Mapping of facial representations

Damien Firmenich
EPFL
Supervisor: Sofien Bouaziz

1 Introduction

Facial expression representation has been the object of numerous studies over the last years. The problem of realism is of great importance in this field as the human brain is highly trained to recognize even the more subtle change of expression on a character's face, thus every detail needs to be taken into account. It depends not only on how one captures an actor's face, but also on the mapping between a real and a synthetic face. In this project, we focus on how this mapping can be improved, and how it can be generalized so that it doesn't depend on the similarity between the capture method and the animation method.

2 Related work

The research about facial expression mapping between a performer's face and a face model generally relies on a tight connection between how the expression is captured and how the face model is animated. Some of those methods include blendshape interpolation from [CB02]. The performer's facial expression is represented by a weighted combination of blendshapes, which weights will be used with the same set of blendshapes to animate the face model.

Another method from [DMB08] uses feature points. They manually select a set of feature points both on the performer's face and on the face model, with the two sets of points being at the same location on both faces. Using a training set, they train an RBF transformation to approximate the underlying mapping, and then estimate the location of the new feature points based on a sequence of facial expressions.

One of the most recent contribution to this field comes from [KMS11]. Their method consist of representing a facial expression with emotion, speech and eye-blink layers. The performer's facial expression is decomposed into a layered repsentation using a composition function, then mapped to the face model's layered representation and the facial expression is reconstructed using the same composition function. The direct advantage of this method is that it doesn't rely on structural similarities of the two faces.

Another approach consist of representing a facial expression using muscle actuation [CLK01]. They establish a relationship between muscle actuation and facial surface deformation. The trackers on the performer's face track the skin deformation, and they estimate how the muscles moves to perform this deformation. Then they use the inverse relationship to compute how the skin will be deformed on the face model.

3 Regression

In this section, we will review the basics of regression and introduce the methods used in this project. Some methods such as Support Vector Regression (SVR) and Gaussian Process Regression (GPR) where already defined by previous authors and we used them unmodified. However, newer regression techniques are based on Semi-Supervised Learning methods originally developed for classification tasks or data-dimensionality reduction.

Basic linear regression aims to fit a function given a dataset containing examples. Examples are data point (inputs) $\mathbf{x}_i$ and their label (outputs) y_i which are known in advance. The linear function will have the expression $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ so we need to estimate the coefficient vector $\mathbf{w}$. The solution is given my minimizing the sum of squared (as derived in [CHH06])

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \left(\mathbf{w}^T\mathbf{x}_i - y_i\right)^2$$

and the closed form, by evaluating the derivative at zero, gives $\mathbf{w} = \left(XX^T\right)^{-1} X\mathbf{y}$ with the columns of X being the input vectors, and $\mathbf{y}$ the vector of outputs.

3.1 Kernel trick

The kernel trick is a way to perform non-linear regression using a linear regression technique. The input data may not always be represented using a linear function, thus we need to map the input set from the input space to a feature space using a function $\phi(x)$ and compute the regression in the resulting space. Going back to the original input space will lead to a non-linear function. When the regression includes the computation of a dot product, we need to evaluate $\langle\phi(x), \phi(x')\rangle$. This implies the explicit computation of $\phi(x)$ for each input point x, which is performance-wise infeasible for high dimensional data.

The kernel trick allows us to use such a mapping implicitely by defining a kernel function as $k(x, x') = \langle\phi(x), \phi(x')\rangle$ that can be used instead of the explicit dot product. Some commonly used kernels include:

$$\begin{aligned} &\text{RBF (radial basis function):} && k(x, x') = \exp\left[-\frac{||x-x'||^2}{2\sigma^2}\right] \\ &\text{Polynomial:} && k(x, x') = (\langle x, x'\rangle + c)^p \end{aligned}$$

For the kernel function to be used for this purpose, it should satisfy Mercer's condition which are further described in [Mer09].

3.2 Support Vector Regression

The basic idea of SVR is the following. Given a set of training data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, then for any given input $\mathbf{x}_i$ find a function $f(\mathbf{x}_i)$ that has at most ϵ variation from the targets y_i of the training dataset. The function should respect the margin and be as flat as possible.

Such a function does not always exist due to outliers. To deal with this problem, SVR introduces a loss function acting as a soft margin. It enables the algorithm to take into account the outliers by determining a trade-off between the flatness of the function and the tolerance for outliers farther than ϵ.

The statement of the problem is the following [SS04] :

$$\text{minimize } \frac{1}{2}\left|\left|\mathbf{w}\right|\right|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{1}$$

$$\text{subject to } \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b & \leq \epsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases} \tag{2}$$

Where $\mathbf{w}$ is the weight vector we need, C is the trade-off factor and ξ_i, ξ_i^* are the slack variables.

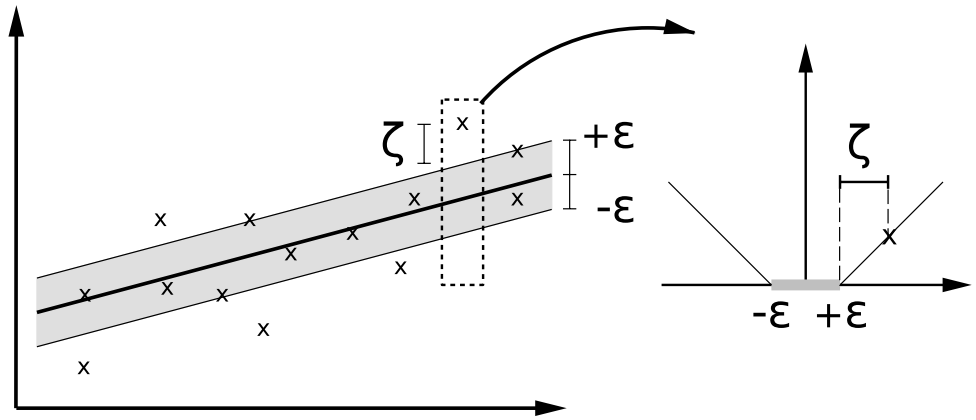

Figure 1: SVR representation with soft margin [SS04]

This formulation applies for linear regression. To extend it to non-linear regression, SVR uses the kernel trick defined in Section 3.1. To use it, the problem must be formulated using dot products, and for this we need to use the dual problem, which the derivation is explicited in [SS04]. The formula for the function $f(x)$ is the following.

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle \mathbf{x}_i, \mathbf{x} \rangle + b$$

where α_i, α_i^* are the Lagrange multipliers for the constraints.

Thus, the function can be computed using only the dot product, and using a kernel gives the following formula.

$$f(\mathbf{x}) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)k(\mathbf{x}_i, \mathbf{x}) + b$$

with $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

The formulation of the problem can also be expressed using a more familiar minimization of a cost function. The expression (1) can be written as the minimization of

$$\sum_i |y_i - f(\mathbf{x}_i)|_\epsilon + \frac{\lambda}{2}\left|\left|\mathbf{w}\right|\right|^2$$

where the factor λ is the regularization constant. In this case, the trade-off factor will be $C = \frac{1}{\lambda l}$, l being the number of labelled data. The cost function $|y_i - f(\mathbf{x}_i)|_\epsilon$ reflects the margin ϵ in which errors are tolerated, where the minimization of the distance between the target output and the estimation has no effect until it reaches ϵ.

3.3 Semi-supervised - Graph-based

Most of the regression methods require a fairly large amount of labelled data to give an accurate estimation. However, most data that we can gather are not labelled and requires often tedious manual work to label them. Recent research investigate a new type of learning named Semi-supervised Learning (SSL), which can accurately predict an output based on large amount of data and few labels only.

One type of SSL algorithm represents the dataset as a graph and uses its topography to perform a more accurate mapping from an input to an ouput space. Intuitively, a graph-based using the Laplacian of the graph looks for points in the dataset that are close to each other, and tries to keep their mapping close to each other too. It uses a graph regularization framework in which the dataset adjacency graph is used as weights between estimates of neighboring points.

From section 3, basic linear regression with a Tikonov regularizer is expressed as

$$f^* = \arg\min \left(\sum_{i=1}^{l} \left(f(\mathbf{x}_i) - y_i \right)^2 + \alpha \left|\left| f \right|\right|^2 \right).$$

For a graph-based approach, we use the edge weights in the data adjacency graph W where W_{ij} is a measure of the distance between points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the dataset. We want to minimize the estimates distance of close points, thus the weights W are added to the regression expression and we have

$$f^* = \arg\min \left(\sum_{i=1}^{l} \left(f(\mathbf{x}_i) - y_i \right)^2 + \alpha \left|\left| f \right|\right|^2 + \beta \sum_{i,j=1}^{N} \left(f(\mathbf{x}_i) - f(\mathbf{x}_j) \right)^2 W_{ij} \right).$$

When using a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the estimated coefficients $\mathbf{w}^*$ are expressed as

$$\mathbf{w}^* = \left(X_1 X_1^T + \alpha I + \beta X L X^T \right)^{-1} X_1 \mathbf{y}$$

where the columns of X corresponds to the data points, the columns of X_1 to the labelled subset and $\mathbf{y}$ to their labels. L is the graph Laplacian where $L = D - W$ and the diagonal matrix $D_{ii} = \sum_{j=1}^{M} W_{ij}$.

When using a non-linear mapping, the function is expressed using a kernel such that $f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}, \mathbf{x}_i)$ and thus the expression of the coefficients α is now

$$\alpha^* = \left(K_1 K_1^T + \alpha I + \beta K L K^T \right)^{-1} K_1 \mathbf{y}$$

where the matrix K is the kernel representation of the dataset $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ and K_1 is the kernel representation for the labelled subset.

3.4 Locally Linear Embedding

Locally Linear Embedding (LLE) is a semi-supervised learning algorithm which aims to improve nonlinear dimensionality reduction by identifying the topography of the dataset.

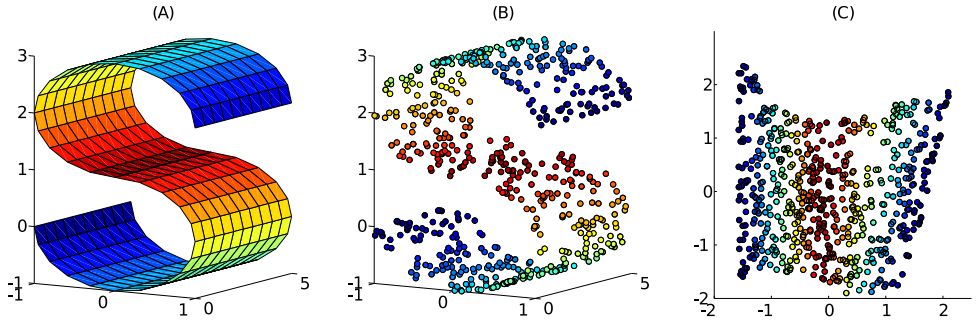

Figure 2: LLE representation [SR00]

When the data points lies on a manifold (e.g. Figure 2), they have a locally linear structure on a small patch of the manifold. LLE tries to describe this linear structure by expressing each of the data point as a linear combination of its neighbors. It assigns a set of weights to each point, representing those of the linear combination, and uses the sames weights to express the point in a lower dimensional space, thus keeping the linear structure of the manifold. In this project, the target dimensionality of the data is that of the target function, in other words the dimension of the parameters of the face mesh. An overview of the method is illustrated in Figure 3.

The problem is solved in the following way. First, we extract a set of neighbors from each point, either by choosing the K-nearest neighbors, by selecting points within a fixed radius, or by using any other appropriate method. Then, we find a set of weights that best describe the original point $\mathbf{x}_i$ using its neighbors $\mathbf{x}_j, j \in J_i$. This corresponds to minimizing the squared distance between the data points and their reconstructions $\mathbf{x}_i^* = \sum_{j \in J_i} W_{ij}\mathbf{x}_j$ (as defined is [RS00]) :

$$\min_W \left|\left| \mathbf{x}_i - \sum_{j \in J_i} W_{ij}\mathbf{x}_j \right|\right|^2$$

with W being the matrix of the weights to estimate. One constraint to this minimization is that if the point $\mathbf{x}_j$ is not a neighbor of $\mathbf{x}_i$, then $W_{ij} = 0$. Another constraint is required for the reconstruction of the points to be invariant to rotation and translation. By keeping the sum of the weights equal to one, $\sum_j W_{ij} = 1$, we ensure translation invariance and we can rewrite the minimization as

$$\min_W \left|\left| \sum_{j \in J_i} W_{ij}(\mathbf{x}_j - \mathbf{x}_i) \right|\right|^2 = \min_W \; ||W_i G_i||$$

with $G_i = [\ldots, \mathbf{x}_j - \mathbf{x}_i, \ldots]$ the distances of the neighbors to the original point. Now solving this constraint least squares problem is expressed as $W_i = (G^T G)^{-1}\mathbf{1}$ where $\mathbf{1}$ is the vector of all ones and W_i the set of weights for the point x_i.

When the problem is ill-posed (the covariance matrix $G^T G$ has small or zero-valued singular values), we can use a regularization framework, in this case the Tikhonov regularization [Tik63] which corresponds to minimizing the problem

$$\min_W \left|\left| \mathbf{x}_i - \sum_{j \in J} W_{ij}\mathbf{x}_j \right|\right|^2 + \alpha \sum_{j \in J_i} W_{ij}^2$$

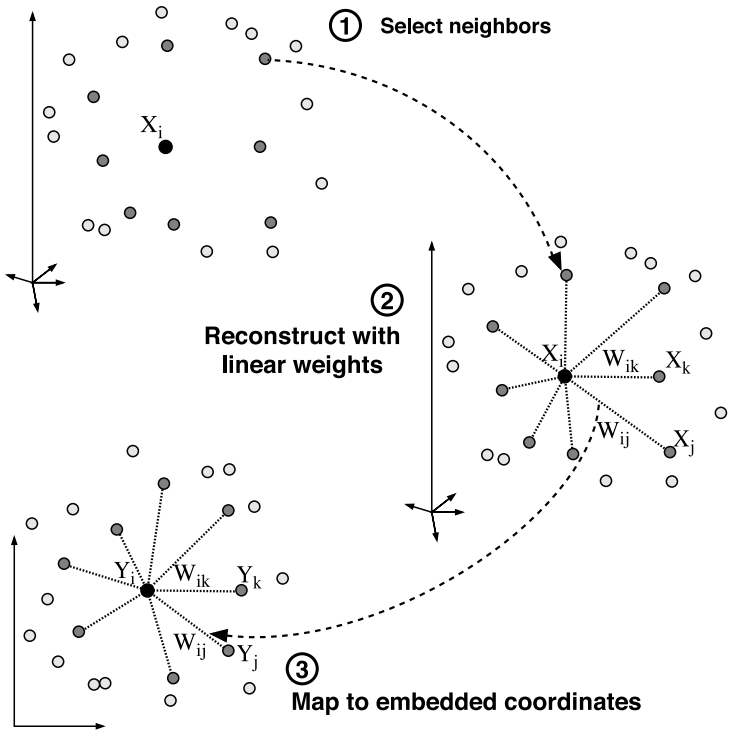

Figure 3: LLE algorithm [RS00]

and the solution is expressed as $W_i = \left(G^T G + \alpha \mathbf{I}\right)^{-1} \mathbf{1}$. The choice of the regularization parameter α can have important effect on the estimation, as explained in the next section.

Once the weights are estimated, we need to find the representation of the point in output space. This corresponds to minimizing the following expression

$$\Phi(Y) = \sum_i \left|\left| \mathbf{y}_i - \sum_{j \neq i} \mathbf{y}_j W_{ij} \right|\right|^2$$

where $\mathbf{y}_i$, the columns of Y are the mapping of $\mathbf{x}_i$ and W the matrice of the weights computed above. Since the problem has an infinity of solutions, we impose the constraint $\frac{1}{n}\sum_i \mathbf{y}_i = 0$. Expanding the minimization expression above gives $\Phi(Y) = Y^T H Y$ where $H = (I - W)^T (I - W)$.

3.5 Modified Locally Linear Embedding

When the dimension of the manifold is larger that one, LLE is not always stable and can produce distorted low-dimensional embedding [ZW07]. When the problem is ill-posed, the estimated solution depends on the choice of the regularization parameter. If it is not suitably selected, the solution may not be a good approximation. Modified Locally Linear Embedding (MLLE) aims to solve this problem by keeping multiple local weight vectors instead of having only one estimation, and it ensures that the reconstruction of the low-dimensional embedding is more stable.

The specification of the method involves selecting the number of optimal weight vectors using the singular values of $G^T G$, and requires condition that are not suited to our problem. The initial intention of MLLE is to perform a dimension reduction, but in our case, the output dimension is greater than the dimension

of the input space and it causes problem with the implementation of MLLE. Thus this problem may be further investigated in future work.

3.6 Regularization

In our case, the dimension of the input space is high dimensional, so a function in this space as many degrees of freedom. We want to avoid overfitting if the problem is ill-posed, i.e. if the function does not have a unique solution. For the purpose of this project, the number of examples given to the regression algorithm is lower than the number of unknowns for the function. Hence, we can use a regularization framework to have a better estimation of the function. In this case, we use a regularization by semi-supervised methods such as the two described above.

We can use three methods to perform the regression, either we use only SVR, or we regularize it with the graph-based Laplacian or the LLE method. Using SVR only means that we will fit a function using only the knowledge we have about the correct mapping, in other words, we will use only labelled data. This may not provide a good approximation as the training set may not have information about the whole range of possibilities for the user's facial expressions. Using a regularization with the graph-based Laplacian methods is more interesting because it uses knowledge about the dataset structure. The algorithm will promote solutions which minimizes the output distance of points that are close to each other.

However, using a LLE regularization has the advantage of keeping more of the structure of the dataset to estimate the outputs. By representing points in the dataset as a linear combination of its neighbors and keeping the same weights to approximate the output point, the mapping will keep the local structure around the point. It is invariant to translation and rotation, but only allows scaling around the point. The expression of the closed form solution using an LLE regularization is the following

$$\mathbf{w} = (X_1 X_1^T + \alpha \mathbf{I} + \beta X H X^T)^{-1} X_1 \mathbf{y}$$

where $\mathbf{w}$ is the estimation of the weights of the mapping $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, the columns of X_1 are the input points, H relates to the estimation of the mapping using LLE (see bottom of Section 3.4). For a non-linear mapping, we use the matrice K and K_1 in place of X and X_1, which are the kernel representation of the dataset and the training set, i.e. $K_{ij} = k(\mathbf{x_i}, \mathbf{x_j})$ (see Section 3.1).

4 Workflow

To evaluate the performance of the methods explained in Section 3, we need to capture a facial expression from an actor and map it onto a virtual face model. An overview of the workflow we used for this purpose is shown in Figure 4. First, the facial expression is captured and expressed as a set of coefficients, then we build a training set using some reference frames of the captured sequence, which will be used together with the input dataset (the coefficients) to perform the regression. The output of the regression will be the parameters of the model that will animate the character according to the captured sequence.

4.1 Facial capture

The first step towards mapping facial representations between an actor and a face model is to capture the facial expression of the actor. The scope of this project is to map a set of input coefficients to

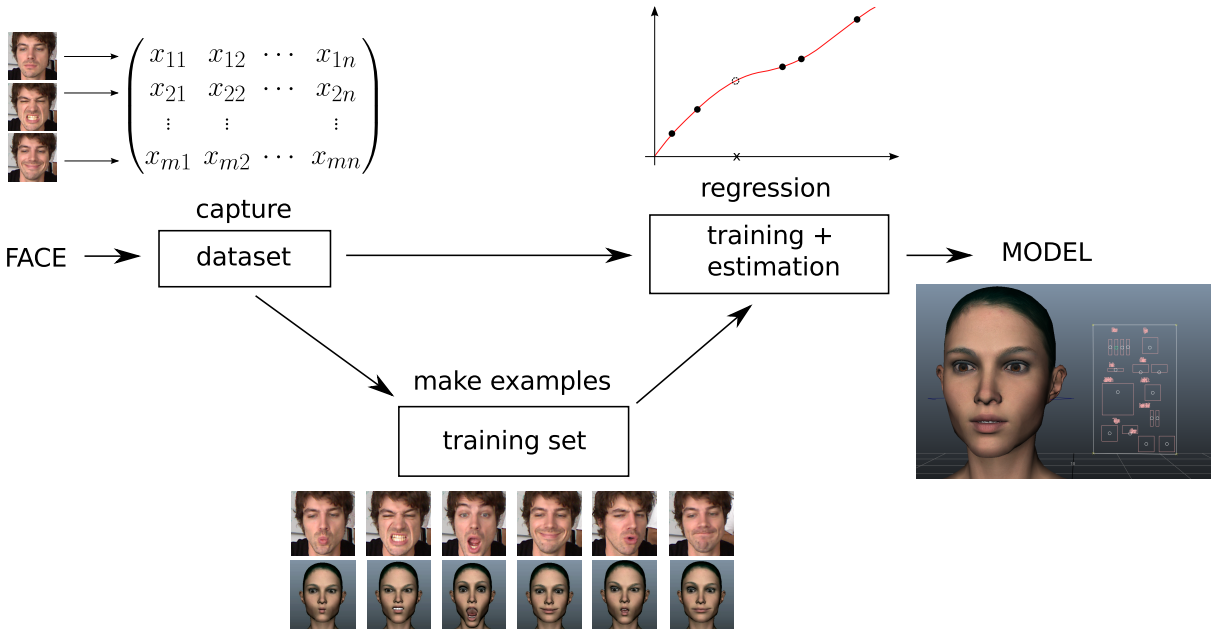

Figure 4: Workflow summary

corresponding parameters on the face model, so it assumes a previously recorded sequence of facial expressions together with a numerical representation of those expressions.

To extract those input coefficients, we use a realtime tracking method by [WBLP11]. It uses a 3D depth map and a 2D images sequence provided by a Kinect system to estimate the coefficients of an underlying blendshape representation of the actor's facial expression. First, an offline preprocessing step is needed to provide a efficient representation of the actor's expression space. This expression model is computed by adapting generic blendshapes with some example expressions performed by the actor. For the realtime expression tracking, the face is decomposed to rigid and non-rigid motion and the blendshape weights are directly estimated from the non-rigid motion.

For each frame of the captured video sequence, this methods provides a set of 39 coefficients describing the facial expression in those frames. This constitutes the input dataset of the regression methods defined above.

4.2 Example set

The regression methods decribed in Section 3 require a set of labelled data points (examples) to estimate new output points. As the target is a 3D mesh, we need to make initial correspondences between the real face from the video and the face of the mesh.

From the video captured in the previous section, we selected a small numbers of frames (between 8 to 15) to use as examples. Those frames need to be representative of the range of facial expressions to be reproduced, i.e. they should contain movements from all components of the face. Once appropriate frames are selected, we accurately pose the 3D mesh according to each of the faces represented in the frames. The mesh is manipulated by intuitive handles and sliders, each of them connected to a specific component of the mesh's face.

The example frames together with their corresponding mesh pose (a sample is shown in Figure 5) will form the training set used to train the regression algorithm.

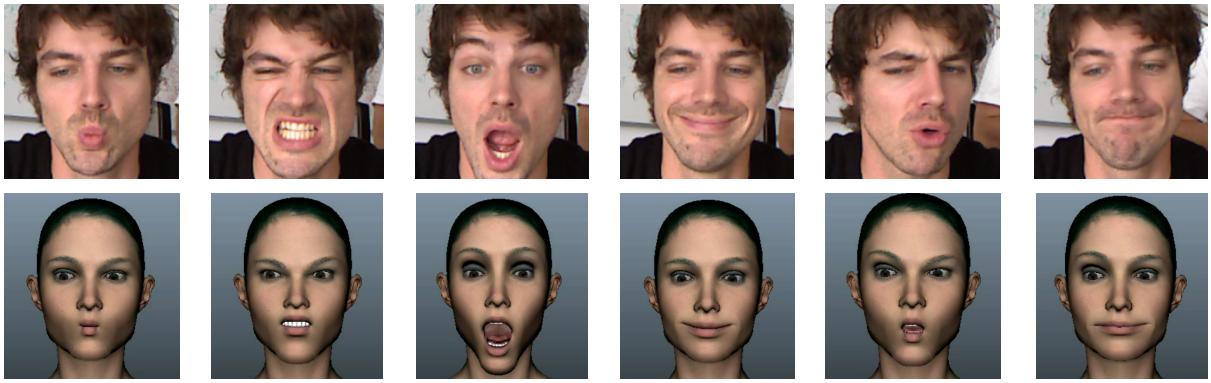

Figure 5: Example set

4.3 Regression

For the purpose of this project, we need to map a set of parameters corresponding to the real face, to a set of parameters correponding to the 3D face. As mentioned in the previous section, the mesh is parametrized by control parameters (handles and sliders) which are internally linked to components of the face (uniquely to each mesh type). For example, a slider is connected to the openness of the mouth, and a handle is connected to the direction of the eyes. Those parameters are extracted from the mesh and, for each frame of the animation, their form a n-dimensional vector used as the target output for the algorithm.

The input parameters corresponds to the coefficients of the real face extracted as explained in Section 4.1. And the labelled data corresponds to the pairs of parameters matched as in the previous section. We use the two sets of parameters to perform the regression using the methods explained in Section 3.6 and the estimated mesh parameters are then mapped to the face mesh, which constitute the animation corresponding to the captured video. For each of the regression method, the output were estimated using linear regression as well as using a kernel.

4.4 Technical details

In this subsection, we will briefly explain the implementation and the tools used for this project.

The 3D mesh were manipulated in Maya (Autodesk) using models found on websites [Im11, Cra11]. The animation data were stored in a `.anim` file which contains a set of parameters corresponding to those of the rigs, and for each of them, their values for each frame of the animation.

5 Experimental Results

The previous workflow has been done using the three methods cited in Section 3.6 : SVR, SVR regularized by graph-based Laplacian and by LLE. Using SVR alone lead to appreciable results from the start, but

suffered from minor shortcomings. It uses only labelled data so it works best when the training set in as complete as possible. Due to this fact, the algorithm also implicitely inferred some inexisting correlation such as the opening of the mouth and the movement of the eyebrows. An example of this problem is shown in Figure 6. This problem could be partially solved by separating the regression of the upper and the lower part of the face, but this implies some parameter-dependent and semantic modification of the algorithm and the dataset.

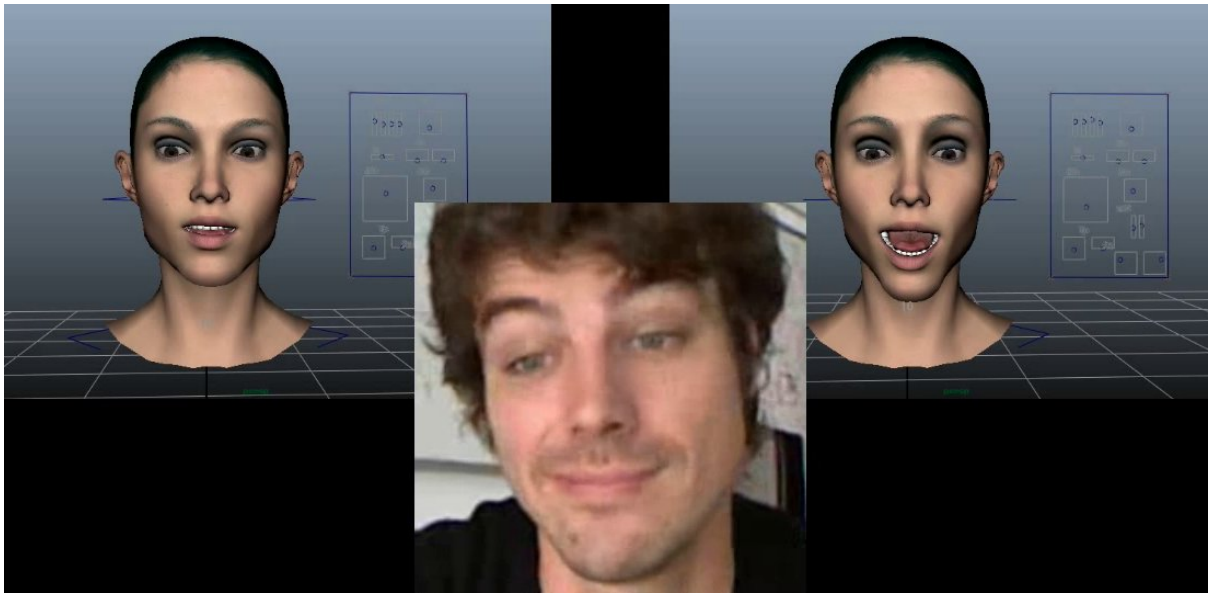

Figure 6: SVR fails to separate the mouth and the eyebrows from the example set. The algorithm uses non-linear regression (kernel trick) on the upper left side, and linear on the upper right side (original in the middle)

Using a regularization by semi-supervised methods showed some improvement of the realism of the facial expressions. The animation overall is more stable and correlates more with the original sequence. This is especially true when the regression uses less labelled data. As we reduce the number of labels, SVR only interpolates between know expressions whereas regularization using LLE shows more ability to represent the space of facial expressions that the user can create. An example of result using this method (with non-linear regression using an RBF kernel) is shown in Figure 7.

The tests were done, for each method, linearly and with an RBF kernel. Although in general the use of a kernel version lead to more realistic results, its effect was more noticable with SVR, where it allowed the mapping to be more flexible on small variations of the expression.

The evaluation of the realism is tricky for this work as we don't have ground truth data about how the face model is supposed to react exactly. Thus, the results were evaluated using perceptual differences between the methods. Also, it is worth noting that the rigid motion of the head (head movements) were not taken into account.

6 Conclusion

We investigated the problem of mapping an expression from an actor's face to a synthetic face model. As a general regression problem, many existing techniques could be used as a starting point, but we focused

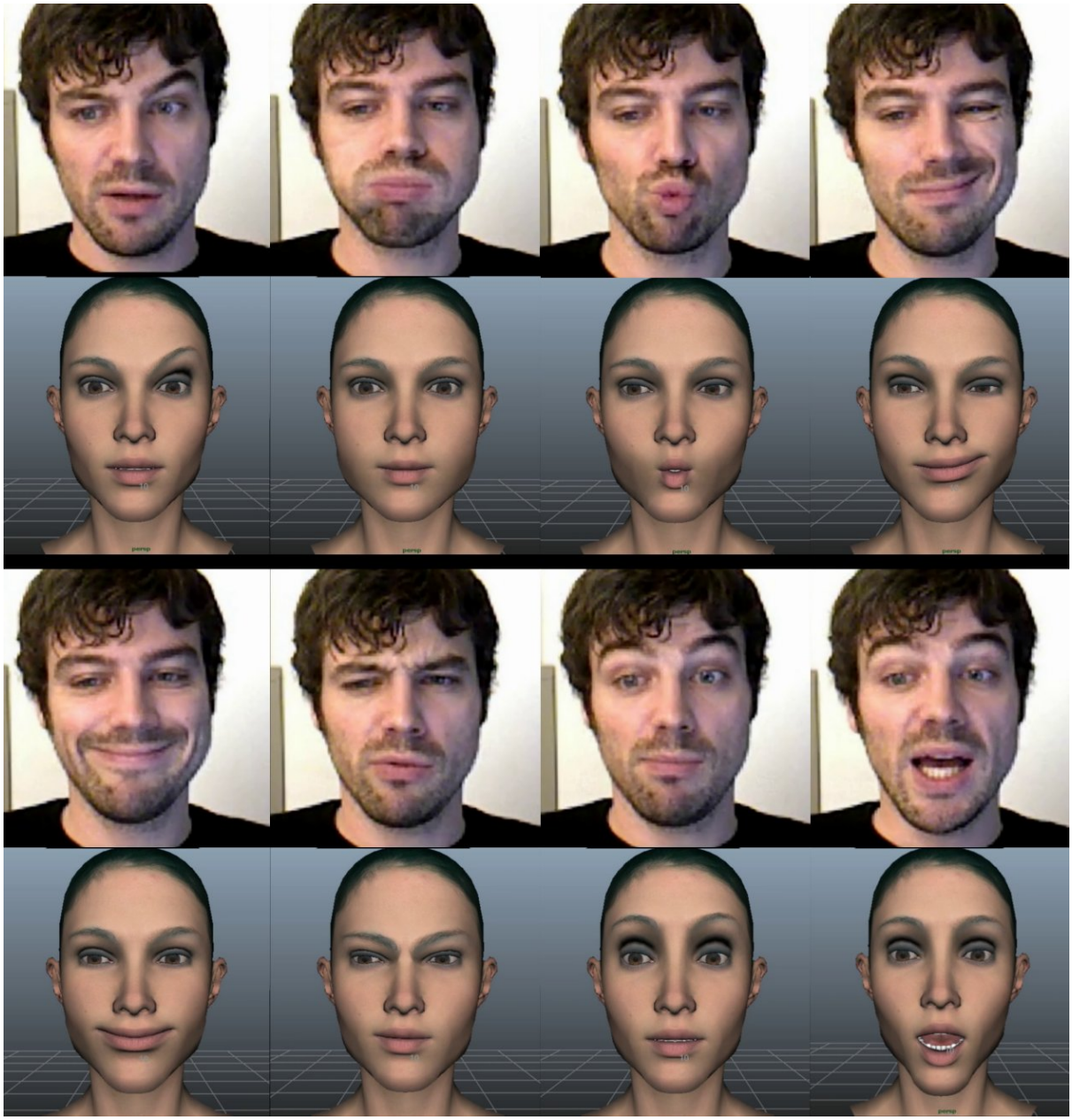

Figure 7: Snapshots from a sequence using LLE and RBF kernel

on Support Vector Regression and its modification using Semi-supervised Learning methods. The theory implies a more accurate approximation using those methods, as they make efficient use of the non-labelled data, unlike SVR which tries to fit a function based on labelled data only. More particularly, using LLE leads to a better approximation than the graph-based Laplacian methods as it keeps the local structure of the dataset, and not only tries to minimize the output distance of input points close to each other. Experimental results confirmed this, where the animation where more realistic even with a lower number of labels.

References

[CB02] E. Chuang and C. Bregler. Performance driven facial animation using blendshape interpolation. *Computer Science Technical Report, Stanford University*, 2(2):3, 2002.

[CHH06] D. Cai, X. He, and J. Han. Semi-supervised regression using spectral techniques. 2006.

[CLK01] B. Choe, H. Lee, and H.S. Ko. Performance-driven muscle-based facial animation. *The Journal of Visualization and Computer Animation*, 12(2):67–79, 2001.

[Cra11] Creative Crash. Alice, http://www.creativecrash.com/maya/downloads/character-rigs/c/alice–2, 2011.

[DMB08] L. Dutreve, A. Meyer, and S. Bouakaz. Feature points based facial animation retargeting. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, pages 197–200. ACM, 2008.

[Im11] Image-metrics. Ilana, http://www.image-metrics.com/community/forumdisplay.php?26-character-rigs, 2011.

[KMS11] N. Kholgade, I. Matthews, and Y. Sheikh. Content retargeting using parameter-parallel facial layers. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 195–204. ACM, 2011.

[Mer09] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209:415–446, 1909.

[RS00] S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323, 2000.

[SR00] L.K. Saul and S.T. Roweis. An introduction to locally linear embedding. *unpublished. Available at: http://www. cs. toronto. edu/˜ roweis/lle/publications. html*, 2000.

[SS04] A.J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[Tik63] A.N. Tikhonov. Regularization of incorrectly posed problems. In *Soviet Math. Dokl*, volume 4, pages 1624–1627, 1963.

[WBLP11] T. Weise, S. Bouaziz, H. Li, and M. Pauly. Realtime performance-based facial animation. In *ACM Transactions on Graphics (TOG)*, volume 30, page 77. ACM, 2011.

[ZW07] Z. Zhang and J. Wang. Mlle: Modified locally linear embedding using multiple weights. *Advances in Neural Information Processing Systems*, 19:1593, 2007.