Bachelor Thesis

Mathematical Methods of Image Processing to Characterize Micro Cups

John Wilfried Schlasche

July 26, 2011

Contents

Introduction **1**
0.1 Notation . . . 3

1 The Perona-Malik Equation **4**
1.1 The Idea Behind this Approach . . . 4
1.1.1 Partial Differential Equations . . . 4
1.1.2 PDEs in Image Processing . . . 5
1.1.3 How to Find the Appropriate Equation? . . . 6
1.1.4 The Perona-Malik Diffusion Equation . . . 6
1.2 Theory . . . 7
1.2.1 Behavior . . . 7
1.2.2 Existence and Uniqueness . . . 9
1.3 Numerical Implementation and tests of the PM-Method . . . 16
1.3.1 Which Approach is Used? . . . 16
1.3.2 What Problems May/Did Occur . . . 19
1.4 Appliance to our Data . . . 20
1.4.1 Application to Workpiece Data . . . 20
1.4.2 Examples of the Problems that did Occur . . . 22

2 Characterization **23**
2.1 What information are we looking for? . . . 23
2.2 Parameters of the workpiece . . . 23
2.2.1 Problems and Conditions Placed on the Image . . . 25
2.3 Assessment of the Rim's Characteristic . . . 25
2.3.1 Application to our Measurement Data . . . 27
2.4 Evaluating the Quality of the Workpiece's Bottom . . . 28
2.4.1 Application . . . 29
2.5 How to Find Cracks in the Rim . . . 30

Appendix **30**

A Further Tests **31**
A.1 Test1 31
A.2 Test2 34

B Source Code **37**
B.1 Source Code of the Perona-Malik Diffusion 37
B.2 Source Code of the Parameter Search 41
B.3 Source Code of the Edge Characterization 46
B.4 Smoothing the Edge Data 48
B.5 Source Code of the Form Assessment 49

Bibliography **51**

Introduction

In a micro deep drawing process a thin metal sheet is fixated under a punch, which then draws the sheet into the required form. Especially during the production of very small parts, many unanticipated manufacturing errors such as cracks, wrinkles, defects of form and chip formation may occur.
In order to minimize these flaws, the resulting parts have to be characterized and the parameters changed accordingly. To get the characterization done in reasonable time, a laser scanning microscope is used, but since the obtained image data of the micro parts are mostly very noisy, characterization becomes exceedingly complicated.
The characterization consists of two main steps of procedure. First the "important" edges in the data have to be found and secondly the micro part itself and its flaws have to be found and classified.

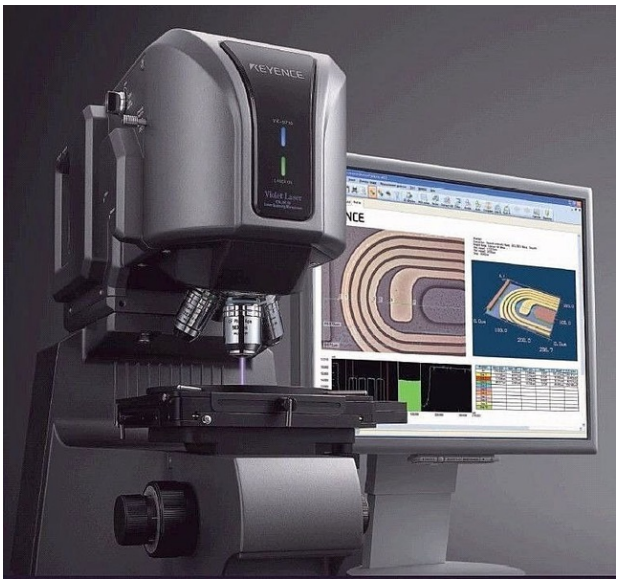

Figure 1: Keyene VK 9700 3D Laser microscope [13]

During the first part of the process the usage of standard algorithms such as Sobel edge detection connected with standard smoothing, leads to serious problemssuch as:

> The shape found in the edge image might not be the original one due to edge displacement.
>
> It might be impossible to distinguish between noise and real edges in the image (noise may be far stronger represented than important edges of fissures).
>
> Even if the noise is reduced sufficiently there may be far too many edges to make a definite statement about the quality of the micro part.

The first subject of this thesis is the application of the Perona-Malik diffusion to this problem. The idea behind this smoothing algorithm is to use a certain partial differential equation instead of solely applying filters to the image.
The second part of the thesis deals with possible ways to characterize the workpieces based on the resulting image data.

0.1 Notation

t	Time variable
x	Space-variable
u=u(t,x), v=v(t,x)	Function of real or complex value. If not stated otherwise, they are defined on Ω (Usually this denotes our image-data).
$\Omega \subset \mathbb{R}^n$	Open and bounded domain with smooth boundaries $\partial\Omega$
A,B,C	Matrices
$a_{i,j}$, $b_{i,j}$, $c_{i,j} \in \mathbb{C}$ or $\mathbb{R}$	Entries of the matrices above
g = g(x)	Function with domain $[0, \infty]$ (generally the diffusivity function).
$\phi \in C_0^\infty(\Omega)$ (or $\in H_0^{1,2}(\Omega)$)	Test function
∂^s	The partial differential operator for the multindex s
$\frac{\partial^2 u}{\partial x_1 x_2} = u_{1,2}$	if it is clear that $(\cdot)_i$ is no indexing
$\frac{\partial^2 u}{\partial x_1 x_2} = u_{x_1,x_2}$	Otherwise
$W^{k,p}(\Omega)$, $k \in \mathbb{N}$	$= \{u \in L^p(\Omega) \vert D^s u \in L^p(\Omega) \, \forall s : \vert s\vert \leq k\}$ The Sobolev space (D^s denotes the weak derivative)
$W^k(\Omega)$	$= W^{k,2}(\Omega)$ note that this is a Hilbert space [5]
$X(\Omega)$	A function space (mostly $W^1(\Omega)$ or its dual $W^{1*}(\Omega)$)
$L^p([0,T];X)$	$= \{u(t,x) \vert \forall t \in [0,T] : \ u(t,\cdot) \in X$ and $(\int_0^T \vert\vert u(t)\vert\vert_X dt)^{1/p}\}$
$\frac{\partial u}{\partial n}$	Mostly used to denote the flow of the function u across the Border of Ω where n is the normal of $\partial\Omega$

Chapter 1

The Perona-Malik Equation

1.1 The Idea Behind this Approach

In this chapter important concepts which help to understand and in a way lead to the Perona-Malik algorithm are introduced.

1.1.1 Partial Differential Equations

As we will come to see, he theory of partial differential equations (short: PDEs) is of great importance in this work and a short introduction is therefore given at this point. Theoretical results including existence and uniqueness are given in Chapter 2. Generally a PDE is an equation or a system of equations which include the partial derivatives of an unknown function u of more than one independent variable.
Important examples are:

1. The linear wave-equation: $\frac{\partial^2 u}{\partial t^2} - c^2 \Delta u(t,x) = 0$ whose solution, for $c =$ speed of light, is the electro-magnetic wave.

2. The heat equation $\frac{\partial u}{\partial t} - \Delta u = 0$.

3. The Laplace equation $\Delta u = 0$ which describes all harmonic functions such as electromagnetic potentials in the static case.

Most PDEs are much more complex than the examples given above and unlike ordinary differential equations (ODEs) there are no simple conditions that ensure the existence and uniqueness of a solution.
PDEs can be categorized into the three groups elliptic, parabolic and hyperbolic PDEs. The general definition is quite complicated and is in part to be found in section 1.2.2, see [8] for more details.
In most cases it is sufficient to examine at the time-derivative in the equation:

- No time-derivative: Elliptic system,

- Time-derivative of the first order: Parabolic system,
- Time-derivative of the second order: Hyperbolic system.

The most important examples happen to be the examples above. The wave-equation is hyperbolic, the heat-equation is parabolic and the Laplace-equation is an elliptic PDE.

1.1.2 PDEs in Image Processing

The concept of scale space is part of the axiomatic image processing and therefore a mathematical way to:

> Derive and prove features of certain algorithms used for image processing.
>
> Create algorithms for image processing on the basis of properties postulated on images.

For a fuller treatment of of scale space theory we refer the reader to [1] Chapter 4. In general, a multi-scale analysis is a family of transformations $\{T_t\}_{t\geq 0}$ which defines a transformation for every $t \geq 0$. In our case the idea is to neglect finer details of the original image for greater t. As an example, a tree might be seen with all its leafs and branches for $t = 0$. As t is increased, one can still recognize thicker branches but the leafs are blurred. For even greater t only the log and the treetop are distinguishable.
This concept fits very well for this type of image processing where the goal is to ignore the noise added by the camera, while preserving the general structure of the micro part.

Theorem 1.
It is shown in [1] that if a multiscale analysis fulfills following requirements:

1) *Translation-invariance:* $(T(u(\cdot + x')))(x) = (T(u))(x + x')$
2) *Monotony:* $u \leq v \Rightarrow T_t u \leq T_t v$
3) *Invariance with respect to gray scale shifting*
4) *Regularity:* $||T_t(u + hv) - T_t u - hv||_\infty$ *is bounded*
5) *Recursiveness:* $T_s(T_t(u)) = T_{s+t}u$
6) *Locality:* $(T_t u)(x)$ *only depends on the values of* $u(.)$ *near* x

Then there exists a continuous functions $F(\cdot, \cdot, \cdot)$ *such that*

$$\frac{\partial u}{\partial t}(t, x) = F(x, u(t, x), \nabla u(t, x), \nabla^2 u(t, x)),$$

where ∇^2 *is the Hessian matrix.*

These requirements are fulfilled by quite a wide range of image processing algorithms. Indicating that there may be a way to define an algorithm which fits our needs, using PDEs.

1.1.3 How to Find the Appropriate Equation?

The Gaussian filter seems the most obvious approach, for it surely eliminates most of the noise. Smoothing a signal $u_0(x)$ with the Gaussian kernel in 2D ($G_\sigma(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{|x|^2}{2\sigma^2}}$) is the same as solving the Heat-equation

$$\frac{\partial u}{\partial t} = c\Delta u(t,x) = div(c\nabla u(t,x))$$

for $u(0,x) = u_0(x)$ and $t = \sigma$.
This equation can be derived using firstly Fourier's law and secondly the enegergy conservation law.
Fourier's law states that the flow rate of any concentration or heat energy is proportional to the negative gradient of the concentration or the temperature respectively;

$$j_u(t,x) = -k\nabla u(t,x)\ k \in \mathbb{R} \text{ (in the isotropic case, k is constant)},$$

where j_u denotes the flow of u and isotropic means that the diffusion is independent of x. The energy conservation law or the the mass conservation law states that the change in energy or mass in a domain equals the flow across its border. Combined with partial integration this is where the $div(\cdot)$ stems from.

Nevertheless, the diffusion equation in this form is not applicable for it moves and eventually eliminates all edges and thereby changes the outcome of the characterization in an undesired way.

1.1.4 The Perona-Malik Diffusion Equation

To adapt the diffusion equation to our needs, a so-called anisotropic diffusion, where the flow is not only proportional to the gradient, but is also controlled by a function $g(|\nabla u|)$, is used. Regions with low $|\nabla u|$ are plains. By choosing a high diffusion coefficient the noise can be reduced. Regions with high $|\nabla u|$ can be found near edges. In order for those edges to be preserved (or even enhanced as we will see), a low diffusion coefficient is chosen accordingly. This leads to the function

$$g : [0,\infty] \mapsto [0,1],\ g(0) = 1,\ \lim_{s\to\infty} g(s) = 0 \text{ monotonically decreasing.}$$

In order to obtain a complete system, which is needed to solve the equations, boundary conditions also have to be defined. As is common in image processing the Neumann boundary condition $n \cdot \nabla u = 0$ on $\partial\Omega$ is used. This ensures that there is no flow across the boundary and the overall brightness is thereby preserved. This lead to following system of

equation:

$$\frac{\partial u}{\partial t} = div(g(|\nabla u|)\nabla u) \text{ on } \Omega \tag{1.1}$$

$$n \cdot \nabla u = 0 \text{ on } \partial\Omega \text{ with n = normal of } \partial\Omega \tag{1.2}$$

$$u(0,x) = u_0(x) \text{ on } \Omega \tag{1.3}$$

1.2 Theory

1.2.1 Behavior

To get a first overview of what the Perona-Malik diffusion does, we convert the right side of equation (1.1) by using $div(g \cdot \vec{f}) = \nabla g \cdot \vec{f} + div(\vec{f})g$ and $div(\nabla u) = \Delta u$.

$$div(g(|\nabla u|)\nabla u) = \underbrace{\nabla(g(|\nabla u|)) \cdot \nabla u}_{*} + g(|\nabla u|)\Delta u.$$

The term (*) can be transformed even further:

$$\begin{aligned}\nabla(g(|\nabla u|)) \cdot \nabla u &= \frac{g'(|\nabla u|)}{|\nabla u|}(u_1 u_{1,1} + u_2 u_{2,1}, u_1 u_{1,2} + u_2 u_{2,2}) \cdot \nabla u \\ &= \frac{g'(|\nabla u|)}{|\nabla u|}(u_1^2 u_{1,1} + u_2 u_1 u_{2,1}, u_1 u_2 u_{1,2} + u_2^2 u_{2,2}).\end{aligned}$$

$$\Rightarrow div(g(|\nabla u|)\nabla u) = \frac{g'(|\nabla u|)}{|\nabla u|}\nabla u^T H(u)\nabla u + g(|\nabla u|) \cdot \Delta u\,, \tag{1.4}$$

where $H(u)$ is the Hessian matrix.
To analyze the behavior of this, we now take a closer look at the edges in our image:
In 2 dimensions (which is what we're dealing with) we can (locally) define two coordinates:

- $\eta := \frac{\nabla u}{|\nabla u|}$ which is the direction of greatest decrease/increase. In the vicinity of an edge, this is orthogonal to the edge.
- ν is defined to be an orthogonal to η with $||\nu|| = 1$ (near an edge there always exists such a η), this is parallel to the edge.

Note that the Laplace-operator Δ is rotation invariant and therefore the same for every orthonormal basis:

$$\Delta u = \frac{\partial^2 u}{\partial e_1^2} + \frac{\partial^2 u}{\partial e_2^2} = \frac{\partial^2 u}{\partial \eta^2} + \frac{\partial^2 u}{\partial \nu^2}$$

where $e_1 = (1,0)^T$, $e_2 = (0,1)^T$. Using all this the equation (1.4) can be written as follows:

$$div(g(|\nabla u|)\nabla u) = (g'(|\nabla u|)|\nabla u| + g(|\nabla u|))u_{\eta,\eta} + g(|\nabla u|)\underbrace{(\Delta u - u_{\eta,\eta})}_{=u_{\nu,\nu}}.$$

In this notation the equation is split into a term which describes the diffusion along the edge and one which describes the diffusion orthogonal to the edge.
If we define the function f as $f(s) := s \cdot g(s)$ we can shorten the equation even further, and obtain

$$div(g(|\nabla u|)) = f'(|\nabla u|)u_{\eta,\eta} + g(|\nabla u|)u_{\nu,\nu}\,. \tag{1.5}$$

In this form, one can already see that there might be something resembling backward diffusion if $f'(|\nabla u|)$ is of negative value and $u_{\eta\eta} >> u_{\nu\nu}$ which is the case near edges.
Backward diffusion means that unlike normal diffusion, which eventually smooths everything, the Perona-Malik diffusion may indeed not only preserve strong edges but also enhance them.

Theorems on the Behavior of the Perona-Malik algorithm in one Dimension

The following theorems deal with the behavior of the Perona-Malik (P-M) diffusion near edges in one dimension and are taken from [1] and [2].
They can to some extend be applied to 2D, since as we have seen we can locally consider the orthonormal basis η, ν in which η is the most "important" direction. To put it more graphically: We can (locally) look at slices of the original image and treat them as one dimensional signals.

Definition 1. *We say that $u \in C^3(\mathbb{R}, \mathbb{R})$ has an edge in x if:*

$$\begin{aligned} |u'(x)| &>> 0 \text{ (An edge is always an area of big change in the signal)},\\ u''(x) &= 0 \text{ (the edge is a point, not a whole area)},\\ u'(x)u'''(x) &< 0. \end{aligned}$$

The last point guarantees that $|u'|$ is locally maximal in x, this can best be understood by looking at an example:
If u' is of positive value we have a local maximum of u' in x if $u'' > 0$ to the left of x and $u'' < 0$ to the right. Together with the second condition this can be expressed by $u(x)''' < 0$. The case for $u' < 0$ follows analogously.

Theorem 2. *Theorem on edge-behavior*
The notation $\frac{\partial^1 u}{\partial \eta} = u_\eta$, $\frac{\partial^2 u}{\partial \eta^2} = u_{\eta,\eta}$ is used here.
Let $u(t,x)$ denote a strong solution for the Perona-Malik equation on $[0,T] \times \Omega$ with $u(0,x) = u_0$. Suppose $u(t,\cdot) \in C^5(\mathbb{R},\mathbb{R})\, \forall t \in [0,T]$ and $u(\cdot,x) \in C^1(\mathbb{R},\mathbb{R})\, \forall x \in \Omega$.

Let x denote the position of an edge in u_0 where in addition to the definition above $u_0''''(x) = 0$ then :

$$\partial_t u_\eta = f'(u_\eta) u_{\eta,\eta} \tag{1.6}$$

$$\partial_t u_{\eta,\eta} = 0, \text{ (this ensures that inflection points are preserved)} \tag{1.7}$$

$$\partial_t (u_\eta u_{\eta,\eta,\eta}) = 3f''(u_\eta) u_{\eta,\eta,\eta} \underbrace{u_{\eta,\eta,\eta} u_\eta}_{<0} + f'(u_\eta) u_\eta u_{\eta,\eta,\eta,\eta,\eta} + f'(u_\eta) u_{\eta,\eta,\eta}^2 \tag{1.8}$$

The first equation describes whether an edge is preserved, enhanced or reduced and the second ensures that inflection points are preserved (meaning that if x is an edge point before the application of the P-M algorithm, it still is one afterwards). The third equation looks quite abstract but nevertheless it describes an important potential problem in the 1D case of the Perona-Malik algorithm called staircasing: If $\partial_t(u_\eta u_{\eta,\eta,\eta}) > 0$ it may be the case that for some $t \in [0, T]$ $u_\eta(x) u_{\eta,\eta,\eta}(x) > 0$ which would mean that although we had a point of maximum $|u_\eta|$ to begin with it somehow changed to a point where $|u_\eta|$ is not locally maximal.

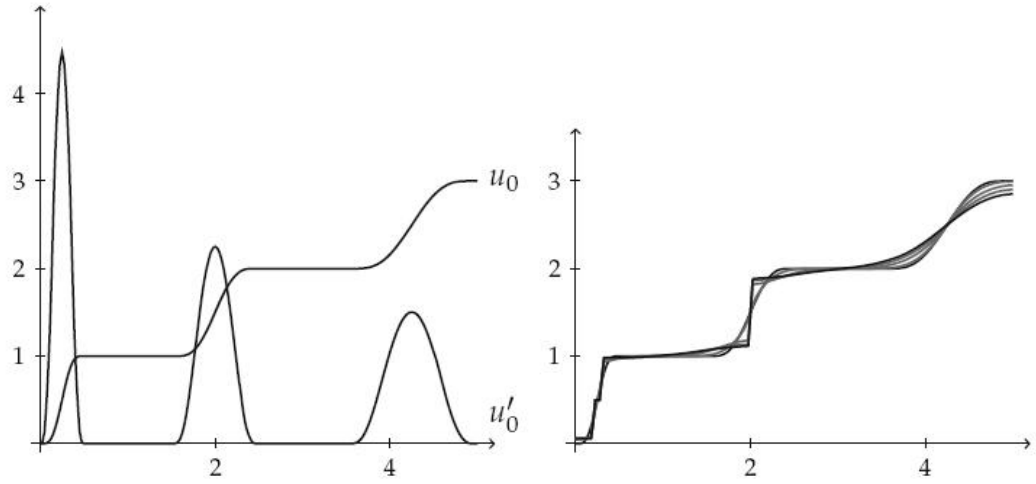

Figure 1.1: Staircasing effect in one dimension [1].
The left figure shows the function and the absolute value of its gradient, the right figure shows the function before and after the the Perona-Malik algorithm. The staircasing effect can be observed on the leftmost edge, whose gradient is maximal.

We did not encounter Staircasing-effects in 2D in any of our Tests, this might be due to our image-data or the chosen parameters. However these theorems show how important the choice of g is, by far not every decreasing function is appropriate. A wide range of functions is introduced and analyzed in [2].

1.2.2 Existence and Uniqueness

So far we assumed that there is a unique solution for the Perona-Malik PDE, which is actually not obvious as the following example will show.

As we know, in one dimension the equation has following form:

$$\frac{du}{dt} = (g'(u')u' + g(u'))u''$$

with the definition $f(s) = s \cdot g(s)$ this can be converted to

$$\frac{du}{dt} = f'(u')u'' \,,$$

where f is symmetric and decreasing for greater $|u'|$. This can in approximation be written as

$$\frac{du}{dt} + cu'' = 0 \,,\; c > 0 \,,$$

which is the inverse heat-equation. This equation is the inversion of smoothing, which gives us quite different results for almost identical starting values! (Note that this is not a proof that all functions $g(s)$, such that $s \cdot g(s)$ is decreasing for some values, are inappropriate. It is just an example why solvability and uniqueness is not a trivial issue.)

In this section we will take a short look at theoretical results on the solvability of our equation, for only then can we be sure that our numerical outcome is reproducible at any time.

Results for linear Parabolic Equation

Only the key idea of the proof is written down at this point, see [8] for the complete proof.

Definition 2. *The differential operator of second order*
Let $[0,T] \subset \mathbb{R}_{\geq 0}$ be the time-interval then the PDE is defined as

$$\frac{\partial u}{\partial t} + Lu = f \;\; on \;\; (0,T] \times \Omega \tag{1.9}$$

$$A\nabla u \cdot n = 0 \;\; on \;\; (0,T] \times \partial\Omega \tag{1.10}$$

$$u(0,\cdot) = u_0(\cdot) \;\; on \;\; \Omega \,, \tag{1.11}$$

where L denotes the partial derivatives of u up to second order (see below), $A\nabla u$ is the flow of u and n as before the normal of $\partial\Omega$. In the notation we used before ($u_{x_i} = \frac{\partial u}{\partial x_i}$) L has following form:

$$Lu = \sum_{i,j}^{n} (a_{i,j}(t,x)u_{x_i})_{x_j} + \sum_{i=1}^{n} b_i(t,x)u_{x_i} + c(t,x)u \,.$$

Remark:
$A(t,x)_{i,j} = a_{i,j}$ and in the following we consider the case where A is symmetric.

Not every PDE has a unique solution, we therefore define the parabolic differential operator.

Definition 3.
We say that the differential operator $\frac{\partial u}{\partial t} + Lu$ *is uniformly parabolic if there exists a constant* $\Theta > 0$ *such that*

$$\sum_{i,j}^{n} (a_{i,j}(t,x)\xi_i\xi_j) > \Theta|\xi|^2 \;\; \forall (t,x) \in [0,T] \times \Omega, \;\; \xi \in \mathbb{R}^n$$

Note that in our case the Perona-Malik equation becomes parabolic (not linear!) as soon as $g(t,x) > c$ for a constant $c > 0$.

Weak solutions for parabolic PDEs are defined quite similar to weak solutions of elliptic PDEs [5] using differentiable test functions ϕ. As is introduced in [5] we define weak derivatives as

$$\int_\Omega \partial^s \phi u = (-1)^{|s|} \int_\Omega \phi \partial^s u \,.$$

This can also be applied to the time-derivative:

$$\int_{[0,T]} \phi' u = -\int_{[0,T]} \phi u' \,.$$

In the following section u' refers to the weak time derivative of the solution u .
If we multiply (1.9) with a test function ϕ and integrate over Ω we get following equation:

$$\int_\Omega u'\phi + \int_\Omega \nabla \cdot (A\nabla u)\phi + \int_\Omega b \cdot \nabla u \phi + cu\phi = \int_\Omega f\phi$$

Using partial integration over Ω and the fact that $\int_\Omega \phi A \nabla u \cdot n = 0$, the second term can be rewritten as

$$\int_\Omega A\nabla u \cdot \nabla \phi \,.$$

We therefore define:

Definition 4. *Weak solutions*
Let $u \in L^2([0,T]; W^1(\Omega))$ *and* $u' \in L^2([0,T]; W^{1*}(\Omega))$. *Then* u *is a weak solution of (1.9)-(1.11) if* u *solves*

$$< u', \phi > + B(u,\phi,t) = \int_\Omega f(t,.)\phi(t,.) \tag{1.12}$$

for almost every $t \in [0,T]$ *and every test function* $\phi \in H_0^1(\Omega) = \{ f | f \in H^1(\Omega)\,,\ supp(f)\ compact\ \}$ *and*

$$u(0,\cdot) = u_0(\cdot)\ on\ \Omega\,, \tag{1.13}$$

where

$$< u', \phi > = \int_\Omega u'(t,\cdot)\phi(t,\cdot) \text{ and}$$
$$B(u,\phi,t) = \int_\Omega A(t,\cdot)\nabla u(t,\cdot)\cdot\nabla\phi(t,\cdot) + \int_\Omega b(t,\cdot)\cdot\nabla u(t,\cdot)\phi(t,\cdot) + c(t,\cdot)u(t,\cdot)\phi(t,\cdot)\,.$$

Using this definition of weak solutions we now study results on the existence and uniqueness of weak solutions for linear parabolic equations.

Theorem 3. *Existence*
If $a_{i,j}, b_i, c \in L^\infty(\Omega)$ independent of t, $u_0 \in L^2(\Omega)$ and $\forall t \in [0,T]$, $f(t,\cdot) \in L^2(\Omega)$ then there exists a weak solution of the boundary value problem (1.9),(1.10),(1.11).

Proof.
Due to its complexity, just a small overview of the proof is given at this point. We refer the reader to [8] for the whole proof.
The general idea is, that we take a look at some finite dimensional subspace of $W_0^1(\Omega)$, defined as $B_n := span(\{w_i\}_{i=1}^n)$ where $\{w_i\}_{i=1}^\infty$is the orthonormal basis of a dense subspace of $W_0^1(\Omega)$ (for examples of $\{w_i\}_{i=1}^\infty$ see [8] as well). For the finite dimensional case on B_n the solution u_n can be represented by a linear combination of functions w_i.
If we also take our test functions from $\{w_i\}_{i=1}^n$, (1.12) becomes a system of Ordinary Differential Equations which is uniquely solvable according to the Picard-Lindelöff Theorem [15] §12.
It is then shown that for n = the dimension of B the sequences of solutions $\{u_n\} \subset L^2([0,T];W_0^1(\Omega))$ and their derivatives $\{u_n'\} \subset L^2([0,T];W^1(\Omega))^*$ are bounded for $n \to \infty$. Since these spaces are Hilbert spaces there exists a convergent subsequence.
Next it is ensured that the limit function u, whose derivative u' is the limit function of $\{u_n'\}_{n=1}^\infty$, is indeed a weak solution on the whole subspace with the basis $\{w_i\}_{i=1}^\infty$ where we chose the coefficients to be smooth functions (in t).
Since the space $span(\{w_i\}_{i=1}^\infty)$ is dense in $W_0^1(\Omega)$ for every $t \in [0,T]$ the statement follows. □

Following is a theorem which is needed to prove the uniqueness of found solution:

Theorem 4. *Gronwalls inequality*
Let f be a real-valued differentiable function and β a continuous function defined on [0,T]. If f satisfies the differential equation

$$f'(t) \le \beta(t)f(t) \text{ on } (0,T]$$

then

$$f(t) \le f(0)e^{\int_0^T \beta(t)} \text{ for all } t \in [0,T]\,.$$

Theorem 5. *Uniqueness*
The weak solution of (1.9),(1.10),(1.11) is unique.

Proof.
Again We only give a short sketch of the proof.
Assume that u and v are both solutions to the same Problem, then $u - v$ is obviously a solution for the same problem with $f \equiv 0$ and $(u - v)_0 \equiv 0$. We therefore have to show that only $u \equiv 0$ solves the Problem (1.9),(1.10),(1.11) with $f \equiv u_0 \equiv 0$.
Let $\phi = u$ on $[0, T] \times \Omega$ then we can exchange the integral and $\frac{\partial}{\partial t}$ in the first term of the equation (1.12) to obtain

$$\frac{\partial}{\partial t}(1/2||u||^2_{L^2(\Omega)}) = -B(u, u, t)\,,$$

where $B(u, u, t)$ is as in definition (4). We can estimate $-B(u, u, t)$ to be $\leq \gamma||u||^2_{L^2(\Omega)}$ for a constant γ, as is proven in [8].
At this point we use Gronwalls inequality with $f(t) := ||u(t, \cdot)||^2_{L^2(\Omega)}$ and $\beta(t) = \gamma$.
Since $||u_0||^2_{L^2(\Omega)} = 0$ this proves the statement. □

Results for the Perona-Malik Equation

At first sight one might believe that these results for linear parabolic equations can be directly applied to the Perona-Malik equation but this is not the case, for here $A(t, x) = g(\nabla u(t, x))$ which results in a non-linear equation. We can nevertheless show existence and uniqueness for a slightly modified version of the P-M equation.
The basis of the theorem on the existence is the Schauder-Tychonoff fixed point theorem.

Theorem 6.
Remark: (The Version of the Schauder fixed point theorem we need here has been proved by Tychonoff in 1935 [9] and deals with topological Banach spaces instead of normed spaces as is the case in most versions found in books. We will later apply this to the weak topology of our function space.)
Let W be a closed and convex subset of a locally convex topological vector space S and $E : W \mapsto W$ a continuous mapping such that the range $E(W)$ is contained in a compact set. Then E has a fixed point.

For proof see [9]

The next Theorem is taken from [7] where the question of the regularity of u is dealt with as well.

Theorem 7. *Existence and uniqueness of a weak solution for the P-M equation*
Let $g : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ *be a decreasing function with* $g(0) = 1$, $\lim_{t\to+\infty} g(t) = 0$ *and* $t \mapsto g(\sqrt{(}t))$ *is smooth. For* $\sigma > 0$ $G_\sigma(x) := \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{|x|^2}{4\sigma}}$. *Let now* $u_0 \in L^2(\Omega)$. *Then there exists a unique function* $u(t,x)$ *such that* $u \in C([0,T]; L^2(\Omega)) \bigcap L^2([0,T]; H^1(\Omega))$ *is a weak solution of*

$$\frac{\partial u}{\partial t} - div(g(|\nabla(G_\sigma * u)|)\nabla u) = 0 \quad on\ (0,T] \times \Omega \tag{1.14}$$

$$\frac{\partial u}{\partial n}\ \ on\ (0,T] \times \partial\Omega \tag{1.15}$$

$$u(0,\cdot) = u_0\,. \tag{1.16}$$

Proof.
We will first prove the existence of a solution.
Since the full proof is quite long, only the general outline of the one written down in [7] is given here.
First we need to define a Hilbert space

$$W(0,T) = \{w \in L^2(0,T; W^1(\Omega)) | \frac{\partial w}{\partial t} \in L^2(0,t; W^1(\Omega))^*\}$$

with the graph norm.
We define the system of equation $E(w)$

$$\frac{\partial u}{\partial t} - div(g(|\nabla(G_\sigma * w)|)\nabla u) = 0 \ \text{ on } (0,T] \times \Omega\}$$

$$\frac{\partial u}{\partial n}\ \text{on } (0,T] \times \partial\Omega$$

$$u(0,.) = u_0\,,$$

which is a linear parabolic equation for every fixed $w \in W(0,T) \bigcap L^\infty([0,T]; L^2(\Omega))$ and therefore has a unique solution (the equation is parabolic is parabolic since w is bounded and thus, $g(|\nabla(G_\sigma * w)|) > \epsilon > 0$).
In [1] it is shown, that

$$||u(t,\cdot)||_{L^p(\Omega)} \leq ||u_0||_{L^p(\Omega)}\ p \in [2,\infty]. \tag{1.17}$$

The proof also holds for our slightly modified version where u is the solution to the problem with $g(|\nabla(G_\sigma * w)|)$.
Let $U(w)$ be the solution for $E(w)$ it is thereby shown that

$$||U(w)||_{L^2([0,T];H^1(\Omega))} \leq C_1 \tag{1.18}$$

$$||U(w)||_{L^\infty([0,T];H^1(\Omega))} \leq ||u_o||_{L^2(\Omega)} \tag{1.19}$$

$$||\frac{\partial U(w)}{\partial t}||_{L^2([0,T];H^1(\Omega)^*)} \leq C_3\,, \tag{1.20}$$

where the constants C_1, C_3 are solely dependent on G, g and u_0
In the next step a subspace W_0 of $W(0,T)$ is introduced which contains all functions w that are bounded like our solution in equations (1.18),(1.19) and (1.20). The set W_0 is nonempty and convex and obviously W_0 is weakly compact in $W(0,T)$. Given that and the fact that $U(w)$ is a weakly continuous mapping $W_0 \mapsto W_0$ this allows us to use the Schauder fixed point theorem mentioned earlier.
This ensures that there is a function $u \in W_0$ such that u solves $U(u)$. This proves the existence of a solution to (1.14), (1.15),(1.16) .

To prove that the solution u is unique, we use a similar approach as we did in Theorem 5 for linear parabolic PDEs. Again only the general idea is given.
If u and v are both solutions to (1.14)-(1.16) we can easily deduce that u and v solve following equations:

$$\begin{aligned} \frac{\partial(u-v)(t,\cdot)}{\partial t} - div(g_u(\nabla u(t,\cdot) - \nabla v(t,\cdot)) - div((g_u - g_v)\nabla v(t,\cdot)) &= 0 \text{ on } (0,T]\,, \\ \frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial u - v}{\partial n} &= 0 \text{ on } (0,T] \times \partial\Omega\,, \\ u_0 - v_0 &= 0 \text{ on } \Omega \end{aligned}$$

for $g_u = g(|\nabla G_\sigma * u|)$ and $g_v = g(|\nabla G_\sigma * v|)$. Multiplying this equation with $(u-v)$ and integration over Ω leads us to

$$\underbrace{\frac{1}{2}\frac{\partial}{\partial t}||u-v||^2_{L^2(\Omega)} - \int_\Omega (u(t,\cdot) - v(t,\cdot))div(g_u(\nabla u(t,\cdot) - \nabla v(t,\cdot)))}_{*}$$
$$= \underbrace{\int_\Omega (u(t,\cdot) - v(t,\cdot))div((g_u - g_v)\nabla v(t,\cdot))}_{**}\,.$$

By using partial integration combined with the boundary conditions the second term on the left side can be transformed to

$$* = \frac{1}{2}\frac{\partial}{\partial t}||u-v||^2_{L^2(\Omega} + \underbrace{\int_\Omega (g_u(\nabla u(t,\cdot) - \nabla v(t,\cdot))^2)}_{\geq 0}$$

The right side is transformed in a similar fashion using in addition the Hölder inequality and can then be shown to be

$$** \leq C||u-v||^2_{L^2(\Omega)}||\nabla u||^2_{L^2(\Omega)}||\nabla u - \nabla v||^2_{L^2(\Omega)}$$

for a Constant C. Where the term $||u-v||^2_{L^2(\Omega)}$ stems from estimating $||g_u - g_v||_{L^\infty(\Omega)} \leq C_2||u-v||^2_{L^2(\Omega)}$ for some constant C_2.
This proves the uniqueness since the Gronwall's inequality (Theorem 4) is applicable with

$(u-v)(0,\cdot)=0$ and therefore $(u-v)=0$.
Remark: At this point the regularity of the soltion u is needed, since in (Theorem 4) $\beta = 2C||\nabla u||^2_{L^2(\Omega)}||\nabla u - \nabla v||^2_{L^2(\Omega)}$ has to be a regular function.

□

1.3 Numerical Implementation and tests of the PM-Method

1.3.1 Which Approach is Used?

Let $u \in \mathbb{R}^{m,n}$ be the (image) data, than $ug \in \mathbb{R}^{m,n}$ denotes a smoothed version of u.
Especially since Matlab is well adapted to matrix operations, a matrix based finite element approach is used here. The basic idea is presented in [2].
We need to discretize all the terms.
The operator *div* is discretized diagonally at cell corners

$$div(f_{i,j}) = \frac{f_{i+1/2,j+1/2} - f_{i-1/2,j-1/2}}{\sqrt{2}h} + \frac{f_{i-1/2,j+1/2} - f_{i+1/2,j-1/2}}{\sqrt{2}h}\,,$$

where h is the height and width of the cells. Since we need to compute $div(g(|\nabla u|)\nabla u)$ we discretize ∇u at (i+1/2,j+1/2). Accordingly we have obtained

$$\nabla u_{i+1/2,j+1/2} = [\frac{u_{i+1,i+1} - u_{i,j}}{\sqrt{2}h}, \frac{u_{i,j+1/2} - u_{i+1/2,j}}{\sqrt{2}h}]^t\,,$$

which leads to following discretization of the whole term:

$$\begin{aligned} 2h^2[div(g(|\nabla ug|)\nabla u]_{i,j} &= g(|\nabla ug_{i+1/2,j+1/2}|)[u_{i+1,j+1} - u_{i,j}] \\ -g(|\nabla ug_{i-1/2,j-1/2}|)[u_{i,j} - u_{i-1,j-1}] &+ g(|\nabla ug_{i-1/2,j+1/2}|)[u_{i-1,j+1} - u_{i,j}] \\ &- g(|\nabla ug_{i+1/2,j-1/2}|)[u_{i,j} - u_{i+1,j-1}] \end{aligned}$$

There are ways to discretize $div(\nabla u)$ with even fewer points but this leads to a less stable implementation and smoothing the signal (which is the disadvantage of using more points) is part of the algorithm anyway.
Furthermore we then want to find a matrix $A(|\nabla u_g|) \in \mathbb{R}^{m\cdot n, m\cdot n}$ such that:

$$2h^2[div(g(|\nabla ug|)\nabla u](:) = A(|\nabla u_g|) * u(:)\,,$$

where in the second equation the matrices $u_{i,j}$ and $div(...)_{i,j}$ are written as column vectors as in Matlab notation

$$u(:)_k = u_{k \bmod m, \lceil k/m \rceil},\ k \in \{1,2,...m\cdot n\}\backslash\{m,2m,3m,...,m\cdot n\} \tag{1.21}$$
$$u(:)_k = u_{m, \lceil k/m \rceil},\ k \in \{m,2m,3m,...,n\cdot m\} \tag{1.22}$$

The matrix A should also include the border-condition (1.15). In areas which are not at the borders of the image u, the value of $A_{r,s}$ is determined quite easily since our discretization of $2h^2[div(g(|\nabla ug|)\nabla u]_{i,j}$ can be written as:

$$-u_{i,j} \cdot [g(|\nabla ug_{i+1/2,j+1/2}|) + g(|\nabla ug_{i-1/2,j-1/2}|) + g(|\nabla ug_{i-1/2,j+1/2}|) + g(|\nabla ug_{i+1/2,j-1/2}|)]$$
$$+g(|\nabla ug_{i+1/2,j+1/2}|)u_{i+1,j+1} + g(|\nabla ug_{i-1/2,j-1/2}|)u_{i-1,j-1}$$
$$+g(|\nabla ug_{i-1/2,j+1/2}|)u_{i-1,j+1} + g(|\nabla ug_{i+1/2,j-1/2}|)u_{i+1,j-1}\,.$$

The border condition is implemented implicitly by treating values with out of bounds indexes as their reflections across the boundary (e.g. $u_{i,n+1} = u_{i,n-1}$).

To avoid unnecessary computation of $|\nabla ug|$, we define $D \in \mathbb{R}^{m+1,n+1}$ as

$$D_{i+1,j+1} = |\nabla ug|_{i-1/2,j-1/2}$$

and in regions outside the borders only the gradients parallel to the borders are considered (e.g. $D_{i,n+1} = |\nabla ug|_{i-1/2,n+1/2} = |\frac{ug_{i,n} - ug_{i-1,n}}{h}|$)

We are now able to define A with a relatively small computational costs.
Let A be a $m \cdot n \times m \cdot n$ matrix filled with zeros.

for k = 1 to $m \cdot n$
compute k according to (1.20),(1.21)

if we're not near the border, define the entries of the k-th row:
entry for u(i,j):

$$A(k,k) = -[g(|\nabla ug_{i+1/2,j+1/2}|)+g(|\nabla ug_{i-1/2,j-1/2}|)+g(|\nabla ug_{i-1/2,j+1/2}|)+g(|\nabla ug_{i+1/2,j-1/2}|)]$$

entry for u(i+1,j+1):

$$A(k, k+m+1) = g(D(i+1, j+1))$$

entry for u(i-1,j-1):

$$A(k, k-m-1) = g(D(i, j))$$

entry for u(i-1,j+1):

$$A(k, k+m-1) = g(D(i, j+1))$$

entry for u(i+1,j-1)

$$A(k, k-m+1) = g(D(i+1, j))$$

In case (i,j) lies on a border of the image, change the definitions above according to our reflection across the border.

In the next step we need to compute $u(t+\Delta t)$ for some given $u(t)$.
An explicit approach would be given by:

$$u(t+\Delta t) = u(t) + \Delta t \frac{\partial u(t)}{\partial t}\,,$$

which in our notation becomes:

$$u(t+\Delta t)(:) = u(t)(:) + \Delta t \cdot A(|\nabla ug|) * u(t)(:)\,.$$

In fact this approach is very simple and even stable for very small Δt.

The fully implicit system:

$$u(t+\Delta t)(:) = u(t)(:) + \Delta t \cdot A(|\nabla ug(t+\Delta t|) * u(t+\Delta t)(:)$$

takes far too long to realize. As proposed in most sources We therefore implement a semi-implicit approach which in principle looks as follows:

$$\begin{aligned} u(t+\Delta t)(:) &= u(t)(:) + \Delta t \cdot A(|\nabla ug(t|) * u(t+\Delta t)(:) \\ \Leftrightarrow (Id - \Delta t \cdot A(|\nabla ug(t)|)) * u(t+\Delta t) &= u(t) \\ \Leftrightarrow u(t+\Delta t) &= (Id - \Delta t \cdot A(|\nabla ug(t)|))^{-1} * u(t)\,. \end{aligned} \tag{1.23}$$

In each iteration we need to invert the matrix $I = (Id - \Delta t \cdot A(|\nabla ug(t)|)$ This is done quite easily since $(Id - \Delta t \cdot A(|\nabla ug(t)|)$ is diagonally dominant since

$$-A(k,k) = \sum_{i \neq k} |A(k,i)| \text{ , where A(k,k)} < 0$$

$$\Longrightarrow I(k,k) = 1 + \Delta t \cdot |A(k,k)| > \sum_{i \neq k} |A(k,i)| = \sum_{i \neq k} |I(k,i)|\,.$$

We can therefore use the CGNR algorithm as it is presented in [14]. This indeed proved to be a fast and stable way to invert I in praxis.
Depending on how the image is structured (dimension of the image, what edges shall be enhanced and how is the noise distributed) we now have to chose a flux function g and for each step a smoothing-kernel for ug.
Choosing the function
As a start we implemented the functions given in [2]. Since in our case this functions seemed to enhance edges with quite different gradients, we scaled these functions in order to compare them directly.

$$g1(s,\lambda) = \frac{5}{1 + 4 \cdot (s/\lambda)^2}\,,$$

$$g2(s,\lambda) = \frac{5}{(1 + (s/(2 \cdot \lambda))^2)^2}\,,$$

$$g3(s,\lambda) = e^{-0.12(s/\lambda)^2}.$$

These functions gíve very similar results, but function $g2$ results in the most suitable outcomes in our case.

Choosing and changing the kernel

Why do we have to change the kernel for each time-step?

We will take a look at the Gaussian kernel with standard deviation σ. In the beginning the noise may be very strong and σ is chosen to be large in order to get a small gradient of ug even near noise-peaks. After a few steps, (depending on step-width and flux-function g) most of the noise is reduced on planes but close to edges noise is still apparent since in ug the area of an edge is much bigger than the actual edge area in u. We therefore lessen σ in order to reduce noise even near edges.

1.3.2 What Problems May/Did Occur

- The main problem of the Perona-Malik diffusion is speed. The matrix A we introduced above has $m^2 \cdot n^2$ elements for an $m \times n$ image.
  Depending on the platform used, Matlab can only handle variables of a certain dimension, this boundary is exceeded quite easily. We therefore define our Matrix to be sparse which saves a lot of memory, for every row has a maximum of 5 non-zero values. The disadvantage of a sparse matrix in Matlab is that the speed is reduced significantly even if the data is prelocated.

- Even though we found a family of functions $g(\cdot,\lambda)$ which worked quite well for all of our data, the user still has to choose a λ that leads to satisfactory results (e.g. it may be necessary to change λ if a different kind of workpiece is analyzed). This lack of genericity is not easy to eliminate, judging only by the data the algorithm cannot know what edges are important to the user.

- Let's take a look at physics again to understand the next possible problem. In 1D there is only one way for material or heat to flow from one plateaus into a deeper lying one, across it's border. In 2D this is not the case. If there is a region where the edge of the plateau is not strong enough, material may flow from one plateau into the next even though most of the edges are high enough. Though in Theory quite apparent we were not able to produce this kind of error in a noticeable way.

- There is another major flaw: Sometimes the workpieces were not scanned at the right angle. The Bottom is tilted sideways in that case the Perona-Malik diffusion may result in two or more plateaus where originally we expected one tilted plane. This may result in additional edges and might influence the assessment of the workpiece's quality. See Figure (1.5), (1.6)

- Another problem stems from errors produced by the microscope. Whenever the rim of the cup reaches the image border additional noise is produced as in Figure (1.2).

After the Perona-Malik diffusion is applied, the rim then appears to be flattened (Figure (1.3) and (1.4)).

1.4 Appliance to our Data

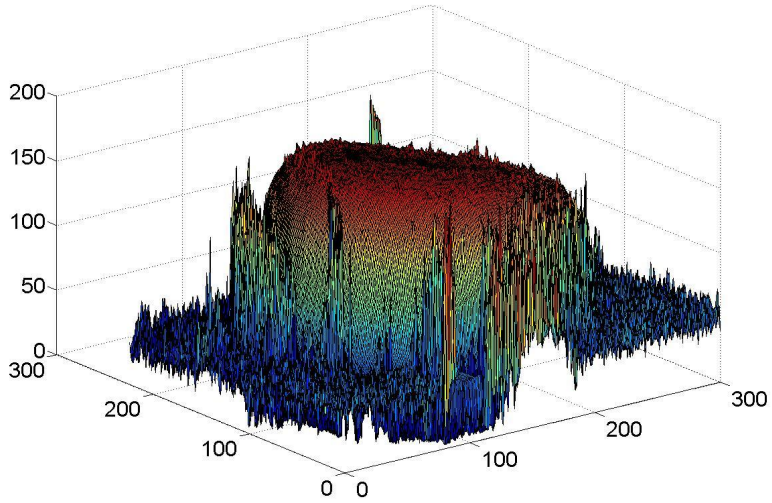

Figure 1.2: The plot of the original height data.

Application to sample data showed that the Perona-Malik equation indeed gives us the desired results when it comes to noise reduction and edge enhancement. On the other hand, it was a setback to see how long even a few iterations take. Further studies showed that it takes the majority (around 95%) of the time to define the matrix A from (1.22) and invert $(Id - A)$ in each step.
For a 154×205 image, the definition of A alone takes $\approx 0.3s$!.

1.4.1 Application to Workpiece Data

Best results where achieved using up to 40 iterations of the CGNR iterative method in each step but only around 8 steps of the Perona-Malik method itself since, even using adaptive step size control, the image hardly changes after the first iterations.

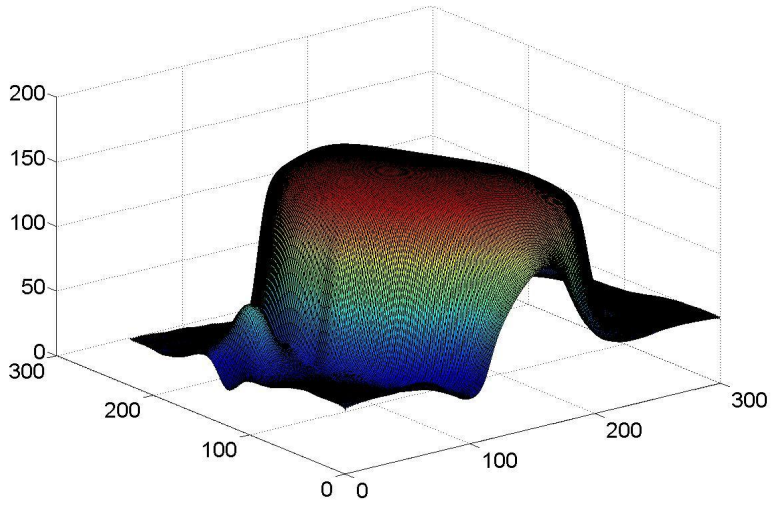

Figure 1.3: Data after 15 iterations

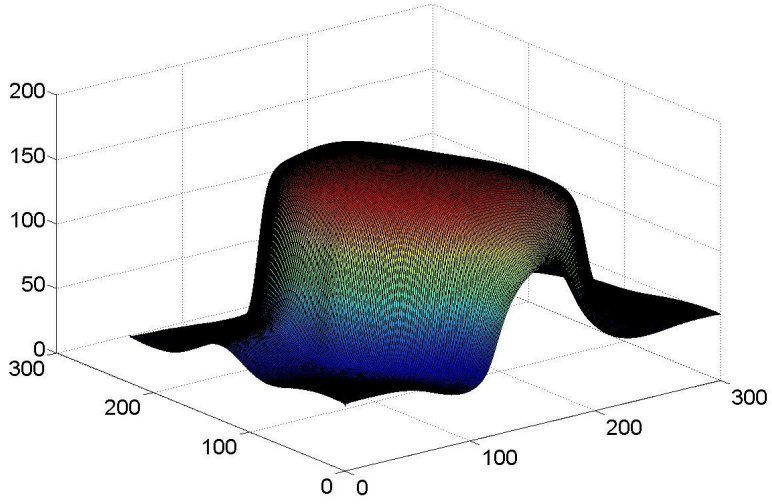

Figure 1.4: Data after 30 iterations.

1.4.2 Examples of the Problems that did Occur

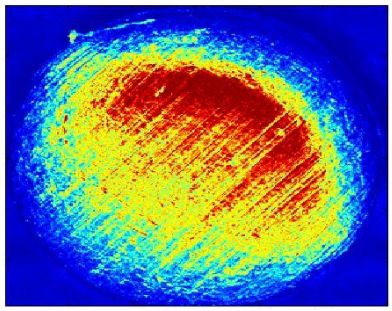

Figure 1.5: Image of the original workpiece. Though the surface is rough there is no strong edge on the inside.

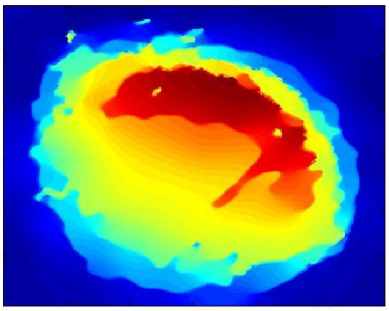

Figure 1.6: The height data after the P-M algorithm was applied. There are now discernible edges inside the workpiece. Note that we deliberately changed the parameters to produce this error.

Further tests and application to our workpiece data may be found in Appendix A.

Chapter 2

Characterization

So far we only enhanced the information on our workpiece contained in the image but we have yet to extract it. In this section we will present three algorithms devised to characterize certain properties of the cup.

2.1 What information are we looking for?

The main goal is to extract parameters and values representing the incidence of production errors in the current workpiece. There are lots of different defect classes that mostly have totally different causes. It would therefore be good to asses as many defect classes separately as possible.

2.2 Parameters of the workpiece

In our case we're looking for the ellipse which fits the outline of our workpiece best, this also gives us a value for the first defect class: "ellipticity".
There are many works available on ellipse-detection in images and like [10] every paper, uses a modified Hough transformation approach. This may be very useful if we have no knowledge of the nature of the ellipse, but in our case we have the advantage to be relatively certain that, after we applied the Perona-Malik algorithm, there are no significant edges outside the ellipse. Plus there may be a second ellipse inside the outer rim (as seems to be quite often the case in the images) a Hough Transformation-based approach is then likely to return the parameters of the wrong ellipse in some cases.
We therefore present a new algorithm which resembles the Snakes algorithm [11] insofar that we position a test ellipse around our suspected workpiece and in each iteration step try to find a better fitting test ellipse. As far as the author knows this has not yet been done in this context. As before, we will only look at the problem at hand which is finding an ellipse. Part of the algorithm may well be applied to a wider range of convex structures

though.

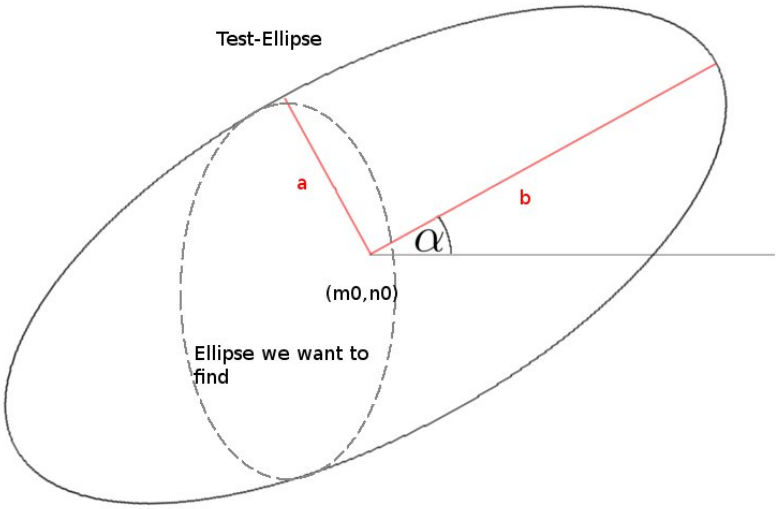

Figure 2.1: Sketch of the ellipse and the test ellipse outside of it. (m0,n0) is the center of the test ellipse, a and b denote the main axes and α describes how the ellipse is rotated relative to the x-axis.

The general outline of the algorithm:
1) load image u
2) set initial values: (m0,n0),a,b,α
(Only restriction to the initial values: The Test ellipse
defined by these parameters has to contain the ellipse we want to find.
while(We have only none, one or two points of contact.)
    3) Search the Test-ellipse for points of contact.
    4) Combine points of contacts if they are near to each other
    (we don't expect our ellipse to be perfectly regular).
    if (We have only one point of contact.)
        Move the center (m0,n0) accordingly.
    else if (We have two points of contact which are approximately opposite each other.)
        if(These points lie on one of the principal axis of our test-ellipse.)
            Reduce the other principal axis.
        else
            Rotate the test-ellipse accordingly by changing the angle α.
        end
    else

In this case we have reached the ellipse to a certain degree.
end end

This Algorithm is based on the fact, that two ellipses of which one lies in the other, cannot touch in more than two points. This algorithm proved to be quite stable and fast as long as there are only negligible edges outside the ellipse.
A combination with an eliminating particle swarming optimization as is done by the authors in [12] is neither fast nor did it help to increase the quality of the found parameters. In some cases the swarming method even led away from the wanted parameters. One reason for this surely is that we search for the outer ellipse and not for the strongest ellipse somewhere in the image.

2.2.1 Problems and Conditions Placed on the Image

This algorithm proved to be stable and fast as long as some points are ensured:

- There is an ellipse to be found in the image.
- The ellipse is inside the test ellipse defined by the initial values of the algorithm.
- There are no major edges outside the ellipse. This is quite important as, unlike Hough Transformation the algorithm is highly sensitive towards noise outside the ellipse.
- The ellipse is represented by enough edges in all directions. If for example only the upper half of the ellipse is discernible the algorithm is likely to fail.

2.3 Assessment of the Rim's Characteristic

Having found the ellipse which approximates the outline of our workpiece best, we now want to evaluate how the edge itself is structured. For this we transform the 2 dimensional outline of the workpiece into an one dimensional function $p(\phi)$ such that $p(\phi)$ is the distance of the actual edge to the optimal ellipse for $\phi \in [0, 360]$

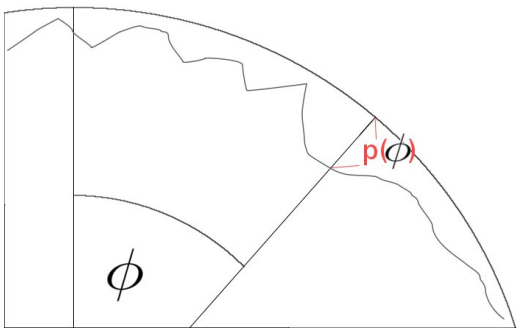

Figure 2.2: Transformation of the rim into a function.

To find this function, the algorithms looks for the outermost edge for every ϕ. If the distance is less than 30 pixels from found the ellipse, the distance and the associated angle ϕ is saved into two vectors. If the distance is greater than 30 pixels we may not be sure that what we are looking at is part of the outer rim and the distance and angle values are ignored.
The desired function $p(\phi)$ is then determined using smoothing an interpolation in case there are no values for some angles ϕ.
There are now at least two possible ways to assess the edge by means of analyzing p:

- Get the average value $\overline{p} = \frac{1}{360} \int_0^{360} p(\phi)\ \mathrm{d}\phi$ and calculate the deviation of p:

$$dev = \int_0^{360} |p(phi) - \overline{p}|\ \mathrm{d}\phi$$

  .
  The resulting *dev* is a value for all production errors of the rim finer than ellipticity. To get a more general value for the quality this is normalized by dividing by the number of edge-points found in the image. Otherwise, a workpiece where we were able to detect the whole rim would always be more likely to get a high ripple-value than a workpiece where we could only find half of the rim-points (this could for example be the case if not the whole workpiece is visible in the image).

- Fourier analysis of p gives information about how the rim is structured exactly. This might be the most useful approach in practice since ripples with higher frequencies mostly have totally different causes compared to ripples with low frequency.

2.3.1 Application to our Measurement Data

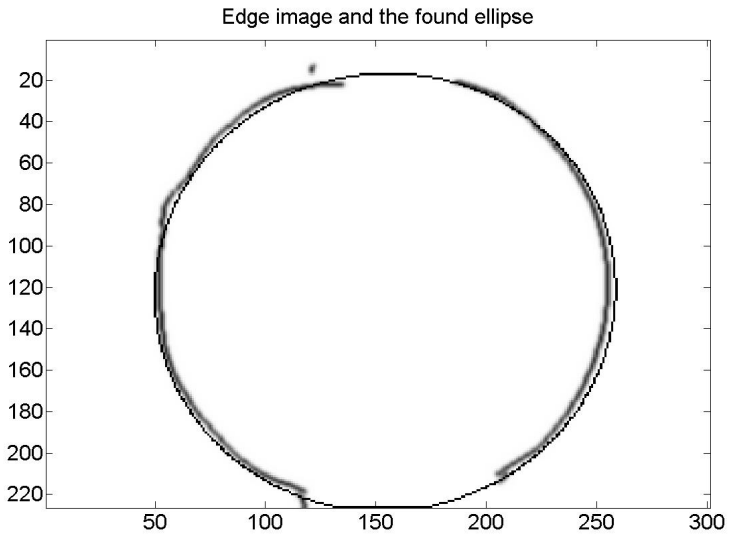

Figure 2.3: Results of our ellipse-finding algorithm when applied to the height data plotted in Figure 1.3

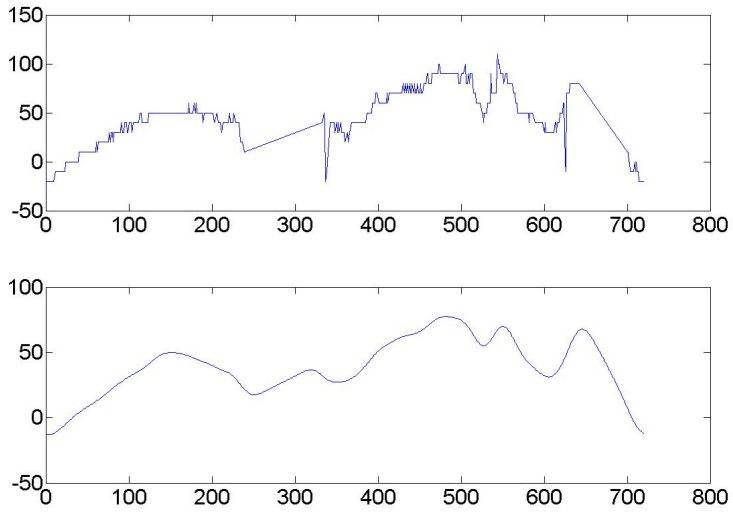

Figure 2.4: Plot of the workpiece's rim belonging to the image above. First plot is the raw data, second plot is a smoothed version thereof.

2.4 Evaluating the Quality of the Workpiece's Bottom

Following is a number of production-errors that may occur according to theory. We wrote down ideas how to extract most of them from the image-data but only implemented an algorithm to evaluate the form quality.
Possible production errors are:

- Cracks of various forms and lengths.
  A way to find cracks could be a Hough transformation of the edge image. Real cracks have an almost infinite gradient and their edge should be easily found using the Sobel algorithm with very high threshold.

- Stretcher strains (Lüder lines).
  We could not find them in the images containing height data. Photos of the workpiece could be helpful here. (The Keyene microscope also produces photographs of the workpiece with the same image dimension and workpiece position. The extracted workpiece parameters can therefore also be applied to the photos).

- Surface roughness.
  There is too much noise before, and to little roughness after the Perona-Malik algorithm is applied. Statements about the surface roughness will be very hard to make. Just as the Lüder Lines, they might be found in the photos of the workpiece.

- General errors in the form of the Bottom.
  An algorithm is presented in the following paragraph.

- Cracks in the rim
  An idea is described in section 2.5.

The form evaluating program should compare the height data of the cup with the shape of the desired cup and return a values that represents the deviation between these two.
Let $g(r)$ be a function which for every $0 \leq r \leq r_{MAX}$ describes the height difference between the center and the points, where r is the distance between the center and the point in the $(X-Y)$ plane in our ideal cup and r_{MAX} is the found ellipse radius. This program will only be run in case the workpiece is almost perfectly circular.
One problem that occurs in almost all cases is a tilt of the cup. As a first step the Algorithm therefore calculates c_1, c_2 and c_3 of the plane $c_1 * (i - m0) + c_2 * (j - n0) + c_3$ which has minimal square distance to the points of the workpiece's bottom($(m0, n0)$ is the center of the workpiece and i, j is the position in the image data).
Using this plane, we calculated the constant d such that $g(r) + d$ fits the cup best, compensating the tilt.
Comparison between $g(r) + d$ the height data indeed gives us a value representing the form quality of the bottom and to a degree the quality of the rim's curvature.

2.4.1 Application

In these two images we replaced part of the actual height-data with the found ideal data. The strong dependency on the correct choice of $g(r)$ becomes apparent in these two images.

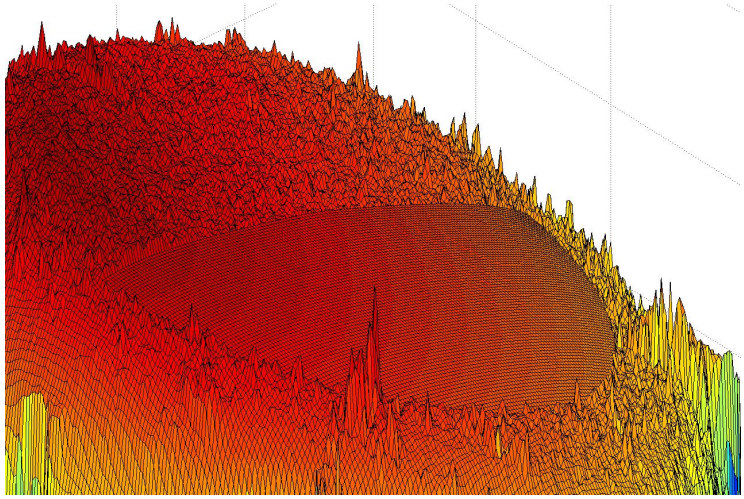

Figure 2.5: Here half of the bottom is replaced with the ideal form, using $g(r) = -\frac{100}{\sqrt{r_{MAX}-10-r}}$

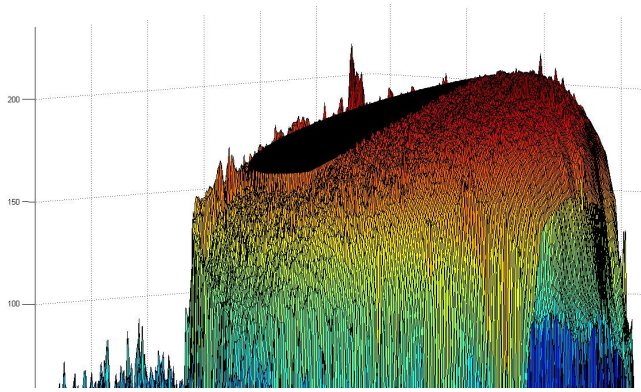

Figure 2.6: In this image the given g(r) equals zero

2.5 How to Find Cracks in the Rim

Besides ripples on the edge, we may also encounter cracks in the edge. Cracks cannot easily be found using the edge-function p so we came up with something different:
The general Idea is to use a Hough transformation detecting lines in the area near the optimal ellipse in *radius* and ϕ coordinates such that a line, whose direction is orthogonal to the ellipse, has slope zero. A crack may then be characterized as a high value in the accumulator with slope near zero. This should bring two advantages:
First of all, the restriction of the area we need to transform and the fact that the accumulator can be reduced to a few values for the slopes of our lines should shorten the time many times over.
Secondly, we do not have to worry about slope-values near ∞ or $-\infty$ which would be the case without our transformation into these polar coordinates.
this function is not yet fully implemented. Further Test in appendix A.

.

Appendix A

Further Tests

A.1 Test1

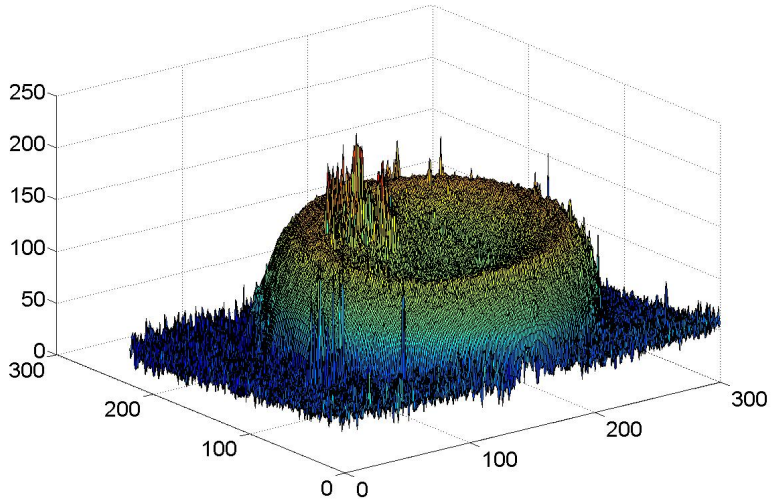

Figure A.1: Test 1: Original height Data

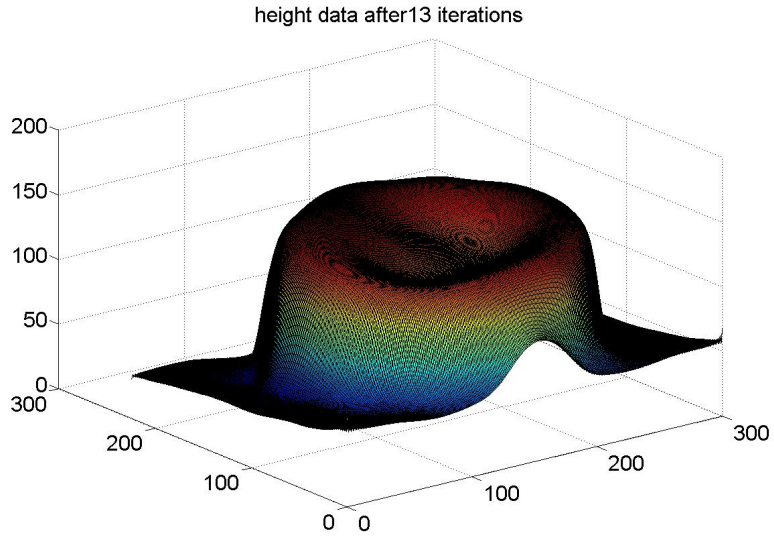

Figure A.2: Test 1: Data after the Perona-Malik diffusion

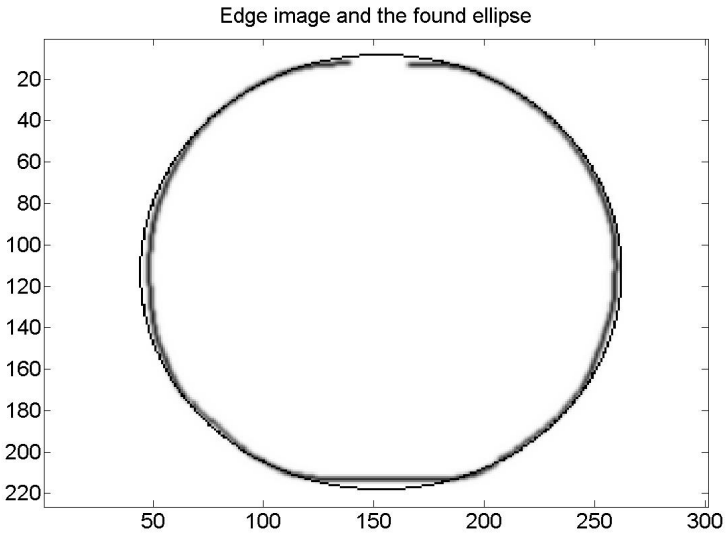

Figure A.3: Test 1: Found ellipse

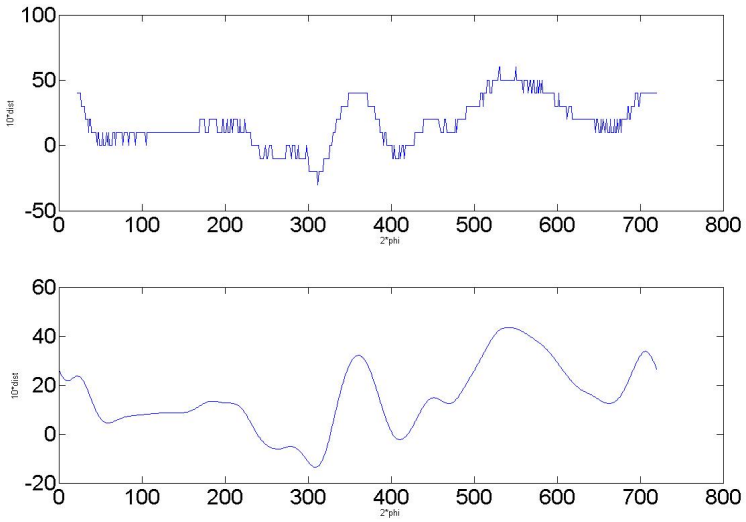

Figure A.4: Test 1: Plot of the rim

Deviation-value of the rim: 47

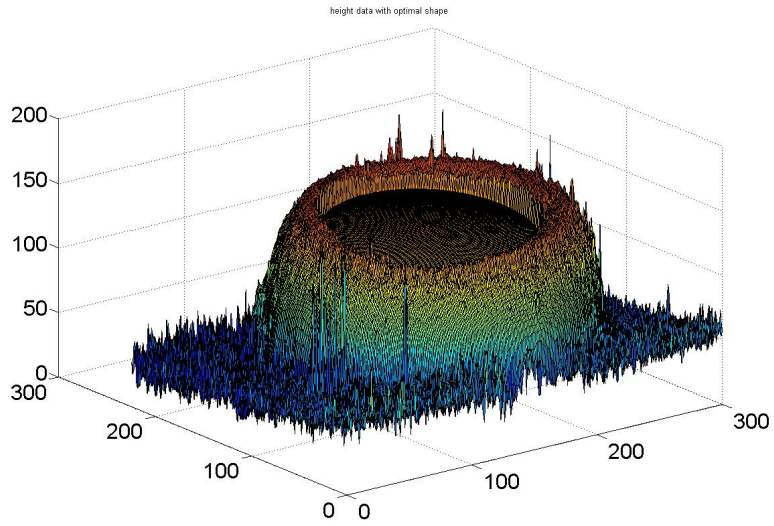

Figure A.5: Test 1: Approximation of the inner region

Deviation from the optimal shape: 13.5

A.2 Test2

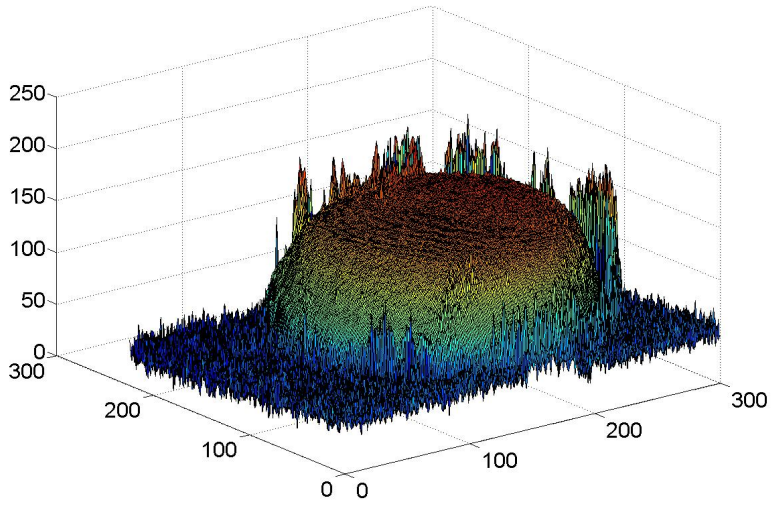

Figure A.6: Test 2: Original height Data

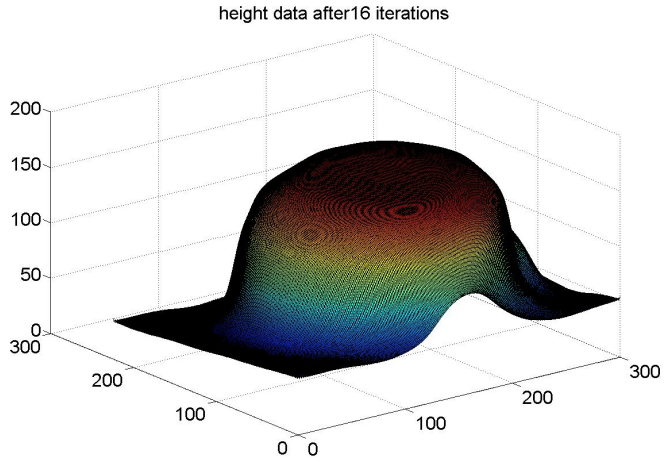

Figure A.7: Test 2: Height data after the Perona-Malik diffusion

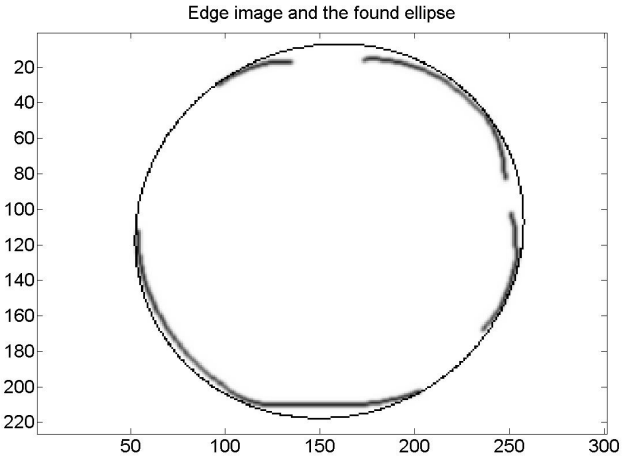

Figure A.8: Test 2: Found ellipse

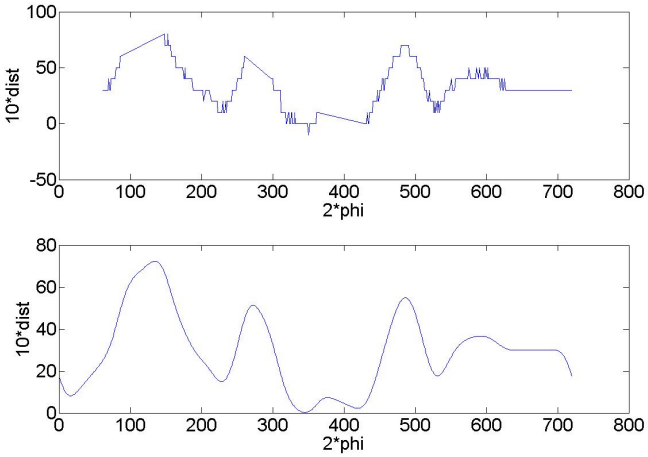

Figure A.9: Test 2: Plot of the rim

Deviation-value of the rim: 57

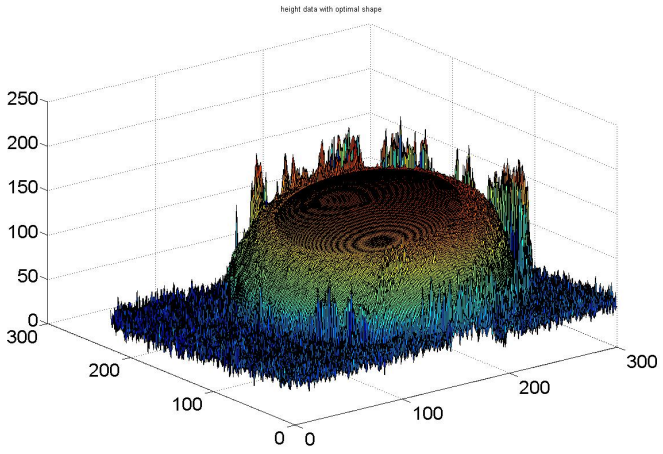

Figure A.10: Test 2: Approximation of the inner region

Deviation from the optimal shape: 2.3

Appendix B

Source Code

B.1 Source Code of the Perona-Malik Diffusion

```
function [ u0 ] = PeronaMalik(u0,maxiter,s,maxinv,gaussw)
%Input parameters:
%u0 the image,
%maxiter =maximum number of iterations
%s =parameter for flow function g is scaled.(higher s -> more smoothing
%maximum =iterations of the CGNR-Algorithm to invert the matrix
%gaussw = parameter to control the gaussian filter for ug

[m,n]=size(u0);
dist = 1;

%kernF is the gaussian kernel which changes with time (sigma decreases)
kernF = fspecial('gaussian',[m,n],1);

%We postulate that all images have the width of 1. h is the width of one
%cell in the image
h=1/n;

%D will contain the absolute value of the gradient of ug
D = zeros(m+1,n+1);

%ug later contains the smoothed image data
ug = zeros(m,n);
ug =real(ifftshift(ifft2(fft2(u0).*fft2(kernF))));

%This is the main loop, during which u_n+1 is computed
for iter=1:maxiter
%The gaussian kernel is changed depending on what loop ieration we're
    %in
    kernF = fspecial('gaussian',[m,n],n*gaussw/(100*(3+iter/30)^(1/10)));

    %We define D for this iteration  D(k,l) = sqrt(sqrt(2)*h)* |grad(k-1/2,l-1/2)|
    %= sqrt((ug(k,l)- ug(k-1,l-1))^2 + (ug(k-1,l)-ug(k,l-1))^2)
```

```
%The inner region of D:
for k=2:m
    for l=2:n
        D(k,l) = sqrt((ug(k,l)- ug(k-1,l-1))^2 + (ug(k-1,l)-ug(k,l-1))^2);
    end
end

%Borders of D:
for l=2:n
    D(1,l) = (2)^(1/2)*abs(ug(1,l)-ug(1,l-1));
    D(m+1,l) = (2)^(1/2)*abs(ug(m,l)-ug(m,l-1));
end
for k=2:m
    D(k,1) = (2)^(1/2)*abs(ug(k,1)-ug(k-1,1));
    D(k,n+1) = (2)^(1/2)*abs(ug(k,n)-ug(k-1,n));
end

%Scaling D:
D= 1/(sqrt(2)*h)*D;

%A is a matrix such that div(g(|nabla ug|)*nabla u) =
%A*u(:)
%We define (I-stepwidth*A);
%We need 2*3*(m+n) entries for the borders and 5*(m-2)*(n-2) entries for
%non-bordersections
l=(n-2)*(m-2)*5 + 2*3*(n+m-4); %number of non-zero entries in I and A
sw=zeros(1,l); %values
i=zeros(1,l); %position i
j=zeros(1,l); %position j
pe=1;
if dist==1
    stepw = (70)/(h^2);
else
    stepw = (9800)^2/((dist*h)^2);
end
for pz=1:m*n
    %pz is the row position in A
    %pi and pj are the positions in u0
    pi= mod(pz,m);
    if(pi==0)
        pi=m;
    end
    pj= ceil(pz/m);

    if(pi>1&&pj>1&&pi<m&&pj<n)
        %Entries on the diagonal
        sw(pe) =1+stepw*(g2(D(pi+1,pj+1),s)+g2(D(pi,pj),s)+...
        g2(D(pi,pj+1),s)+g2(D(pi+1,pj),s));
        i(pe) = pz;
        j(pe)= pz;
```

```
    %Entries for u(i+1,j+1):
    sw(pe+1) =-stepw*g2(D(pi+1,pj+1),s);
    i(pe+1) =pz;
    j(pe+1) = pz+1+m;
    %Entries for u(i-1,j-1)
    sw(pe+2) = -stepw*g2(D(pi,pj),s);
    i(pe+2) = pz;
    j(pe+2) = pz-m-1;
    %Entries for u(i-1,j+1)
    sw(pe+3) = -stepw*g2(D(pi,pj+1),s);
    i(pe+3) = pz;
    j(pe+3) = pz+m-1;
    %Entries for u(i+1,j-1)
    sw(pe+4) = -stepw*g2(D(pi+1,pj),s);
    i(pe+4) = pz;
    j(pe+4) = pz-m+1;
    pe = pe+5;

elseif(pi==1&&pj~=1&&pj~=n)
    %Entries on the diagonal:
    sw(pe) = 1+stepw*(g2(D(pi+1,pj+1),s)+g2(D(pi+1,pj),s));
    i(pe) = pz;
    j(pe) = pz;
    %Entries for u(i+1,j+1):
    sw(pe+1) =-stepw*g2(D(pi+1,pj+1),s);
    i(pe+1) = pz;
    j(pe+1) = pz+m+1;
    %Entries for u(i+1,j-1)
    sw(pe+2) = -stepw*g2(D(pi+1,pj),s);
    i(pe+2) = pz;
    j(pe+2) = pz-m+1;
    pe=pe+3;

elseif(pj==1&&pi~=1&&pi~=m)
    %Entries on the diagonal:
    sw(pe) =1+stepw*(g2(D(pi+1,pj+1),s)+g2(D(pi,pj+1),s));
    i(pe)= pz;
    j(pe)= pz;
    %Entries for u(i+1,j+1):
    sw(pe+1) =  -stepw*g2(D(pi+1,pj+1),s);
    i(pe+1) = pz;
    j(pe+1) = pz+m+1;
    %Entries for u(i-1,j+1)
    sw(pe+2) = -stepw*g2(D(pi,pj+1),s);
    i(pe+2) = pz;
    j(pe+2) = pz+m-1;
    pe = pe+3;

elseif(pi==m&&pj~=1&&pj~=n)
    %Entries on the diagonal:
    sw(pe) = 1+stepw*(g2(D(pi,pj),s)+g2(D(pi,pj+1),s));
```

```
        i(pe) = pz;
        j(pe) = pz;
        %Entries for u(i-1,j-1)
        sw(pe+1)= -stepw*g2(D(pi,pj),s);
        i(pe+1) = pz;
        j(pe+1) = pz-m-1;
        %Entries for u(i-1,j+1)
        sw(pe+2) = -stepw*g2(D(pi,pj+1),s);
        i(pe+2) = pz;
        j(pe+2) = pz+m-1;
        pe = pe+3;

    elseif(pj==n&&pi~=1&&pi~=m)
        %Entries on the diagonal:
        sw(pe) = 1+stepw*(g2(D(pi,pj),s)+g2(D(pi+1,pj),s));
        i(pe) = pz;
        j(pe) = pz;
        %Entries for u(i-1,j-1)
        sw(pe+1)=-stepw*g2(D(pi,pj),s);
        i(pe+1)=pz;
        j(pe+1)=pz-m-1;
        %Entries for u(i+1,j-1)
        sw(pe+2)=-stepw*g2(D(pi+1,pj),s);
        i(pe+2) = pz;
        j(pe+2) = pz-m+1;
        pe = pe+3;
    end
end

%Defining I
I = sparse(i,j,sw,m*n,m*n,l);

%In order to compute u_n+1 we have to invert (Id-A).
%We use the Conjugate Gradient on Normal Equations (CGNR),
%starting value is u_n (since u_n+1 should be near u_n).
%I is the matrix we need to invert. Step width is cntrolled by
%dist =||u_n - n_(n-1)||.

x=u0(:);
r = I'*(I*x-u0(:));
ra = r;
beta=0;
d=0;
a=0;
rel = norm(x);

for k=0:maxinv
    if norm(r)<(iter)^(1/2)*10^8*rel && k>5
        disp(k)
        break;
    else
```

```
            if k>0
                r2 = (r'*r);
                beta = r2/(ra'*ra);
                d=-r+beta*d;
                Id = I'*(I*d);
                a=r2/(Id'*d);
                x= x +a*d;
                ra=r;
                r = r + a*Id;
            else
                d=-r;
                r2 = (r'*r);
                Id = I'*(I*d);
                a=r2/(Id'*d);
                x= x +a*d;
                ra=r;
                r = r + a*Id;
            end
        end
    end

    %Do the step...
    dist = norm(u0(:)-x);
    u0(:) = x;

    %Get new smoothed image.
    ug =real(ifftshift(ifft2(fft2(u0).*fft2(kernF))));

    %Stop if u barely changed.
    if(dist<60)
        break;
    end
end
figure(2)
surf(u0);
title(['height data after' num2str(iter) ' iterations']);

end
```

The flux function:

```
function [ y ] = g2( s,l)
%y=5/(1+4*(s/l)^(2));
y=5/(1+1/4*(s/l)^(2))^2;
%y=exp(-0.12*(s/l)^2);
end
```

B.2 Source Code of the Parameter Search

```
function[m0,n0,a,b,dreh]= paraSearch(u0)
```

```
%The parameter u0 should contain a noise-free image u0 containing an
%ellipse.
%It then computes the center: (m0,n0), the main axes a and b and the
%rotaion: dreh of this ellipse.

%get the edges that are locally maximal and greater than the threshhold
ue = edge(u0,'sobel',2.9);
[m,n] = size(ue);
ue=double(ue);
kern = fspecial('gaussian',[9,9],1);
%the smoothing helps to increase the stability.
uef= [[ue(4:-1:1,4:-1:1);ue(1:m,1:4);ue(m:-1:m-4,4:-1:1)],...
[ue(4:-1:1,1:n);ue;ue(m:-1:m-4,1:n)],...
[ue(4:-1:1,n:-1:n-4);ue(1:m,n:-1:n-4);ue(m:-1:m-4,n:-1:n-4)]];
ue=conv2(uef,kern,'valid');
[m,n] = size(ue);
%The ellipse is: (x-xo)^2/a^2 +(y-yo)^2/b^2 = r^2 rotated by phi.

%initial values:
m0 =m/2;
n0=n/2;
a=m/2+30;
b=m/2+35;
dreh=0; %in radiant

%remeber the last center-points and rotations to avoid "shaking"
versch = false;
peakalt = 0;
peakalt2=0;
peakalt3=0;
peakalt4=0;
drehold =0;
drehold2=0;
gedreht=false;

for iter=1:120
    %Scan where our test-ellipse touches edges.
    beruhr=zeros(1,720);
    %If desired color the image at this point.
    ub = ones(m,n,3);
    ub(:,:,1) =ub(:,:,1)- 2*ue;
    ub(:,:,2) =ub(:,:,2)- 2*ue;
    ub(:,:,3)=ub(:,:,3)-2*ue;
    Dreh = [cos(dreh),-sin(dreh);sin(dreh),cos(dreh)];
    for phi=1:720
        rad=pi*phi/(360);
        y=a*cos(rad);
        x=b*sin(rad);
        p=Dreh*[x;y];
        if(round(m0-p(2))>0&&round(m0-p(2))<=m&&round(n0+p(1))>0&&round(n0+p(1))<=n&&...
                ue(round(m0-p(2)),round(n0+p(1)))>0)
```

```
        beruhr(phi)=ue(round(m0-p(2)),round(n0+p(1)));
    end
    ub(min(m,max(1,round(m0-p(2)))),min(n,max(1,round(n0+p(1)))),2) =0;
    ub(min(m,max(1,round(m0-p(2)))),min(n,max(1,round(n0+p(1)))),1) =0;
    ub(min(m,max(1,round(m0-p(2)))),min(n,max(1,round(n0+p(1)))),3) =0;
end
kern= fspecial('gaussian',[1,720],30);

%Stop if the ellipse already lies on a certain amout of edgepoints.
if sum(beruhr)>pi*sqrt(2*(a^2+b^2))/8
    break;
end
bg = conv([beruhr(359:-1:1),beruhr,beruhr(720:-1:361)],kern,'valid');
[pks,locs]=findpeaks([0,bg,0],'sortstr','descend',...
'minpeakdistance',5,'minpeakheight',0.02);

%Find peaks that are too near to each other.
l=numel(locs);
for i = 1:l
    for j=i+1:l
        if j>l
            break;
        end
        if abs(locs(i)-locs(j))>430
            locs = [locs(1:i-1),mod((locs(i)+locs(j)+720)/2,720),...
            locs(i+1:j-1),locs(j+1:l)];
            l=l-1;
            if j==l
                break;
            end
        elseif abs(locs(i)-locs(j))<290
            locs = [locs(1:i-1),(locs(i)+locs(j))/2,locs(i+1:j-1),locs(j+1:l)];
            l=l-1;
            if j==l
                break;
            end
        end
    end
end

if numel(pks>0)
    locs = locs-ones(1,numel(locs));
end

%Depending on where and how the test ellipse touches the rim of the cup
%we change the parameters of the test-ellipse.
if numel(locs) ==1
    %Compare poisitions between iterations to avoid repetitions.
    if versch>=1&&abs(abs(locs(1)-peakalt)-360) <30
        locs = [locs,peakalt];
        versch=0;
```

```
    elseif versch>=2&&abs(abs(locs(1)-peakalt2)-360) <60
        locs = [locs,peakalt2];
        versch=0;
    elseif versch>=3&&abs(abs(locs(1)-peakalt3)-360) <60
        locs = [locs,peakalt3];
        versch=0;
    elseif versch>=4&&abs(abs(locs(1)-peakalt4)-360) <60
        locs = [locs,peakalt4];
        versch=0;
    end
end

if(numel(locs)==2)
    peakalt = pks(1);
    %We have two maximums.
    if abs(abs(locs(1)-locs(2))-360)<70
        %They lie opposite each other.

        %We calculate the main axis:
        loc=mod((locs(1)+locs(2))/2 +180,360);
        if mod(loc,360) <40 || abs(mod(loc,360)-360)<40
            %Rotation ok, lessen b
            b=b-1;
            versch=0;
            gedreht=0;
        elseif abs(loc-180)<40 || abs(loc-540)<40
            %Rotation ok, lessen a
            a=a-1;
            versch=0;
            gedreht=0;
        else
            %Wrong rotation, change dreh.
            %To avoid repetion, we compare old and new rotations in
            %case we turned in the last iteration as well.
            if b>a
                dreh = dreh +(pi*(180-loc)/360);
                if abs(drehold2-dreh)<0.1 &&gedreht>1
                    dreh = (dreh+drehold)/2;
                end
                drehold2=drehold;
                drehold=dreh;
            elseif a>b
                dreh = dreh - (pi*loc/360);
                if abs(drehold2-dreh)<0.1&&gedreht>1
                    dreh = (dreh+drehold)/2;
                end
                drehold2=drehold;
                drehold=dreh;
            else
                dreh = dreh+ (pi*(loc)/360);
                if abs(drehold2-dreh)<0.1&&gedreht>1
```

```
                    disp([dreh,drehold,drehold2])
                    dreh = (dreh+drehold)/2;
                end
                drehold2=drehold;
                drehold=dreh;
            end
            gedreht=gedreht+1;
            versch=0;
        end
    else
        %The points of contact do not lie opposite each other,
        %move according to the combined maximum.
        versch=versch+1;
        gedreht=0;
        if(abs(locs(1)-locs(2))<360)
            kombmax = (locs(1)+locs(2))/2;
            peakalt4=peakalt3;
            peakalt3=peakalt2;
            peakalt2=peakalt;
            peakalt = kombmax;
        else
            kombmax = (locs(1)+locs(2)+720)/2;
        end
        v1=cos(pi*kombmax/360); %m-Wert
        v2=sin(pi*kombmax/360); %n-Wert
        v=Dreh'*[v1;v2];
        m0 =m0 - v(1);
        n0 =n0 + v(2);
    end
elseif(numel(locs)==1)
    %Move away fromt he maximum.
    v1=cos(pi*locs(1)/360); %m-value
    v2=sin(pi*locs(1)/360); %n-value
    v=Dreh'*[v1;v2];
    m0 =m0 - v(1);
    n0 =n0 + v(2);

    peakalt4=peakalt3;
    peakalt3=peakalt2;
    peakalt2=peakalt;
    peakalt = locs(1);
    versch = versch+1;
    gedreht=0;
elseif numel(locs)==0
    %No peaks, we therefore lessen both a and b.
    a=a-1;
    b=b-1;
    gedreht=0;
    versch=0;
else
    %Too many points of contact.
```

```
        break;
    end

    if mod(iter,4)==0
    figure(3)
    imagesc(ub);
    title('Current edge image and the found ellipse');
    end

end
a=round(a);
b=round(b);
figure(3)
imagesc(ub);
colormap gray;
title('Edge image and the found ellipse');
end
```

B.3 Source Code of the Edge Characterization

```
function [ wert1 ] = rimcharact(u0,m0,n0,a,b,dreh )
%This function shall characterize the workpiece's rim.
%This includes first a value for the overall ripples on the edge
%and may also contain a frequancy analysis of the edge.

[m,n]=size(u0);
r = ceil(max(a,b)); %The maximum radius. We assume the workpiece to be round

%Since we now the position and size of our workpiece, we may look at a
%smaller region.
uk=u0(max(1,round(m0-r)):min(m,round(m0+r)),max(1,round(n0-r)):min(n,round(n0+r)));
m0 = r-min(0,m0-r);
n0 = r-min(0,n0-r);
[m,n]=size(uk);

%The ripples of the edge are characterized by p(phi)=distance between outer
%rim and found ellipse at angle phi.

%We're again working with the edge-image
uke= edge(uk,'sobel',4);

%transformation-matrix
Dreh = [cos(dreh),-sin(dreh);sin(dreh),cos(dreh)];
diff=35;
erst=true;
for phi=1:719
    %get position in image for phi
    rad=pi*phi/(360);
    y=cos(rad);
    x=sin(rad);
```

```
    radius=sqrt((a*y)^2 + (b*x)^2);
    p=Dreh*[x;y];

    %find the distance between rim and ellipse
    for k=radius+3:-1:radius-35
        if 0<round(m0-k*p(2))&&m>=round(m0-k*p(2))&&...
        0<round(n0+k*p(1))&&n>=round(n0+k*p(1))
            if uke(round(m0-k*p(2)),round(n0+k*p(1))) >0
                if ~erst
                phiwerte = [phiwerte,phi];
                radwerte = [radwerte,10*(radius-k)];
                uke(round(m0-k*p(2)),round(n0+k*p(1)))=...
                uke(round(m0-k*p(2)),round(n0+k*p(1)))+30;
                break;
                else
                    phiwerte = phi;
                    radwerte = 10*(radius-k);
                    erst=false;
                end
            end
        end
    end
end

if numel(phiwerte)<2
   disp('keinen Rand gefunden!')
   return;
end


[l1,l2]=size(phiwerte);
%We need a value for phi=720 for later computations.
phiwerte = [phiwerte,720];
radwerte=[radwerte,radwerte(1,l2)+(720-phiwerte(1,l2))*...
(radwerte(1,1)-radwerte(1,l2))/(phiwerte(1,1)+720-phiwerte(1,l2))];

%So far phiwerte and radwerte only contain values in case we found an
%edge. This may not be the case for all phi. This is a problem in
%ferquency analysis and plotting...
%The function plottable deals with this.
[phiwerte2,radwerte2]= plottable(phiwerte,radwerte);

%Smooth the signal.
kern= fspecial('gaussian',[1,50],10);
%convolution with periodic continuation of the signal
radwg = conv([radwerte2(696:720),radwerte2,radwerte2(1:24)],kern,'valid');

%Show "the edge"
figure(4)
 subplot(2,1,1);
 plot(phiwerte,radwerte);
```

```
 xlabel('2*phi');
 ylabel('10*dist');
   % set(gca,'FontSize',24);
subplot(2,1,2);
 plot(phiwerte2,radwg);
  %title('plot of the found edge points and a smoothed and optimized version');
   xlabel('2*phi');
 ylabel('10*dist');
  %set(gca,'FontSize',24);

%Get the value for the overall rippleness. this is just a way to receive a simple value.
%in later implementations this schould probably be changed to frequency analysis
wert1=0;
p= polyfit(phiwerte2,radwerte2,2);
pw=polyval(p,phiwerte2);
for k=1:720
wert1=wert1 +abs(radwerte2(k)-pw(k));
end
wert1=wert1/n;
end
```

B.4 Smoothing the Edge Data

```
function [ phiwerte2, radwerte2 ] = plottable( phiwerte, radwerte )
% We want to achieve several things:
%a value for every values of phi
%a certain degree of smoothnes in the data
%optional: points with no near neightbours shall not influence our overall picture
%too much.

[m,length]=size(phiwerte);
radwertesm = zeros(1,length);
if length>1
    radwertesm(1)=radwerte(1);
    radwertesm(length)=radwerte(length);
end

%partly ignore values that are "alone" those values may otherwise give
%falsify our characterization
%{
for k = 3:length-2
    radwerte(k) = radwerte(k)*sqrt(6)/sqrt(phiwerte(k+1)+...
    phiwerte(k+2)-phiwerte(k-1)-phiwerte(k-2));
end
%}

%smooth the values
for k=2:length-1
    if abs(phiwerte(k-1)-phiwerte(k))<=4&&abs(phiwerte(k+1)-phiwerte(k))<=4
        radwertesm(k) = 2/3*radwerte(k)+1/6*(radwerte(k-1)+1/6*radwerte(k+1));
```

```
    elseif abs(phiwerte(k-1)-phiwerte(k))<=4
        radwertesm(k)=3/4*radwerte(k)+1/4*radwerte(k-1);
    elseif abs(phiwerte(k+1)-phiwerte(k))<=4
        radwertesm(k) = 3/4*radwerte(k)+1/4*radwerte(k+1);
    else
        radwertesm(k)=radwerte(k);
    end
end

%We interpolate the values
phiwerte2 = 1:720;
radwerte2 = zeros(1,720);

%position inside our phi-vector
pos = 2;
%Last-position-value;
lpv=0;
%Last position
lp =0;
for i = 1:720
    if i<phiwerte(pos)
        radwerte2(i) = lpv+(radwertesm(pos)-lpv)*(i-lp)/(phiwerte(pos)-lp);
    elseif i== phiwerte(pos);
        radwerte2(i)=radwertesm(pos);
        lpv = radwertesm(pos);
        lp = phiwerte(pos);
        pos=pos+1;
    end
end
end
```

B.5 Source Code of the Form Assessment

```
function [ dev ] = ingeo(u0,m0,n0,a,b,dreh  )
%We're trying to get a value representing the geometric quality of the cup.
[m,n]=size(u0);
%step 0: get the tilt out of the cup by finding the approximating plane:
%aa(i-mo)+ba(j-n0)+ca = u(i-mo,j-no)
a2 = max(a,b);
b2 = min(a,b);
e = sqrt(a2^2-b2^2);
f1= [m0 - e*sin(dreh);n0+e*cos(dreh)];
f2= [m0 + e*sin(dreh);n0-e*cos(dreh)];
pos=1;
A=zeros(ceil(pi*a*b)/9,3);
f=zeros(ceil(pi*a*b)/9,1);
for i=1:3:m
    for j=1:3:n
        if(norm([i-f1(1),j-f1(2)])+norm([i-f2(1),j-f2(2)])<2*(a2-20))
                A(pos,:)=[i-m0,j-n0,1];
```

```
                %smoothing of the image to save time
                f(pos,1)=1/5*(u0(i,j)+u0(i+1,j+1)+u0(i-1,j-1)+u0(i+1,j-1)+u0(i-1,j+1));
                pos=pos+1;
        end
    end
end
[Q,R]=qr(A);
f=Q'*f;
ca = f(3)/R(3,3);
ba = (f(2)-R(2,3)*ca)/R(2,2);
aa = (f(1)-R(1,3)*ca-R(1,2)*ba)/R(1,1);
rcirc= round( (a+b)/2)-35;
radi=round((a+b)/2);

%To reduce computation-time we assume the workpiece to be round.
%elliptical workpieces would not need to be characterized at this point.
dist=0;
nvalues = 0;
for i=round(min(1,m0-rcirc)):round(max(m0+rcirc,m))
 for j=round(n0-sqrt(rcirc^2-(i-m0)^2)):round(n0+sqrt(rcirc^2-(i-m0)^2))
         dist = dist +(u0(i,j)-aa*(i-m0)-ba*(j-n0)-geofunc(norm([i-m0,j-n0]),radi));
         nvalues = nvalues+1;
 end
end

d=dist/(nvalues);
dev=0;
for i=round(min(1,m0-rcirc)):round(max(m0+rcirc,m))
 for j=round(n0-sqrt(rcirc^2-(i-m0)^2)):round(n0+sqrt(rcirc^2-(i-m0)^2))-1
     temp= (u0(i,j)-aa*(i-m0)-ba*(j-n0)-geofunc(norm([i-m0,j-n0]),radi)-d);
     dev = dev+abs(temp);
     u0(i,j)=aa*(i-m0)+ba*(j-n0)+geofunc(norm([i-m0,j-n0]),radi)+d;
 end
end
 figure(5)
 surf(u0)
 title('height data with optimal shape');
 dev= dev/(a*b);
end
```

Bibliography

[1] Kristian Bredies and Dirk Lorenz. Mathematische Bildverarbeitung: Einführung in Grundlagen und moderne Theorie. Vieweg+Teubner, 2011.

[2] L.Keeling and Rudolf Stollberg. Nonlinear anisotropic diffusion filterung for multiscale edge enhancement. Inverse Problems, 18:175-190, 2002

[3] Pietro Perona and Jitendra Malik. Sclae-space and edge detection using anisotropic diffusion. IEEE Trans. Pattern Anal. Machine Intell., 12:629-639, 1990.

[4] Wieland Richter. Partielle Differentialgleichungen, Einführung in Theorie und Praxis mit Fortran Programmen. Spektrum Akademischer Verlag GmbH Heidelberg Berlin Oxford 1995.

[5] Hans Wilhelm Alt. Lineare Funktionalanalysis, 5., überarbeitete Auflage. Springer-Verlag Berlin Heidelberg 2006.

[6] Dirk Werner. Funktionalanalysis. Springer-Verlag, 6., korrigierte Auflage. Springer-Verlag Berlin Heidelberg 2007.

[7] Francine Catté, Pierre-Louis Lions, Jean Michel Morel and Tomeu Coll. Image Selective Smoothing and Edge Detection by Nonlinear Diffusion. SIAM Journal on Numerical Analysis, Vol 29,No. 1.(Feb., 1992),pp. 182-193.

[8] Lawrence C. Evans, Partial Differential Equations. Graduate Studies in Mathematics Volume 19. American Mathematical Society Providence, Rhode Island 1998

[9] Tychonoff A.: Ein Fixpunktsatz. Math. Ann. 111, 767-776 (1935)

[10] Yonghong Xie and Qiang Ji. A New Efficient Ellipse Detection Method. In International Conference on Pattern Recognition 2002

[11] Michael Kass, Andrew Witkin and Demetri Terzopoulos. Snakes: Active Contour Models. International Journal of Computer Vision 1998

[12] Yanhui Guo, H.D. Cheng1, Wei Zhao1 and Yingtao Zhang1. A Novel Hough Transform Based on Eliminating Particle Swarm Optimization and its Applications. of the 11th Joint Conference on Information Sciences (2008)

[13] http://www.keyence.de/

[14] Robert Plato. Numerische Mathematik kompakt. 4. Auflage. Vieweg+Teubner Wiesbaden 2010.

[15] Otto Forster. Analysis 2 Differentialgleichungen im $\mathbb{R}^n$. Vieweg + Teubner 2008

Declaration

I guarantee that I personally wrote this thesis and used only the aforementioned resources. The thesis has not been in any form published or submitted to another examination office.

Bremen, July, 28, 2011

_____

John Wilfried Schlasche