit. Calculating the second derivative of the Φ, we see that it only maps into

$\text{span}\{e_i \otimes e_i : 0 \leq i < k\} \subset \mathrm{R}^{k \times k}$:

$$\begin{aligned}
\Phi(x) &= \max\left(0, 1 - \frac{1}{k}\sum_{i=0}^{k-1}\frac{x_i^2}{\epsilon_i^2}\right) \\
\mathrm{D}\Phi(x) &= \sum_{i=0}^{k-1}\mathrm{D}\Phi_i(d_m(x))e_i \\
&= \begin{cases} \sum_{i=0}^{k-1} -\frac{2x_i}{k\epsilon_i^2}e_i, & if\ \frac{1}{k}\sum_{i=0}^{k-1}\frac{x_i^2}{\epsilon_i^2} \leq 1 \\ 0, & otherwise \end{cases} \\
\mathrm{D}^2\Phi(x) &= \sum_{i=0}^{k-1}\sum_{j=0}^{n-1}\sum_{\tilde{j}=0}^{n-1}\mathrm{D}^2 d_{m,i,j,\tilde{j}}(x)\ e_i \otimes f_j \otimes f_{\tilde{j}} \\
&= \begin{cases} \sum_{i=0}^{k-1} -\frac{2}{k\epsilon_i^2}\ e_i \otimes e_i, & if\ \frac{1}{k}\sum_{i=0}^{k-1}\frac{x_i^2}{\epsilon_i^2} < 1 \\ \sum_{i=0}^{k-1} \infty\ e_i \otimes e_i, & if\ \frac{1}{k}\sum_{i=0}^{k-1}\frac{x_i^2}{\epsilon_i^2} = 1 \\ 0, & otherwise \end{cases}
\end{aligned} \tag{5.21}$$

Thus T_1 can be further simplified to

$$T_1 = \sum_{j=0}^{n-1}\sum_{\tilde{j}=0}^{n-1}\left(\sum_{i=0}^{k-1}\mathrm{D}^2\Phi_{i,i}(d_m(x))\mathrm{D}d_{m,i,j}(x)\mathrm{D}d_{m,i,\tilde{j}}(x)\right)\ f_j \otimes f_{\tilde{j}}. \tag{5.22}$$

5.4 Dynamic Switching and Matchlists

Dynamic switching can be adapted to work with the method described above. Instead of just checking if the interval passed to the one-dimensional Φ, we have to check if the interval passed to the multi-dimensional Φ overlaps the ground-ellipse, as seen in figure 5.2.

For our problems $q_m(x)$ is of the form $q_m(x) = \Phi(d_m(x))$. Hence we have to check if

$$S(x) = \left(\forall\ m \in M : 1 \notin \frac{1}{k}\sum_{i=0}^{k-1}\frac{(d_m(x))_i{}^2}{\epsilon_i^2}\right) \tag{5.23}$$

Again, this is cheap to evaluate because the sum has to be evaluated anyway for the derivatives of Φ.

There is a possible optimization. If we restrict Φ to be larger than zero

only on the rectangular interval $\prod_{i=0}^{k-1}[-\epsilon_i, \epsilon_i]$, Φ becomes

$$\Phi(x) = \begin{cases} 1 - \frac{1}{k}\sum_{i=0}^{k-1} \frac{x_i^2}{\epsilon_i^2}, & \text{if } \forall\ 0 \leq i < k-1 : |x_i| < \epsilon_i \\ 0, & \text{otherwise} \end{cases} \tag{5.24}$$

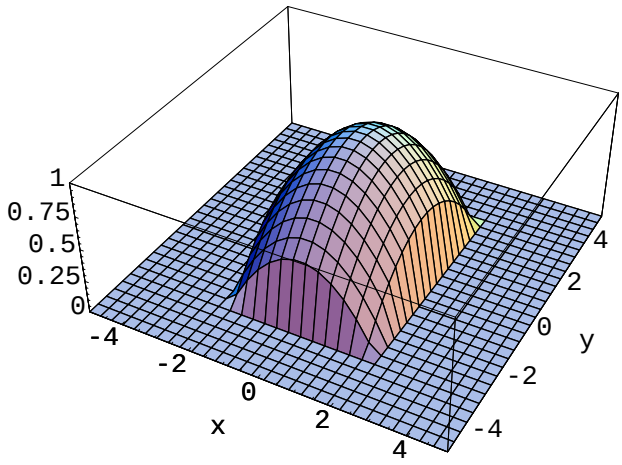

Figure 5.3: Plot of the optimized objective function Φ

Figure 5.3 shows a plot of this Φ. This function is still a likelihood function as the other Φ before. Besides, the rectangular area is exactly the same rectangular area as it is in Φ_{alt}.

Φ can be rewritten by omitting $\frac{1}{k}$ as

$$\begin{aligned} \Phi(x) &= \begin{cases} \sum_{i=0}^{k-1} 1 - \frac{x^2}{\epsilon_i^2}, & \text{if } \forall\ 0 \leq i < k-1 : |x_i| < \epsilon_i \\ 0, & \text{otherwise} \end{cases} \\ &= \begin{cases} \sum_{i=0}^{k-1} \Phi_{\epsilon_i}(x), & \text{if } \forall\ 0 \leq i < k-1 : |x_i| < \epsilon_i \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{5.25}$$

where Φ_{ϵ_i} is the one-dimensional Φ with $\epsilon = \epsilon_i$.

It is not sufficient to just set $\Phi(x) = \sum_{i=0}^{k-1} \Phi_{\epsilon_i}(x)$. Using this function may return undesired results. This can be seen in figure 5.4. The area where the function is larger than zero is infinite. Points do not have to fulfill every constraint to contribute to the sum.

The advantage in comparison to (5.11) is that not all constraints have to

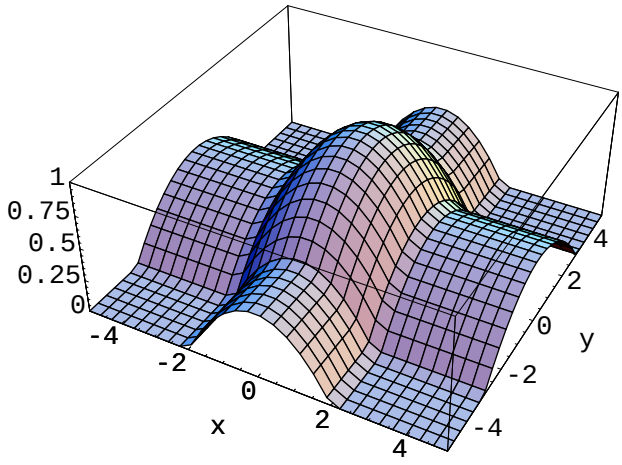

Figure 5.4: Plot of a poorly chosen objective function Φ

be evaluated every time. If $|d_{m,i}(x)| < \epsilon_i$ is true for one constraint, the other constraints do not have to be evaluated anymore because the contribution to the sum for that m is already known to be zero. This is even more important if the constraints are differently expensive to evaluate. Thus, sorting the constraints to first evaluate the cheaper ones accelerates the method.

The added discontinuity does not matter, because the dynamic switching function can be changed to check for our interval d_m not to overlap the rectangular boundary.

$$\begin{aligned} S(x) \quad = \quad & (\forall m \in M : (\forall\ 0 \leq i < k : \epsilon_i \notin |d_{m,i}(x)|) \\ & \vee (\exists\ 0 \leq i < k : |d_{m,i}(x)|.lo > \epsilon_i)) \end{aligned} \tag{5.26}$$

Analogously, the method can be easily combined with matchlists. A feature m has to be removed from the matchlist if

$$\exists 0 \leq i < k : |d_{m,i}(x)|.lo > \epsilon_i. \tag{5.27}$$

5.5 Periodic Distances

Until now, the distance $d_{m,i}$ of a constraint i was restricted to be non-periodic. Later on, we want to add a constraint on the rotation of the normal of the point. The $d_{m,i}$ becomes a periodic distance over $[-\pi, \pi]$. For now, periodic distances will be discussed in general.

Let the periodic distance be $d_{m,i}$ on real values map to $[-p, p]$. Extending $d_{m,i}$ to interval arithmetic, the destination interval has to be extended to the double length to distinguish between the two possible meant areas that can be specified by the two boundaries of the interval. The lower bound of the interval has to stay in $[-p, p]$. This is necessary because the interval arithmetic previously defined requires for every interval $[a, b]$, $a < b$. The additional information needed could alternatively be memorized by exchanging the two bounds, but then, the interval arithmetic needs to be changed. Therefore the upper option is chosen.

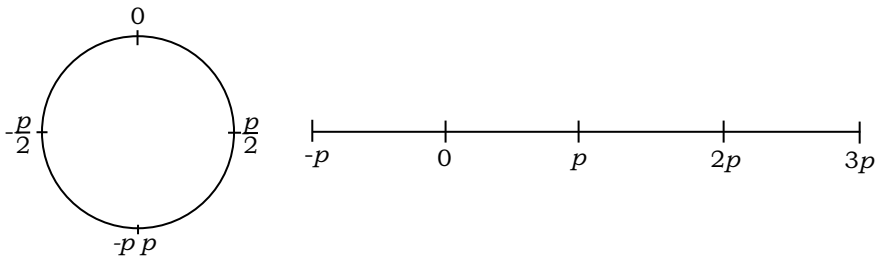

Figure 5.5: Periodic distance mapped onto an interval.

Let the interval extended $d_{m,i}$ map to $[-p, 3p]$. For example in Figure 5.5, the upper area between $-\frac{p}{2}$ and $\frac{p}{2}$ is described by $\left[-\frac{p}{2}, \frac{p}{2}\right]$ while the lower area is described by $\left[\frac{p}{2}, \frac{3p}{2}\right]$. If the $d_{m,i}$ for some reason maps to another interval, it can be shifted to the interval above by the addition of subtracting multiples of $2p$, because it is periodic.

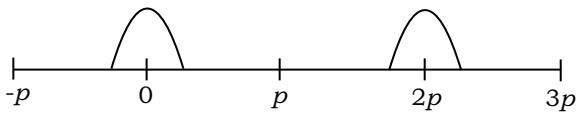

Figure 5.6: Φ on the interval twice as long as the period.

On $[-p, 3p]$, the constraint is fulfilled optimally if $d_{m,i}(x) \in \{0, 2p\}$.

Therefore, $\Phi_{periodic}$ gets defined as:

$$\Phi_{periodic}(d_{m,i}(x)) = \Phi(d_{m,i}(x)) \cup \Phi((d_{m,i}(x) - 2p) \cap [-p, p]) \tag{5.28}$$

where $\Phi(\emptyset) = \emptyset$ and $y \cup \emptyset = y$. $\emptyset$ is the empty interval. If $d_{m,i}$ lies clearly within $[-p, p]$, $\Phi_{periodic}$ should not build the union of $\Phi(d_{m,i}(x))$ and $\Phi(d_{m,i}(x) - 2p)$ because $\Phi(d_{m,i}(x) - 2p)$ becomes zero. Therefore cutting, with [-p,p] like it is done above handles this situation. It cannot happen the other way around because we required the lower bound to lie within $[-p, p]$. However, this only affects the lower bound we get for the current interval. If we do not need a prove for perfect optimality, this cut can be left out to further optimize this equation.

Analogously, for the derivatives we get

$$\begin{aligned} \mathrm{D}\Phi_{periodic}(y) &= \mathrm{D}\Phi(y) \cup \mathrm{D}\Phi((y - 2p) \cap [-p, p]) \\ \mathrm{D}^2\Phi_{periodic}(y) &= \mathrm{D}^2\Phi(y) \cup \mathrm{D}^2\Phi((y - 2p) \cap [-p, p]) \end{aligned} \tag{5.29}$$

where $y = d_{m,i}(x)$.

5.6 Orientation as a second Constraint

As mentioned earlier, using rotation as another constraint additionally to the position will be tried. The distance between the optimal rotation and the rotation prescribed by the current interval is 2π periodic. $p = \pi$ with the same notation as in the previous chapter.

The following sections describe the different distance functions for rotation for the four problems discussed earlier. Let zero be the constraint on the position and one the rotational constraint. The vector m describing only the position of a point m before it is extended by another component:

$$m = \begin{pmatrix} x \\ y \\ w \end{pmatrix} \tag{5.30}$$

m_x and m_y describe the position of the point as before and m_w describes the angle between the normal of the point and the x-axis.

5.6.1 The Line Finding Problem

$d_{m,1}$ is the easiest for the line finding problem. The rotation of the line only has to be compared directly with the angle of the normal of point m.

$$d_{m,1}(w,t) = w - m_w \tag{5.31}$$

The derivatives are

$$\nabla d_{m,1}(w,t) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{5.32}$$

$$\mathrm{D}\nabla d_{m,1}(w,t) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} = 0 \tag{5.33}$$

Note that $\mathrm{D}\nabla d_{m,1}(w,t) = 0$ does not mean $\mathrm{D}q_{m,1}(w,t) = 0$. The function $d_{m,1}$ is obviously twice continuous differentiable everywhere.

5.6.2 The Circle Finding Problem

For the circle finding problem, the rotation of the point m has to be compared with the normal of the circle defined by x, y and r at the position m_x, m_y, scaled onto the circle.

$$d_{m,1}(x,y,r) = \mathrm{atan2}(m_y - y, m_x - x) - m_w \tag{5.34}$$

$$\nabla d_{m,1}(x,y,r) = \frac{1}{(m_x - x)^2 + (m_y - y)^2} \begin{pmatrix} m_y - y \\ -(m_x - x) \\ 0 \end{pmatrix} \tag{5.35}$$

$$\mathrm{D}\nabla d_{m,1}(x,y,r) = \frac{1}{\left((m_x-x)^2+(m_y-y)^2\right)^2} \cdot \begin{pmatrix} 2(m_x-x)(m_y-y) & (m_y-y)^2-(m_x-x)^2 & 0 \\ \cdot & -2(m_x-x)(m_y-y) & 0 \\ \cdot & \cdot & 0 \end{pmatrix} \quad (5.36)$$

As already shown in the case where only the position was regarded to find a circle, the distance between a point contributing to the sum of Q and the center of the circle $\sqrt{((m_x-x)^2+(m_y-y)^2)}$ is larger than zero and therefore, $m_x \neq x$ or $m_y \neq y$. Thus, $d_{m,1}$ is twice continuous differentiable.

5.6.3 The Ellipse Finding Problem

An axis aligned ellipse that has its center at (x, y) and lengths of the axes a and b can be represented by the points m of the ISO line with an ISO value of one of

$$F(m_x, m_y) = \frac{(x-m_x)^2}{a^2} + \frac{(y-m_y)^2}{b^2} = 1 \quad (5.37)$$

The gradient of F is perpendicular to the ellipse. Thus, the normal of the ellipse at a point m is

$$\nabla F(m_x, m_y) = 2 \cdot \begin{pmatrix} \frac{m_x}{a^2} \\ \frac{m_y}{b^2} \end{pmatrix} \quad (5.38)$$

We see that for getting the angle of the normal at m, the distances have to be scaled with the square of the lengths of the ellipse axes before atan2 gets applied:

$$d_{m,1}(x,y,a,b) = \mathrm{atan2}\left(\frac{m_y-y}{b^2}, \frac{m_x-x}{a^2}\right) - m_w \quad (5.39)$$

Thus, the derivatives are:

$$\begin{aligned}\nabla d_{m,1}(x,y,a,b) &= \frac{1}{(m_x-x)^2b^4+(m_y-y)^2a^4} \\ &\cdot \begin{pmatrix} -(m_y-y)a^2b^2 \\ (m_x-x)a^2b^2 \\ 2(m_x-x)(m_y-y)ab^2 \\ -2(m_x-x)(m_y-y)a^2b \end{pmatrix}\end{aligned} \tag{5.40}$$

$$\begin{aligned}\mathrm{D}\nabla d_{m,1}(x,y,a,b) &= \frac{1}{\left((m_x-x)^2b^4+(m_y-y)^2a^4\right)^2} \\ &\cdot \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{23} & d_{33} & d_{34} \\ d_{41} & d_{24} & d_{34} & d_{44} \end{pmatrix}\end{aligned} \tag{5.41}$$

$$\begin{aligned}
d_{11} &= 2(m_x-x)(m_y-y)a^2b^6 \\
d_{12} &= (m_y-y)^2a^6b^2-(m_x-x)^2a^2b^6 \\
d_{13} &= 2(m_y-y)^3a^5b^2-2(m_x-x)^2(m_y-y)^2ab^6 \\
d_{14} &= -2(m_y-y)^3a^6b+2(m_x-x)^2(m_y-y)a^2b^5 \\
\\
d_{22} &= -2(m_x-x)(m_y-y)a^6b^2 \\
d_{23} &= -2(m_x-x)(m_y-y)^2a^5b^2+2(m_x-x)^3ab^6 \\
d_{24} &= 2(m_x-x)(m_y-y)^2a^6b-2(m_x-x)^3a^2b^5 \\
\\
d_{33} &= -6(m_x-x)(m_y-y)^3a^4b^2+2(m_x-x)^3(m_y-y)b^6 \\
d_{34} &= 4(m_x-x)(m_y-y)^3a^5b-4(m_x-x)^3(m_y-y)ab^5 \\
\\
d_{44} &= -2(m_x-x)(m_y-y)^3a^6b+6(m_x-x)^3(m_y-y)a^2b^4
\end{aligned} \tag{5.42}$$

As already shown in the case where only the position was regarded to find an ellipse, for every point m contributing to the sum of Q, $m_x \neq x$ or $m_y \neq y$. Thus, $d_{m,1}$ is twice continuous differentiable.

5.6.4 Finding Ellipses at Arbitrary Orientations

For the rotated ellipse, every point m has to be rotated according to the angle w. Additionally, the calculated perfect angle has to be compared to the difference between the angle m_w of the normal of m and the angle of the ellipse w. The rest of the equation is equal to the axes aligned ellipse:

$$d_{m,1}(x, y, a, b, w) = \operatorname{atan2}\left(\frac{v_y - y}{b^2}, \frac{v_x - x}{a^2}\right) - (m_w - w) \tag{5.43}$$

v_x and v_y are defined as:

$$\begin{aligned} v_x &= \cos(w)m_x + \sin(w)m_y \\ v_y &= -\sin(w)m_x + \cos(w)m_y \end{aligned} \tag{5.44}$$

The first derivative is

$$\nabla d_{m,1}(x, y, a, b, w) = \frac{1}{\frac{v_x^2 b^2}{a^2} + \frac{v_y^2 a^2}{b^2}} \cdot \begin{pmatrix} v_y \\ -v_x \\ \frac{2v_x v_y}{a} \\ -\frac{2v_x v_y}{b} \\ \rho \end{pmatrix} \tag{5.45}$$

$$\rho = -m_x^2 - m_y^2 + (m_x x + m_y y)\cos(w) - (m_x y - m_y x)\sin(w) \tag{5.46}$$

The second derivative is

$$\mathrm{D}\nabla d_{m,1}(x, y, a, b, w) = \frac{1}{\gamma^2} \cdot \begin{pmatrix} d_{11} & d_{12} & d_{13} & d_{14} & d_{15} \\ d_{12} & d_{22} & d_{23} & d_{24} & d_{25} \\ d_{13} & d_{23} & d_{33} & d_{34} & d_{35} \\ d_{14} & d_{24} & d_{34} & d_{44} & d_{45} \\ d_{15} & d_{25} & d_{35} & d_{45} & d_{55} \end{pmatrix} \tag{5.47}$$

$$
\begin{aligned}
d_{11} &= \frac{2v_x v_y}{a^6 b^2} \\
d_{12} &= \frac{2v_y^2 - b^4\gamma}{a^2 b^6} \\
d_{13} &= \frac{2v_y(2v_x^2 - a^4\gamma)}{a^7 b^2} \\
d_{14} &= \frac{2v_y(2v_y^2 - b^4\gamma)}{a^2 b^7} \\
d_{15} &= \frac{-v_y\beta + (-m_x\cos(w) - m_y\sin(w))\gamma}{a^2 b^2}
\end{aligned}
$$

$$
\begin{aligned}
d_{22} &= -\frac{2v_x v_y}{a^2 b^6} \\
d_{23} &= \frac{2v_x(-2v_x^2 + a^4\gamma)}{a^7 b^2} \\
d_{24} &= \frac{2v_x(-2v_y^2 + b^4\gamma)}{a^2 b^7} \\
d_{25} &= \frac{v_x\beta - (m_y\cos(w) - m_x\sin(w))\gamma}{a^2 b^2}
\end{aligned}
$$

$$
\begin{aligned}
d_{33} &= \frac{2v_x v_y(4v_x^2 - 3a^4\gamma)}{a^8 b^2} \\
d_{34} &= \frac{4v_x v_y(2v_y^2 - b^4\gamma)}{a^3 b^7} \\
d_{35} &= -(2(b^2 x + a^2 y - (m_y a^2 + m_x b^2)\cos(w) + (m_x a^2 - m_y b^2)\sin(w)) \\
&\quad (b^2 x - a^2 y + (m_y a^2 - m_x b^2)\cos(w) - (m_x a^2 + m_y b^2)\sin(w)) \\
&\quad (m_x^2 + m_y^2 - (m_x x + m_y y)\cos(w) + (-m_y x + m_x y)\sin(w))) \cdot \frac{1}{a^7 b^6}
\end{aligned}
$$

$$
\begin{aligned}
d_{44} &= \frac{2v_x v_y(-4v_y^2 - 3b^4\gamma)}{a^2 b^8} \\
d_{45} &= (2(m_x^2 + m_y^2 - (m_x x + m_y y)\cos(w) + (-m_y x + m_x y)\sin(w)) \\
&\quad (b^4 x^2 - a^4 y^2 + (-2m_x b^4 x + 2m_y a^4 y)\cos(w) \\
&\quad +(-m_y^2 a^4 + a^2 b^4)\cos(w)^2 - 2(m_y b^4 x + m_x a^4 y)\sin(w) \\
&\quad +(-m_x^2 a^4 + m_y^2 b^4)\sin(w)^2 + m_x m_y(a^4 + b^4)\sin(2w)) \cdot \frac{1}{a^6 b^7}
\end{aligned}
$$

$$
\begin{aligned}
d_{55} &= (-a^4b^4((-m_yx + m_xy)\cos(w) + (m_xx + m_yy)\sin(w)\gamma \\
&\quad +2(m_x^2 + m_y^2 - (m_xx + m_yy)\cos(w) + (-m_yx + m_xy)\sin(w) \\
&\quad (m_xm_y(-a^4 + b^4)\cos(2w) + (m_xa^4x + m_ya^4y)\sin(w) \\
&\quad +cos(w)(-m_yb^4x + m_xa^4y + (m_x - m_y)(m_x + m_y)(a^4 - b^4)\sin(w)))) \\
&\quad \cdot\frac{1}{a^6b^6}
\end{aligned} \tag{5.48}
$$

where

$$
\begin{aligned}
\beta &= \frac{2(-v_y)(m_x\cos(w) + m_y\sin(w))}{b^4} + \frac{2v_x(m_y\cos(w) - m_x\sin(w))}{a^4} \\
\gamma &= \frac{v_x^2}{a^4} + \frac{v_y^2}{b^4}
\end{aligned} \tag{5.49}
$$

As already shown in the case where only the position was regarded to find a rotated ellipse, for every point m contributing to the sum of Q, $v_x \neq x$ or $v_y \neq y$. Thus, $d_{m,1}$ is twice continuous differentiable.

Chapter 6

Evaluation with the Orientation Constraints

This chapter provides the results of the performance tests between branch and bound with matchlists and the Interval Newton method with dynamic switching and matchlists on datasets with points having normals. Additionally, the results get compared to the Interval Newton method not using the normals.

6.1 The Datasets

We will proceed with the experiments as before without the rotation to enable comparisons between the results. The algorithms are tested on different geometric matching problems. For each problem equivalence classes are defined and representative datasets are selected. Each dataset consists of a set of points with normals from which the optimal geometry of the particular problem has to be found. The considered problems are as before:

- line finding
- circle finding
- ellipse finding

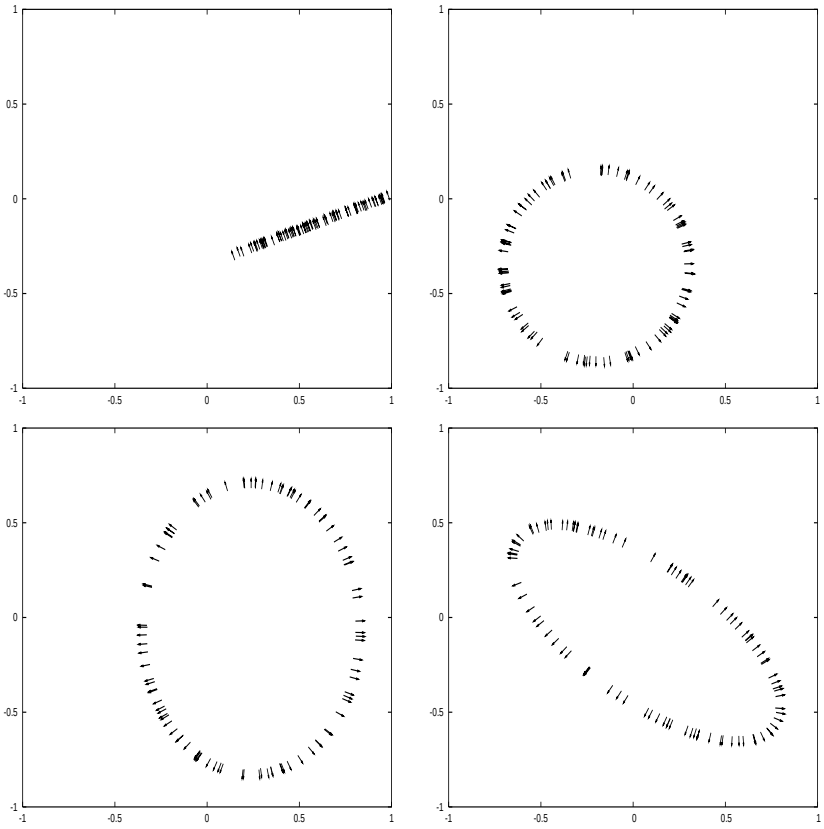

Figure 6.1: Examples of the datasets used in the experiments.

- finding ellipses at arbitrary orientations

See Figure 6.1 for examples of these problems. The domain of the points is $[-1, 1]^2$. The angle of the normals is $[-\pi, \pi]$. The points get an additional error on the position and the angle of the normal of 0.01. The equivalence classes for each problem are:

- Simple data only consisting of points describing the primitive.
- Simple data and additional randomly distributed points.
- Only randomly distributed points.

See Figure 6.2 for examples. The number of points in the three classes is always 100. For the first class, all points belong to the primitive that has to be found. For the second class, 50 points belong to the primitive and 50 are additional random points. The third class only consists of random points.

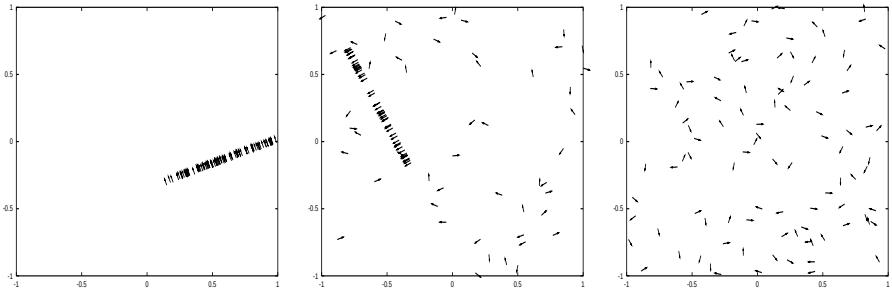

Figure 6.2: Examples of the classes of test cases for line finding. From the left to the right, the pictures show class one to class three.

6.2 Evaluation

The following combination of algorithms and datasets were evaluated:

- The Interval Newton method with dynamic switching and matchlists ignoring the normal information.
- The branch and bound method with matchlists using normal information.
- The Interval Newton method with dynamic switching and matchlists using normal information.

6.3 The Empirical Results

In this section the results of the tests are discussed. All the described tests were evaluated this time on an Athlon XP 2000+ with 1 GB of RAM. The gcc version 3.3.3 with optimization for speed -O2 was used.

6.3.1 The Line Finding Problem

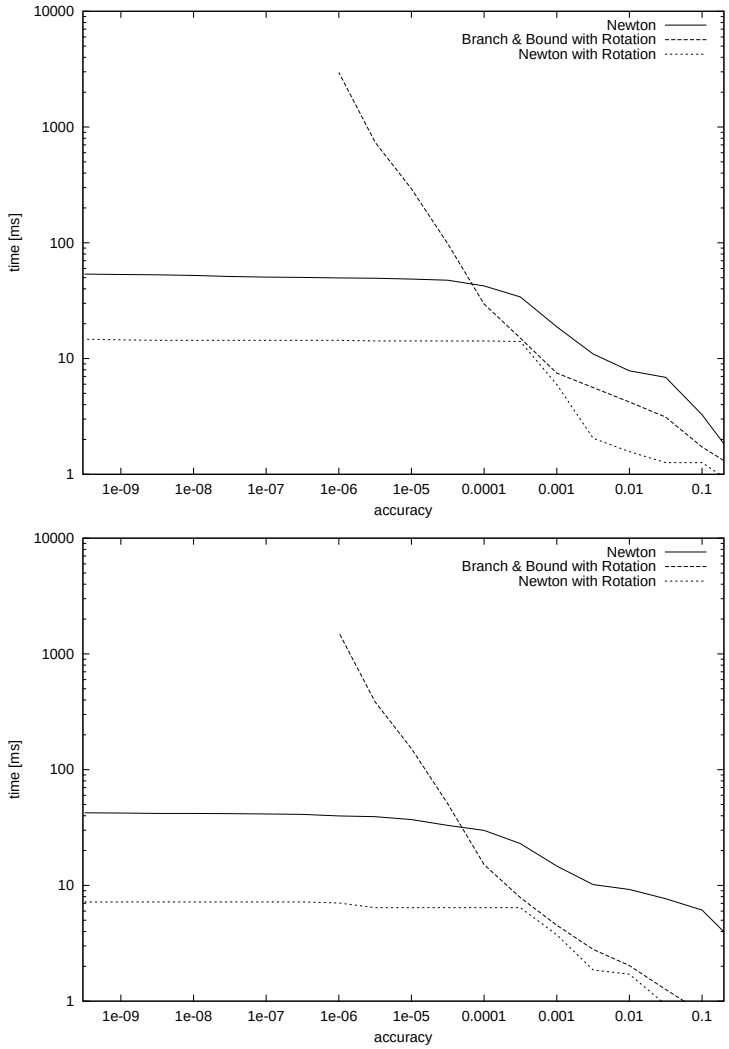

Figure 6.3: Speed comparison between the three methods discussed in the text on the first two classes of datasets for the line finding problem. The first one is for the plain line, the second for the line with additional random points.

Figures 6.3 and 6.4 show the results for the three classes for the line finding problem on the datasets with normals. We can see that the Newton method with the additional constraint on the rotation of the normals performs better than the Newton method without using normals. Also the branch and bound method is accelerated in the beginning by using the additional information. It is even faster than Newton without the rotational

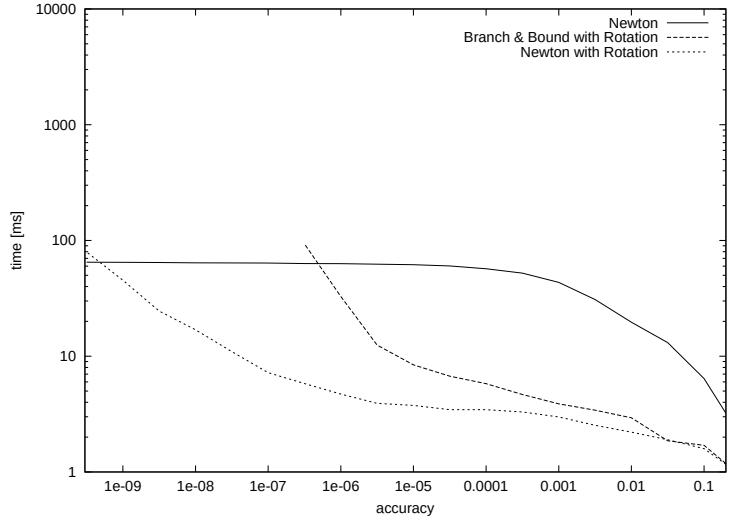

Figure 6.4: Speed comparison between the three methods discussed in the text on the third class of datasets, having only random points, for the line finding problem.

information at the beginning.

There is also another observation to be made: Finding the match for the second class of datasets is faster than for the first class. Because of the matchlists, the quality function Q only has to be evaluated over the 50 points forming the primitive in the end while in the first class, all 100 points belong to the primitive. Additionally, evaluating one addend for Q with the rotational constraint is more expensive than evaluating one for the quality function only regarding the position of the points in the datasets. For these two reasons, finding the solution for the second class of datasets is faster than for the first with respect to this problem.

For the third class, the Newton method using the rotational information is the fastest up to an accuracy of 10^{-9}. Afterwards, a situation comparable to the one described above occurs. The additional constraint produces an overhead in the end when none of the points that are still in the matchlist will be removed, while not offering a convergence rate that is high enough to be faster than the Newton method without using the normals. If really high accuracy is needed and the input data can be as bad as the datasets in class three, not using the additional information of the normals is faster. But in practice, this is hardly the case.

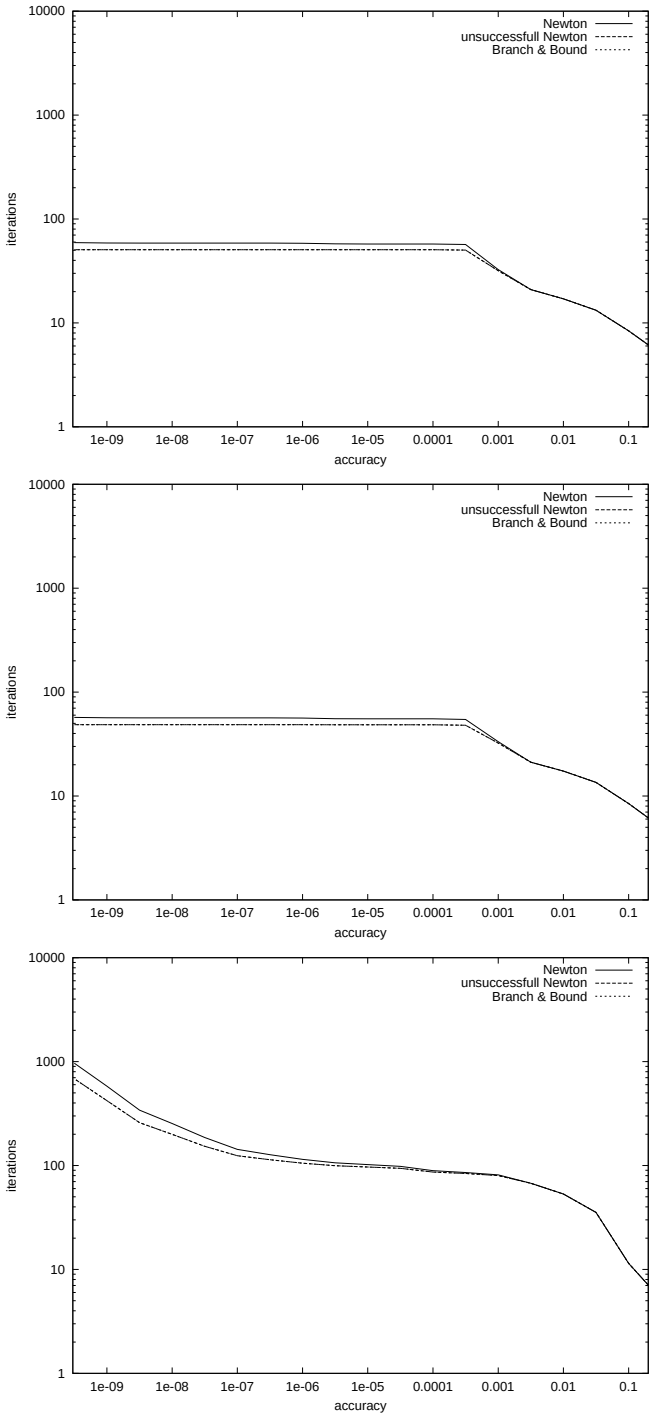

Figure 6.5: The effectiveness of dynamic switching with respect to the line finding problem. From top to the bottom, the three equivalence classes are plotted with respect to the Newton method applied to the datasets with normals of the points.

Figure 6.5 shows the number of iterations for a desired accuracy. The values are averaged over 100 datasets. From the top to the bottom, the three equivalence classes are plotted with respect to the Newton method applied on the datasets with normals of the points. The graphs have to be interpreted as in the previous chapters. For example have a look at figure 4.8. For class one and two, the graphs look nearly the same. For the third class, the number of branch and bound iterations as well as the number of Newton iterations again start to increase for high accuracy.

In the rare case that the datasets are not known to represent a primitive and very high accuracy is needed, the algorithm may take an unintentionally long time to complete. Otherwise, it is an acceleration compared to the line finding only using the positional information of the data points.

6.3.2 The Circle Finding Problem

Figures 6.6 and 6.7 show the results for the three classes for the circle finding problem on the datasets with normals. From the plot of the second class of datasets, we see that the Newton method with the additional constraint on the rotation of the normals performs better than the Newton method without using normals. Again, the branch and bound method gets accelerated in the beginning by using the additional information like it was for the line finding problem.

But for the plain circle, the situation is different. Comparing the branch and bound method and the Newton method using the rotational information, they behave the same at higher accuracies, but for low accuracies, branch and bound is more efficient. Comparing the Newton method using the rotational information and the Newton method not using it, the first is faster for low accuracies while the second is faster for high accuracies. All in all, for low accuracies up to an accuracy of approximately 0.005, using the branch and bound method with rotational information was the best and for higher accuracies, using the Newton method without rotational information was the best. This can be explained. For these datasets, the filtering effect of the added constraint on rotation, early ignoring points not belonging to the

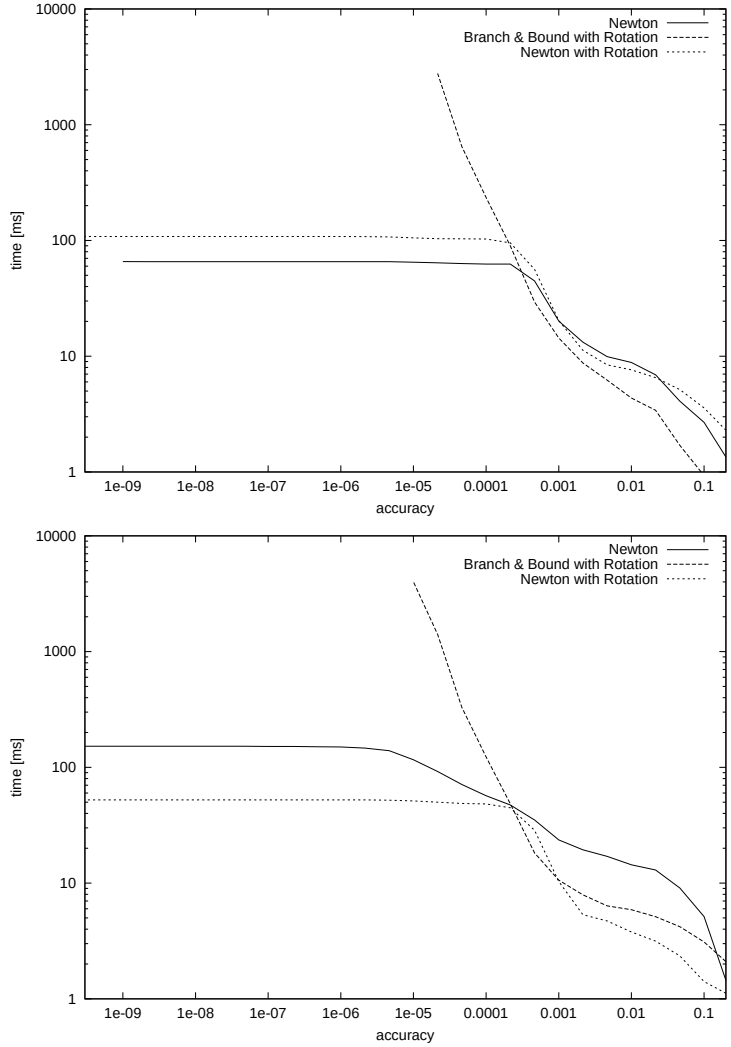

Figure 6.6: Speed comparison between the three methods discussed in the text on the first two classes of the datasets for the circle finding problem. The first one is for the plain circle, the second for the circle with additional random points.

primitive, does not occur because all points belong to the primitive. Therefore the Newton method ignoring the rotational information is faster than the Newton method using it.

In the second class, the Newton method with two constraints profits from the matchlists. It is overall the fastest, except for a small area of accuracy approximately from 0.001 to 0.002.

For the third class, the added information on rotation accelerates the

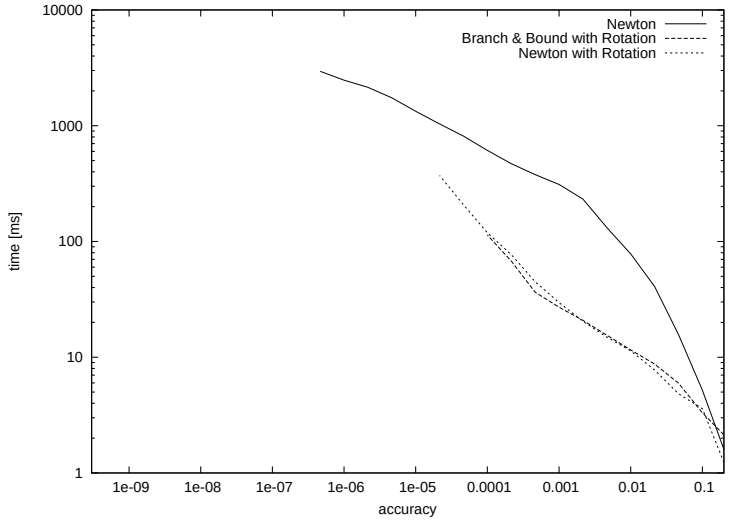

Figure 6.7: Speed comparison between the three methods discussed in the text on the third class of the datasets, having only random points, for the circle finding problem.

methods in the beginning. The plain Newton method wastes time on processing points that do not have a matching angle of their normals. But for higher accuracies, the overhead of evaluating the second constraint gets too high. Note that a graph ending at an accuracy lower than 10^{-9} means, that the time needed for calculating the solution for the next power of accuracy is above the prescribed time limit of 10000 ms in this case. The actual time needed for that accuracy is not known exactly and therefore not plotted. Therefore, at an accuracy of approximately 10^{-5}, the plain Newton method outperforms the methods using the normals.

Figure 6.8 shows the number of iterations for a desired accuracy. For class one and two, the graphs look nearly the same. For the third class, the number of branch and bound iterations as well as the number of Newton iterations again start to increase for high accuracy.

As for the line finding problem, in the rare case that the datasets are not known to represent a primitive and high accuracy is needed, the algorithm may take an unnecessarily long time to terminate. If the datasets are simple, the Newton method not using the normals is more efficient. But at least in this case, the Newton method using the normals still reaches the point of high convergence a bit later. For the general problem where points representing

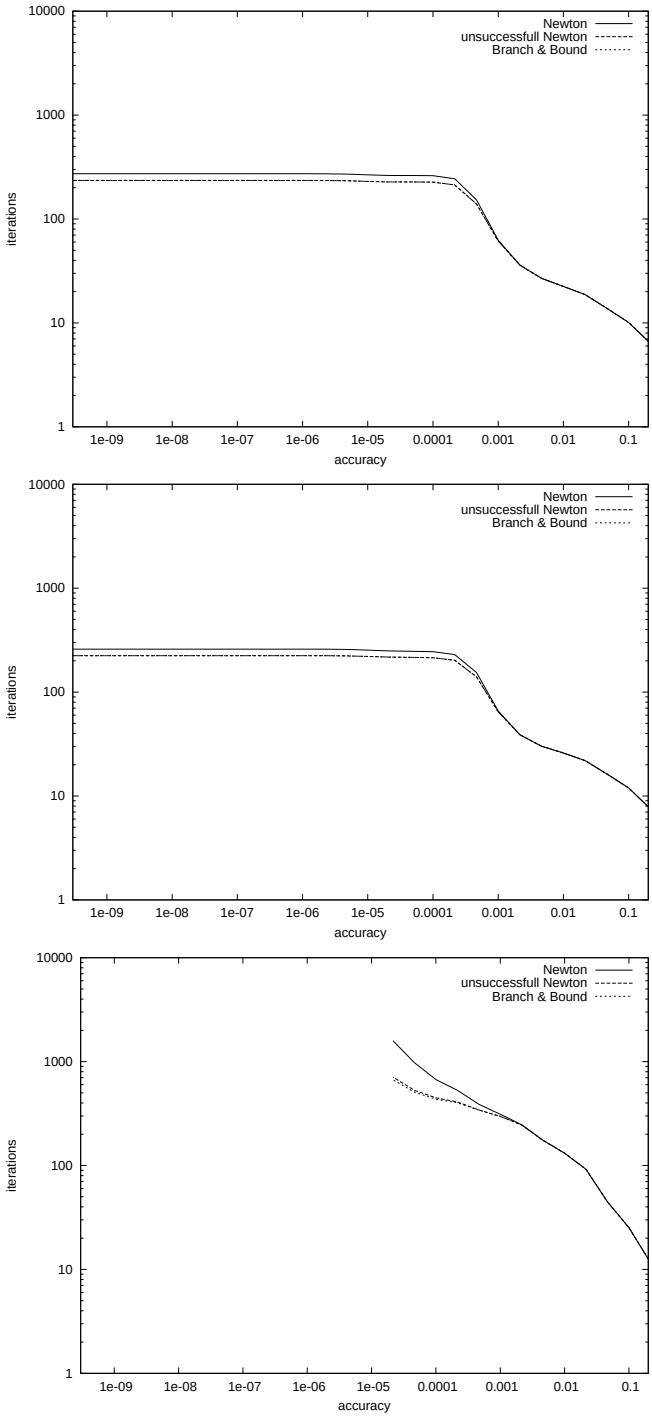

Figure 6.8: The effectiveness of dynamic switching with respect to the circle finding problem. From top to the bottom, the three equivalence classes are plotted with respect to the Newton method applied on the datasets with normals of the points.

the primitive and points produced by noise are mixed, it is an acceleration compared to the line finding only using the positional information of the data points.

6.3.3 The Ellipse Finding Problem

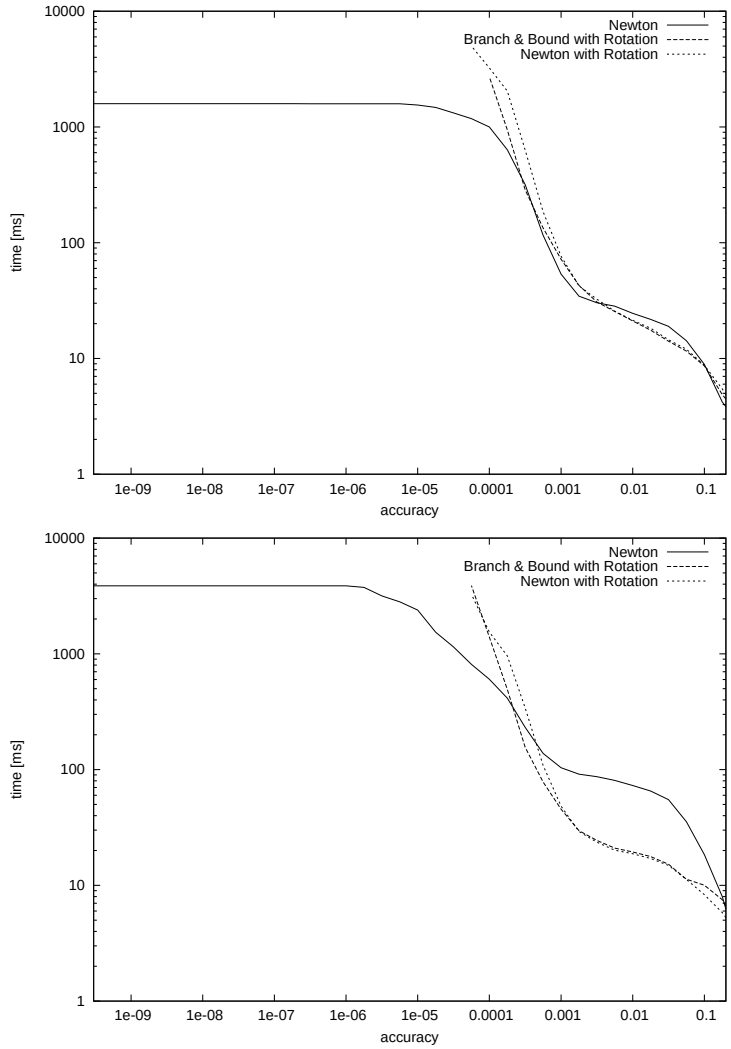

Figure 6.9: Speed comparison between the three methods discussed in the text on the first two classes of the datasets for the ellipse finding problem. The first one is for the plain ellipse, the second for the ellipse with additional random points.

Figures 6.9 and 6.10 show the results for the three classes for the ellipse finding problem on the datasets with normals. From the plot of the first class

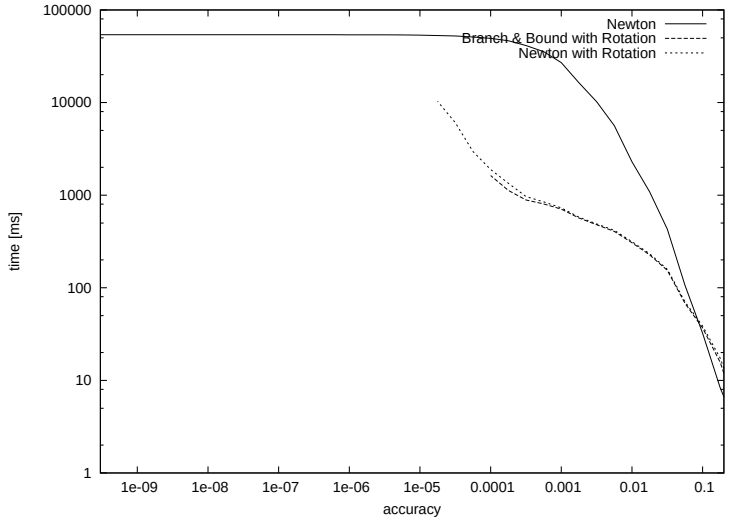

Figure 6.10: Speed comparison between the three methods discussed in the text on the third class of the datasets, with only random points, for the ellipse finding problem.

of datasets, we see that the plain Newton method is the best when no noise is in the data. In the beginning all methods behave the same. At an accuracy of approximately 0.007, the Newton method not using the normals becomes more efficient. The overhead of calculating the second constraint does not turn out to be profitable for this class. For the second and third class, in the beginning, the methods using the normals are faster than the method not using them. But at an accuracy of 0.002 for the second class and 10^{-5} for the third, ignoring the normals performs better. For the methods that use the normal information, the point where Newton converges faster by using Newton steps was not reached within the maximum calculation time of 100 s. It cannot be said from these plots if it actually reaches it, but for real time applications, 100 s is already too slow.

Figure 6.11 shows the number of iterations for a desired accuracy. We can also see that the point of high convergence was not reached because the number of Newton steps is for all three classes very low.

For high accuracies, using the information of the normals decreases the speed of the solution compared to the ellipse finding method not using the normals in the earlier chapters. For low accuracies, using the rotation of the normals as an additional information accelerates the calculation of the solu-

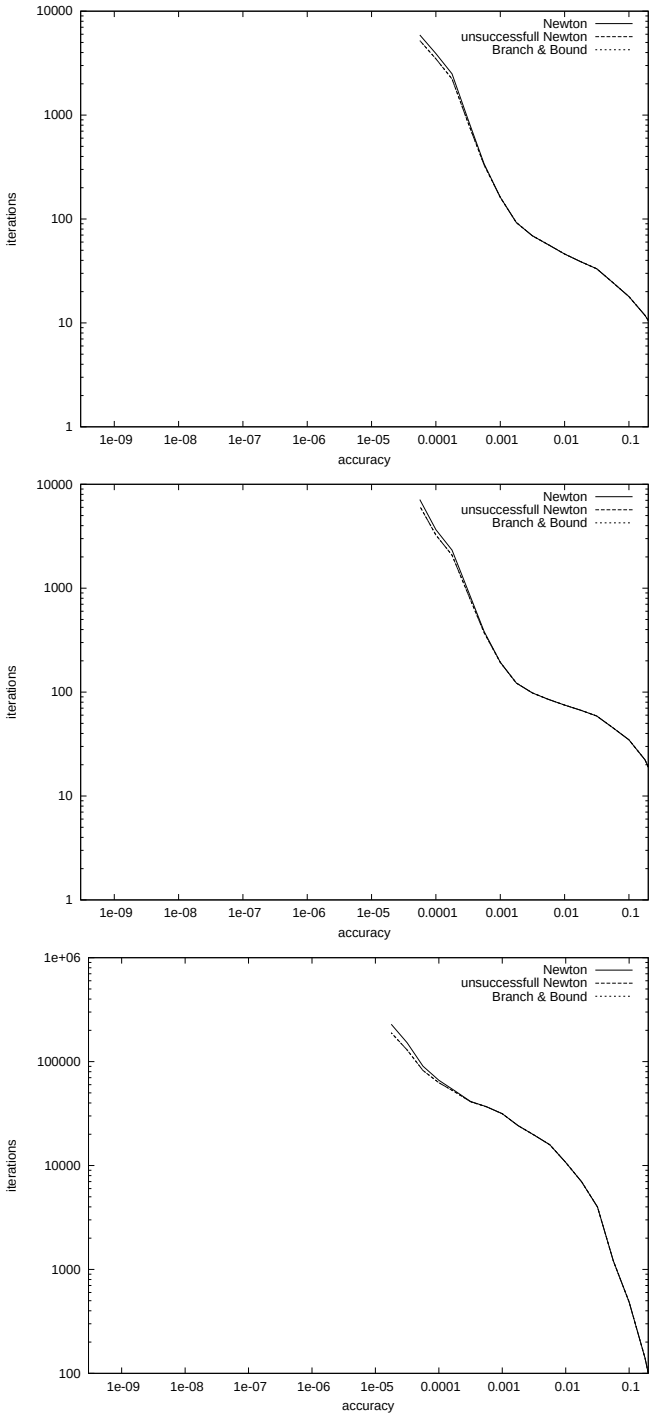

Figure 6.11: The effectiveness of dynamic switching with respect to the ellipse finding problem. From top to the bottom, the three equivalence classes are plotted with respect to the Newton method applied on the datasets with normals of the points.

tion. Whether this information gets used by the branch and bound method or the Newton method is not important, both methods behave practically the same for this problem.

6.3.4 Finding Ellipses at Arbitrary Orientations

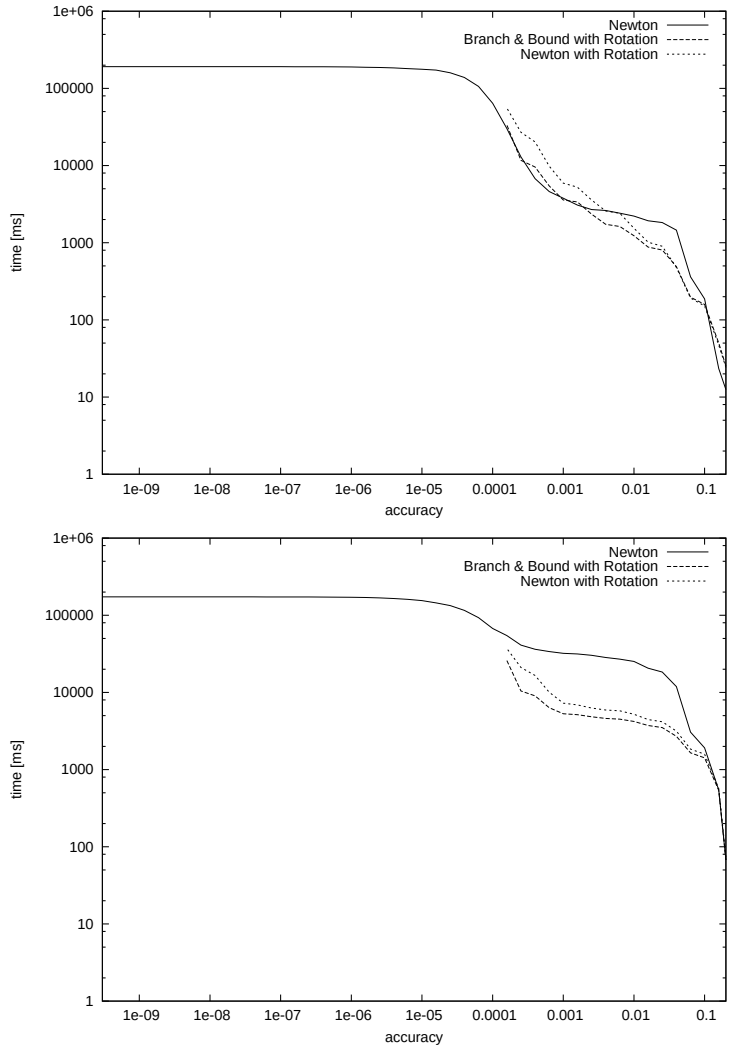

Figure 6.12: Speed comparison between the three methods discussed in the text on the first two classes of the datasets for the rotated ellipse finding problem. The first one is for the plain ellipse, the second for the ellipse with additional random points.

Figures 6.12 and 6.13 show the results for the three classes for the rotated ellipse finding problem on the datasets with normals. As for the previous

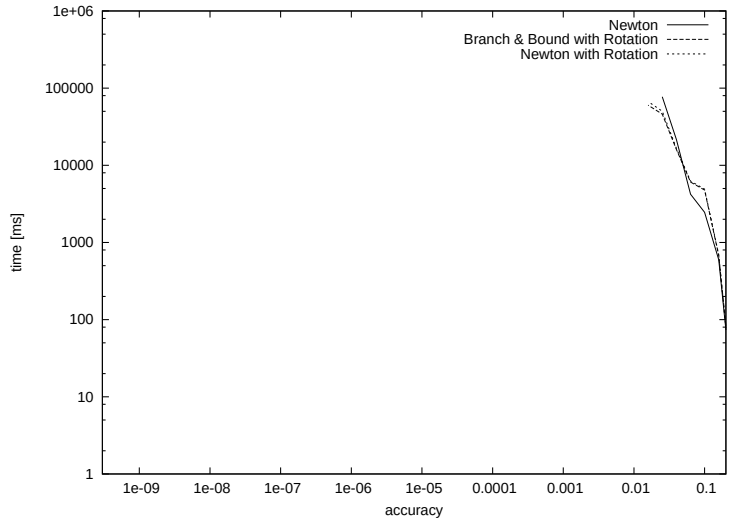

Figure 6.13: Speed comparison between the three methods discussed in the text on the third class of the datasets, having only random points, for the rotated ellipse finding problem.

problems, we see that at the beginning the methods using the normals are faster than the Newton method ignoring them, whereas the advantage is greatest for the second class of datasets. For the first class, the advantage is already smaller while for the third class, it vanishes completely. For class one and two and higher accuracies, the plain Newton method outperforms the methods using the normals. Once again the additional calculation of the rotational constraint produces a computational overhead while not removing any more points from the matchlists.

The plots of the number of iterations spent with the different types of steps are left out. They look the same for each of the three classes of the datasets. Hardly any Newton steps were made within the prescribed time.

Figure 6.14 shows the 100 representatives of the second class of datasets for three accuracies of the branch and bound method and the Newton method using the normal information. The behavior of the methods averaged over the 100 datasets stays the same. But comparing single datasets, we see that at some point, here it is the case for the highest accuracy plot, the methods perform differently. There are still some datasets where both methods need the same time, but also some datasets where the branch and bound method or the Newton method is faster.

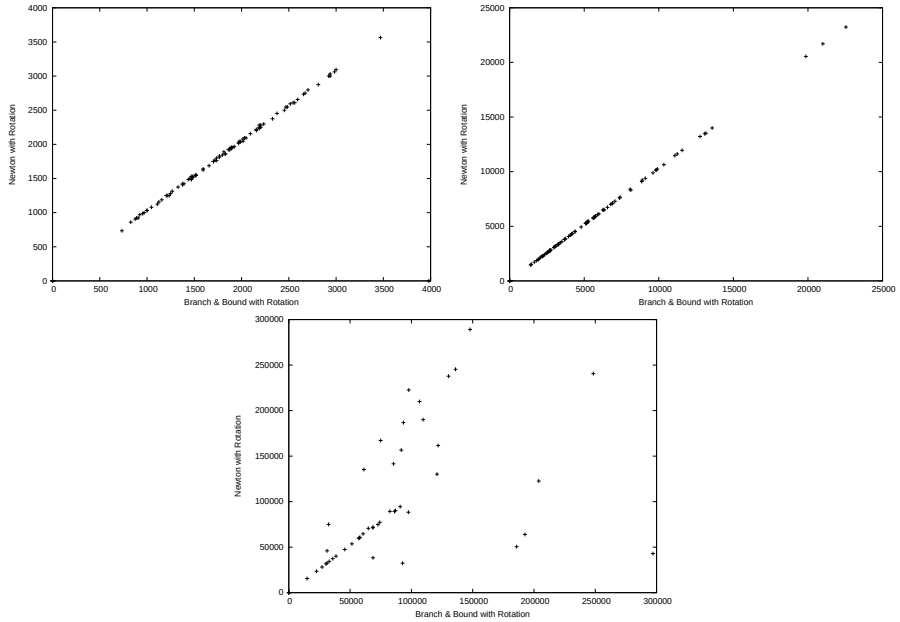

Figure 6.14: The 100 representatives of the class of the rotated ellipse datasets with random points were used. The first shows the times to get an accuracy of $10^{-1.25}$, the second the times needed for an accuracy of 10^{-2} and the third for an accuracy of 10^{-4}, per parameter in parameter space. Times are given in milliseconds.

For high accuracies, using the information of the normals decreases the speed of the solution compared to the rotated ellipse finding method not using the normals in the earlier chapters. For low accuracies, using the rotation of the normals as additional information accelerates the calculation of the solution. Whether this information gets used by the branch and bound method or the Newton method is not important, both methods behave practically the same for this problem.

6.4 Analysis of the Results

Adding a second constraint on the rotation of the normals of the point features accelerates the primitive finding problem for the line and the circle in the common case where a primitive is described in the input data mixed with points from the background noise. In combination, Newton is a more efficient choice than branch and bound for high accuracies. At some point for datasets with a lot of background noise, the evaluating of the second constraint may waste time compared to the plain position based methods. Normal information should not be used this way if the input data is noisy.

For the ellipse finding and rotated ellipse finding problem, using the normal information accelerates the problem for low accuracies. For higher accuracies, not using them is more efficient. It depends on the desired accuracy, if the branch and bound method using the normal information or if the Newton method only using the positions of the points should be used.

A general observation is: Using the normal information has the effect of determining early if a point is consistent to the searched primitive or not. In combination with the Matchlists, this explains the acceleration of the methods in the beginning phase of the algorithm execution.

Chapter 7

Conclusion

In this thesis, a combination of the branch and bound method on interval arithmetic and the Interval Newton method was presented. The method was adapted to work efficiently with only requiring locally twice continuous differentiable functions by introducing dynamic switching and therefore is well suited for robust matching problems. The method also makes use of matchlists for acceleration.

The following robust geometric matching problems were discussed: The line finding, the circle finding, the ellipse finding and the rotated ellipse finding problem. They were formalized to be used with the method introduced. Once with only a constraint on the position of the points in the dataset and once with an additional constraint on the angle of the normal of the points. Experimental results were shown, comparing variations of the branch and bound method and the Interval Newton method with and without matchlists and dynamic switching. For the line and circle finding problem, the new combined method accelerated the solution clearly. Especially, when the normal information is used, the problems were solved fast. For practical cases, it outperformed the other methods that it was compared with or at least was as good as them, reaching times of < 20 ms for the line finding problem and < 100 ms for the circle finding problem, on the datasets used using the normal information. For the ellipse and rotational ellipse finding problem, the method also accelerates the solution, but it is still much slower than the

line and circle finding problem. For higher accuracies, it is better not to use the normal information.

7.1 Future Work

There are some ideas that could still be tried:

For the ellipse and rotational ellipse finding problem, it could be tried to switch between using the rotational information and not using it at some point. However, by switching this, the problem definition changes and theoretically, the result is not clear. Practically it probably works and perhaps the problem definition can be tweaked to also be theoretically a clearly defined optimization problem. A criteria which decides whether to use the rotational information or not is needed for this approach.

For the line finding problem, the idea of reducing the number of features by matching on datasets consisting of line segments instead of points as it was done by Breuel in his RAST implementation in [2] and [4], perhaps also might be accelerated by using the Newton method. The problem is, defining the optimization problem for the line finding problem on line segments gets more complicated and therefore, the terms of the derivatives also get more complicated and expensive to evaluate. The overhead may overcome the benefit of this approach, resulting in lower efficiency. But maybe, this effect only occurs for very high accuracies. If faster line matching is needed, this could be tried.

The methods used here only find one primitive in the dataset and afterwards the algorithm terminates. In practice, one may want to find several primitives. It is possible to extend the algorithm to find a number n of primitives by removing the points belonging to primitives already found from the dataset and continuing the search with the current list of intervals to process. Removing points from the dataset while running the algorithm is safe. The only thing that could happen is that the quality of an interval in the list is too high. But since the quality in the list is an upper limit for the quality of all points in an interval, this is no problem.

Bibliography

[1] M. D. Alder. Inference of syntax for point sets. *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, 1994.

[2] T. M. Breuel. A practical, globally optimal algorithm for geometric matching under uncertainty. In *International Workshop on Combinatorial Image Analysis (IWCIA 2001), Philadelphia, CA*, 2001.

[3] T. M. Breuel. Implementation techniques for geometric branch-and-bound matching methods. *Comput. Vis. Image Underst.*, 90(3):258–294, 2003.

[4] T. M. Breuel. Implementation techniques for geometric branch-and-bound matching methods. Accepted for publication in *Computer Vision and Image Understanding*, 2003.

[5] T. M. Breuel. On the use of interval arithmetic in geometric branch-and-bound algorithms. Accepted for publication in *Pattern Recognition Letters*, 2003.

[6] C.-Y. Chen. "Adaptive numerische Quadratur und Kubatur mit automatischer Ergebnisverifikation", 1998.

[7] E. Davies. *Machine Vision: Theory, Algorithms, Practicalities.* Academic Press Ltd, 24/28 Oval Road, London NW1 7DX, United Kingdom, 1990.

[8] M. Fitzgibbon, A. W.and Pilu and R. B. Fisher. Direct least-squares fitting of ellipses. *Pattern Analysis and Machine Intelligence*, 21(5):476–480, May 1999.

[9] D. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[10] H. K. Yuen, J. Illingworth and J. Kittler. Detecting partially occluded ellipses using the hough transform. *Image and Vision Computing*, 7(1):31–37, 1989.

[11] M. Hagedoorn and R. Veltkamp. Reliable and efficient pattern matching using an affine invariant metric, 1997.

[12] E. Hansen and R. I. Greenberg. An interval newton method. *Appl. Math. Comput.*, 12:89–98, 1983.

[13] J. Illingworth and J. Kittler. A survey of the hough transform. *Comput. Vision Graph. Image Process.*, 44(1):87–116, 1988.

[14] F. Jurie. Solution of the simultaneous pose and correspondence problem using Gaussian error model. *Computer Vision and Image Understanding: CVIU*, 73(3):357–373, 1999.

[15] R. B. Kearfott, C. Hu, and M. N. III. A review of preconditioners for the interval Gauss-Seidel method. *Interval Computations*, 1(1):59–85, 1991.

[16] N. Kiryati, Y. Eldar, and A. M. Bruckstein. A probabilistic hough transform. *Pattern Recogn.*, 24(4):303–316, 1991.

[17] V. F. Leavers. Which hough transform? *CVGIP: Image Underst.*, 58(2):250–264, 1993.

[18] R. A. McLaughlin and M. D. Alder. Recognising cubes in images. *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, 1994.

[19] R. A. McLaughlin and M. D. Alder. The hough transform versus the upwrite. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(4):396–400, 1998.

[20] D. Mount, N. Netanyahu, and J. L. Moigne. Efficient algorithms for robust feature matching, 1998.

[21] C. F. Olson. Locating geometric primitives by pruning the parameter space. *Pattern Recognition*, 34(6):1247–1256, 2001.

[22] J. Princen, J. Illingworth, and J. Kittler. A hierarchical approach to line extraction based on the hough transform. *Comput. Vision Graph. Image Process.*, 52(1):57–77, 1990.

[23] P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, 1997.

[24] W. Wells. Statistical approaches to feature-based object recognition, 1997.

[25] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: randomized hough transform (rht). *Pattern Recogn. Lett.*, 11(5):331–338, 1990.