Bachelor Thesis

Articulated Motion Estimation
Implementation and Evaluation

Thomas Leichtle

Ulm University
Institute of Neural Information Processing
89069 Ulm, Germany
Thomas.Leichtle@uni-ulm.de

Supervisors:

Prof. Dr. Heiko Neumann
Dipl.-Inf. Georg Layher

June 10, 2013

Abstract

On the basis of the work of Ju, Black, Yacoob [JBY96] and Datta, Sheikh and Kanade [DSK11], we try to estimate articulated motion, especially human motion, from image sequences. It's done by a gradient-based method with an affine transformation model. The limbs are approximated by quadrangles like in [JBY96]. We use different kinds of constraints to influence the calculations, like [JBY96] and [DSK11].
We show that larger transformations can be found using constraints. We analyze the influence of textures and noise and apply the method on animated image sequences.

Eigenständigkeitserklärung

Contents

1 Introduction

As the computing power of even small and cheap devices with cameras increases continuesly, this technology offers exciting and new possibilities. Tracking objects or people is one of them. Since this can be used for action recognition, it establishes a completely new way of user interface, where gestures control the device. We can also think of application in a different area like medical science. If tracking for body parts works well enough, it could support doctors at diagnosis. Another use case is computer graphics. By getting information about articulated motion, we can animate image sequences more accurate.

There are mainly two different kinds of approaches for tracking: feature tracking and gradient-based ones. In this paper we look for a way to improve tracking for articulated motion with the second. Therefore we implement and analyze the idea of constraints from Ju, Black and Yacoob mentioned in [JBY96] and from Datta, Sheikh and Kanade in [DSK11]. We add another constraint by ourselves to improve the approach even further.

To do this, we first introduce the mathematical background in chapter 2. It includes the explanation of different motion models, a method for image generation of articulated data and the basic idea of gradient-based motion estimation. In chapter 3 our implementation for the approach is described. We also derive there some formulas and explain the constraints in more details. Chapter 4 shows the evaluation of the results. It is done in multiple steps. First we apply the algorithm on artificial data to take a look on the behavior and important properties (section 4.3). Thereafter we examine the results on animated motion sequences.

2 Mathematical basics

Each image is represented as an array of pixels. Each pixel has some coordinates $\begin{pmatrix} x & y \end{pmatrix}^T$ and a color value at each position $I(x, y)$. Since we work on grayscale images, this value is a scalar for its intensity. In homogeneous coordinates, they are extended to $\begin{pmatrix} x & y & 1 \end{pmatrix}^T$ to describe transformations more easily.

2.1 Transformation models for motion

We need a model to estimate motion. Therefore we use the following descriptions. A transformation describes the alteration of an image caused by motion. Since we assume that the brightness constancy assumption applies, the color values of a pixel $I(x, y)$ just move to another position $\begin{pmatrix} x' & y' \end{pmatrix}^T$ in the image. This procedure can be written as a matrix vector multiplication:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \mathbf{A} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

while $\mathbf{A}$ is the transformation matrix (see below). By this, the reference point of the transformation is automatically the origin of the coordinate system of x and y. It's also possible to split up the image into various parts and apply different transformations to each of them.

List of typical 2d transformation models:

Translation: The translation model has only two parameters, namely the translation in x-/y-direction t_x and t_y or often called u and v.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rigid: In this model, additionally to t_x and t_y, there is a parameter for rotation ϕ. $a = \cos\phi$, $b = (-)\sin\phi$ for clockwise (/counterclockwise) rotation of the image.

$$\mathbf{A} = \begin{pmatrix} a & b & t_x \\ -b & a & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Rigid with scaling: This model has four parameters, t_x, t_y, ϕ, s. It can describe translation, rotation and global scaling. A scaling with s means the same as multiplying the matrix with $\frac{1}{s}$, because the resulting coordinates need to be normalized to $\left(x' \; y' \; \mathbf{1} \right)^T$.

$$\mathbf{A} = \begin{pmatrix} a & b & t_x \\ -b & a & t_y \\ 0 & 0 & s \end{pmatrix}$$

Affine: This 6 parameter model explains translation, rotation, scaling and shear. It preserves collinearity, i.e. all points lying on a line initially still lie on a line afterwards. a, b, c, d are combinations of different single transformations like rotation and shearing.

$$\mathbf{A} = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Projective: The projective transformation model is the most powerful described here. It has 9 degrees of freedom and can model all transformations mentioned above and more.

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

2.2 Kinematic chain for image generation

In order to evaluate our algorithm, we need to generate data for which the ground truth is known. Therefore we use the same model of the human body (cardboard people [JBY96]) as for the estimation in our algorithm. So we have a collection of connected patches - one for each limb or part of the body, we want to model.
If one moves his upper arm, the lower arm moves as well, because they're connected at the elbow. To copy this behavior in the synthetical data generation, we use a kinematic chain (see [JNW06]). We represent each part of the body as a local coordinate system with the origin at its joint. Choosing one patch as the root results in a hierarchy and we can define the systems relatively to their respective parent. If we then apply a transformation to one patch, we consecutively apply it also to all its successors.

A small example:
Assume we have three transformations T_1, T_2, T_3 for the patches P_1, P_2, P_3 with their coordinate origins O_1, O_2 and O_3.

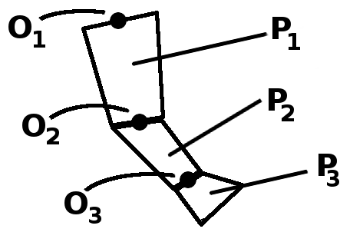

Figure 2.1: Hierachy of the patches. P_1 is the root patch with its reference point O_1. P_2 is second in hierarchy with O_2 as reference point and P_3 as the lowest with joint O_3.

1. Apply transformation T_1 on patch P_1 with coordinates with respect to origin O_1. We'll write this as $T_1(P_1^1)$. Since the other patches are connected, they move as well. They change the same way around the same origin $\rightarrow T_1(P_2^1)$, $T_1(P_3^1)$.
2. Again the same with $T_2 \rightarrow T_2(P_2^2)$, $T_2(P_3^2)$.
3. And the last transformation T_3 only on $P_3 \rightarrow T_3(P_3^3)$

If we try to calculate the transformation of each patch with respect to its joint, we'll get something similar to T_1 for P_1, but combined results for the others. Even if we choose $T_2 = T_3 = id$ (identity) and T_1 as a simple rotation, we'll get some rotation and translation for the estimated transformations T_2', T_3' of P_2 and P_3.

2.3 Gradient-based motion estimation

In this paper, we use a gradient-based method for motion estimation. While feature tracking approaches extract distinctive points from images and try to match them to calculate the motion of an object, the basic idea behind this approach is different. We create a connection between the spatial and temporal image derivations to estimate a transformation. Here's a small example in one-dimensional space:

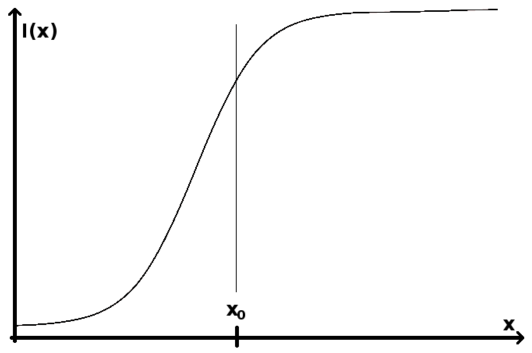

Figure 2.2: Intensity function at $t = 0$

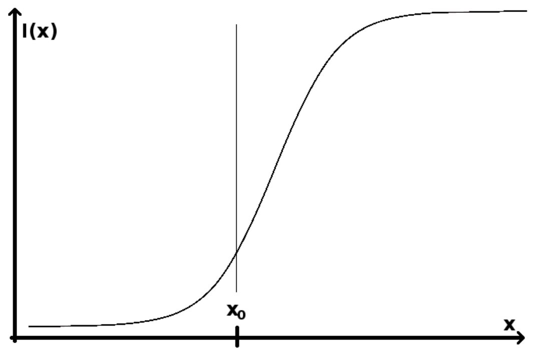

Figure 2.3: Intensity function at $t = 1$

In this example $\left.\frac{\partial I(x,t)}{\partial x}\right|_{x=x_0} > 0$ and $\left.\frac{\partial I(x,t)}{\partial t}\right|_{x=x_0} < 0$ holds.

There are four possible combinations for the signs ($\neq 0$) of the derivations and we can distinguish whether there was a motion to the right or to the left. There's even a relation between the ratio of the derivations and the amplitude of the motion. Since we assume that the brightness constancy assumption holds, it follows

$$I(x,t) = I(x + \Delta x, t + \Delta t)$$

which we can write by linear taylor approximation as

$$I(x,t) = I(x,t) + \frac{\partial I(x,t)}{\partial x} \cdot \Delta x + \frac{\partial I(x,t)}{\partial t} \cdot \Delta t$$

After rearranging, we get

$$\frac{\Delta x}{\Delta t} \frac{\partial I(x,t)}{\partial x} = -\frac{\partial I(x,t)}{\partial t}$$

If Δx and Δt have the same dimension and we apply $\lim_{\Delta t \to 0}$, we get $\frac{\partial I(x,t)}{\partial x} \frac{dx}{dt} = -\frac{\partial I(x,t)}{\partial t}$ and with $u := \frac{dx}{dt}$, $I_t := \frac{\partial I(x,t)}{\partial t}$, $I_x := \frac{\partial I(x,t)}{\partial x}$ follows:

$$I_x \, u = -I_t \quad \Rightarrow u = -\frac{I_t}{I_x}$$

This u is the 1-d motion vector. We can do the same calculation for 2-d by starting with $I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$. The problem is, that there's only one equation ($I_x \, u + I_y \, v = -I_t$) per pixel, but two unknowns (u, v = optical flow). This is called the aperture problem. When we try to fit a different transformation model besides the translation (see section 2.1), we get even more unknown variables. To solve this, we have to take a neighborhood and create a system of equations. In our case, we define the patch that approximates the limb, as the neighborhood.

3 Algorithm for articulated motion estimation

With the background from chapter 2 we implement an algorithm for articulated motion estimation in MATLAB. It fits an affine transformation model for motion of selected patches. There are different variations of the algorithm: One without constraints, two with exact/approximate position (articulation) constraints and two with exact/approximate position and the respective width constraints.

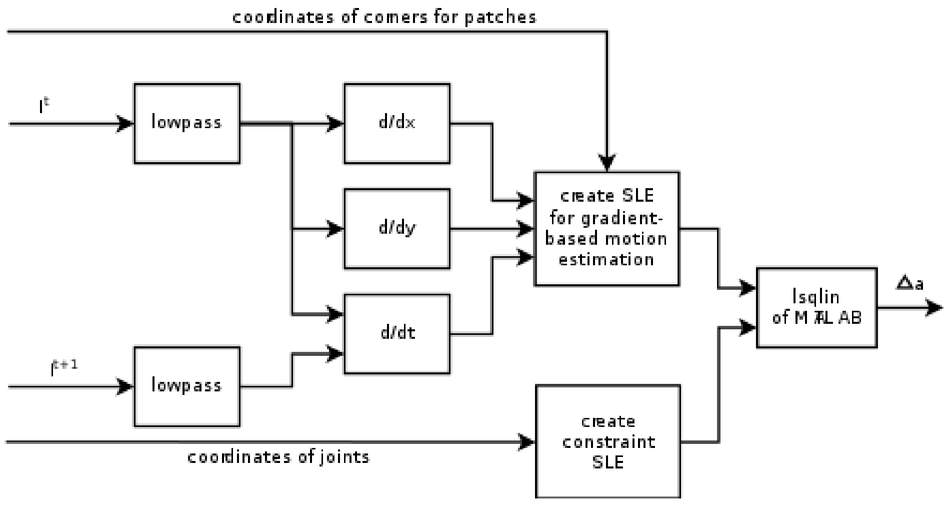

Figure 3.1: Flowchart for one iteration of the algorithm.
Inputs: I^t, I^{t+1}, coordinates of corners for patches and joint positions. Outputs: Δa

The input images can be any two frames (of an image sequence). We labeled the corners of the patches manually, but it is also possible to find them by some preprocessing steps. The same holds for the joints.

First, we lowpass filter the images with a gaussian. Then we calculate the spatial derivations for the first image and the temporal derivation of the pair to create a system of linear equations (SLE) for gradient-based motion estimation. The spatial derivations are calculated by a normalized sobel mask; the temporal derivation by subtracting I^t from I^{t+1} and it is smoothed over a 3x3 neighborhood. Afterwards we create the SLE consisting of C, d as described in 3.1 and depending on the variation, the SLE for the constraints with M, b as in 3.2.

We solve the SLE $Cx = d$ s.t. $Mx \; op \; b$ with $op \in \{\leq, =\}$ (depending on the variation) by the lsqlin package of MATLAB. For evaluation, we repeat this process n times and break afterwards. In real application we might set a threshold for the transformation changes (since we don't get the correct transformation at once, but only a small change per iteration).

3.1 System of equations for gradient-based approaches

We use the equation $I_x\ u + I_y\ v = -I_t$ from section 2.3 for creating the SLE. Since we use the affine transformation model, we rewrite this (because $u = (x' - x)$ & $v = (y' - y)$) to:

$$I_x \cdot (x' - x) + I_y \cdot (y' - y) = -I_t$$

with $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$ and the warped positions $(x', y', 1)^T = A\ (x, y, 1)^T$. Our transformation matrix is $\mathbf{A} = \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \end{pmatrix}$ and we get:

$$\begin{aligned} I_x\ (xa_1 + ya_2 + a_3 - x) + I_y\ (xa_4 + ya_5 + a_6 - y) &= -I_t \\ I_x x a_1 + I_x y a_2 + I_x a_3 - I_x x + I_y x a_4 + I_y y a_5 + I_y a_6 - I_y y &= -I_t \\ \begin{pmatrix} I_x x & I_x y & I_x & I_y x & I_y y & I_y \end{pmatrix} \begin{pmatrix} a_1 - 1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 - 1 \\ a_6 \end{pmatrix} &= -I_t \end{aligned}$$

$a_1 - 1$ and $a_5 - 1$ remove the bias to ensure that all values are approximately around 0. For each pixel, we get one equation for the system which we denote $Cx = d$. That means we need at least six pixels for the affine model, but usually we've got a lot more. The positions (x, y) are calculated with respect to the joint with the patch above (patch hierarchy, see section 2.2) since we choose it to be the reference point for the transformation. For the root patch, we define it on our own.

So far, this approach would work for each patch on its own. But we want to connect the patches, so we have to put all equations into one SLE and calculate all transformations at once. Therefore we denote $\mathbf{a^i} = \begin{pmatrix} a_1^i & a_2^i & a_3^i & a_4^i & a_5^i & a_6^i \end{pmatrix}^T$ the parameters of the $i-$th patch with n_i pixels in it and x^i, y^i the coordinates in respect to the transformation origin of the $i-$th patch.

We get to the following system, i.e. for two patches:

$$\begin{pmatrix} I_x x_1^1 & I_x y_1^1 & I_x & I_y x_1^1 & I_y y_1^1 & I_y & 0 & 0 & 0 & 0 & 0 & 0 \\ I_x x_2^1 & I_x y_2^1 & I_x & I_y x_2^1 & I_y y_2^1 & I_y & 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ I_x x_{n_1}^1 & I_x y_{n_1}^1 & I_x & I_y & I_y y_{n_1}^1 & I_y & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I_x x_1^2 & I_x y_1^2 & I_x & I_y x_1^2 & I_y y_1^2 & I_y \\ 0 & 0 & 0 & 0 & 0 & 0 & I_x x_2^2 & I_x y_2^2 & I_x & I_y x_2^2 & I_y y_2^2 & I_y \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & I_x x_{n2}^2 & I_x y_{n2}^2 & I_x & I_y & I_y y_{n2}^2 & I_y \end{pmatrix} \begin{pmatrix} \mathbf{a^1} \\ \mathbf{a^2} \end{pmatrix} = \begin{pmatrix} -I_t \\ -I_t \\ \vdots \\ -I_t \\ -I_t \\ -I_t \\ \vdots \\ -I_t \end{pmatrix}$$

I_x, I_y, I_t depend on the examined pixel. We dropped the notation $I_x(x_j^i, y_j^i)$, $I_y(\dots)$, $I_t(\dots)$ due to space limitations. The left matrix of the SLE has a size of $(\sum_{i=1}^{k} n_i)$ x $(6 \cdot k)$ with $k = \textit{number of patches}$ and $6 \cdot k$ parameters to estimate.

Actually, by solving the SLE, we get $\mathbf{\Delta a^i}$, which we add to our current $\mathbf{a^i}$ and so on. Only the I_t change during iterations, because they depend on the current estimated transformation, while everything else stays the same.

3.2 Articulation constraints

To improve the motion estimation, we use constraints as mentioned in the introduction. We use the approximate position constraint from [JBY96] and the exact position constraint from [DSK11]. The idea of both constraints is that the point of articulation (**joint**) should be transformed the same way for both connected patches. $w(\mathbf{a^i}, \mathbf{x})$ is the warping function applied to the point $\mathbf{x} = \begin{pmatrix} x & y \end{pmatrix}^\intercal$ with the (affine) parameters $\mathbf{a^i} = \begin{pmatrix} a_{11}^i & a_{12}^i & a_{13}^i & a_{21}^i & a_{22}^i & a_{23}^i \end{pmatrix}$ for patch i and $\mathbf{A^i} = \begin{pmatrix} a_{11}^i & a_{12}^i & a_{13}^i \\ a_{21}^i & a_{22}^i & a_{23}^i \end{pmatrix}^\intercal$. Because the position of the joint is given in respect to the articulation point of each patch, we need to transform the coordinates. Therefore we move them to their own articulation point after warping. We do this by applying $-\mathbf{x_{joint}^i}$.

$$\begin{aligned} w(\mathbf{a^1}, \mathbf{x_{joint}^1}) - \mathbf{x_{joint}^1} &= w(\mathbf{a^2}, \mathbf{x_{joint}^2}) - \mathbf{x_{joint}^2} \\ \mathbf{A^1} \cdot \mathbf{x_{joint}^1} - \mathbf{x_{joint}^1} &= \mathbf{A^2} \cdot \mathbf{x_{joint}^2} - \mathbf{x_{joint}^2} \end{aligned}$$

Since we use an iterative algorithm and can only calculate $\Delta\mathbf{A^i}s$, we have to formulate the equation as follows (with the elementwise addition $+$):

$$\begin{aligned} w(\mathbf{a^1} + \mathbf{\Delta a^1}, \mathbf{x_{joint}^1}) - \mathbf{x_{joint}^1} &= w(\mathbf{a^2} + \mathbf{\Delta a^2}, \mathbf{x_{joint}^2}) - \mathbf{x_{joint}^2} \\ (\mathbf{A^1} + \mathbf{\Delta A^1}) \cdot \mathbf{x_{joint}^1} - \mathbf{x_{joint}^1} &= (\mathbf{A^2} + \mathbf{\Delta A^2}) \cdot \mathbf{x_{joint}^2} - \mathbf{x_{joint}^2} \\ \mathbf{A^1} \cdot \mathbf{x_{joint}^1} + \mathbf{\Delta A^1} \cdot \mathbf{x_{joint}^1} - \mathbf{x_{joint}^1} &= \mathbf{A^2} \cdot \mathbf{x_{joint}^2} + \mathbf{\Delta A^2} \cdot \mathbf{x_{joint}^2} - \mathbf{x_{joint}^2} \end{aligned}$$

The matrices $\mathbf{A^i}$ are the onces calculated so far. Now we look for a SLE to calculate the $\mathbf{\Delta A^i}$ for this iteration. As the joints are the origins for the transformation of each patch, we know $\mathbf{x^2_{joint}} = \left(\begin{smallmatrix} 0 & 0 \end{smallmatrix} \right)^\intercal$. [1]

$$\mathbf{A^1} \cdot \mathbf{x^1} + \begin{pmatrix} \Delta a^1_{11} \cdot x^1 + \Delta a^1_{12} \cdot y^1 + \Delta a^1_{13} \\ \Delta a^1_{21} \cdot x^1 + \Delta a^1_{22} \cdot y^1 + \Delta a^1_{23} \end{pmatrix} - \begin{pmatrix} x^1 \\ y^1 \end{pmatrix} = \begin{pmatrix} a^2_{13} \\ a^2_{23} \end{pmatrix} + \begin{pmatrix} \Delta a^2_{13} \\ \Delta a^2_{23} \end{pmatrix}$$

$$\begin{pmatrix} \Delta a^1_{11} \cdot x^1 + \Delta a^1_{12} \cdot y^1 + \Delta a^1_{13} \\ \Delta a^1_{21} \cdot x^1 + \Delta a^1_{22} \cdot y^1 + \Delta a^1_{23} \end{pmatrix} - \begin{pmatrix} \Delta a^2_{13} \\ \Delta a^2_{23} \end{pmatrix} = \begin{pmatrix} x^1 \\ y^1 \end{pmatrix} - \mathbf{A^1} \cdot \mathbf{x^1} + \begin{pmatrix} a^2_{13} \\ a^2_{23} \end{pmatrix}$$

$$\begin{pmatrix} x^1 & y^1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x^1 & y^1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{\Delta a^1} \\ \mathbf{\Delta a^2} \end{pmatrix} = \begin{pmatrix} x^1 \\ y^1 \end{pmatrix} - \mathbf{A^1} \cdot \mathbf{x^1} + \begin{pmatrix} a^2_{13} \\ a^2_{23} \end{pmatrix}$$

$$\mathbf{M} \cdot \begin{pmatrix} \mathbf{\Delta a^1} \\ \mathbf{\Delta a^2} \end{pmatrix} = \mathbf{b}$$

Doing this procedure for each joint, we get two times the number of joints equations, which we insert in one (constraint) SLE: $Mx = b$. Now we have a constraint system for an exact equality constraint. By using the less-equal option of lsqlin and a ε, we can relax this (but the number of constraint equations doubles, because of $\pm\varepsilon$).

We can also think of adding other constraints, like a *width constraint*. Therefore we can assume, that the distance between the lower two points of the upper patch should be the same as the upper two points of the lower patch. We thought about this due to an implementation error in the testing phase, which led to strange deformations resulting in line-like patches. Since the implementation of this width constraint with the l_2-distance isn't linear, we do an approach with the l_1-distance.

[1]For readability we drop the subscript joint

4 Evaluation of results

To evaluate the quality of the algorithm, we apply it on different data sets. This is done in in multiple steps. We use artificial images of quadrangles to look for the limits of the parameters of the transformations on its own, i.e. for rotation, translation, scaling. We start with one patch and add for the second step another one, which is connected with the first one by a joint.
We do some statistical analysis with respect to the robustness against noise and the role of textures influencing the algorithm.
Finally, we apply the algorithm on pictures of a moving person as a sequence of images of animated motion.

4.1 Error norm

For the error estimation, we use a image-based set-like error norm. Therefore we define the ground truth of the estimated patch as set A and the estimated patch itself as B.

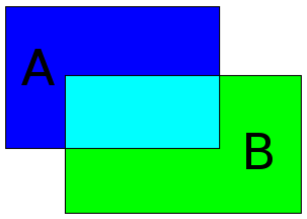

Figure 4.1: Visualisation of the error norm. A: ground truth, B: estimated solution. $E = \frac{\text{green and blue area}}{\text{colored area}}$

The normalized error is: $E = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$
In our case, each set consists of the pixels it contains. A is the set of the pixels in the patch calculated with the ground truth, B is the set of pixels in the patch calculated with the estimated transformation, $A \cup B$ are the pixels that appear in one or both of the patches and $A \cap B$ are only the pixels contained in both patches.

4.2 Input data

For evaluation, we use the following generated input data - especially some patches of textures with relatively broad spectra. The dimensions of the original images are below.

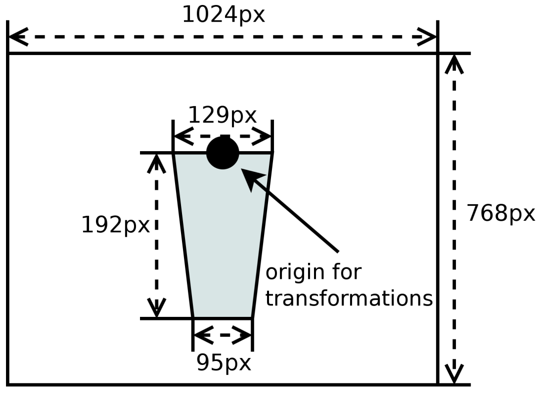

Figure 4.2: dimensions of input data (for one patch)

Figure 4.3: texture of default input data (for two patches)

4.3 Two-frame application for one patch

In this section, we calculate the transformation of a patch between two images. We do this for each parameter on its own, starting with rotation, then scaling and translation. Shearing is left out, although it could be modelled by the affine transformation we use.

4.3.1 Finding limits

In figure 4.4 and 4.5, we see that it is possible to find correct solutions for a rotation up to $18°$. Figure 4.6 shows the steps of the algorithm. For angles greater than $18°$ we end up in a local minimum, far away from the optimal. Since the algorithm needs fewer than 300 iterations to come up with the best solution, we could not improve it by increasing the number of iterations. The few steps needed to reach the minimum error for rotation $> 18°$ shows, that in this cases, we get bad estimated transformations, namely nearly the starting point of our algorithm. The small errors that remain in the plots result from rounding operations and might be erased by using interpolation instead. But if we visualize the estimated and correct patch in one image, they look very similar (see figure 4.6).

Figure 4.6 shows how the algorithm fits the estimated patch over iterations onto the ground truth. For this szenario, figure 4.7 shows the development of the error and in figure 4.8 there is the quadratic euclidean norm of the estimated transformation change in each step. The

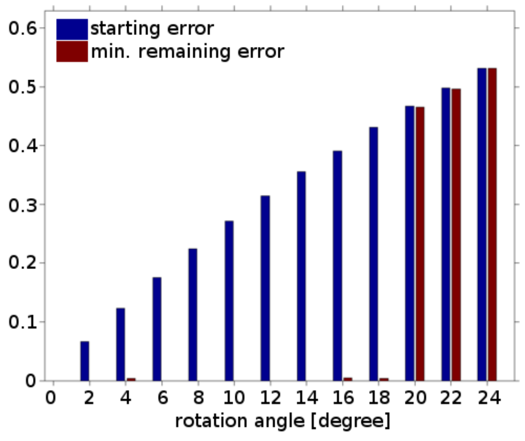

Figure 4.4: Comparison of starting and minimal remaining error for one patch with **rotation**

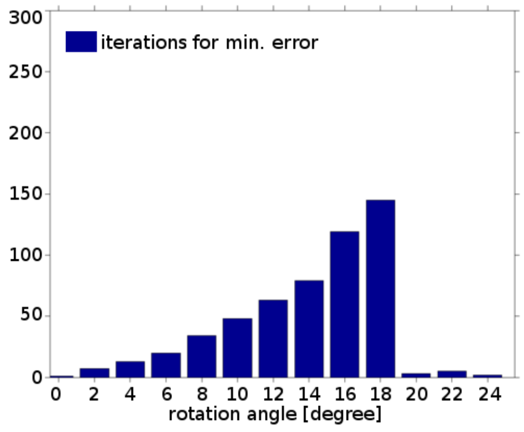

Figure 4.5: Iterations needed to reach the minimum error for one patch with **rotation**

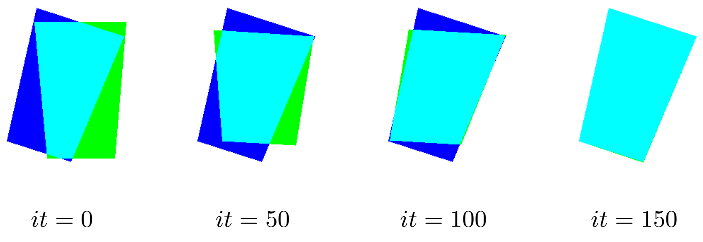

Figure 4.6: Visualisation of the patches for rotation of 18°: ground truth (blue), estimated patch (green) and overlap (cyan) each 50 iterations (it)

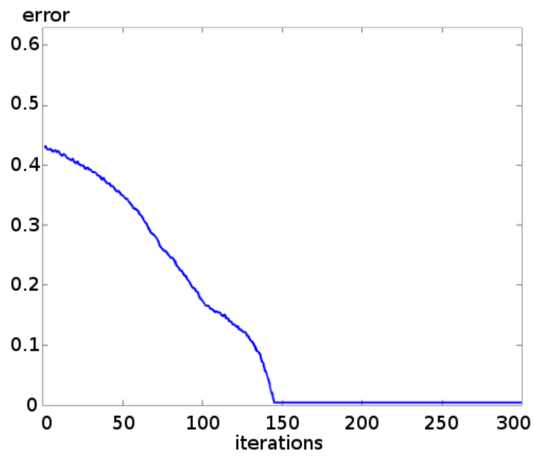

Figure 4.7: Development of the error at each step for rotation of 18°. It falls nearly monotonical.

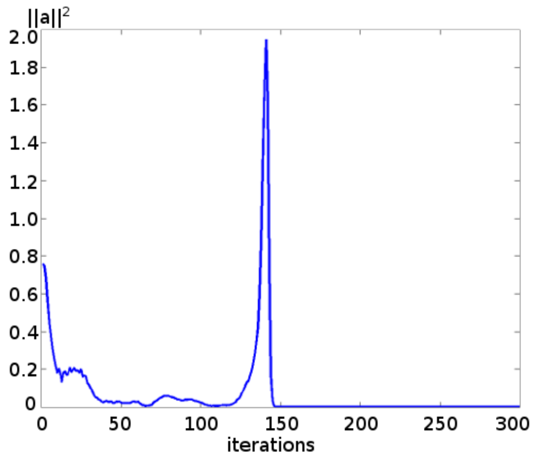

Figure 4.8: $||\Delta a||_2^2$ for rotation of 18°. Varies over iterations, but has a peak few iterations before the final solution is found.

error nearly falls monotonically, while the amplitude of transformation changes varies until a solution is found. It's interesting to see, that there's a peak in the change plot, a few iterations before we find the final result. This peak is also in plots for other transformations, but with a smaller amplitude. A explanation might be that the algorithm is very near to the correct solution. If this is the case then there are a lot of equations in the SLE which lead to the same correct result.

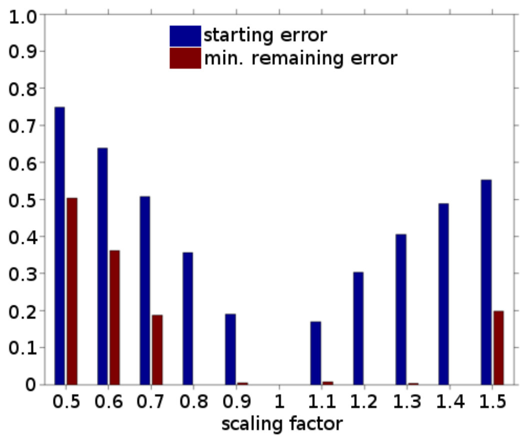

Figure 4.9: Comparison of starting and minimal remaining error for one patch with **scaling**

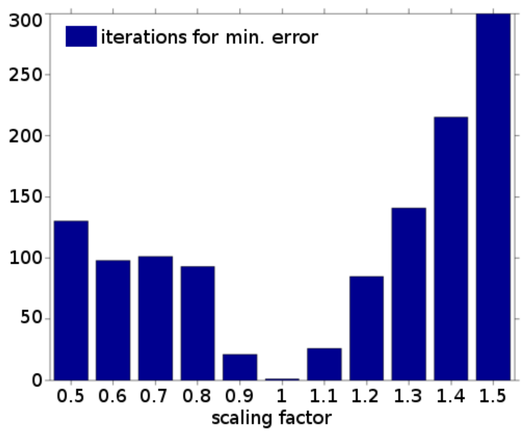

Figure 4.10: Iterations needed to reach minimum error for one patch with **scaling**

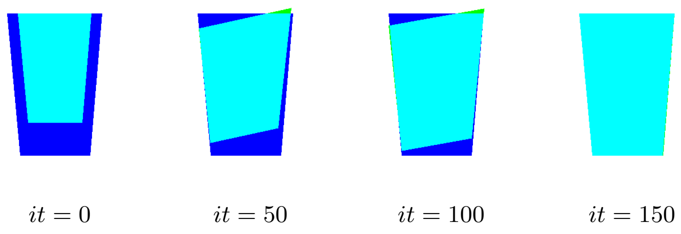

Figure 4.11: Visualisation of the patches for the scaling factor (sfac) = 1.3: ground truth (blue), estimated patch (green) and overlap (cyan) each 50 iterations

Next, we take a look on the scaling (figures 4.9 and 4.10). The range for the correct solution is derived from the plot and lies between a scaling factor of 0.8 to 1.4. Again, if we can not find the correct solution, we will find one far away from it. The scaling case is highly non-symmetrical. We would explain this by losing information due to a smaller patch size. Here we should mention, that we use nearest neighbor interpolation for image generation and it is possible to find a solution for 1.5 if we increase the number of iterations. There are fewer steps needed (for cases 0.5 - 0.7) to find the minimum errors, if the transformation is too large, since only a small local optimization is done in these cases.

We can follow the steps of the algorithm in figure 4.11. It does not only try scaling, but changes all parameters of the affine transformation. It first does some rotation with scaling and later rotates the patch back until it finds its solution:

$$A_{\text{est}} = \begin{pmatrix} 1.3013 & -0.0000 & 0.1326 \\ -0.0012 & 1.3013 & 0.0913 \end{pmatrix} = \frac{1}{1.3} \cdot \begin{pmatrix} 1.0010 & -0.0000 & 0.1020 \\ -0.0010 & 1.0010 & 0.0703 \end{pmatrix}$$

what is close to the ground truth of:

$$A_{\text{ground truth}} = \frac{1}{1.3} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

The values of $t_x = 0.1020$ and $t_y = 0.0703$ for the translation are in pixels, meaning the solution is subpixel-accurate in these parameters, even if we use rounding in the algorithm. The other parameters influence the transformation position dependent, since they are for shearing and rotation. This means the resulting error due to their wrong estimation might be greater, but they are also estimated more accurate.
We don't show the solution matrices for the other cases, because the estimated is always similar to the ground truth, if we find the correct solution.

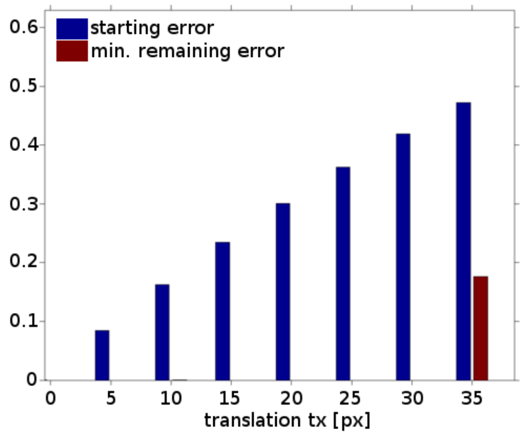

Figure 4.12: Comparison of starting and minimal remaining error for one patch with **horizontal translation**

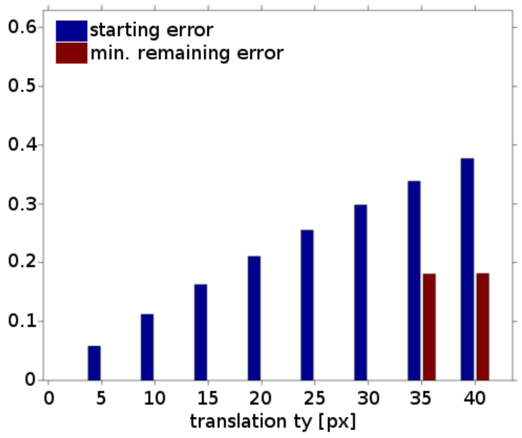

Figure 4.13: Comparison of starting and minimal remaining error for one patch with **vertical translation**

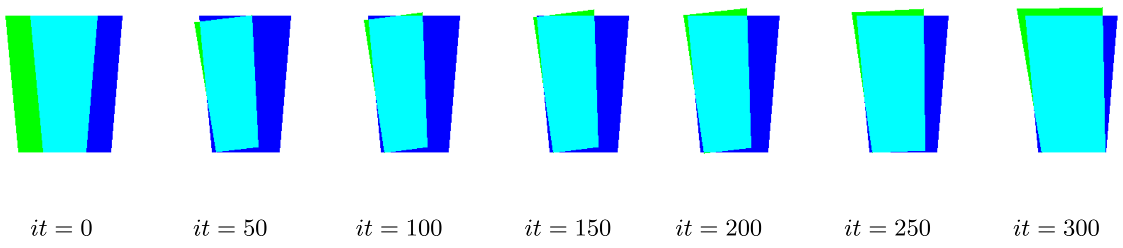

Figure 4.14: Visualisation of the patches for horizontal translation (tx) = 35px: ground truth (blue), estimated patch (green) and overlap (cyan) each 50 iterations

For the translation (figures 4.12 and 4.13) the results confirm the expectation. If we increase the amplitude, we need more iterations to find the optimum. This applies on all transformations.

In another version of the algorithm the spatial derivations were smoothed over both input images. This made it possible to find vertical translation up to 60px, but leads to strange deformations in other cases. Additionally we could add a image pyramid representation to increase the range of single patch estimation, but this is left out.

We can interpretate figure 4.14 in the same way as the scaling. Since all parameters changed, translation, shearing, scaling and rotation are combined until the algorithm finds the correct solution and changes the other parameters (except t_x) back to values similiar to the identity.

We did this to get a feeling for the amplitude of transformations for which our algorithm finds a solution. We will call this the range. No 100% hard threshold exists, but there is often a clear cut between 'correct' solutions and 'very wrong' ones.

4.3.2 Statistical analysis

In our next step we evaluate the influence of different factors on the algorithm.

Influence of texture
We have to ask ourselves: How important is the texture of the input data? Therefore we generate the same input data as before, but with different combinations of textures for the fore- and background. For each transformation there is a set of 34 combinations. Due to computational complexity, we rescale the data by $\frac{1}{2}$, what also might have an effect on the results.

We show the results via boxplots in the figures 4.15, 4.16, 4.17 and 4.18.

For small transformations, there is only a small difference, but as amplitude increases, texture becomes more and more important. By comparing the results with the figure from 4.3, we see that the texture of our 'default input data' is good for the calculation.

The same behavior is visible for the translation. If we would extend the plot range, we could even see that the higher the amplitude of the transformation, the fewer textures influence the algorithm to end at the correct result.

Influence of noise
Furthermore we analyze the influence of noise. Therefore we add independent gaussian noise, generated with different σ (0.01, 0.05, 0.1), to the two input frames (with the default textures) and apply the algorithm on the resulting noisy images. We repeat this 100 times for each transformation, to get statistical relevant information. Again we rescale the images to half of the original size. The results are shown in figure 4.19 to 4.22.

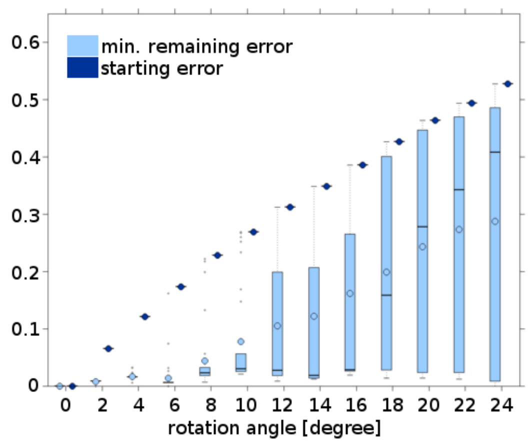

Figure 4.15: Influence of texture: Comparison of starting and minimal remaining error in 300 iterations. Results include 34 different combinations for one patch with **rotation**

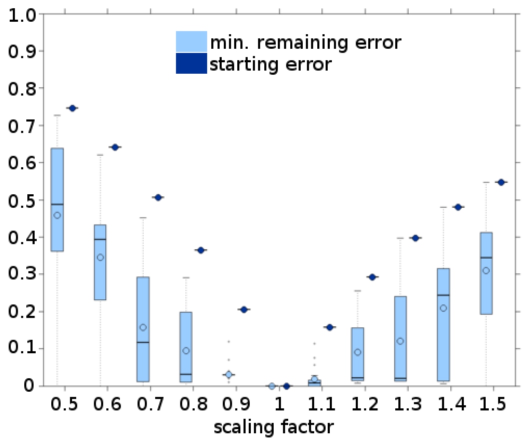

Figure 4.16: Influence of texture: Comparison of starting and minimal remaining error in 300 iterations. Results include 34 different combinations for one patch with **scaling**

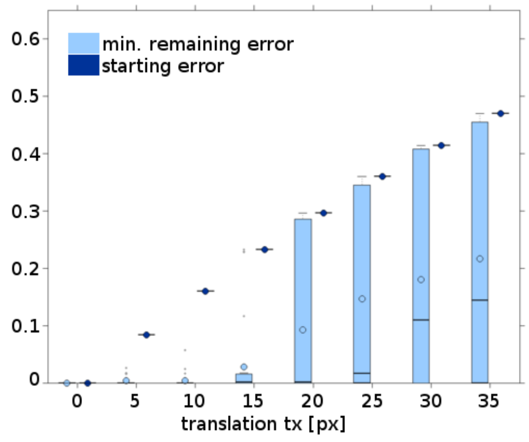

Figure 4.17: Influence of texture: Comparison of starting and minimal remaining error in 300 iterations. Results include 34 different combinations for one patch with **horizontal translation**

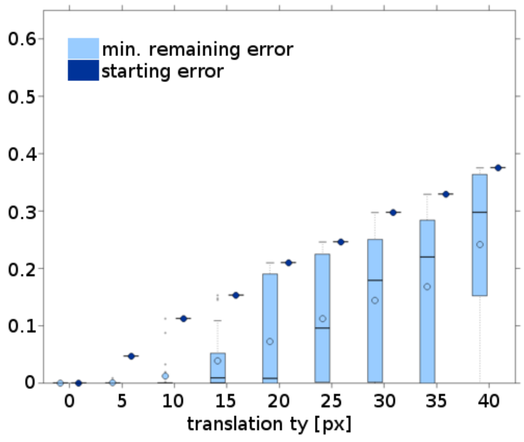

Figure 4.18: Influence of texture: Comparison of starting and minimal remaining error in 300 iterations. Results include 34 different combinations for one patch with **vertical translation**

For small transformations the algorithm is robust against noise, but the larger they are, the more sensitve it becomes. Some results look as expected, for example the cases $sfac = 0.7$ and $sfac = 1.5$ in figure 4.20. If there's less noise, the algorithm finds the correct solution more often and vice versa. The same holds for figure 4.21 or 4.22 at $ty = 30$, where the patch often is estimated correctly, but for some cases with high noise, it is wrong.
For figure 4.22 with $ty = 35$ and $ty = 40$ or figure 4.19 it's different. For these cases, the probability for the correct solution is higher if we add a noise with a larger σ. One explanation

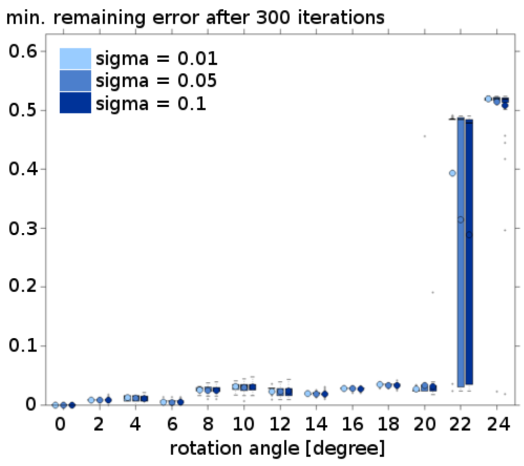

Figure 4.19: Influence of noise: Comparison of starting and minimal remaining error in 300 iterations. Results include 100 noisy images per σ for one patch with **rotation**

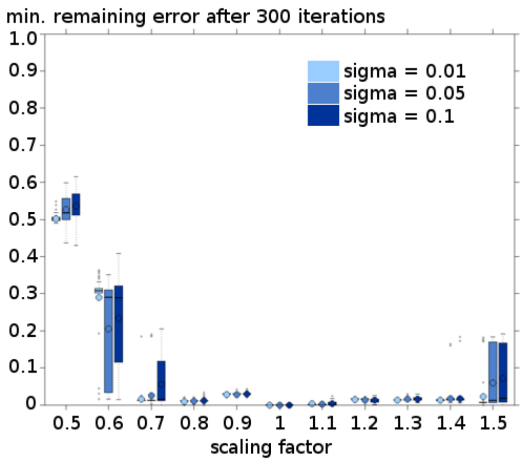

Figure 4.20: Influence of noise: Comparison of starting and minimal remaining error in 300 iterations. Results include 100 noisy images per σ for one patch with **scaling**

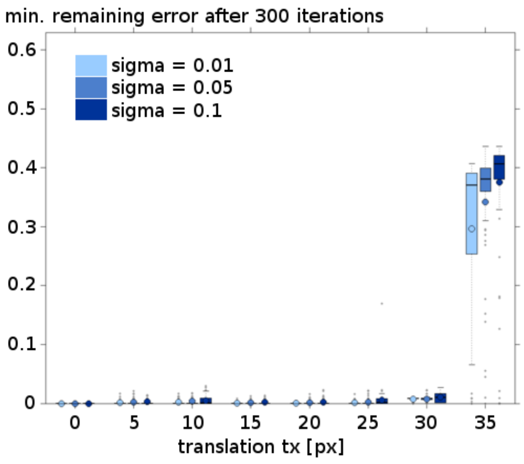

Figure 4.21: Influence of noise: Comparison of starting and minimal remaining error in 300 iterations. Results include 100 noisy images per σ for one patch with **horizontal translation**

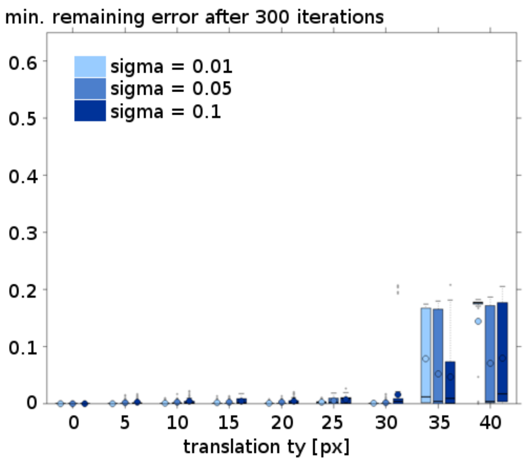

Figure 4.22: Influence of noise: Comparison of starting and minimal remaining error in 300 iterations. Results include 100 noisy images per σ for one patch with **vertical translation**

might be that the noise changes the intensity values and gradients in the correct way, so the algorithm comes up with a correct solution. Noise with smaller amplitude might lack the energy for such an effect.

To get a better feeling for the amplitude of noise with respect to image energy, we have created figure 4.23 - 4.26.
The profile (figure 4.23) shows the intensity of one row of the input image, where the red lines

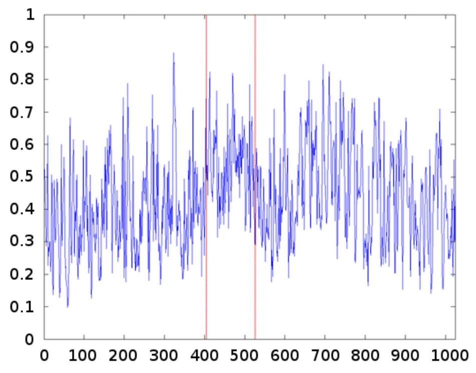

Figure 4.23: Intensity values of one input image row. The red lines mark the patch.

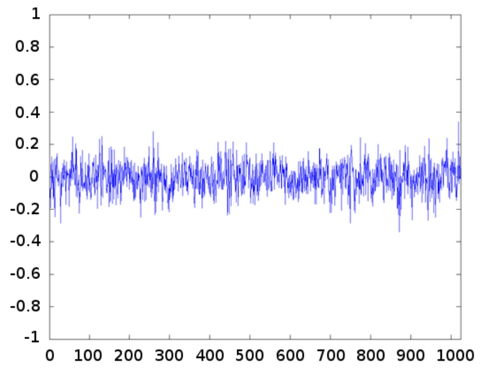

Figure 4.24: Gaussian noise signal with $\mu = 0$ and $\sigma = 0.01$

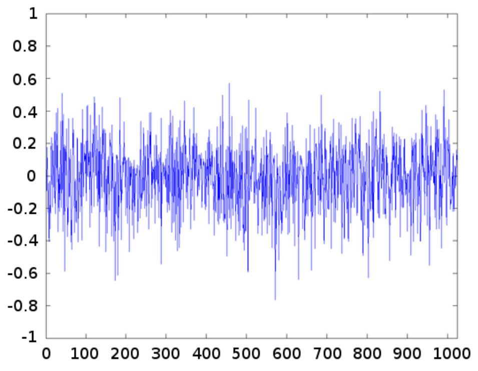

Figure 4.25: Gaussian noise signal with $\mu = 0$ and $\sigma = 0.05$

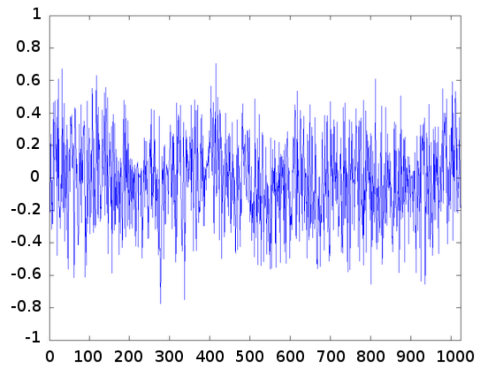

Figure 4.26: Gaussian noise signal with $\mu = 0$ and $\sigma = 0.1$

mark the patch area. It's difficult to recognize the difference between fore- and background textures. On the one hand the energy of the noise (fig. 4.24, 4.25, 4.26) is pretty high, but on the other hand gaussian lowpass filtering of the algorithm to reduce noise is not applied so far.

Energy analysis of noise

The energy of an intensity image I is calculated as followed: $E(I) = \sum_{\text{x,y}\in\text{image}} I(x,y)^2$.
For the gaussian noise on an intensity image I ($\tilde{I} = I + \text{noise}$) with standard deviation σ, it is $E_{\text{noise:}\sigma} = E(\tilde{I}_\sigma - I)$. It's calculated like this, because the amplitudes of noisy images are limited to $[0,1]$. The values for noise energy are the average of 1000 iterations.
Therefore $E_{\text{signal}} = E(\text{default input image}) = 2.5701 \cdot 10^5$ and

$$\begin{aligned}
E_{\text{noise,0.01}} &= 7.6307 \cdot 10^3 &\rightarrow\quad& \text{SNR}_{E_{\text{signal}},n_1:0.01} = 33.6808 \approx 15\text{ dB}\\
E_{\text{noise,0.05}} &= 3.3626 \cdot 10^4 &\rightarrow\quad& \text{SNR}_{E_{\text{signal}},n_1:0.05} = 7.6432 \approx 9\text{ dB}\\
E_{\text{noise,0.1}} &= 5.7285 \cdot 10^4 &\rightarrow\quad& \text{SNR}_{E_{\text{signal}},n_1:0.1} = 4.4865 \approx 6.5\text{ dB}
\end{aligned}$$

4.4 Two-frame application for two patches

In the next step of our evaluation, we apply the algorithm on two patches. We add the constraints described in section 3.2 and compare the results (see figure 4.27). Sizes and textures for first patch and background are still the same. For the second patch we choose a marble texture. We only do here the evaluation of rotation. The number of iterations are increased to 1000 instead of 300 as before.

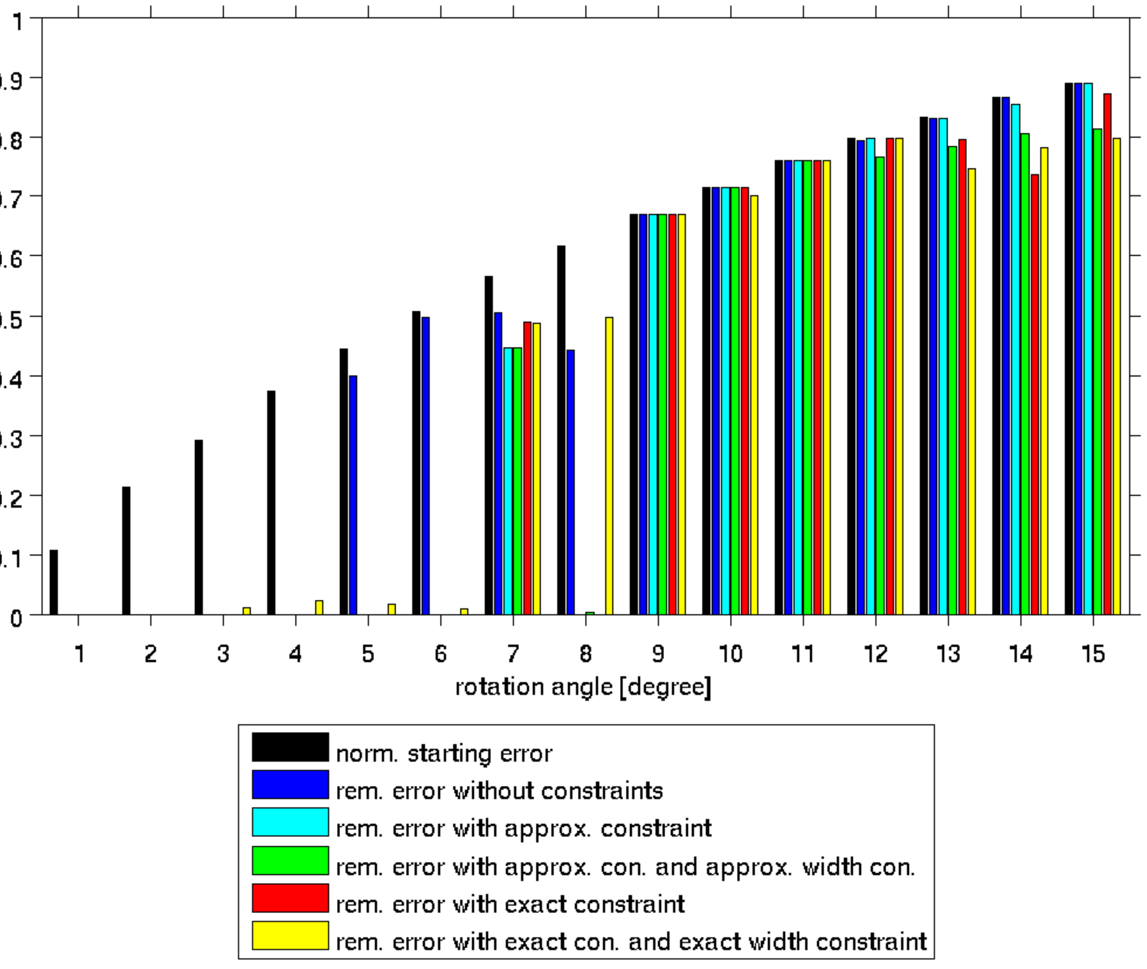

Figure 4.27: Comparison of the starting and remaining error in 300 iterations for the second patch estimated with different approaches

We realize that the approaches with constraints have a higher range than the one without, because it only finds the correct solution up to 4°. The highest three have the approximate position and approximate width constraints, only the approximate position constraint and only the exact position constraint. The one with the exact position and exact width constraint has additional disadvantages, because often a small error remains as seen in the figure 4.27 and it reduces the range for the first patch. This happens because the constraints influence the calculation and the upper, better estimated patch, does not only pull the lower one to the correct solution, but the lower one also disturbs the upper one. This behavior also reflects in the iterations needed to find the minimal error (figure 4.28).

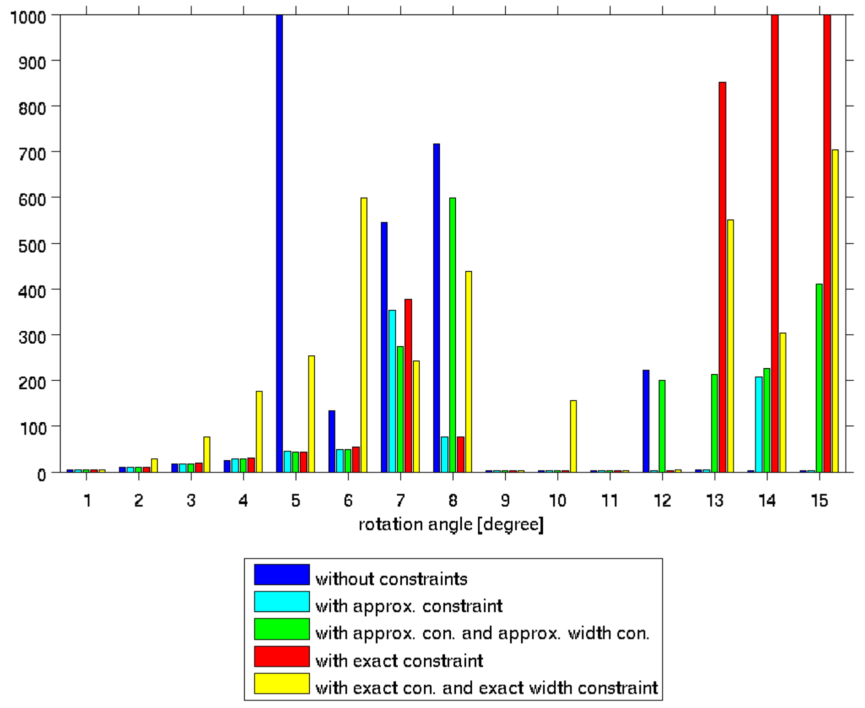

Figure 4.28: Iterations needed to reach the minimum error for the second patch with different approaches

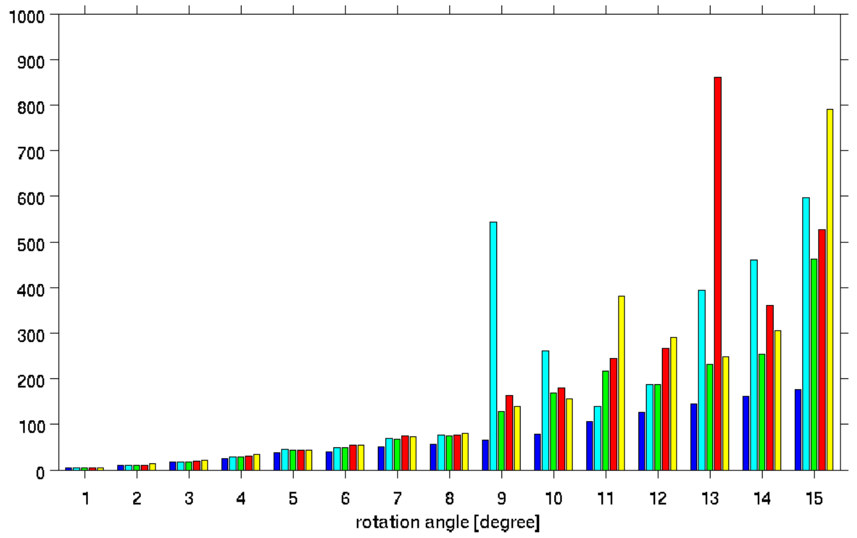

Figure 4.29: Iterations needed to reach the minimum error for the first patch with different approaches

If the approach without constraints finds the correct transformation, it needs the same or fewer iterations than approaches with constraints (figure 4.28 and 4.29). As mentioned, the constraints help to find a correct solution for the second patch, but also slow down the calculations for the first one.

The reason why we only find correct solutions for transformations with small angles (compared to section 4.3) is obvious. Here we rotate both patches around the rotation point of the upper patch and therefore do not only get a rotation for the lower patch, but also a translation component. To calculate its impact, we denote the upper rotation point as $O_1 = (x + \Delta x, y + \Delta y)$ and the joint as $O_2 = (x, y)$, because we estimate the tranformation in respect to O_2.

$$O_1 : \quad \begin{pmatrix} \cos(\theta) & -\sin(\theta) & t_x \\ \sin(\theta) & \cos(\theta) & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x + \Delta x \\ y + \Delta y \\ 1 \end{pmatrix} = \begin{pmatrix} u + \Delta x \\ v + \Delta y \\ 1 \end{pmatrix}$$

$$O_2 : \quad \begin{pmatrix} \cos(\phi) & -\sin(\phi) & t'_x \\ \sin(\phi) & \cos(\phi) & t'_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

Therefore we get the following equations:

$$\begin{aligned} x\cos(\theta) + \Delta x\cos(\theta) - y\sin(\theta) - \Delta y\sin(\theta) + t_x - \Delta x &= x\cos(\phi) - y\sin(\phi) + t'_x \\ x\sin(\theta) + \Delta x\sin(\theta) + y\cos(\theta) + \Delta y\cos(\theta) + t_y - \Delta y &= x\sin(\phi) + y\cos(\phi) + t'_y \end{aligned}$$

By setting $\theta = \phi$ we receive

$$\begin{aligned} t'_x &= t_x + \Delta x\cos(\theta) - \Delta y\sin(\theta) - \Delta x \\ t'_y &= t_y + \Delta x\sin(\theta) + \Delta y\cos(\theta) - \Delta y \end{aligned}$$

With the values for the example ($O_1(467, 203)$ and $O_2(468, 395)$) plugged in, we get the following table, that shows the ground truth for the translations:

rot [degree]	t'_x [px]	t'_y [px]
0	0	0
1	-3.3510	-0.0118
2	-6.7013	-0.0821
3	-10.0499	-0.2108
4	-13.3957	-0.3979
5	-16.7377	-0.6435
6	-20.0749	-0.9473
7	-23.4064	-1.3093
8	-26.7310	-1.7294
9	-30.0477	-2.2074
10	-33.3556	-2.7433
11	-36.6537	-3.3368
12	-39.9409	-3.9877
13	-43.2162	-4.6960
14	-46.4787	-5.4613
15	-49.7273	-6.2834

This table shows, that additionally to the rotation of 5°, there's a translation of approximately -17 pixels for the second patch.

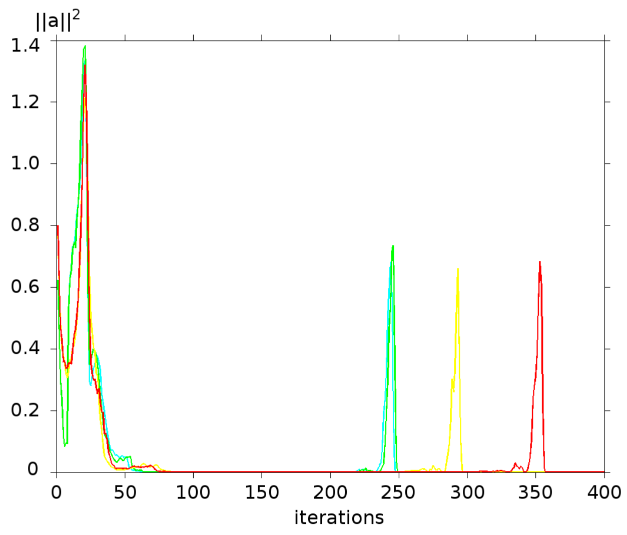

Figure 4.30: $||\Delta a||_2^2$ for 6° rotation around the upper rotation point

Figure 4.30 shows the quadratic euclidean norm of the estimated transformation change in each step for 6° as before in 4.8, but this time summed up for both patches. The approach without constraints is left out here. We can clearly see two peaks for each approach, which indicate, that the algorithm reaches the correct solution for a patch. The first one is found nearly after the same amount of steps, whereas for the second patch it varies (see also 4.28).

4.5 Multiframe application

There are different possibilities to extend the process of finding an image-to-image transformation to finding transformations for a whole image sequence. Mainly because we can predict a transformation and start there initially. Since our algorithm finds the local optimum, we expect to get better results.

Reference frame
We have to differentiate, if we calculate the transformation of each image with respect to one key-image/frame (A) (for example the first one) or with respect to the previous frame (B). Both have some advantages and disadvantages:
For A position and size of the mask is possibly very exact, since labeling for the first frame is done by hand. Also gradient images only have to be calculated once. But the resulting longer time interval might hurt assumptions (e.g. the brightness constancy assumption).
With B only small transformations appear in a short time interval, but deformation of the patch (due to errors) influences the calculation and errors sum up. We tried both approaches.

Transformation prediction
We assume to get better results if our starting point (i.e. the initial transformation) is closer to the solution of the searched transformation. Since we have no information at the beginning, we start with the identical transformation. For the next frames, we have multiple possibilities to guess the initial transformation:
We could use the **identical transformation** like in the first frame. Therefore we need no additional information and it is close to the solution, if there is only a small change. Or, if we assume transformations to be nearly the same over time, we might use the **previous transformation**, i.e. we start with the solution of the previous step. Another possibility is to use the **previous translation**. This means we use the translation of the previous step to get the estimated position and choose the rest as identical transformation. This might be good, if there is a constant translation but different transformations in the sequence. Finally we calculate the **expected transformation**. Therefore we use the two previous solutions and look how they have changed. Then we take the last one and change it in the same way. It is an improvement to the previous transformation, especially when reaching a reversal point, because the overshoot is reduced. How we implement the predictions depends on the reference frame, but all combinations are possible.

Additional tweaks
Since we get bad solutions with the upper approaches, we try to improve them even more. Usually the transformation between two frames is small. Therefore we can decide if a transformation might be a possible solution or not. In the second case we could replace the

transformation by the expected and hope the difference is small enough to find a valid solution in the next step. This approach might be fruitful in a one-reference-frame case since it leads to a wrong estimated patch persisting in the previous-frame case.
We can also think of an approach that limits itself to a few iterations per step so only small changes/transformations can appear. As a consequence we can also find only small changes in transformations at all. In many cases this leads to unexact solutions, but since we try to estimate the starting point near the solution it might be sufficient. The advantage is that even if the algorithm tries to find a incorrect solution, the error is bounded by the steps.
If it still fails with the ideas above, we could try to insert additional information by hand. We could label critical transformations, which will otherwise probably lead to a wrong solution, or we label after (each) t steps. Then it would be possible to do a bidirectional transformation prediction (forward and backward). But it would not be applicable in live-tracking, so we did not implement it.

In figure 4.31 we can see some results. We choose the best for different parameters with different combinations of approaches mentioned above.

Results for our jumper sequence

We produce the results in figure 4.31 with the previous frame as reference image and the expected transformation as prediction method. We use 50 iterations per step. In the beginning the fit is quite good, but from $t = 13$ to $t = 14$, the deformation is extremely large and we have to stop calculation. We tried a more textured background to get better results, but unfortunately it didn't help. For all trials differences between correct and estimated transformation is either very small or extremely large, cases with medium differences are sparse.

Results for our walker sequence

In figure 4.32 we see results for the walker sequence. Also with the previous frame as reference image, but this time with identical transformation as prediction and 150 iterations per step. We can not show the patch in the next step ($t = 25$) since it is (totally) deformed and reaches out of the image.

We have not succeeded to track forearm and hand for more than a few iterations - with and without our constraints. Maybe it works with some different preprocessing steps.

When we want to improve the calculation with a more textured background, it leads to the sequence seen in figure 4.33, where the patch expands from the thigh over the whole leg.

We calculate the results of figure 4.34 on images of size 320x180. Therefore the original input data is rescaled by $\frac{1}{4}$ for each side. In this figure tracking of the thigh works relatively good. But as soon as occlusion appears, like in $t = 11$, the algorithm does a bad job. Here it

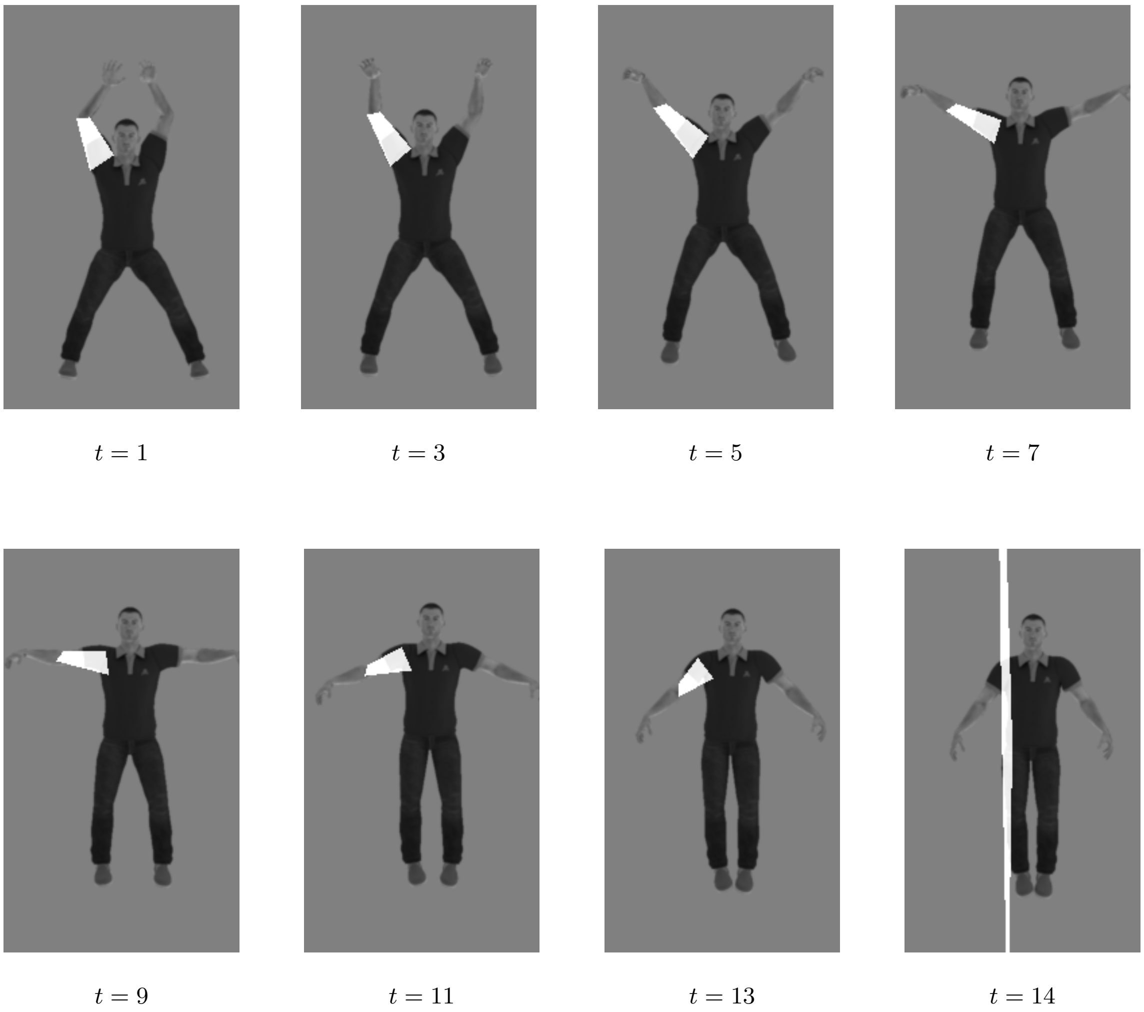

Figure 4.31: Estimated patch for different time steps of jumper sequence using the previous frame for reference and expected transformation as prediction method by 50 iterations per step

continues to track the other thigh, like in $t = 17$. It follows the limb until the next occlusion and deforms afterwards.

For tracking multiple patches, we see in figure 4.35 that for a few frames patches are tracked correctly but then results between correct and estimated ones diverge very fast. From this point on the algorithms seldomly returns a correct solution later on.

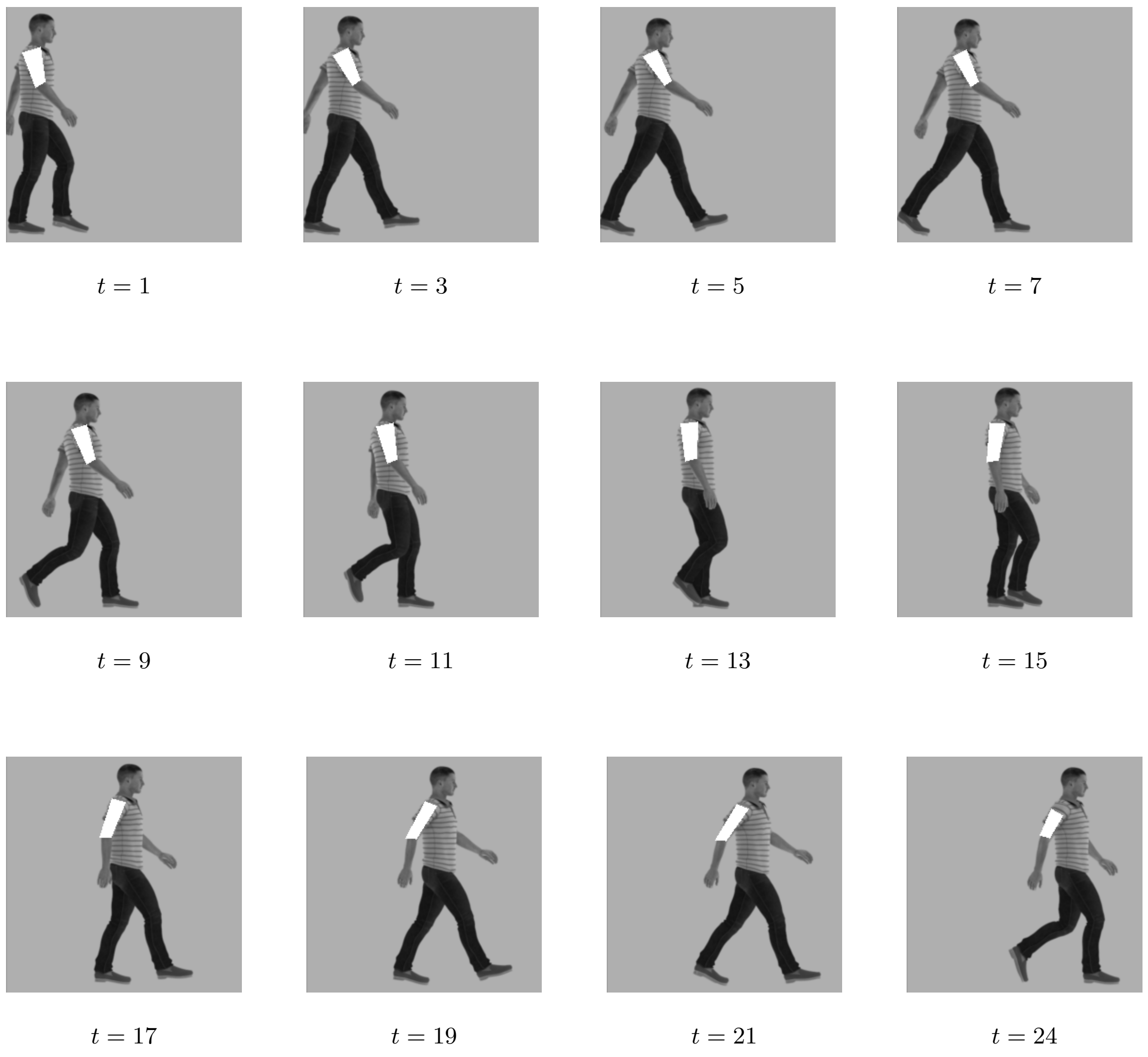

Figure 4.32: Estimated patch for different time steps of walker sequence using the previous frame for reference and identical transformation as prediction method by 150 iterations per step

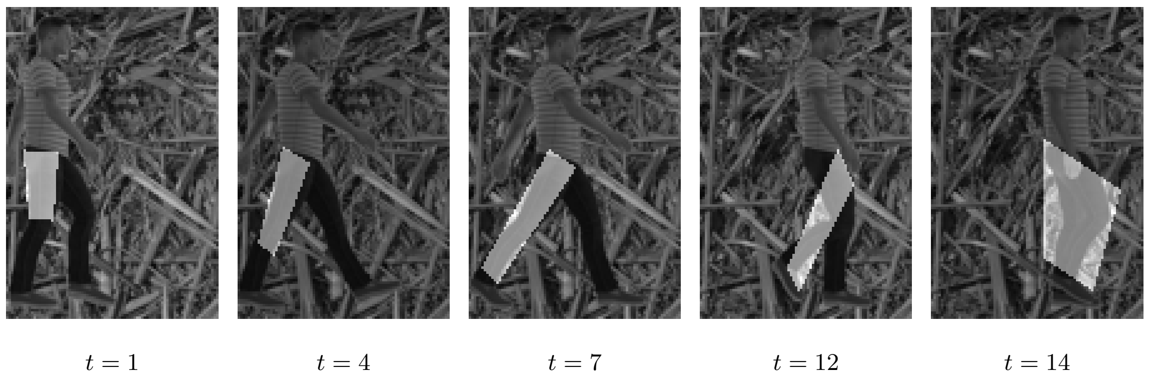

Figure 4.33: Estimated patch for different time steps for walker sequence with textured background. Patch expands from thigh over whole leg.

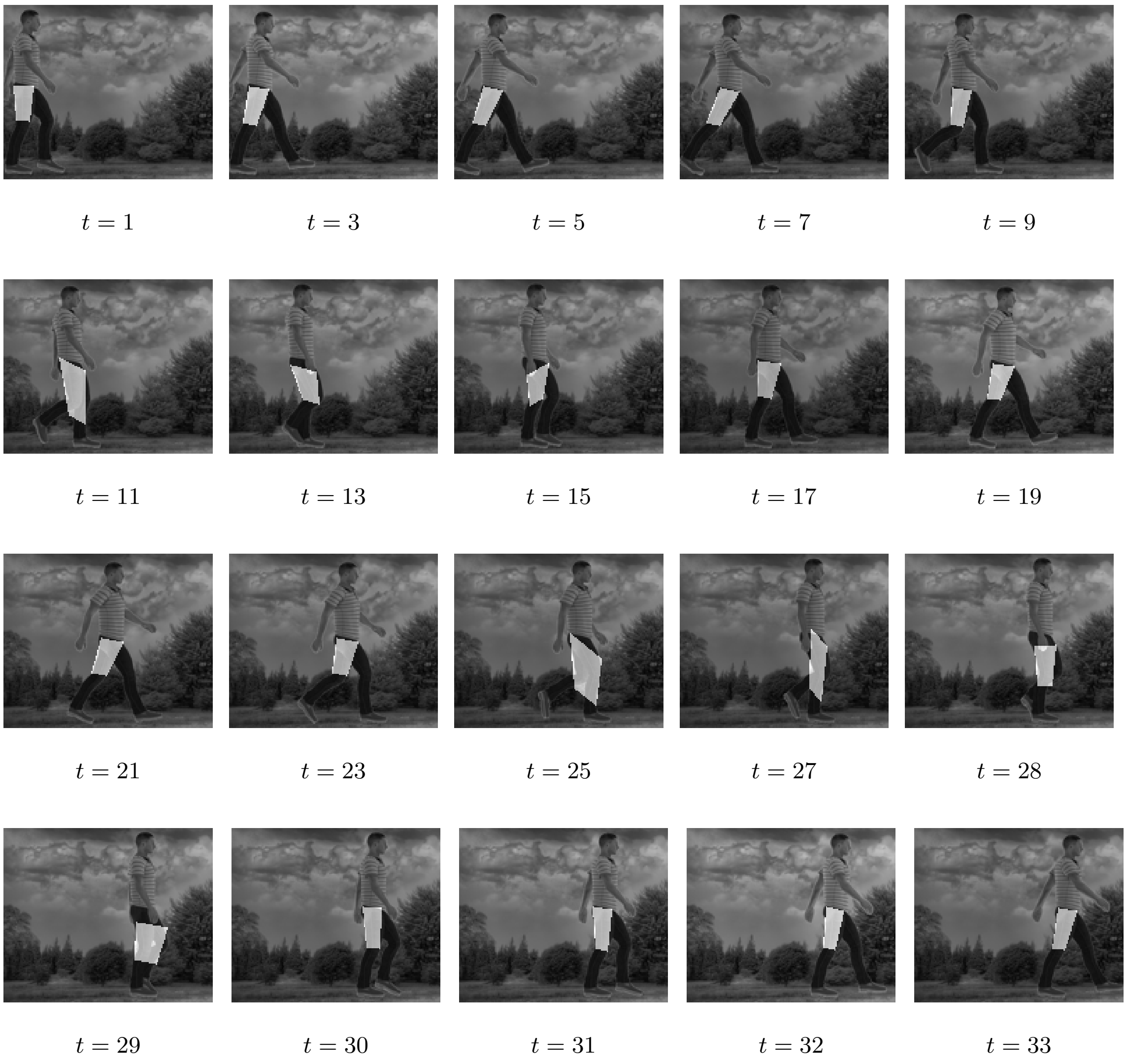

Figure 4.34: Estimated patch for different time steps: walker with background texture

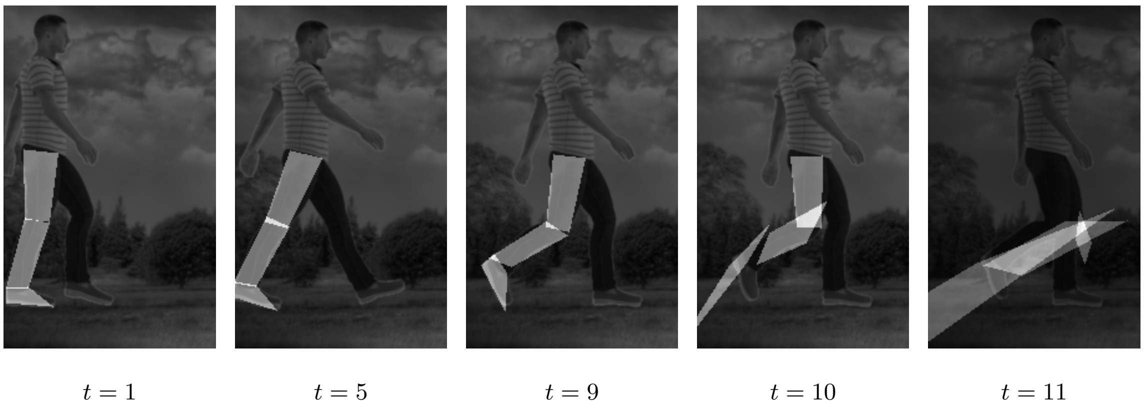

Figure 4.35: Estimated patch for different time steps: trial for multiple patches

5 Conclusion

We analyzed the behavior of an algorithm for gradient-based motion estimation on articulated data. For artificial data and small transformations the results look very good as seen in section 4.3. But if the amplitude of a transformation goes beyond the range of the algorithm, there's a high chance that a large error remains. It is interesting to see that the error in our plots falls nearly monotonical, even if we use the image-based error value. This does not correspond to the search space of the algorithm in which it does gradient descent.

The algorithm is robust against gaussian noise for small transformations, but for bigger transformations it becomes more sensitive. The same behavior holds for the influence of the textures.

In the case of more patches, the constraints extend the range of our algorithm. But therefore it needs at least one easily calculable patch. Then the constraints help to find a correct solution for other patches, but also slows down the calculations for the good ones. The best approach is the one with approximate position and approximate width constraint. It is better than the exact one, because it allows small differences that may appear due to noise or for other reasons.

Unfortunaly, for the multiframe application, the algorithm provides different results than we expected. Because it didn't track even one patch reliable on the animated data, there was no possibility to evaluate the constraints on that datasets.

An interesting task for future research would be to find and evaluate different preprocessing methods for animated data and then analyze the behavior on real video sequences. Or the algorithm could be analyzed in terms of: 'how many iterations are needed and when should it stop?' and 'what would be the best resolution to calculate on?' A pyramid approach might help to find the correct solution in even fewer steps, but then the question of temporal resolution needs to be answered. Of course, everything depends on the speed of the objects in the sequences.
Another interesting point that has not been evaluated here, is the behavior for data with projective perspective. This means if someone walks away from the camera or to the camera. It would be interesting to know if the models needs to be changed from the affine to the perspective model.

One might be interested, if it is possible, to apply our algorithm in realtime. So far it isn't. But our algorithm could be implemented more efficiently, due to the use of the special structure that only a few parts of the SLE change during interations. Since this wouldn't change the results, there are possibilities for improvement. But this is only for the sake of completeness.

Bibliography

[BBPW04] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *ECCV (4)*, pages 25–36, 2004.

[BM98] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '98, pages 8–, Washington, DC, USA, 1998. IEEE Computer Society.

[BM04] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004.

[DSK11] Ankur Datta, Yaser Sheikh, and Takeo Kanade. Linearized motion estimation for articulated planes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(4):780–793, 2011.

[HS81] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artif. Intell.*, 17(1-3):185–203, 1981.

[JBY96] Shanon X. Ju, Michael J. Black, and Yaser Yacoob. Cardboard people: A parameterized model of articulated image motion. In *2nd International Conference on Automatic Face and Gesture Recognition (FG '96), October 14-16, 1996, Killington, Vermont, USA*. IEEE Computer Society, 1996.

[JNW06] D. Jackel, S. Neunreither, and F. Wagner. *Methoden Der Computeranimation.* Springer London, Limited, 2006.

[UFF06] Raquel Urtasun, David J. Fleet, and Pascal Fua. Temporal motion models for monocular and multiview 3d human body tracking. *Computer Vision and Image Understanding*, 104(2-3):157–177, 2006.