Using line features for 3D face registration

Fabian Brix

18.06.2013

Abstract

In this bachelor thesis we attempt to modify the existing face registration pipeline for the morphable face model of Prof. Thomas Vetter by using a registration algorithm developed by PD Marcel Lthi at the University of Basel. ALTERNATIVE: In this bachelor thesis we discuss the construction of a face registration pipeline using an algorithm based on a vector-valued gaussian process and at the same time attempting to ensure registration quality through the use of contours marking important parts of the face - referred to as line features.

The algorithm is capable of mapping any two shapes on to one another. All that is needed is a set of corresponding points on the two shapes. Different constraints to the displacement field can be applied through regularisation.

The aim of this bachelor thesis is more specifically to apply this general algorithm for point correspondences to scanned face data, that is to implement feasible registration of face scans onto the mean face of the morphable model. In order to achieve this we mark important parts of the face meshs not only with point landmarks, but also structures and organs (eyebrows, eyes, ears) with lines - line features - and thereby to create further correspondences for the algorithm to perform better by. Instead of using sparse points of key features points of the face we mark complex features, e.g. the eyes, with contour lines - line features in order to create further correspondences

These line features are marked by hand using bzier curves on three 2D images to the front, left and right of the 3D face. In order to utilize them, however, they have to be projected on to the computed mesh of the face that was recorded by a 3D scanner. These meshs have holes in the region of the eyes and the ears rendering the projected line features useless at first. This thesis first gives an overview over the morphable model and the face registration pipeline, then goes on to obtaining 3D points from the 2D line features, to explain the theory behind the general algorithm and in the main part discusses the problems and solutions we encountered trying to optimize the algorithm for and without line features for the face registration process.

Contents

Chapter 1

Introduction

1.1 Problem Statement

So es bizzeli alles schriebe 1. Use Gaussian Processes - 2. Use Line Features =¿ prepare for Gaussian Process Regression In this bachelor thesis Implement 3D face registration using Gaussian Processes and Line Features. One part of the problem is to sample equidistant 3D points from 2D line features marked on images of a 3D face scan. These line features should then be used as an additional input to a registration algorithm which is based on Gaussian Process Regression. The aim is to build a pipeline which starts off with the raw scan data as well as the landmarks and line features. The feature points are used to register the mean face of the MM/BFM (Basel Face Model) on to/with the raw scan thereby obtaining a fully defined and textured 3D model representation of the face in 3D. Registration is the technique of aligning to objects using a transformation, in this case the registration is performed by adding displacements to every points in the mean face model. A model is represented as vector N*d. What is a model? A vector representation of a 3D scan? For the morphing a Posterior Shape Model is used in combination with a Gaussian Process. Image registration is a process of aligning two images into a common coordinate system thus aligning.

(gaussian process + line features for accurate, reproducable registration)

1.2 Review Literature

2. Definition of terms (morphable model, 3D face registration, Gaussian Process regression, posterior shape models) 3. Review of literature (papers)

Chapter 2

Model building

2.1 Building a model from registered face data

Model building is our motivation. We are going to great lengths to build a model In this chapter we explain how to go about building a 3D face model from 3D face scans. 2.3 Building a Model from the registered data (short) cite morphable model, give a short overview how a model is built A set of faces parametrized by coefficients a, set of Textures parametrized by coefficients b Fit multivariate normal distribution to data set, based on average of faces and textures. Build covariance matrices over differences between the mean and face samples in surface and texture. =¿ two distributions. Perform PCA to get orthogonal basis system In the MM three subspaces are morphed independently

Eine Gruppe (G,) heisst abelsch [abelian] oder kommutativ wenn ab = ba gilt fr alle a,b G.

2.2 Prerequisite Data

image with landmarks and line features a short overview what data we have given

Facial Scans: face scans given as point clouds The data we have given is a set of about 300 face scans that have had a set of key points marked. Furthermore important and detailed regions like the eyes, ears and lips have been marked by contour lines known as line features. The scans have been obtained with a scanner. The surface is very detailed, however the eyes and the nostrils are not recorded. From these scans we want to create fully textured 3D faces, which can be used to build a new face model.

Mean Face: The mean face has been derived from a collection of 100 male and 100 female 3D face models.

2.3 Finding Correspondences

WE WANT POINT TO POINT CORRESPONDENCE BETWEEN THE TWO FACES in general: point to point correspondence between to images Are scans already in semantical correspondence? No semantical correspondence FINDING CORRESPONDENCE IS EXACTLY THE AIM OF REG-

ISTRATION =¿ HAVING SAME POINTS AS CLOSE TO ONE ANOTHER AS POSSIBLE Now in order to obtain a 3D representations of the face we need to transform the mean face so that it fits a particular 3D face scan. To find the transformation, however, we first have to find feature points in both 3D representations which correspond to the same semantical structure. Previous work has shown that point landmarks are not sufficient to preserve the level of detail which is imminent in the regions of the eyes, ears and lips and that the computed transformations are not able to preserve these regions. For this reason, additional line features have been introduced. In order to relate these

How registration works so far

What we want to change

Chapter 3

Gaussian Processes in 3D Face Registration

As described briefly in the introduction, the first of our two objectives is to build a face registration pipeline. In achieving this we use an algorithm which can handle arbitrary shapes for the registration of the 3D faces. It is derived from a stochastic process, more specifically a vector-valued Gaussian process or Gaussian random field. In this chapter we deal with the theory necessary for understanding the functionality of the registration pipeline. To begin with, we recapitulate the definition of stochastic processes and extend it to the definition of Gaussian processes. In the next step, we then delve into Gaussian process regression and finish by applying a vector-valued Gaussian process to our problem of 3D face mesh registration.

3.1 Stochastic Processes

In probability theory a stochastic process consists of a collection of random variables $\{X(t)\}_{t\in\Omega}$ where Ω is an index set. It is used to model the change of a random value over time. The underlying parameter time is either real or integer valued. In our case, however, we use a collection of vector-valued random variables with indices in the $\mathbb{R}^3$ space because we want to model deformations of the vectors on the faces' surfaces. This generalization of a stochastic process - which can handle multidimensional vectors - is called a random field. Defining the random variables on an index set in an n-dimensional space, allows for spatial correlation of the resulting values, which is an important aspect of the algorithm discussed later on.

3.2 Gaussian Processes

A Gaussian process is a stochastic process in which each finite collection $\Omega_0 \subset \Omega$ of random variables has a joint normal distribution. More formally, we define the collection of random variables $\{X(t)\}_{t\in\Omega}$ to have a d-dimensional normal distribution if the collection $\{X(t)\}_{t\in\Omega_0}$ - for any finite subset Ω_0 - has a joint $d \times |\Omega_0|$-dimensional normal distribution with mean $\mu(\Omega_0)$ and covariance $\Sigma(\Omega_0)$. If $\Omega \not\subseteq \mathbb{R}$ holds, the process is a Gaussian random field, which holds true for our case, because we use an index set $\Omega \subseteq \mathbb{R}^3$. In the further proceedings the term “Vector-valued Gaussian Processes” will be used to refer to Gaussian random field” will be used to refer to Gaussian random fields.

An alternative way of viewing a Gaussian process is to consider it as a distribution over functions. This allows us to look for inference in the space of these functions given a dataset, specifically to find the deformation function given a 3D face mesh. Each random variable now yields the value of a function $f(x)$ at a location $x \in \mathcal{X}$ in the index set of possible inputs. We now denote the index set by $\mathcal{X}$ to stress that we are ceasing to discuss Gaussian processes defined over time. In this function-space view a Gaussian Process at location x is thus $f(x) \sim GP(\mu(x), k(x, x'))$ defined by its mean $\mu : \mathcal{X} \to \mathbb{R}$ and covariance $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ functions which in turn are defined over the set of input vectors. With $\mu(\mathcal{X}) = (\mu(x))_{x \in \mathcal{X}}$ and $\Sigma(\mathcal{X}) = (k(x, x'))_{x, x' \in \mathcal{X}}$ we obtain the full distribution of the process $GP(\mu(\mathcal{X}), \Sigma(\mathcal{X}))$. For the purpose of simplifying calculations we may assume that every random variable has zero mean without a loss of generality. When modeling a deformation field with a Gaussian process this circumstance implies that the expected deformation is itself zero.

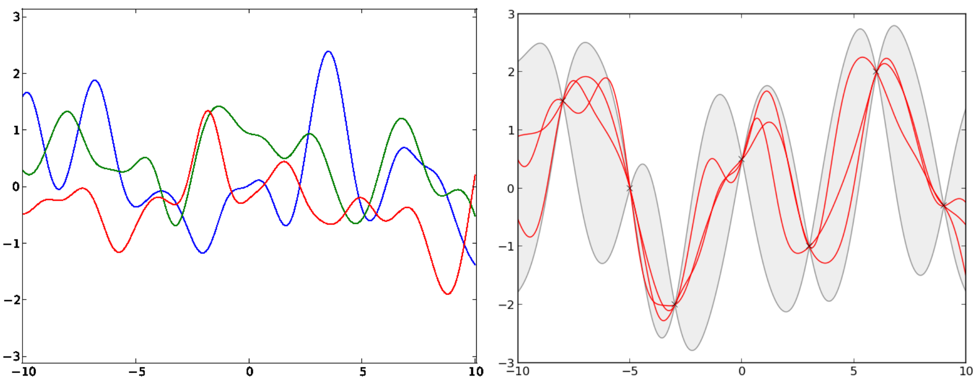
Covariance Functions The key feature of a Gaussian Process is its covariance function also known as “kernel”. It specifies the covariance $\mathbb{E}[f(x)f(x')]$ between pairs of random variables for two input vectors x and x', allowing us to make assumptions about the input space by defining the spatial codependency of the modelled random variables. Note that when assuming zero mean we can completely define the process' behaviour with the covariance function.
A simple example of a covariance function is the squared exponential covariance function, defined by $cov(f(x), f(x')) = k(x, x') = \exp(-\frac{(x-x')}{2l^2})$. (derivation Rasmussen et al. p.83) *still to be continued and refined...*

It is possible to obtain different prior models by using different covariance functions. In our case, we use a stationary (x-x, invariant to translation), isotropic exponential covariance function - Squared Exponential Covariance Function (p. 38)

Gaussian Process Prior The specification of the covariance function implies that a GP is a distribution over functions. To illustrate this one can draw samples from a prior distribution of functions evaluated at any number of points, X_*. The Gaussian Process Prior is solely defined by the covariance matrix made up of the covariances of the input points.

$$\Sigma(X_*) = \begin{bmatrix} k(x_{*1}, x_{*2}) & \cdots & cov(f(x_{*1}, f(x_{*n}) \\ \vdots & \ddots & \vdots \\ cov(f(x_{*n}, f(x_{*1}) & \cdots & cov(f(x_{*n}, f(x_{*n}) \end{bmatrix} \in \mathcal{M}^{|X_*| \times |X_*|} \tag{3.1}$$

A sample is a random Gaussian vector $f_* \sim \mathcal{N}(0, \Sigma(X_*))$ containing a function value for every given input point. Plotting random samples above their input points is a nice way of illustrating that a GP is indead a distribution over functions, see figure 3.1. The GP Prior forms the basis for inference in Gaussian Process Regression.



(a) Samples drawn from the prior (b) Samples after a few observations

Figure 3.1: In figure a) three functions have been drawn from the GP prior, each has a sample size of 1000 points. Figure b) again shows three functions, but this time the prediction incorporates the information of random values observed at seven points in the input space

Vector-valued Gaussian Processes In order to use Gaussian processes to model deformation fields of three dimensional vectors as intended, there is the need for a generalization of the above definition from the function-space view. The random variables $X_1, X_2, \ldots, X_k, \ldots, X_n$ are now d-dimensional vectors, yielding a covariance function of the form $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}^{d \times d}$ and $k(x, x') = \mathbb{E}[X_k(x)^T X_k(x')]$. *Should this paragraph be continued?*

3.3 Gaussian Process Regression

The task of registering two 3D face meshs can be treated as a regression problem in which the goal is to predict the deformation of all floating mesh points, given the displacement of the landmarks present in both meshs. Trying to fit an expected function - be it linear, quadratic, cubic or nonpolynomial - to the data is not a sufficiently elaborated approach to our problem. Using a Gaussian Process disposes of the need to describe the data by a specific function type, because the response for every input point is now represented by a normally distributed random value, in turn governed by the specification of the covariance function.

Key assumption: data can be represented as a sample from a multivariate gaussian distribution P

Regression Problem Assume a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$ where $x \in \mathbb{R}^d$ and y is a scalar output or target. Later on, in the case of the training set consisting of landmarks, a Vector-valued Gaussian Process must be used, because y is then also a vector $y \in \mathbb{R}^d$. The task is now to infer the conditional distribution of the targets for yet unseen inputs and given the training data $p(\mathbf{f}_* | \mathbf{x}_*, \mathcal{D})$

Noise-free Prediction First we assume the observations from the training data to be noise-free so that we can fix the training data to these observations

$\mathbf{y}$ without complicating the model. The joint prior distribution with training $\mathbf{f}$ and test $\mathbf{f}_*$ outputs indicated is the following:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \Sigma(X) & \Sigma(X, X_*) \\ \Sigma(X_*, X) & \Sigma(X_*) \end{bmatrix}\right) \tag{3.2}$$

We can now obtain the posterior samples illustrated in 3.1 b) by conditioning the above joint Gaussian prior distribution on the observations $\mathbf{f}_*|\mathbf{f} = \mathbf{y}$ which results in the following distribution:

$$\mathbf{f}_*|X_*, (X, \mathbf{f}) \sim \mathcal{N}\left(\Sigma(X_*, X)\Sigma(X)^{-1}\mathbf{f}, \Sigma(X_*) - \Sigma(X_*, X)\Sigma(X)^{-1}\Sigma(X, X_*)\right) \tag{3.3}$$

Later on, we will extend this definition to 3-dimensional inputs and outputs.

Prediction with Gaussian Noise Model In most real world applications as is the case for the problem we will look into later, however, observations from the training data are not free of noise. The landmarks clicked on the 3D face meshs, for example, can never be marked at the exact same feature location. These circumstances call for the incorporation of a noise model. We specify a simple additive i.i.d Gaussian noise model $y = f(x) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ for every input vector x. In section **??** the variances will be varied for every sole landmark. For now it is enough to add the variance of the noise model to the covariance of the training.

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \Sigma(X) + \sigma^2\mathcal{I}_{|X|} & \Sigma(X, X_*) \\ \Sigma(X_*, X) & \Sigma(X_*) \end{bmatrix}\right) \tag{3.4}$$

The distribution - now conditioned on the noisy observations - is thus

$$\mathbf{y}_*|\mathbf{f} = \mathbf{y} \sim \mathcal{N}\left(\bar{\mathbf{y}}_*, \Sigma(\mathbf{y}_*)\right) \tag{3.5a}$$

where the mean depends on the observed training targets

$$\bar{\mathbf{y}}_* = \Sigma(X_*, X)\left(\Sigma(X) + \sigma^2\mathcal{I}_{|X|}\right)^{-1}\mathbf{y} \tag{3.5b}$$

whilst the covariance depends only on the input points

$$\Sigma_* = \Sigma(X_*) - \Sigma(X_*, X)\left(\Sigma(X) + \sigma^2\mathcal{I}_{|X|}\right)^{-1}\Sigma(X, X_*) \tag{3.5c}$$

Conclusion, how does this help us to proceed?

3.4 Application to 3D Face Meshs

deformation field bold instead of calligraphic? We strive to predict a deformation field $\mathcal{D} : \mathcal{M} \subset \mathbb{R}^3 \to \mathbb{R}^3$ which assigns a displacement vector to every vertex in a triangulated mesh. The mesh in question is called a floating or moving mesh $\mathcal{M}$. Adding the displacement field to the moving mesh should then render a mesh corresponding as closely as possible to a target mesh $\mathcal{T}$ and thereby performing a registration / should provide an accurate mapping on to the target mesh. The mean mesh of the Morphable Model serves as the moving mesh which we want to register with multiple triangulated meshs of scanned target faces.

Reference Mesh Prior As defined by the deformation field the output the regression problem is in $\mathbb{R}^3$ calling for the use of a Vector-valued Gaussian Process with random variables $u \subseteq \mathbb{R}^3$. After the reference and target have been aligned **??** a Vector-valued Gaussian Process can be initialized by defining the prior over all vertices of mean mesh. For this purpose the covariance function has to be redefined to handle 3-dimensional vectors. ***Prior consists of smooth deformations of the mean face***

$$k\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \begin{bmatrix} x_2' \\ x_2' \\ x_3' \end{bmatrix}\right) = xy^T \in M^{3\times 3} \tag{3.6}$$

Each covariance entails 9 relationships between the different components of the vectors, yielding a 3×3 matrix. The covariance matrix then grows to become:

$$\Sigma_{\mathcal{X}} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \in M^{3n\times 3n} \tag{3.7}$$

The mean mesh is defined by a set of vectors $\mathcal{X} \in \mathbb{R}^3$ and a set of landmarks $L_{\mathcal{M}} = \{l_1, \cdots, l_n\} \subset \mathbb{R}^3$. The mean vector μ is made up of the component-wise listing of vectors so that is has dimensionality $3n$. Setting the whole mean vector to zero, as discussed before, implies a mean deformation of zero and makes perfect sense in this setting, because we are modelling deformations of the mean face surface. The prior distribution over the mean face mesh is therefore defined as

$$\mathcal{D} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathcal{X}}) \tag{3.8}$$

meaning that a deformation field can be directly drawn as a sample from the prior distribution of the vertices of the mean mesh. ***show two or three samples of prior here, next to mean mesh***

Reference Mesh Posterior The target landmarks also consist of a set $L_{\mathcal{T}} = \{l_{\mathcal{M}1}, \cdots, l_{\mathcal{M}N}\} \subset \mathbb{R}^3$. Fixing the prior output to the deformation vectors $\mathcal{Y} = \{\mathbf{t} - \mathbf{m} | \mathbf{t} \in L_{\mathcal{T}}, \mathbf{m} \in L_{\mathcal{M}}\}$ defined by the distance between the reference and target landmarks and assuming additive i.i.d Gaussian noise the resulting posterior distribution is

$$\begin{bmatrix} \mathcal{Y}_\varepsilon \\ \mathcal{D} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \Sigma(\mathcal{Y}) + \sigma^2 \mathcal{I}_{3|\mathcal{Y}|} & \Sigma(\mathcal{Y}, X_*) \\ \Sigma(X_*, \mathcal{Y}) & \Sigma(X_*) \end{bmatrix}\right) \tag{3.9}$$

Is this a correct definition for the distribution?

The deformation model is now rendered fixed at certain landmark points in the target mesh and the goal is to find valid deformations through the set of fixed targets, analogous to the case of eq. 3.5a. The posterior model is defined as the joint distribution of all mean mesh points and the mean landmarks, conditioned on the output deformation vectors for every mean landmark with added noise.

$$\mathcal{D} | \mathcal{X} \rightarrow \mathcal{Y}_\varepsilon. \tag{3.10}$$

e now have defined a distribution over our mean face fesh. The variance of the gaussian kernel can thereby be described as a smoothing parameter P ***mean is now max aposteriori solution***

Sampling the conditional distribution creates deformed 3D surfaces of the mean mesh which are fixed at the target landmarks. ***show images of mean, prior and posterior with added landmarks***

3.5 Fitting & Optimization

Due to the fact that the posterior model is fixed at the target landmarks it is possible to perform the registration by drawing a shape from the posterior model. The sample, however, has to be optimized according to the shape of the target face. In order to find the deformation corresponding to the optimal fit d_* a linear optimization with the posterior process as a constraint is be employed/ regularization term. (small lambda) a bit of the posterior mean)

$$d_* = \mathop{\mathbf{arg\ min}}_{d \in \mathcal{D}} \quad L[O_\mathcal{T}, O_\mathcal{M} \circ d] + \lambda R[d] \tag{3.11}$$

Minimizing a loss function L - mean square distance for example - on the target and the deformed mean provides a feasible deformation field. D denotes the space of possible deformations in the posterior model. The information needed is held by the covariance matrix of the posterior process. However, using Mercer's theorem (in short what it does, yada yada yada) a basis of functions corresponding to all possible deformations can be extracted. The kernel is thus described as a linear model of these basis functions. ***whole matrix or simple kernel? refer to paper "a unified formulation of statistical model fitting and non-rigid registration***

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x) \phi_i(x') \tag{3.12}$$

λ_i are the eigenvalues and ϕ_i the eigenvectors of K. They denote the deformation directions while the eigenvalues ... We are looking for a finite linear combination of eigenvectors that form a deformation field with $\exists \alpha_1 \cdots \alpha_n \in \mathbb{R}$ as linear parameters.

$$f(x) = \sum_{i=1}^{n} \alpha_i \lambda_i \phi_i(x) \tag{3.13}$$

f GP(0, K) we take our gaussian process ¿ f — x=y, ***ask Marcel for a helping hand with the theory?***

Next, we want to minimize residuals for the whole of our surface according to optimal parameter values α_i

$$\mathop{\mathbf{arg\ min}}_{\alpha \in \mathbb{R}^n} \quad \Sigma_{x_i \in \mathcal{T}_R} (f(x_i) - \phi_T(x_i))^2 \tag{3.14}$$

where $f(x_i)$ is the deformation function and $\varphi_T(x_i)$ returns the nearest point on the target mesh. Yields the overall loss function Φ_L

$$\Phi_L(f(x_i) - \varphi_T(x_i)) \tag{3.15}$$

The eigen vectors - which are deformation vectors defining a deformation for every model vertex - of the covariance matrix define a basis space? Shape Modell =¿ select best eigenvectors via PCA in order to simplify computation. =¿ Vorstellen wie wenn mehrere Wellbleche durch die Target"_"landmarks gelegt werden und dann mit bestimmten parametern alpha zwischen ihnen interpoliert wird Alternative way to understand basis functions for gaussian process: sample from the GP(0, K) and then build a linear model from the functions, f(x) = sum(i, n) alpha(i) si(x) Posterior Distribution of Landmarks Defining the Gaussian Process Posterior Distribution - Landmarks (Referenz deformieren From Gaussian Processes to Shape Models =¿ by selected principal components of the covariance matrix

3.6 Robust Loss Functions

robust against outliers the Alignment of the mean face mesh and the target mesh causes overlaps on the forehead, the side of the head and the neck. Using a simple Mean Square error between the reference and target mesh for optimization penalizes the overlapping regions with a strong gradient and therefore causes strong distortions. Our approach to tackling this problem was to try out a range of different robust estimators, namely the Tukey, Huber, and Fair estimators. (table with formulas?) The advantage is that these estimators are less sensitive to outliers, reducing the artefacts of registration considerably. However, as can be seen from the formulas, these techniques require finding appropriate parameters first which produce reasonable/acceptable visual results Fair

$$\rho(x) = c^2 \left[\frac{|x|}{c} - log(1 + \frac{|x|}{c}) \right] \tag{3.16a}$$

$$\psi(x) = \frac{x}{1 + \frac{|x|}{c}} \tag{3.16b}$$

Huber

$$\rho(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| < k \\ k(|x| - \frac{k}{2}) & \text{if } |x| \geq k \end{cases} \tag{3.17a}$$

$$\psi(x) = \begin{cases} x & \text{if} \\ ksgn(x) & \text{if} \end{cases} \tag{3.17b}$$

Tukey

$$\rho(x) = \begin{cases} \frac{c^2}{6} \left(1 - \left[1 - \left(\frac{x}{c} \right)^2 \right]^3 \right) & \text{if } |x| \leq c \\ \frac{c^2}{6} & \text{if } |x| > c \end{cases} \tag{3.18a}$$

$$\psi(x) = \begin{cases} x \left[1 - \left(\frac{x}{c} \right)^2 \right]^2 & \text{if} \\ 0 & \text{if} \end{cases} \tag{3.18b}$$

Rigid Alignment This part is the theoretical part, alignment can follow in the Pipeline Therefore, we first have to perform a rigid transformation to align the meshs according to the feature/ landmark points.

Chapter 4

Registration Pipeline using Line Features

In this chapter we follow up on the definition of the Vector-valued Gaussian Model for 3D Face Registration by describing the registration pipeline built to put this concept into practice. The pipeline is of a sequential nature, where in each step the output of a data processing unit is the input for the next step. To enhance the registration outcome of this pipeline we use contour lines of key regions of the face.

4.1 Line Features

Definition of Line Features

For every scan we want to register, 8 contours have been marked on three images of the face - taken from the front, the left and the right of the face - with a special GUI for marking points and lines on images. These contours depict the eyebrows, eyes, ears and lips of a face and we call them “line features”. They are made up of a set of segments, each of which is modelled with a **Bézier curve** (parametric curve frequently used in computer graphics, bernstein basis polynomials, used for modelling smooth curves) of varying order. Due to the nature of the objects depicted, there are open as well as closed curves.

$$B(t) = \sum_{i=0}^{n} (1-t)^{n-i} t^i P_i \tag{4.1}$$

The line features are saved in explicit files along with the face mesh of the scan.

Why use Line Features for Registration?

Line features serve the purpose of augmenting the quality of registration by initiating it with a larger set of corresponding points (points which are on the lines). They are used to mark complex regions of the eyes, i.e. the eyes, ears etc., so that the registration process produces an accurate mapping of the contours of these organs which would otherwise not be possible. Areas containing “curves” have a dense abundance of points/parameter changes, while straight areas only have scarce points.

Figure 4.1: Line Features of the left eyebrow and eye, consisting of bézier curves defined by visible control points (white)

4.2 Sampling 3D Points from 2D Line Features

In order to be able to use line features in the Vector-valued Gaussian Model, they have to be sampled at discrete intervals resulting in a set of additional landmarks $L_{Add} = \{l_1, \cdots, l_N\}$. These define the mapping $\Omega : L_{Add\mathcal{M}} \rightarrow L_{Add\mathcal{T}}$ of the contours - describing the different imporant features present in the faces - in the mean face mesh on those of the target face mesh. In order for the mapping Ω to be plausible, it is essential for the curves to have equidistant parametrization so that when curves undergo sampling of N points, these N points are all at equal parametric intervals.

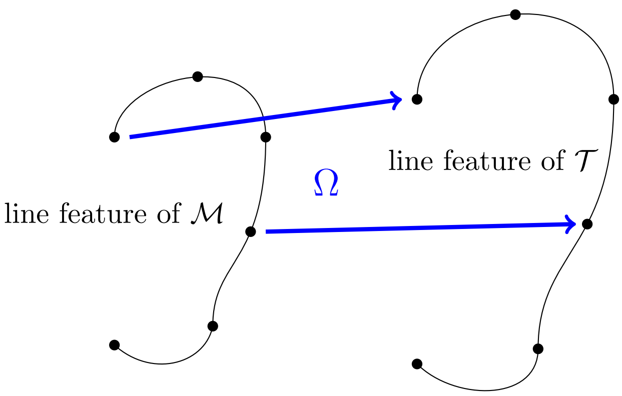
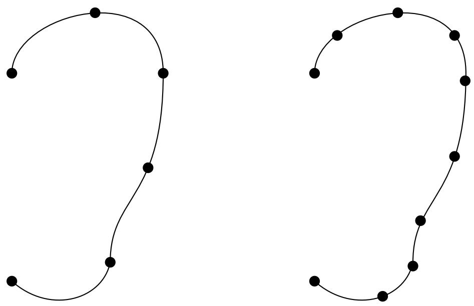


Figure 4.2: Mapping of equidistant samples of **ear** line features from the reference (left) on to the target (right)

Arc Length Parametrization

The first problem which becomes apparent when trying to sample the line features is that the bézier curve segments don't allow for equidistant parametrization, because the underlying parameter $t \in \mathbb{R}$ is not linear in respect to the length of the curve. The growth of the parameter of a bézier curve is instead dictated by velocity.



add another point to the right ear, so there are 11.

Consequently, the imperative must be to evaluate the curves based on their arc-length, which is defined as the length of the rectified curve, instead. The underlying parameter must then correspond - at every point of the curve - to the ratio of the curve length that has been traversed and the total curve length.

In theory It is possible to get the arc length $L(t) = \int_{t_0}^{t_1} |C'(t)| \, dt$ for given parameters t_0, t_1 where $C'(t)$ is the derivative of the curve $C : t \in [0, 1] \to \mathbb{R}^2$. What we are in need of, however, is a reverse mapping from the length of a fraction of the curve to the curve parameter $t = L^{-1}(l)$. This mapping can of course be derived analytically, but it is far easier to implement it using a numeric approximation.

In practice As we are not in need of a subpixel accurate resolution, we can skip the formal math and use a lookup table to compute the arc-length. First, we calculate n=1000 points on each segment the curve - made up of bézier curves using the normal parameter t. For each point we save the euclidean distance from the origin of the segment into a new slot in the lookup array. We get the euclidean distance for one point by summing up the distance to the predecessor/preceeding points and its distance from the origin.
draw a line with a few segments draw curve with points just off and

lookup table beneath In effect, we are provided with have a lookup array that contains the approximated distances of a large number of points from the origin of a curve segment. Assembling the segments' lookup arrays gives us the overall array for the curve with the last value presenting the arc-length of

the whole curve.
Second, finding points on the curve according to a linear parameter governed by the amount of points that we want to parametrize the curve with is quite easy. The curve can easily be sampled by computing the length of parametric intervals $\frac{L}{N}$ for a specified number of points N to be sampled. $l = k \cdot \frac{L}{N}$ returns the current length of the curve for the sampling point of index k, where $k = [0, \cdots, N]$ for open curves and $k = [0, \cdots, N-1]$ for closed curves. Then we simply perform a binary search on the lookup table (to get largest value smaller than n?) for this distance. We choose the index that returns the exact length we specified or the index with the next smaller length. The coordinates of the point with this index t are now the coordinates we use for the sampling point. We compute the distance we want to travel the curve using the length of equidistant sections and the point we want to get. ***reference lines in text above?***

Listing 4.1: Equidistant Sampling

```
void getEquidistantPoints(int numSampleSegments = 20) {
    // static members:
    // arcLookup - lookup table
    // totalLength - total arc-length of curve
    // auxiliaryPoints - ??? exact definition

    if(arcLookup.size() == 0) return;
    int pointsToDraw = numSampleSegments+1;
    if(closed) pointsToDraw--;

    T sectionLength = totalLength/numSampleSegments;

    for(size_t i=0; i < pointsToDraw; ++i) {
        T progress = i*sectionLength;
        // perform c++ binary search on lookup table
        int low = 0;
        int currIndex = 0;
        int high = arcLookup.size()-1;
        T currPieceLength;

        while(low < high) {
            currIndex = low + (high - low)/2;
            currPieceLength = arcLookup[currIndex];
            if(currPieceLength < progress) {
                low = currIndex+1;
            } else {
                high = currIndex;
            }
        }
        // currPieceLength is now >= progress
        if(currPieceLength > progress) {
            currIndex--; // currPieceLength is now < progress
        }
        equidistantPoints.push_back(auxiliaryPoints[currIndex]);
    }
}
```

Mesh Projection of Sampled Points

Having implemented arc length parametrization it is possible to draw an arbitrary amount of samples $x \in \mathbb{R}^2$ from the line features. They are thereby defined as a set of points $S \subset \mathbb{R}^2$. Our goal is, however, to have these additional landmarks describing the features on the mesh itself and not a 2-dimensional snapshot. We therefore need to use the camera callibration and some computer graphics to project the sampled points onto a face mesh for each line feature we want to obtain.

Previous implementation In the previously used registration method, a large number of points was used for each curve. These points were, however, not projected directly on to the 3/4 shells of the mesh. Instead their location was constrained by computing a 1-dimensional band of points before and after their approximate position, seen from the origin.

Get direction of 3D representation of a curve, compute distance from origin to mesh, normalize to direction.

Compute distances from origin for mesh vertices dot product (direction of point, for every vertex: direction of vertex) dot product: 1 for similar directions, 0 for perpendicular directions. Angle value ¿.9999: mindist, maxdist are updated with the distance value of the distance vector to the nearest vertex on to the direction of the actual 3D representation of the point

In the previous registration method implemented by Dr. Brian Amberg, the points from line features constrained to 1D and are then projected on to the 3 shell meshs with the program points_from_surface. That is how the feature points are generated. shells from the scanner are cleaned points are marked on the 3 images to the front, left and right of the person xplain what program does, latex sketches: camera calibration is done by the scanner? meshs and camera settings are loaded into the software. For a lot of points per curve the direction their 3d representation is computed and their distance, normalized to give direction, from the origin is saved. Distances and directions are further computed for all vertices in the mesh. Now for every point the dot product of its direction is formed with direction of every vertex in the mesh. Remember, the dot product results in 1 for similar directions and 0 for perpendicular directions. For an angle value larger than .9999 the parameters min_dist or max_dist are updated with the distance value of the projection of the distance vector to the nearest vertex on to the direction of the actual 3D representation of the point. min(min_dist, d), max(max_dist, d), so that at the end min_dist contains the distance to the point on the line nearest to the camera and max_dist the distance to the point farthest away from the camera. now min_dist is multiplied with a value ¡ 1 and max_dist with a value ¿ 1. then a band of points is computed perpendicular to the line along the direction at each computed point that cuts through the mesh and serves as a horizontal constraint for the points to lie after the registration. picture of dot product

Listing 4.2: Point Projection

```
for (size_t l=0; l<templates.size(); ++l) {
    if (!templates[l].isSet)
    continue;
    {
        vector<f3Vector> meanEqPoints3d;
        templates[l].evaluate(512);
        // Find min_dist and max_dist for this template
        /* COMPUTE EQUIDISTANT LINE POINTS AND WRITE THEM TO FILE */

        // computing 1000 points per curve segment
        templates[l].curve.initializePoints();
        // create arc length lookup table computed over all initalized
            segment points
        templates[l].curve.approxTotLength();
        // equidistant sampling
        templates[l].curve.getEquidistantPoints(numPoints);
        vector<d2Vector> eqPoints = templates[l].curve.equidistantPoints;
        vector<d3Vector> eqDirs;

        for (size_t i=0; i<eqPoints.size() ; ++i) {
            // compute direction vector of 3d representation of points on
                curve from the point of origin
            auto point = eqPoints[i];
```

```
                d3Vector dir= -O + C.imageToWorld(point);
                dir /= dir.normL2();
                eqDirs.push_back(dir);
            }
            vector<double> selectedVecDist;
            // go over all directions of points on the line template and compute the dotproduct with the current mesh vertex direction
            for (size_t p=0; p < eqDirs.size(); ++p) {
                const d3Vector &dir = eqDirs[p];
                // save distances along the directions of near vertices and angles for every point
                vector<double> remDistances;
                vector<double> remAngles;
                for(size_t i=0; i < meshes.size(); ++i) {
                    for(size_t j=0; j < meshes[i].vertex.size(); ++j) {
                        // compute direction from origin for every vertex in the mesh
                        d3Vector vert_dir = (d3Vector(meshes[i].vertex[j]) - O);
                        d3Vector vert_dir_n = vert_dir / vert_dir.normL2();

                        double a = vert_dir_n.dot(dir);
                        // if direction likeness is bigger than 99.99%
                        if(a > 0.9999) {
                            // projection of distance vector of mesh vertex onto direction of 3d representation of true point on curve segment
                            // project vert_dir onto dir and
                            double dist = vert_dir.dot(dir);
                            remDistances.push_back(dist);
                            remAngles.push_back(a);
                        }
                    }
                }
                // choose distance via best angle match, PROBLEM: holes in mesh
                if(remAngles.size() > 0) {
                    int index = std::max_element(remAngles.begin(), remAngles.end())-remAngles.begin();
                    selectedVecDist.push_back(remDistances[index]);
                } else {
                    selectedVecDist.push_back(0.0);
                }
            }
            // save directions to equidistant points on all line features in 3D
            for (size_t i=0; i<eqPoints.size(); ++i) {
                d3Vector dir = -O + C.imageToWorld(eqPoints[i]);
                dir /= dir.normL2();
                float a = 0.5f;
                f3Vector point;
                if(selectedVecDist[i] != 0) {
                    f3Vector tmp(O + selectedVecDist[i] * dir);
                    point = tmp;
                }
                meanEqPoints3d.push_back(point);
            }
        }
```

Modification ***up close image of eye holes of face scan*** The modification we introduced, was solely to the select the mesh vertex with the highest similarity of direction to the 3D representation of a 2D line feature sample. Due to the areas of the mesh around the ears and the eyes containing large holes the projected sample points from the line features can be off target, especially if a large number of points is sampled for every curve. This circumstance leads to the sampled line features being represented by more of a point cloud, for example around the eyes, (which is not distinguishable as a line) instead of clearly denoting a contour line. The direction of the vertex is used to find a point −¿ this distorts the shape (position of sampled points) of the line. On different data sets the performance of the projection of the line features for a large number of samples varied enormously. ***Compute some landmarks with 30 samples*** Using only 5-10 sample points per curve some datasets rendered near perfect results on a “control sample”. ***Compile list of datasets*** However, as long as the method is dependent on the data from the scans - the size of the holes in the meshs - it lacks generality and generality is exactly the basis for feasible and reproducable registration results.

```
/**
* Returns the position in world coordinates lying on the focal plane
* which is corresponding to a pixel coordinate. The camera ray is
* c.imageToWorld(v_i) - c.origin()
**/
inline
t3Vector<T> imageToWorld(const t2Vector<T> &v_i) const {
    /// Pixel to Distorted Image Plane
    //
    // Offset
    const t2Vector<T> v_i_o = v_i-C;
    // Scale
    const t2Vector<T> v_d(v_i_o.x/sx*d.x, v_i_o.y*d.y);

    /// Distorted Image to Pinhole
    const t2Vector<T> v_p = v_d * (1.0 + k1*v_d.normL2sqr());

    /// Pinhole to Camera
    const t3Vector<T> v_c(v_p.x, v_p.y, f);

    /// Camera to World
    const t3Vector<T> v_w = Rinv * (v_c - t);
    return v_w;
}
```

4.3 Preparing the Mean Mesh

On top of that, another problem occured, because the mean face mesh of course doesn’t have any line features projected on to it either. Rendering, marking line features, projecting back possible, because we know direction

However, it contains about 60 feature points manually clicked, which are not present in newly scanned datasets. Eliminate the ones, which are not clicked on scans

Output pipeline specifications

4.4 Rigid Mesh Alignment

simple rigid transformation of the scanned face onto the mean, transformation computed from landmark vectors. To begin the registration we first have to align the two meshs. The floating mesh has to be clipped at the neck and around the ears where the scanner has left artifacts. Furthermore the mouth cavity of the mean face has to be removed. We then selected the 11 feature points present in the floating mesh in the mean face from the abundant 60. To achieve this we wrote a python script loading the feature point files. A feature point is described by its 3D coordinates, a visibility parameter in the range [-1,1] and a label denoting its exact location (mouth.inner.upper). All we had do to now was to create to dictionaries label : (x,y,z) and to compare them for labels. Then we passed the resulting point correspondencies to the python vtk api for the mean of computing a transformation comprised of simple translation and rotation (no scaling, only 3 point correspondencies needed). Note, we are not trying to map the meshs on to one another here. We are simply trying to align them through the use of the feature points. The computed transformation we applied to all points in the floating mesh. The resulting mesh was written to a file and then opened in paraview. We now had the meshs in a position from where we could start the actual mapping. The mean face was broader in shape than the scan and was perfectly coated in texture for the simple reason that hours of manual labour have been invested to render this important piece of data a perfect reference. Now in order to receive a perfect mapping of the floating mesh on to the mean/reference mesh we have to allow for 3 degrees of freedom, that is in all 3 dimensions x,y and z, for every pixel in the floating mesh except for the reference points we have used as correspondencies. The parameters having the most influence to the mapping will be those specified in the constraints we introduced into the equation via regularization. The idea behind the use of sampled points from the line features was to have more point correspondencies in complex regions as for example the eyes and the ears where there is a great abundancy of pixels and the algorithm isnt likely to create a flow field which is accurate not enough to describe these regions, because of its smoothness constraint. For the actual registration we use the software framework statismo developed at the Computer Science Department of the University of Basel. It is a framework for PCA based statistical models. These are used to describe the variability of an object within a population, learned from a set of training samples. We use it to generate a statistical model from the floating mesh. Furthermore we use the software package gpfitting for the actual fitting. We generate a infinite row of faces from the statistical model using gaussian processes and then sample out a fixed number. Then the faces are left. Carry on.

4.5 Prior Model

what to say here? describe programme?

4.6 Posterior Model

what to say here? describe programme?

4.7 Fitting

4.8 Optimizing the Loss Function

4.9 Varying the Variances