Learning Robust Objective Functions for Model Fitting in Image Understanding Applications

Matthias Wimmer, Freek Stulp, Stephan Tschechne, Bernd Radig
Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, 85748 Garching, Germany
`wimmerm@cs.tum.edu` `http://wwwradig.cs.tum.edu`

Abstract

Model-based methods in computer vision have proven to be a good approach for compressing the large amount of information in images. Fitting algorithms search for those parameters of the model that optimise the objective function given a certain image. Although fitting algorithms have been the subject of intensive research and evaluation, the objective function is usually designed ad hoc and heuristically with much implicit domain-dependent knowledge.

This paper formulates a set of requirements that robust objective functions should satisfy. Furthermore, we propose a novel approach that learns the objective function from training images that have been annotated with the preferred model parameters. The requirements are automatically enforced during the learning phase, which yields generally applicable objective functions. We compare the performance of our approach to other approaches. For this purpose, we propose a set of indicators that evaluate how well an objective function meets the stated requirements.

1 Introduction

In computer vision applications, models serve as an intermediate step for interpreting scenes in images. Model-based image understanding methods exploit a priori knowledge about the objects and the scene by imposing constraints on texture, regions, or region boundaries. This reduces the large amount of information in images to a small set of model parameters. This parameters facilitate and speed-up further image understanding. Model fitting is the computational problem of finding the parameter values for a parametric model that describe the content of the image best [2].

Model-based methods consist of three main components. (1) The model. There are various types of models and each of them is appropriate for a certain use, such as 2D and 3D models, contour and texture models, rigid and deformable models, and many more. A parameter vector $\mathbf{p}$ represents the possible configuration of the model, such as position, rotation, scaling, and deformation. (2) The fitting method that searches for the model parameters that match the image best. Since our methods are independent of the fitting algorithm used, we shall not elaborate on them in this paper. We refer to [2] for a recent overview and categorisation of fitting algorithms. (3) The objective function that evaluates how good the fit of a parameterised model to an image is. Examples and synonyms of the objective function are *likelihood function*, *energy function*, *cost function*, *goodness function*, and *quality function*.

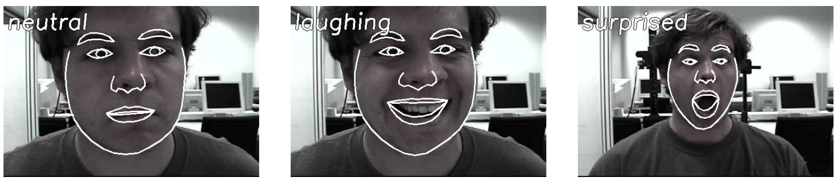

Figure 1: Fitting a deformable face model to images that show different mimics.

Mimic recognition serves as a good example for model-based image interpretation. Deformable face models are suitable for expressing the variations within a human face. The parameters describe the constitution of the face, such as the opening of the mouth, the roundness of the eyes, or the raising of the eye brows, as depicted in Figure 1. A fitting algorithm searches for the parameters that describe the visible face best by minimising an objective function. The mimic of the person is then derived from the model parameters. Pantic et al. [3] give a comprehensive overview about current research within this area.

Fitting algorithms have been the subject of intensive research and evaluation. In contrast, the objective function is usually determined ad hoc and heuristically, using the designer's intuitions about a good measure of fitness. Afterwards, its appropriateness is subjectively determined by inspecting the result of the objective function on example images and example parameterisations of the model. If the result is not satisfactory the function is tuned or redesigned from scratch. This iterative process is shown to the left in Figure 2. In short, the traditional way of designing objective functions is rather an art than a science. The consequence is that this methodology is time-consuming, and often leads to non-robust functions that do not objectively determine the fitness of a model parameterisation.

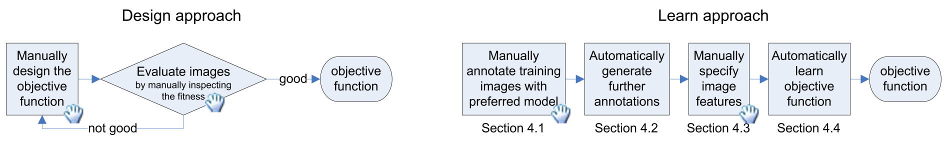
Design approach
Manually design the objective function
Evaluate images by manually inspecting the fitness
good
objective function
not good
Learn approach
Manually annotate training images with preferred model
Automatically generate further annotations
Manually specify image features
Automatically learn objective function
objective function
Section 4.1
Section 4.2
Section 4.3
Section 4.4

Figure 2: Modus operandi for the design approach and the proposed learn approach.

1.1 Solution Idea

To avoid the problems of the design approach, we invert this procedure. Instead of iteratively designing an objective function and testing it on example images, we first annotate example images with preferred model parameters and learn the objective function from these examples. The two approaches are contrasted in Figure 2. This novel procedure has several benefits. First of all, this method is less error-prone, because giving examples of good fits is much easier than explicitly designing a function that should cover all possible examples. Analogously, most of us will have no problem determining whether we like individual paintings or not. On the other hand, it would be difficult to incorporate the exact reasons for these preferences into a function that would correctly predict our judgement for previously unseen paintings. A further benefit is that the labour-intensive phase for the objective function design is replaced by an automated learning algorithm, which also eliminates the iterative *design-inspect* loop. Finally, the learning phase can be directed, for instance to enforce certain requirements the objective function should meet. This yields more robust and accurate objective functions which greatly facilitate the task of the fitting algorithms.

The contributions of this paper are threefold: First, we express a set of domain-independent requirements that robust objective functions need to satisfy. Second, we propose a new approach that learns objective functions from training data. Third, we formulate a set of indicators that deliver comparable values about the accuracy of objective functions.

In the remainder of the paper we proceed as follows. In Section 2 we explain the drawbacks of manually designing objective functions. In Section 3 we formulate a set of requirements for robust objective functions and define optimal objective functions based on those insights. Section 4 describes our approach that learns an objective function from annotated example images. Section 5 evaluates the accuracy of our approach and compares it to both a design approach and an optimal approach. Section 6 concludes with a summary and an outlook on future work.

2 Designing Objective Functions

An objective function $f(I, \mathbf{p})$ calculates a value that indicates how well model parameters $\mathbf{p}$ describe the content of the image I. In this paper, lower values correspond to a better fit, so f is a cost function. The goal of fitting algorithms is therefore to find the model parameters that minimise the objective function.

In this paper, we will use a contour model, and base the objective function on this contour. Such models incorporate a contour which is described by n contour points $\mathbf{c}_i(\mathbf{p})$ with $1 \leq i \leq n$. These points are often connected for visualisation purposes. Both the model and its contour are appropriate for describing the object in the image, but usually the contour consumes more memory space than the model because $n \gg dim(\mathbf{p})$. To facilitate the design of the global objective function $f(I, \mathbf{p})$, it is usually defined to be a sum of n local objective functions $f_i(I, \mathbf{c}_i(\mathbf{p}))$, one for each contour point.

Hanek [2] categorises objective functions into two groups: non-feature-extracting methods and feature-extracting methods. Non-feature-extracting methods use the plain image data to calculate the fitness of a model instance. Since those algorithms do not rely on appropriately extracted image features, they promise to work even with difficult image conditions. Feature-extracting methods reduce the vast image information to a smaller set of appropriate features. Those features are most often obtained via edge detection algorithms or region growing algorithms. The calculation rules of the objective function base on the deviation between the model boundary and those features. The process of fitting a model to a set of extracted image features is also called *model matching*. Our approach contributes to this category. We use local Haar-like features that are combined in a non-linear way to create the calculation rules of the objective function. The calculation rules are empirically learnt.

The traditional methodology for obtaining objective functions is to design them heuristically. Humans pick salient image features and combine them in an intuitive way to formulate the calculation rules. The contour is often split up into semantically meaningful parts and the $f_i(I, \mathbf{c}_i(\mathbf{p}))$ are then designed differently for each part. This approach is characterised by a subjective decision of what features are appropriate and how to design the calculation rules. Commonly used objective functions consider the edge values of the image, see Cootes et al. [1]. It treats the fitness to be good if the contour of the model overlaps the edges of the image. We design a similar objective function with the following formula, where $E(I, \mathbf{c}_i(\mathbf{p}))$ denotes the magnitude of the edge at the position $\mathbf{c}_i(\mathbf{p})$. The magnitudes are defined to range from 0 to 1.

$$f^e(I, \mathbf{p}) = \frac{1}{n}\sum_{i=1}^{n} f_i^e(I, \mathbf{c}_i(\mathbf{p})) = \frac{1}{n}\sum_{i=1}^{n}(1 - E(I, \mathbf{c}_i(\mathbf{p}))) \tag{1}$$

Those design approaches have comprehensive shortcomings. Their calculation rules primarily consider edges and region boundaries which are no salient image features in unconstrained environments. The objective function shows no continuous behaviour because of the small local extent of those features. It consists of many local minima and maxima, instead. Figure 3d) illustrates the behaviour of the designed objective function $f_i^e(I, \mathbf{c}_i(\mathbf{p}))$ by moving the contour point along its perpendicular towards the contour. It produces several local minima and the desired position in the centre does not produce the global minimum. Figure 3b) and 3c) show the image data and the magnitude of the edges along the perpendicular.

3 Requirements for Robustness

We now formulate a set of soft requirements for robust objective functions. The extent to which objective functions meet these requirements determines how appropriate they are for fitting algorithms. Note that R1 and R2 are no distinct requirements but they are derived from R3. In the following, the desired model parameters $\mathbf{p}^*$ are those parameters that describe the content of the image best. Those parameters are not calculated but need to be annotated manually. Analogously, $\mathbf{c}_i(\mathbf{p}^*_\mathbf{k})$ denote the desired contour points that are related to $\mathbf{p}^*$.

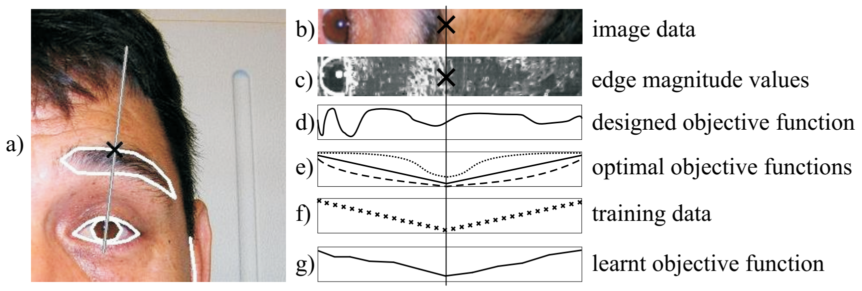
a)
b)
image data
c)
edge magnitude values
d)
designed objective function
e)
optimal objective functions
f)
training data
g)
learnt objective function

Figure 3: a) depicts the 43rd contour point (black cross) and its perpendicular in the 22nd image in the database b-g) show different values along this perpendicular. In e), the solid line is $f_i^{\mathcal{O}}$.

R1: The objective function $f(I,\mathbf{p})$ has a global minimum at $\mathbf{p} = \mathbf{p}^*$.

R2: The objective function $f(I,\mathbf{p})$ has no local extreme value or saddle point at $\mathbf{p} \neq \mathbf{p}^*$.

R3: At any $\mathbf{p} \neq \mathbf{p}^*$ the slope falls most towards $\mathbf{p}^*$. Therefore, the direction of the gradient $\nabla f(I,\mathbf{p})$ needs to point away from $\mathbf{p}^*$.

We define *optimal* objective functions to be the ones that exactly satisfy all of those requirements. There are several of those functions and three examples are depicted in Figure 3e). One of them is described by formula 2. There is a twofold need for an optimal objective function: First, the accuracy of any other objective function is easily evaluated by a comparison. Second, its result is essential for generating the training data, see Section 4.2. The training phase requires us to split up $f^{\mathscr{O}}$ into a sum of local parts $f_i^{\mathscr{O}}$. This splitting does not exactly deliver the same result, but it serves as a good approximation.

$$f^{\mathscr{O}}(I,\mathbf{p}) \; := \; |\mathbf{p}^* - \mathbf{p}| \quad \approx \quad \frac{1}{n}\sum_{i=1}^{n} |\mathbf{c}_i(\mathbf{p}^*) - \mathbf{c}_i(\mathbf{p})| \; =: \; \frac{1}{n}\sum_{i=1}^{n} f_i^{\mathscr{O}}(I,\mathbf{c}_i(\mathbf{p})) \qquad (2)$$

4 Learning Robust Objective Functions

In this section we will introduce the novel methodology that automatically determines an objective function $f^{\ell}(I,\mathbf{p})$. This function is learned from images that have been annotated with preferred model parameterisations, using model trees. This section describes the four steps of our methodology.

4.1 Annotating the Images with the Preferred Model

Each of the K training images I_k is first annotated with the preferred model parameters $\mathbf{p}_{\mathbf{k}}^*$, as shown in Figure 4 left. This is the only labour intensive step in our procedure. Note that *preferred* is defined with respect to the user's judgement, not some predefined objective measure, because such a predefined measure does not exist. We therefore avoid using the word *optimal parameters*, because this would imply that it does exist. Note also that this is not a consequence of our approach; the same holds for all annotated benchmarks.

4.2 Generating Further Image Annotations

To learn the objective function, the learning algorithm should not only be trained with images that are annotated with the preferred parameters $\mathbf{p}_k^*$ and the derived preferred contour points $\mathbf{c}_i(\mathbf{p}_k^*)$, but also with other possible positions of the contour points. Otherwise, $f_i^{\mathscr{O}}$ would always return 0 by definition, and there would not be much to learn. Therefore we automatically generate slightly relocated positions from the contour point along the perpendicular, as shown to the right Figure 4. As can be expected from Equation 2, the values for these deviations are larger than 0.

4.3 Specifying the Features and Generating Training Data

The result of the annotation step is a list of K images with preferred parameters $\mathbf{p}_k^*$, with n corresponding contour points and a list of D deviations from this point, together with the

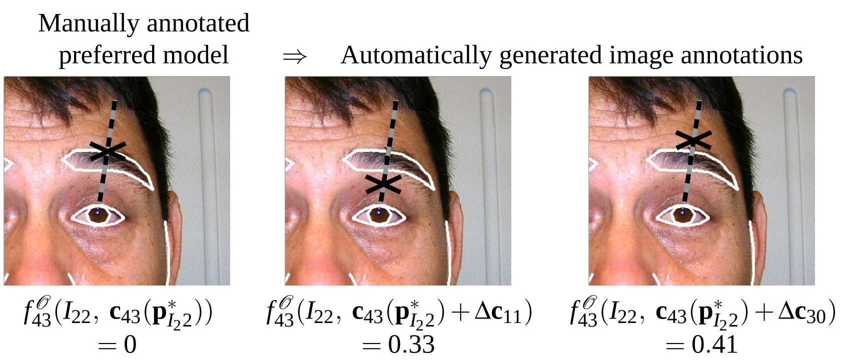
Manually annotated preferred model
⇒
Automatically generated image annotations
$f^{\varnothing}_{43}(I_{22},\ \mathbf{c}_{43}(\mathbf{p}^*_{I_22})) = 0$
$f^{\varnothing}_{43}(I_{22},\ \mathbf{c}_{43}(\mathbf{p}^*_{I_22}) + \Delta\mathbf{c}_{11}) = 0.33$
$f^{\varnothing}_{43}(I_{22},\ \mathbf{c}_{43}(\mathbf{p}^*_{I_22}) + \Delta\mathbf{c}_{30}) = 0.41$

Figure 4: Each image is manually annotated with the preferred parameters and thus with the preferred contour points at the same time. Here we show the 22nd image and how the 43rd contour point of the model is automatically relocated along the perpendicular (dotted line) which generates more annotations for the subsequent training step.

value returned by $f^{\varnothing}_i$: $[I_k,\ \mathbf{c}_i(\mathbf{p}^*_k) + \Delta\mathbf{c}_d,\ f^{\varnothing}_i(I_k,\ \mathbf{c}_i(\mathbf{p}^*_k) + \Delta\mathbf{c}_d)]$ with $1 \le k \le K$, $1 \le i \le n$, and $1 \le d \le D$. In Figure 4, examples for $k = 22$, $i = 43$ and $d = (0, 11, 30)$ are shown.

The goal is to now learn from these examples the function f^{ℓ} that maps I and $\mathbf{p}$ to $f^{\varnothing}(I, \mathbf{p})$. Analogously to the design approach explained in Section 2, we learn a local function $f^{\ell}_i(I, \mathbf{c}_i(\mathbf{p}))$ from $f^{\varnothing}_i$, for each contour point. It might seem that we are relearning the already known function $f^{\varnothing}_i$ in Equation 2. However, $f^{\varnothing}_i$ references $\mathbf{p}^*$, which of course is unknown for novel images. f^{ℓ}_i will only base its value on an image I and contour point $\mathbf{c}_i(\mathbf{p})$. To facilitate the learning of f^{ℓ}_i, we extract a set of features from I, given $\mathbf{c}_i(\mathbf{p})$. The idea is then to provide a multitude of features associated with the contour points, and let the learning algorithm choose which features are relevant to the objective function.

We use Haar-like features [5] of different shapes, sizes, and locations around each contour point. These locations are on several concentric circles around the contour point, see Figure 5. When moving the contour point along its perpendicular, these features also move along with it, and their values change. Contrary to edge-based and region-based features, Haar-like features cope with noisy image data. They even allow for modelling contours at non-distinctive borders, such as the chin line, see Figure 1. Haar-like features are high-performance, as they are efficiently computed from the so-called *integral image*.

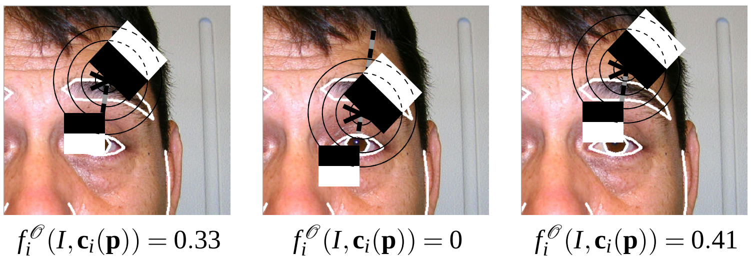
$f^{\varnothing}_i(I, \mathbf{c}_i(\mathbf{p})) = 0.33$
$f^{\varnothing}_i(I, \mathbf{c}_i(\mathbf{p})) = 0$
$f^{\varnothing}_i(I, \mathbf{c}_i(\mathbf{p})) = 0.41$

Figure 5: This figure shows two exemplary Haar-like features (black and white boxes) in the vicinity of the 43rd contour point. This vicinity has been defined to consist of several concentric circles.

Applying this feature mapping $\mathbf{h}$ to the list of examples $[I_k, \mathbf{c}_i(\mathbf{p}_k^*) + \Delta\mathbf{c}_d,$ $f^{\mathcal{O}}(I_k, \mathbf{c}_i(\mathbf{p}_k^*) + \Delta\, \mathbf{c}_d)]$ yields examples of the form $[\mathbf{h}(I_k, \mathbf{c}_i(\mathbf{p}_k^*)+\Delta\mathbf{c}_d)),\ f^{\mathcal{O}}(I_k, \mathbf{c}_i(\mathbf{p}_k^*)+ \Delta\mathbf{c}_d)]$. The formula is getting quite complex, but actually, $\mathbf{h}$ simplifies matters. Since $\mathbf{h}$ returns a list of real number values, one for each Haar-like feature, and $f_i^{\mathcal{O}}$ is also just a real number, the training examples are nothing more than a list of values.

4.4 Training

The objective function f_i^{ℓ} for each contour point, which maps Haar-like feature values to the values returned by $f_i^{\mathcal{O}}$, is learnt by training a model tree [4, 6], with the examples described above. Model trees are a generalisation of decision trees. Whereas decision trees have nominal values at their leaf nodes, model trees have line segments, allowing them to also map features to continuous values. They are learnt by recursively partitioning the feature space. A linear function is fitted to the data in each partition using linear regression. Figure 3f) and 3g) show the training data generated by $f_i^{\mathcal{O}}$, and the plot of a partial objective function f_{43}^{ℓ} that has been learnt for this contour point over all images and deviations.

An important advantage of model trees is that they only use features that are relevant to predicting the target values. This becomes apparent when inspecting the automatically generated model trees. They use just a few features of the entire feature vector $\mathbf{h}$. Apparently, only a few Haar-like features are relevant to predicting the objective value. Interestingly, the objective functions for the different contour points use different subsets of Haar-like features. Due to the partitioning in the model tree algorithm, it is even possible to use different features for different deviations from the contour point. This would mean that for acquiring an initial approximation and fine tuning, different features could be used. We still need to verify this assumption in the learned model trees.

These observations clearly demonstrate the main benefits of our approach. Instead of manually specifying the objective function and the few features it uses, our approach automatically learns, from an abundance of features, which ones are relevant to the objective function. Whereas contemplating exactly which features might be relevant is laborious and the result error-prone, providing a plethora of features whose relevance or irrelevance need not be contemplated is easily done. Furthermore, the selection of features is not based on human intuition, but on the objective information theoretic measures that model trees use to partition the state space. It is not an art, but science.

It also means that different types of features can be combined. Any other salient feature that is representative for the position of a contour point may be used or appended to a feature vector as well. We are currently investigating the combination of edges and Haar-like features in a single learnt objective function.

5 Experimental Results

Current image understanding applications are often subdivided into two operational areas: On the one hand, there are *lifelike scenarios* such as mimic recognition, where the utilised algorithms require robustness towards a lot of variations in the image data, e.g. fitting a face model to an image. One the other hand, there are *industrial scenarios* with highly constrained environments, such as quality analysis on assembly lines. We evaluate our approach to both scenarios.

Concerning the lifelike scenario, we train an objective function that is capable of fitting a face model. We gather about 700 images of frontal faces from the Internet and the TV. Due to their widespread origin, they show large variations to background, illumination, focal length, colour saturation, size, and orientation of the face. We evaluate our approach with a 2D *Point Distribution Model* [1] that consists of 134 contour points. Its parameter vector $\mathbf{p} = (x, y, \alpha, \sigma, \mathbf{b})^T$ contains the model's translation x and y, its rotation α, its scaling σ, and a vector of deformation parameters $\mathbf{b}$. Concerning the industrial scenario, a robot takes 150 images while approaching the kitchenette in our laboratory. We train another objective function to fit a 2D contour model to the stove. The model of the stove is not deformable and contains 84 contour points and its parameters consist of translation and rotation. In this case, the camera parameters, the illumination, and most other environmental parameters are stable. Each image is manually annotated with the desired model parameters $\mathbf{p}^*$, which takes an experienced person less than 20 seconds per image. Since the creation of the deformable face model already required us to manually landmark hundreds of images, those annotated images are now reused, which saves us a lot of time. The feature vectors consist of 50 different Haar-like features of different sizes and different locations around the contour points. We train and test each objective function with hundreds of images, see Table 1.

scenario	# test images	objective function		⊘ I1	⊘ I2a	⊘ I2b	⊘ I3
lifelike	300	designed:	$f^e(I, \mathbf{p})$	0.06	101	1.27	0.19
		learnt:	$f^\ell(I, \mathbf{p})$	0.02	27	1.91	0.57
		optimal:	$f^{\mathcal{O}}(I, \mathbf{p})$	0.00	1	0.00	1.00
industrial	50	designed:	$f^e(I, \mathbf{p})$	0.38	229	0.80	0.56
		learnt:	$f^\ell(I, \mathbf{p})$	0.05	96	0.31	0.85
		optimal:	$f^{\mathcal{O}}(I, \mathbf{p})$	0.00	1	0.00	1.00

Table 1: The average indicator values show the result of our evaluation.

The result of the learnt objective function is compared to two other objective functions: $f^e(I, \mathbf{p})$ and $f^{\mathcal{O}}(I, \mathbf{p})$. We evaluate how well each of them satisfies the requirements R1 – R3. For that purpose, we formulate indicators I1 – I3 that deliver a comparable value for each requirement. They are calculated separately for each image and our evaluation considers their mean values from all test images. It is impossible to evaluate every parameterisation of the model because most model parameters are continuous and the search space is very high dimensional. We randomly assemble model instances and gather them in a set P.

I1: Depending on the position of the global minimum this indicator is set to 0 or to 1.

$$\mathsf{I1} = \begin{cases} 0 & : \quad f(I, \mathbf{p}^*) < f(I, \mathbf{p}) \quad \forall \mathbf{p} \in P \wedge \mathbf{p} \neq \mathbf{p}^* \\ 1 & : \quad f(I, \mathbf{p}^*) \geq f(I, \mathbf{p}) \quad \exists \mathbf{p} \in P \wedge \mathbf{p} \neq \mathbf{p}^* \end{cases}$$

I2: This indicator denotes both the number of local minima and their function value compared to the one of the desired model parameters $f(I, \mathbf{p}^*)$.

$$\mathsf{I2a} = |L| \qquad \textit{where } L = \{\, \mathbf{p} \in P \mid \mathbf{p} \neq \mathbf{p}^* \wedge \mathbf{p} \textit{ is a local minimum}\}$$

$$\mathsf{I2b} = \begin{cases} 0 & : \quad L = \emptyset \\ \frac{1}{|L|} \sum_{\mathbf{p} \in L} \frac{f(I, \mathbf{p}^*)}{f(I, \mathbf{p})} & : \quad L \neq \emptyset \end{cases}$$

I3: This indicator sums up the angles between the gradient vector and the optimal gradient vector which points away from $\mathbf{p}^*$. Remember, the scalar product of two vectors calculates the cosine value of the angle in between.

$$\mathsf{I3} = \frac{1}{|P|} \sum_{\mathbf{p} \in P} \frac{\nabla f^{\mathcal{O}}(I,\mathbf{p}) \circ \nabla f(I,\mathbf{p})}{|\nabla f^{\mathcal{O}}(I,\mathbf{p})||\nabla f(I,\mathbf{p})|}$$

Our evaluation shows that the learnt objective function performs better in the industrial scenario than in the lifelike scenario because of profoundly limited variations to the image data in the industrial scenario. Table 1 illustrates the mean indicator values for both scenarios and compares three objective functions. The indicator values of $f^{\mathcal{O}}(I,\mathbf{p})$ denote the best values to be reached. I1 shows that the global minimum of the design approach is different from the desired position $\mathbf{p}^*$ in many images, whereas the global minimum of our approach is located at the expected position in most images. The learnt objective function produces a low number of local minima, see I2a, and I2b visualises that the function value of those local minima differs greatly from the function value of the desired minimum at $\mathbf{p}^*$. The design approach produces more local minima and, in addition to that, their function values are similar to the one at $\mathbf{p}^*$. Furthermore, our approach shows just a small difference in orientation between the gradient vector and the optimal orientation.

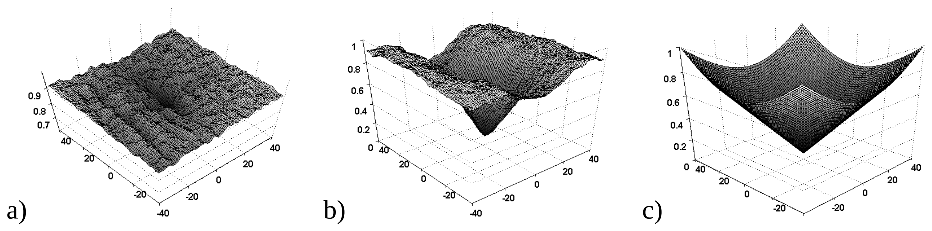

Figure 6: The three objective functions behave differently by changing their position parameters x and y. a) $f^{e}(I,\mathbf{p})$, b) $f^{\ell}(I,\mathbf{p})$, c) $f^{\mathcal{O}}(I,\mathbf{p})$

For visualisation purpose, we take the desired model parameters $\mathbf{p}^*$ and continuously change its position parameters x and y. Figure 6 illustrates three objective functions. The x and y axes indicate the translation and the z axis indicates the value of the objective function. The correlation between an increasing distance from $\mathbf{p}^*$ and an increasing value of the objective functions is visible very well in Figure 6b). The proposed objective function produces a distinct minimum at $\mathbf{p}^*$, which is the position of no spatial translation. Around that position, the function rises constantly and shows no local extrema. It delivers noise when the distance increases beyond a certain range. This range is closely linked to the area $\Delta\mathbf{c}$ that is used for training. In opposite to that, the designed objective function shows several local minima in which fitting algorithms tend to get stuck. Furthermore, those local minima are likely to be as low as the desired minimum at $\mathbf{p}^*$. Gradient vectors of the designed objective function have often random orientation, resulting in a high value for I3. Gradient vectors of the proposed objective function are shorter and they are collectively oriented away from $\mathbf{p}^*$ around the centre. The gradient vectors of the optimal objective function are solely oriented away from $\mathbf{p}^*$. Our evaluation shows that changing entries of the parameter vector other than the translation entries causes similar variations of the result values of the objective function.

We fit a face model on each image of our specific image database and measure the distance between the obtained contour points and the contour points $\mathbf{c}_i(\mathbf{p}_k^*)$. The design approach delivers a mean distance of 30.1 pixels and the learn approach improves this value up to 7.8 pixels.

6 Conclusion and Outlook

In this paper, we have presented a novel approach to learning objective functions. We have defined three requirements an objective functions should meet, and designed a simple *optimal* objective function that meets these requirements. The training examples for learning are manually annotated images, along with the corresponding value returned by the optimal objective function. Model trees then learn a mapping from Haar-like features in the image to the value of the optimal objective function for each contour point. A large number of features can be used, as model tree algorithms only use relevant features.

This methods has several benefits: 1) annotating images is more intuitive and less error-prone than explicitly designing an objective function. 2) having the model tree select only relevant features from a large set is more objective than using only a few based on human intuition. 3) The learned objective function is close to optimal, facilitating the fitting algorithms' task of finding its optimum. Furthermore, we define three indicators that compute how well an objective function meets the specified requirements. In our experimental evaluation, these indicators clearly show that our novel approach outperforms the traditional design approach, and is close to the optimal objective function.

Currently we are integrating edge-based features, and combining them with the Haar-like features. Preliminary results show that, contrary to traditional approach, edge features are hardly used in the learned objective function. Our future work concentrates on applying our approach to other models such as 3D models or texture models. We will also evaluate learned objective functions in the context of tracking algorithms.

References

[1] T.F. Cootes and C.J. Taylor. Active shape models – smart snakes. In *Proc. of the British Machine Vision Conference 1992*, pages 266 – 275. Springer Verlag, 1992.

[2] Robert Hanek and Michael Beetz. The contracting curve density algorithm: Fitting parametric curve models to images using local self-adapting separation criteria. *International Journal of Computer Vision (IJCV)*, 59(3):233–258, 2004.

[3] M. Pantic and L. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2000.

[4] R. Quinlan. Learning with continuous classes. In A. Adams and L. Sterling, editors, *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, 1992.

[5] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001.

[6] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*. Morgan Kaufmann, San Francisco, 2005.