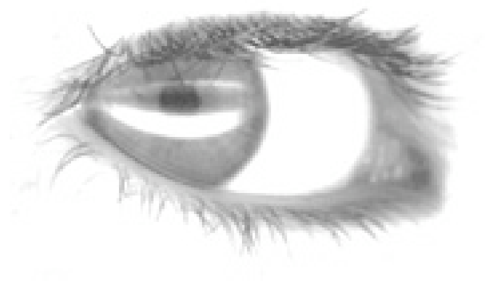
FIRST LIGHT

Diploma thesis, 2004



Unsupervised object tracking without a-priori knowledge

Student:

Thabo Beeler
thabo.beeler@hsr.ch
University of Applied Sciences, Rapperswil

Advisor:

Prof. Dr. Josef Joller
joseph.joller@hsr.ch
University of Applied Sciences, Rapperswil

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

INI Institute for Neuroinformatics
uni | eth | zürich

Computer Science Department
University of Applied Sciences Rapperswil
Oberseestrasse 10
8640 Rapperswil, Switzerland
http://www.hsr.ch

Institute of Neuroinformatics
University and ETH Zurich
Winterthurstrasse 190
8057 Zurich, Switzerland
http://www.ini.unizh.ch

For my family

Especially for my dad, my mom, my brother and my sister

Preface

«Computer vision as a field is an intellectual frontier. Like any frontier, it is exciting and disorganized; there is often no reliable authority to appeal to – many useful ideas have no theoretical grounding, and some theories are useless in practice.» [4]

Vision is one of the most important senses we have and many tasks in everyday-life could not be fulfilled, if we could not see – like reading this paper.
While our visual system is highly developed, the artificial counterpart is still at its beginning.

There are many unsolved and difficult – and therefore very interesting – problems at every stage of the perception process. They range from low-level vision (perceiving the images) over mid-term layers (processing the images) up to higher-level stages (interpreting the images).

In our opinion, the capability to perceive the environment (not only, but especially visually) is crucial for unsupervised learning and (artificial) intelligence.

MOTIVATION

A generic and reliable object-tracking algorithm is the base of any higher-level visual system. While tracking of arbitrary objects is no challenge to a human at all, it has not yet been completely solved for artificial systems.

The current state of the art tracking algorithms base mostly on a provided model of the objects of interest. This model might describe some characteristics of the object, such as the shape, its texture or its motion. Clearly, such contextual tracking is used by humans as well.

The problem with this approach is its limited usability. Whenever we want to track a different type of objects, we need to provide a new model.

The visual system of animals is capable of tracking objects for which it does not have a model. Based on the tracking it is able to build its own models, its own mental representation of the perceived environment. An artificial system with similar tracking abilities would provide a great step towards unsupervised learning and artificial intelligence.

Up to now, many generic tracking algorithms have been developed and implemented. But still, their application is limited to simple objects only. The most popular use background subtraction to identify the regions of interest, which is possible with 'stationary cameras' only.

OVERVIEW

In our research on visual tracking we first analyzed and implemented a variety of current state of the art algorithms for unsupervised, dynamic tracking. Based on this research we developed two different approaches for the tracking task.

Part I

Chapter 1

To enhance the results, the frames need first to be pre-processed to reduce noise and other distorting effects. This is achieved by the combination of two filters, a ***median*** a ***Gaussian*** convolution.

Chapter 2

Chapter 3

Images contain a lot of information and one approach to retrieve some of it is to segment the image. Based on ***region growing***, we propose a fast segmentation algorithm, which is suitable for cartoons. The border of the generated segments is found by a very simple ***edge tracer*** in order to produce the shape of the segments.

Chapter 4

The shape as a whole is sensitive to noise, partial occlusion and other distorting effects. Furthermore it contains a lot of redundant information. The introduced ***chain code*** deals with these issues in a very fast way.

Part II

Chapter 5

The first approach on optic flow estimation is ***gradient-based***, because it relies on the intensities of the individual pixels. The ***optic flow*** is detected by finding a corresponding pixel in the next frame for every pixel of the previous frame.
The approach is called ***Facette***, because it is inspired by the facetted eyes of insects. Facette uses multi-scale image pyramids well known from many scale-invariant feature detectors to sense motion of different intensities. The construction of the pyramids is slightly more complex, but still simple enough to provide high performance. It could even be re-implemented in hardware to provide a high resolution, real-time tracking system.

Part III	The second approach is ***feature-based***. Instead of trying to correlate all pixels of two frames, it first looks for specific regions in the frames. These are found by feature detectors.
Chapter 6	We compare three different ***feature detectors*** in order to find the best one for this work.
Chapter 7	The detected regions are encoded by ***feature descriptors*** in a way they can be compared later on.
Chapter 8	The task to find corresponding features in two adjacent frames is called tracking. We introduce and compare three different ***feature trackers*** and enhance them with some fast and efficient post-processing algorithms to receive a better result.
Chapter 9	Finally, the detected structure is stored in a ***feature graph***, which is tolerant to temporal inconsistencies in the feature detection. Furthermore it is able to transform unobserved features, which enables it to retain the structure of objects even beyond the border of the screen.
Chapter 12,13,14	Last but not least we provide a quick and dirty overview of the main concepts we used for the ***realization*** of the algorithms.

Contents

PART I

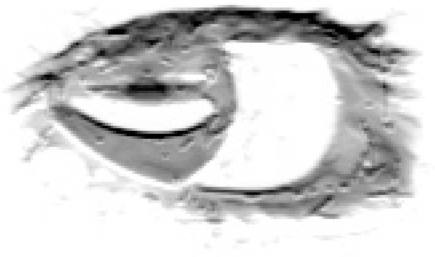


Image pre-processing

1 Image filters

Raw images are almost never in perfect condition. Encodings like mpeg or jpeg reduce the quality. Noise, dust and scratches add undesired effects. And last but not least, most images try to reduce a three dimensional space onto a two dimensional plane.

Dealing with such perspective distortions is difficult and not possible with low-level filtering [4].

Image noise is one of the primary problems in early vision. We use the *additive stationary Gaussian noise* model [4] to describe noise in images. In this model, each pixel has added to it a value chosen independently from the same Gaussian probability distribution. One possibility to deal with such noise are Gaussian filters.

1.1 IMAGE SMOOTHENING

Image smoothening, or blurring, results in signals where pixels tend to be increasingly similar to the value of neighboring pixels. Smoothening can be done in many different ways. We use a convolution with a symmetric Gauss kernel [4].

$$G \ = \frac{1}{2 \quad ^{2}} \quad ^{\left(\frac{\left(x^2+y^2\right)}{2 \ ^2} \right)} \qquad (1.1)$$

To speed up the calculation, we use a slightly modified version to produce the kernel:

$$G \ ' = 1.5^{\left(\frac{\left(x^2+y^2\right)}{2 \ ^2} \right)} \qquad (1.2)$$

Still, this convolution operates in 2-D space. Luckily it can be separated in two 1-D convolutions. The first one convolves the image horizontally and the second one vertically.

$$\begin{aligned} G \ ' &= 1.5^{\left(\frac{\left(x^2+y^2\right)}{2 \ ^2} \right)} \\ &= 1.5^{\left(\frac{x^2}{2 \ ^2} \right)} 1.5^{\left(\frac{x^2}{2 \ ^2} \right)} \\ &= G_H \ ' G_V \ ' \end{aligned} \qquad (1.3)$$

Figure 1-1 and Figure 1-2 show how the smoothening works with a rather big kernel ($= 21$).

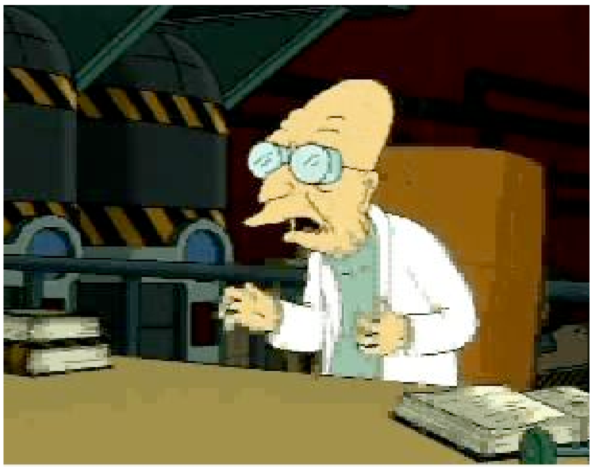


Figure 1-1: original image

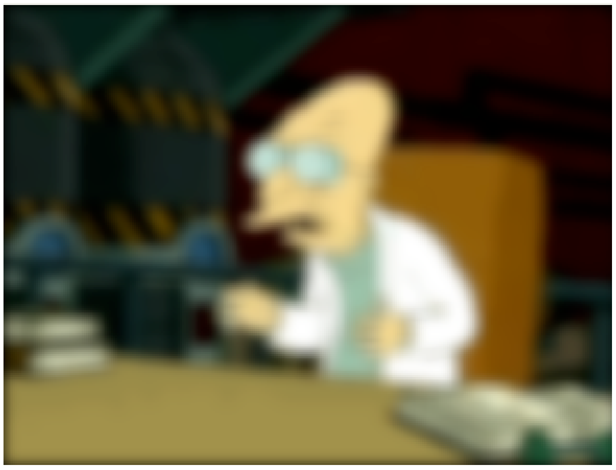


Figure 1-2: smoothed image ($= 21$)

1.2 MEDIAN

Image smoothening with a Gauss kernel effectively reduces the impact of noise, especially when the noise has a low standard deviation. Smoothening performs sub-optimal when images contain so called *blind pixels* or noise with a high standard deviation. Blind pixels are pixels, which are permanently erroneous, e.g. broken detection cells.
Another approach, which performs well in such situations, is the median. Its weak point is that it can only deal with low-density noise.

The median sets the intensity I of a pixel to the value of the majority of the surrounding pixels.

$$I(x,y) = M(x,y)_k \tag{1.4}$$

with

$$M(x,y) = sorted \bigcup_{i=\ N}^{N} \bigcup_{j=\ N}^{N} I(x+j, y+i) \tag{1.5}$$

and

$$k = \frac{(2N+1)^2}{2} \tag{1.6}$$

N is the kernel strength, defining how many neighboring pixels will be considered. The size of the whole kernel is therefore $(2N+1)^2$. In Figure 1-4 the starry field, the logo and even the bike were filtered by the median. Usually a lower strength is used $(N \quad 3)$.

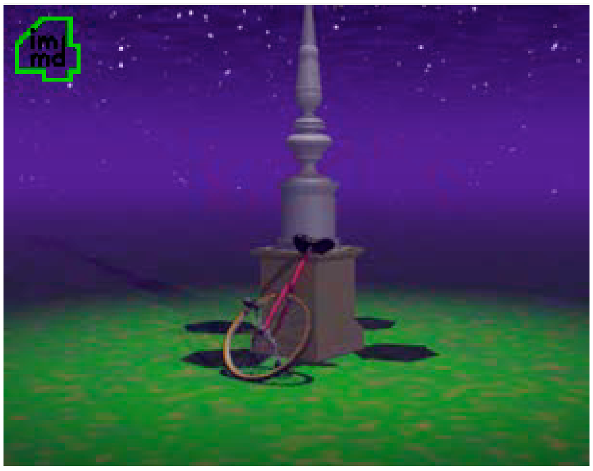


Figure 1-3: original image

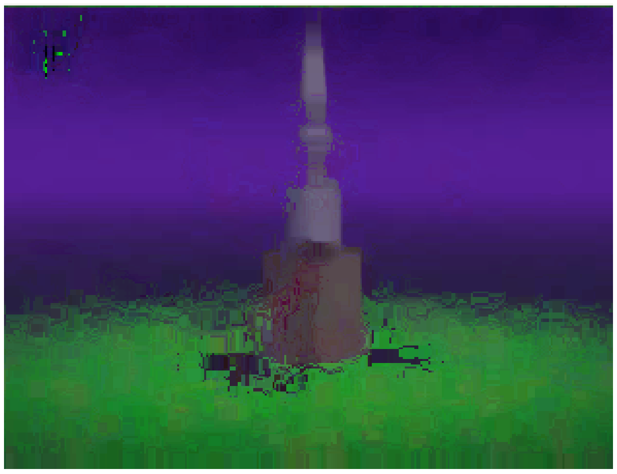


Figure 1-4: Median filtered image $(N = 21)$

Image smoothening and median enhance each other very well. First we apply a median and then a Gaussian convolution.

2 Image segmentation

One approach to find structures in an image is to segment it. Over the past 20 years many different approaches have been introduced [13]. They range from pixel clustering[4][11] over graph representation [14][3][23] to probabilistic segmentation methods [4][15].
Still, segmentation has not been solved in general, but some approaches show good results in their specific areas.
Some operate on luminance intensity, others on color or texture [24][34]. Also edges and other features have been examined [20][25].
Last but not least, motion based segmentation is another approach to this problem [35][36][22]. Motion segmentation will be discussed in [chapter 5].

2.1 REGION GROWING

The algorithm proposed in this document was designed to segment comics. The advantage of cartoons are the simple color gradients. It was not designed for real-world images, as their texture and lightings are way more complex.
Its advantage over other segmentation algorithms, like graph-cut, is its speed. The algorithm can segment comic movies in real time.

The algorithm partitions the image into areas of similar color. When running the algorithm iteratively it would converge to a deterministic segmentation. Experiments showed, that after one pass the segments differ only slightly from the iterative run, while time efforts sink dramatically.
For comics, the color channels seemed to be the best criterion. The color can be written as a three-dimensional vector containing the color channels red, green and blue[4].

$$\vec{C} = \begin{matrix} R \\ G \\ B \end{matrix} \tag{2.1}$$

2.1.1 DESCRIPTION

The color of an area A ($\vec{C}_A$) is the mean value of all its pixels

$$\vec{C}_A = \overline{\vec{C}} = \frac{1}{n} \overset{n}{\underset{i=1}{}} \vec{C}_i \tag{2.2}$$

The algorithm chooses one point (it currently starts in the upper left corner, but other methods would be possible) and sets it as the center of the region. It then iteratively adds all connected points as long as their similarity

$$S\left(\vec{C}_i, \vec{C}_j\right) = \left|\vec{C}_i \quad \vec{C}_j\right| \tag{2.3}$$

differs not more than .

While the area grows, the color center shifts.

$$\overline{\vec{C}}_{n+1} = \frac{n\overline{\vec{C}}_n + \vec{C}_i}{n+1} \tag{2.4}$$

As we have three channels of color, this computation is not efficient. We can optimize it though:

$$\begin{aligned}\overline{\overline{\vec{C}}}_{n+1} &= \frac{n\overline{\overline{\vec{C}}}_{n} + \vec{C}_i}{n+1} \\ &= \overline{\overline{\vec{C}}}_{n} \frac{n}{n+1} + \vec{C}_i \frac{1}{n+1}\end{aligned} \tag{2.5}$$

Using the fact, that

$$\begin{aligned}1 &= \frac{n}{n+1} + \frac{1}{n+1} \\ \frac{n}{n+1} &= 1 \quad \frac{1}{n+1}\end{aligned} \tag{2.6}$$

the center can be calculated with a minimum effort:

$$\begin{aligned}\overline{\overline{\vec{C}}}_{n+1} &= \overline{\overline{\vec{C}}}_{n} \frac{n}{n+1} + \vec{C}_i \frac{1}{n+1} \\ &= \overline{\overline{\vec{C}}}_{n} \; 1 \quad \frac{1}{n+1} \quad + \vec{C}_i \frac{1}{n+1} \\ &= \overline{\overline{\vec{C}}}_{n} \quad \overline{\overline{\vec{C}}}_{n} \frac{1}{n+1} + \vec{C}_i \frac{1}{n+1} \\ &= \overline{\overline{\vec{C}}}_{n} + \left(\vec{C}_i \quad \overline{\overline{\vec{C}}}_{n}\right) \frac{1}{n+1}\end{aligned} \tag{2.7}$$

Note that the factor $\frac{1}{n+1}$ is constant for all channels. So we need 1 division, 1 multiplication and 7 additions per step only. The first formula would need 3 divisions, 3 multiplications and 4 additions.

If the point already belongs to a region B, it is only added if $\left|\vec{C}_A \quad \vec{C}_i\right| < \left|\vec{C}_B \quad \vec{C}_i\right|$. If this condition holds, we need to remove the pixel from B to fulfill equations (2.9) and (2.10). The removal of a pixel shifts the color center just like the addition does. Analog to equation (2.7) we can calculate the new center

$$\overline{\overline{\vec{C}}}_{n \;\; 1} = \overline{\overline{\vec{C}}}_{n} + \left(\overline{\overline{\vec{C}}}_{n} \quad \vec{C}_i\right) \frac{1}{n \quad 1} \tag{2.8}$$

When an area cannot be grown further, a point just outside of the area is chosen as the center of a new region. This new region then competes with the old one.

After the image has been segmented, the areas will be pruned to limit the amount of areas to process at later stages. Pruning is done by two criteria:

- Size (the size of the area must be > MIN_SIZE)
- Coherence (the coherence determines how compact an area is. It is the amount of pixels in the border divided by the size of the area. The coherence must be > MIN_COHERENCE)

2.1.2 Precondition

An RGB color image. Results are better, when the image has been preprocessed by some filters to remove noise [chapter 1].

2.1.3 Post condition

A segmentation map containing all the segments.

2.1.4 Invariant

The amount of pixels in all areas is the same as the amount of pixels in the image.

$$|I| = \left|\bigcup_{i=1}^{k} A_i\right| + \left|I \quad \bigcup_{j=1}^{k} A_j\right| \tag{2.9}$$

k is the amount of areas, I is the image and A_i is the i-th area.

This implies that the areas separate the image exclusively. The condition

$$A_i \quad A_j = \qquad (i \quad j) \tag{2.10}$$

is invariant.

2.1.5 Pseudo code

```
Queue q;

function doSegment( Image image )
{
     Array areas;
     until all points are segmented do
     {
          Point p = getNextPoint(image);
          put p in q;
          Area area = new Area();
          while q not empty do
               grow(area);
          put area in areas;
     }
     pruneAreas(areas);
}

function grow( Area area )
{
     Point p = get first Point of q;
     if belongsTo(p,area)
     {
          removeFromArea(p, getArea(p) );
          addToArea(p,area);
          put all sourrounding points of p in q;
     }
}
```

2.1.6 Complexity

The complexity of the algorithm is not easy to determine as it depends on the content of the image. An image containing only one color would have a complexity of O(n), where n is the amount of pixels in the image.
The worst case would be an image with many small areas with smooth transitions, because this would cause many areas to be created and these would need to compete a lot.

2.1.7 Optimization

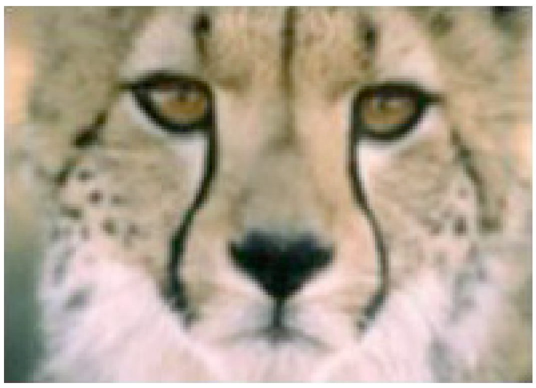
The variance of an area could be used to determine its inner coherence. A big variance would indicate that the area needs to be split.
Two adjacent areas could be combined if their mean color vectors are similar and the resulting variance would be lower than a given threshold.

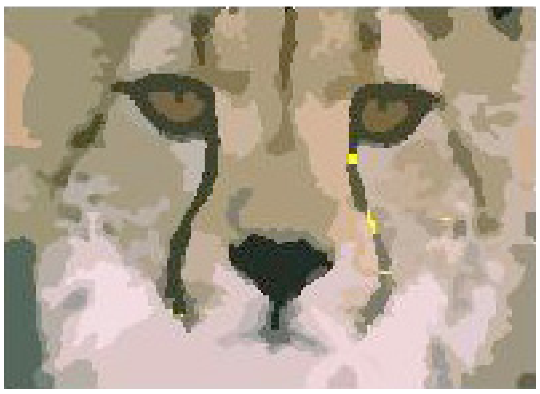
2.1.8 Examples

Bright yellow lines represent gaps between the segments.

2.1.8.1 Gepard

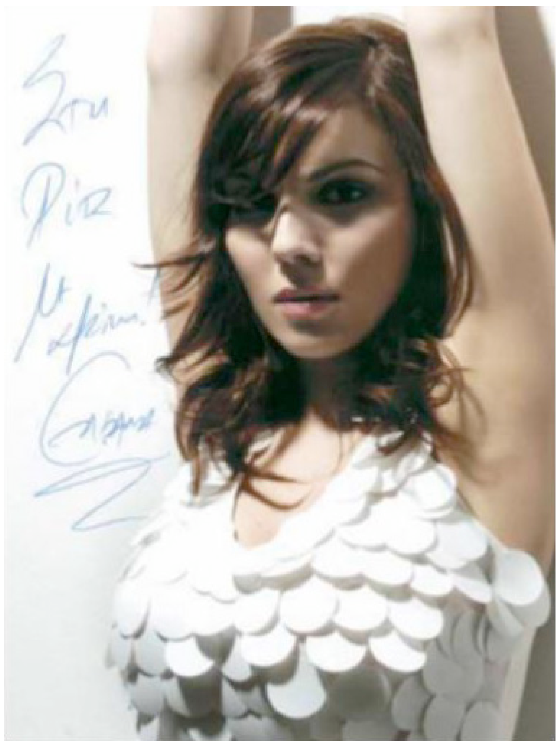
Real world, heavy texture

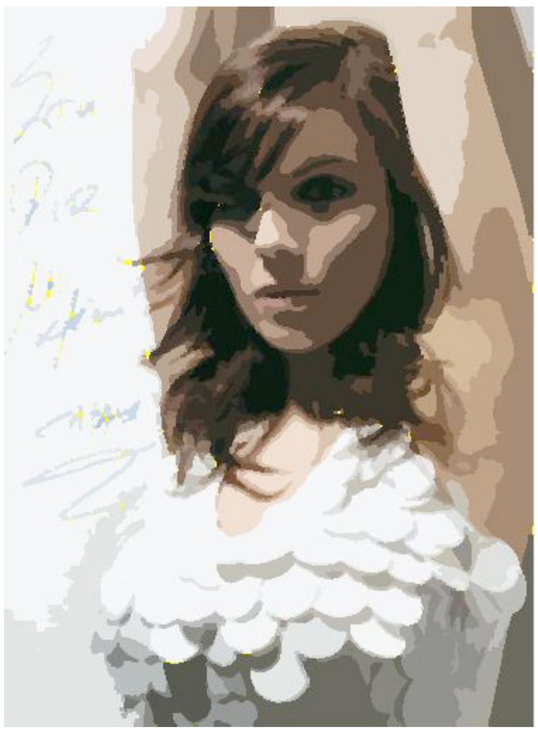




2.1.8.2 Girl

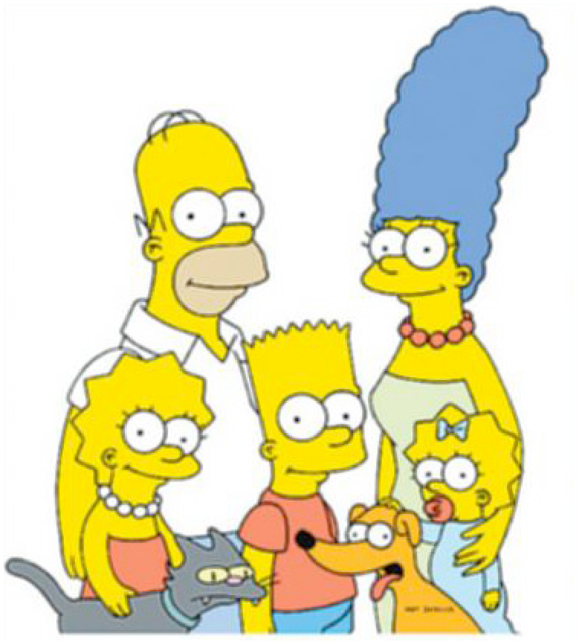
Real world, fading colors

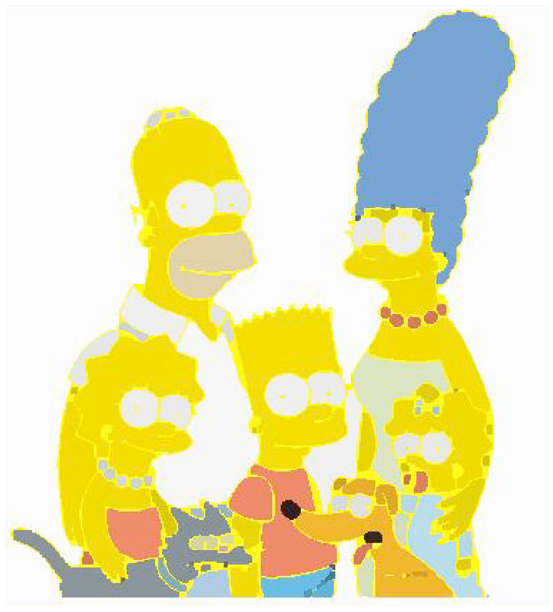




2.1.8.3 The Simpsons

Comic style, big segments





3 Edge tracing

One way to represent an object is its shape. The edge tracer algorithm traces the outline of a shape. It does NOT detect edges though. Its application is limited to already segmented shapes.
The algorithm is guaranteed to find the minimum number of points in the shape and it is able to trace lines (areas without space) as well.

3.1 Precondition

The input to the algorithm is a bitmap containing the shape information in Boolean representation: ‘Inside shape’ and ‘Outside shape’.

3.2 Post condition

The output is a sequence of all points in the shape in correct order. The order is clockwise and the shape is always closed.

3.3 Pseudo code

```
function tracePoint( P(t) )
{
     while( P(t) != P(2) and P(t-1) != P(1) ) do
     {
          m = calculateDirection( P(t),P(t-1) ) + 90° );
          P(t+1) = getPoint(P(t), m);
          while( isOutside( P(t+1) ) do
     {
               m += 45°;
               P(t+1) = getPoint(P(t), m);
     }
          tracePoint( P(t+1) )
     }
}
```

P_t is the point at time t.
The function calculateDirection calculates the slope between two points.
The function getPoint returns the point in direction m of the current point.
The function isOutside evaluates a point if it belongs to the current shape.

The condition ($P_t \quad P_2$ and $P_{t\ 1} \quad P_1$) assures that we terminate when the shape has completely been traced. The second part assures, that we terminate only when we evaluated the whole shape.

As we trace in clockwise order, the next point to be traced lies 90° to the left (-90°) of the current point in relation to the last traced point. We then evaluate every point in clockwise order (in 45° steps) until we find a point inside the shape. Note that we need to evaluate at max until 180° to the right, because this point is the same we came from, and it is assured that it lies within the shape.

In the following figures, grey pixels represent the shape. Green pixels stand for the edge pixels while red pixels mark the border outside of the shape. The yellow pixel is the currently active pixel.

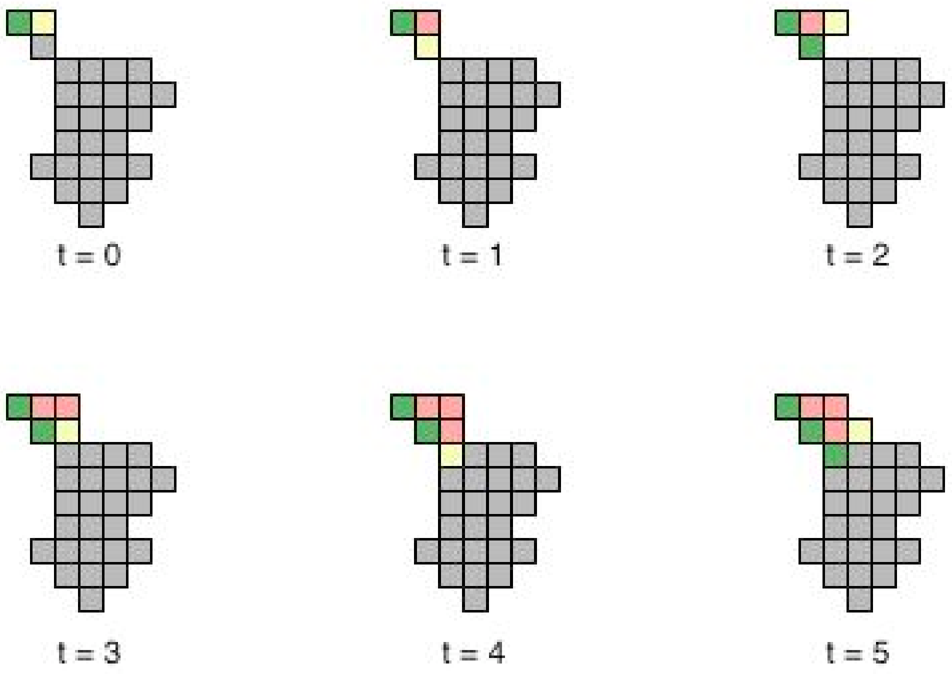


Figure 3-1: The first six steps of the tracing process

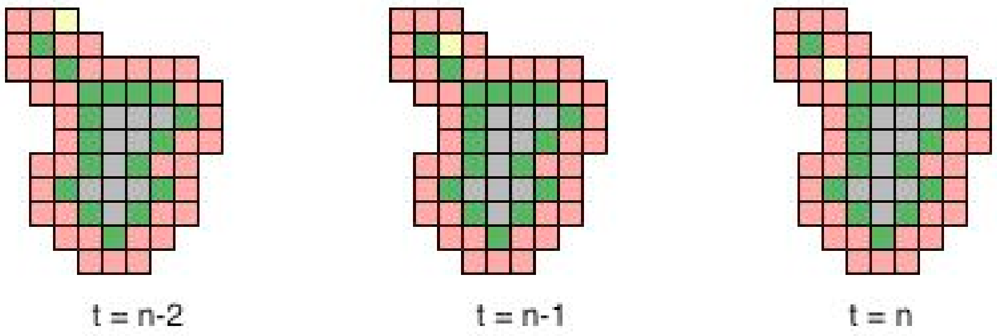


Figure 3-2: The last three steps of the tracing process

3.4 Optimization

When tracing diagonal, the procedure is optimal. When tracing straight (up <-> down, left <-> right), we can optimize the algorithm by one pixel. We know that the pixel $P_{t\quad 1}$ already traced the pixel 90° left of P_t. So we can start with the pixel at 45° left.

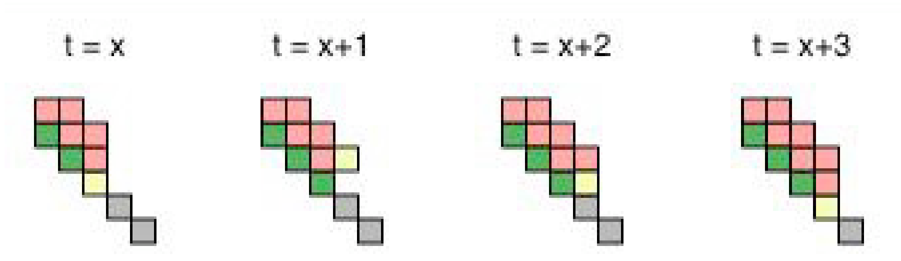


Figure 3-3: Tracing diagonal (start 90° left)

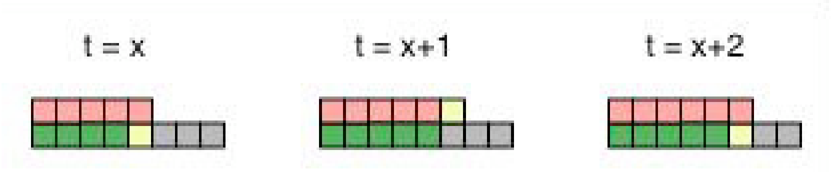


Figure 3-4: Tracing straight (start 45° left)

3.5 Complexity

One step needs to evaluate 1 to 7 pixels. As it is assured that the shape will be traced as a closing circle, turns in left direction must be compensated by turns towards the right side. We need at least 3 turns to the right for the simplest shape (triangle), which are about 18 pixel evaluations, 6 per corner.
If we add more pixels without adding more corners, the evaluation is 3 pixels per additional point.
If we add more corners, the angles converge towards 0° because turns to the left will result in compensating turns to the right.

$$\lim_{n} nf(\frac{2}{n}) + 3f(\frac{2}{3}) \quad nf(0) \quad 3n \tag{3.1}$$

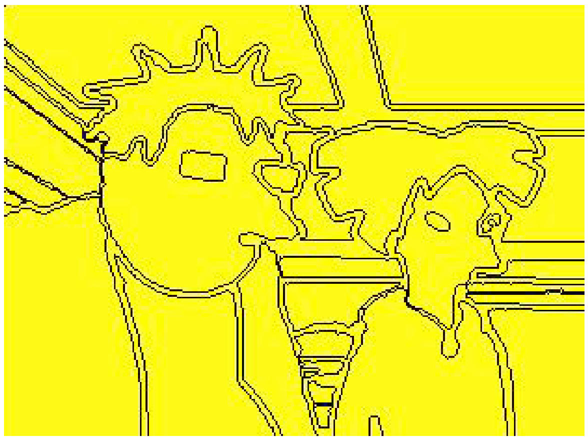
$f(0)$ is 3 evaluations for the un-optimized approach and either 2 or 3 evaluations for the optimized one. So we have $O_{\min}(2.5n)$.
The effective complexity will be higher than 3n, because n will never be big enough. Most shapes will have a complexity of $O(4n)$, which is still linear.

3.6 Examples

All examples have first been segmented [chapter 2] and then been traced. Bold lines result from two adjacent lines.

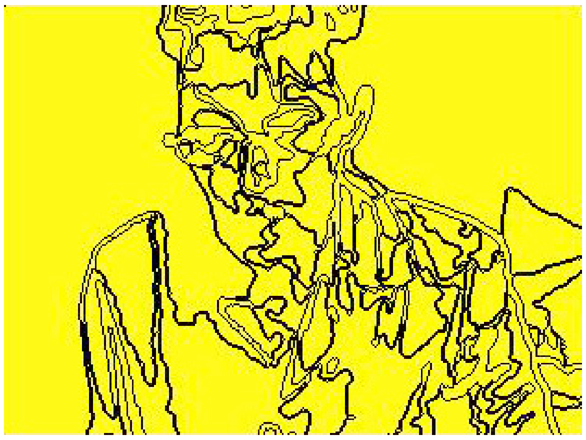
3.6.1 Hermes and Ami

Comic style, big segments



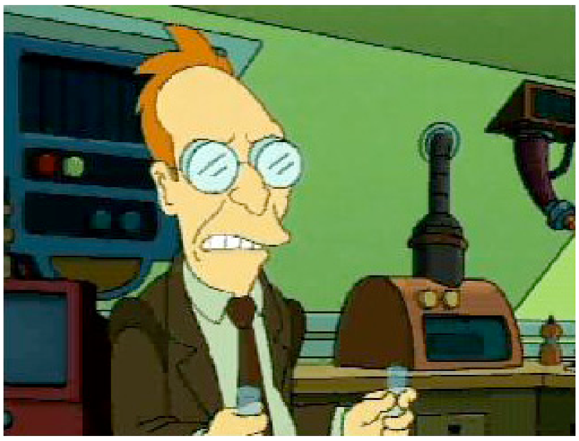
3.6.2 Human

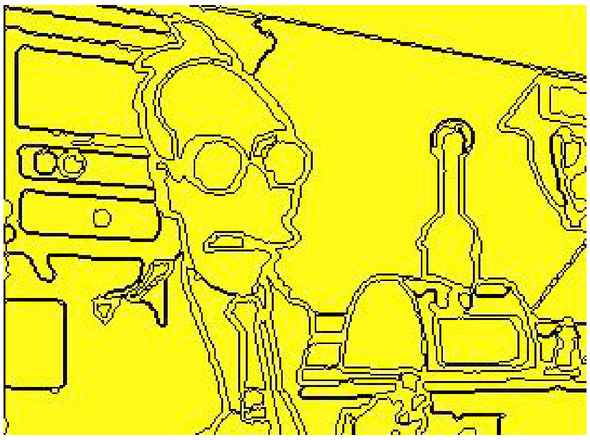
Real style



3.6.3 Prof. Farnsworth

Comic style, big and medium segments





4 Shape representation

Shapes can be represented in many different ways [33][28][21][31]. It is an essential task to produce a stable and lightweight description of shapes so they can be stored, retrieved and recognized later on. The representation can be categorized by their tolerance to noise, partial occlusion as well as their rotation and scale invariance. Perspective projection and distortion tolerance is also a major criterion when working on lower dimensional representations of higher dimensional objects.
The following representation is tolerant towards rotation. It is partially scale invariant as well and it showed to be quite stable to noise and partial occlusion.

4.1 CHAIN CODE

The algorithm first computes the vectors between every two points in the shape. Given a shape *S* consisting of *n* points

$$S = \{P_0, P_1, ..., P_{n\ \ 1}\} \tag{4.1}$$

the result is a set of vectors

$$\begin{aligned} V &= \{\overrightarrow{V_0}, \overrightarrow{V_1}, ..., \overrightarrow{V_{n\ \ 2}}\} \\ &= \{\overline{P_0 P_1}, \overline{P_1 P_2}, ..., \overline{P_{n\ \ 2} P_{n\ \ 1}}\} \end{aligned} \tag{4.2}$$

containing *n-1* vectors. Note that for cyclic shapes, the sets have the same power. To reduce the impact of noise we can calculate the vectors using points with an interval of .

Next we calculate the slope of every vector in V using

$$m_i = \tan(\ \ ) = \frac{y_i}{x_i} \tag{4.3}$$

giving us

$$M = \{m_0, m_1, ..., m_{n\ \ 2}\} \tag{4.4}$$

Note that these slopes are neither rotation nor scale invariant. A rotation invariant representation could be achieved by using the relation of the slopes or their delta. But for the next step this is not necessary.

4.2 PEAK DETECTION

If we think of the generated chain code as a function output, we can calculate the first derivation. The max/min of the function should then provide the desired peak points.

$$f : S \quad M \tag{4.5}$$
$$f' : M \quad \mathbb{R} \tag{4.6}$$

The used derivation formula is simply

$$f'(x_i) = f(x_i) \quad f(x_{i \ 1}) \tag{4.7}$$

Theoretically the calculated peaks should correspond to extrematas (corners) in the original shape. Experiments showed, that this is only true when is chosen to be big. This is reasonable because of the discrete nature of computer images. Adjacent pixels may only take one of eight possible directions. In smoothed (continuous) shapes, this problem is not that important. But smoothening flattens not only noise but also the peaks.

To receive an even more stable result, we introduce two more parameters

- Bandpass threshold
- Momentum μ

4.2.1 BANDPASS THRESHOLD

The first parameter is a bandpass threshold . A bandpass is widely used in signal processing to cut off undesired frequencies.

Because we use the slope *m* and not the angle we are sensitive to changes close to $\pm\frac{}{2}$.

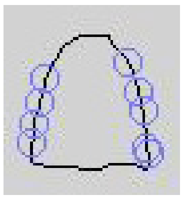


Figure 4-1: = , over detection of corners where slopes are close to $\pm\frac{}{2}$.

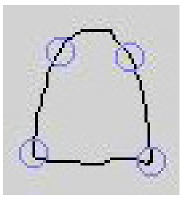


Figure 4-2: =2, 'Optically pleasing' corner detection

The bandpass reduces this sensitivity in limiting *m* to $[\quad ,+ \quad]$. What would be a reasonable bandpass threshold? This question is not easy to answer, because it depends on how tolerant we want to be. We achieved 'optically pleasing' results with 2, which limits the possible angles to $\frac{\ }{3},+\frac{\ }{3}$. To use such a low bandpass can be a severe drawback, because sharp corners will not be detected, as the following example shows.

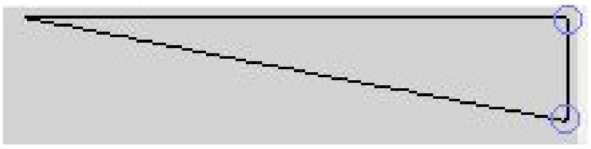


Figure 4-3: =2, upper left corner not detected

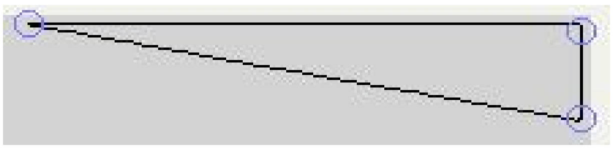


Figure 4-4: = , all 3 corners detected

Note: If the extra computational effort using tangens is not a concern, one should prefer to use instead!

4.2.2 Momentum

The second parameter is called momentum *μ*. It addresses small corners and changes in direction over a small portion of the shape only. These irregularities may appear because of distorting noise. While evaluating the shape, *μ* assures that only consistent changes in direction are considered.

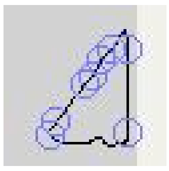


Figure 4-5: μ=0, over detection because of small direction changes

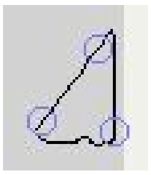


Figure 4-6: μ=7, momentum driven suppression results in a correct detection

4.3 Complexity

The complexity of the algorithm is linear: $O(n)$.

4.4 Discussion

The algorithm finds features in a shape similar to the ones a human selects. Difficulties arise with smooth shapes, e.g. circles.

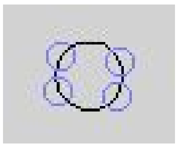


Figure 4-7: Key points detected in a circle

Instead of just using the position of the peaks we could use local feature descriptors encoding other properties like the curvature as well. In order to produce a scale invariant version, we could apply the algorithm to shape pyramids [7][8][9] instead.
Note that rotation of the shape causes a phase shift of the chain code signal. Rotation invariance is therefore a matter of reshifting the signal. Because the generated chain code is very compact, this can easily be done. Another approach to rotation invariance would be to connect the features in a way they form triangles and to store them in histograms.
Shape should never be the only criterion in object representation. Whenever possible, other properties like texture or color should be encoded as well.

4.5 Examples

All examples are organized in two parts. On the right we see the original shape, while the chain code representation as function plot is on the left. The found peaks are marked with blue circles on the shape and lines on the chain code plot, resp.

The parameters are described in detail earlier in this document.

∞	$_{s}$	Shape blur kernel size
∞	$_{cc}$	Chain Code blur kernel size
∞		Bandpass threshold
∞	μ	Momentum
∞		Slope offset

4.5.1 Bottom of a space shuttle

shape and chain code

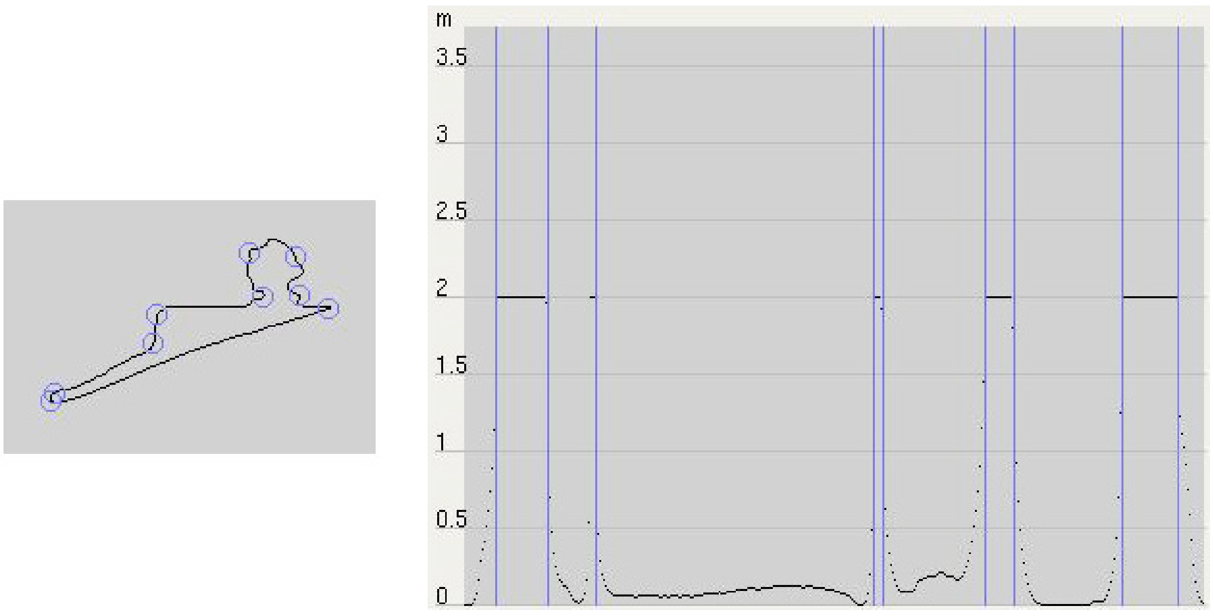


Figure 4-8: $_{s}$*=3,* $_{cc}$*=3, =2, μ=2, =20*

4.5.2 Comic head (small)

shape and chain code

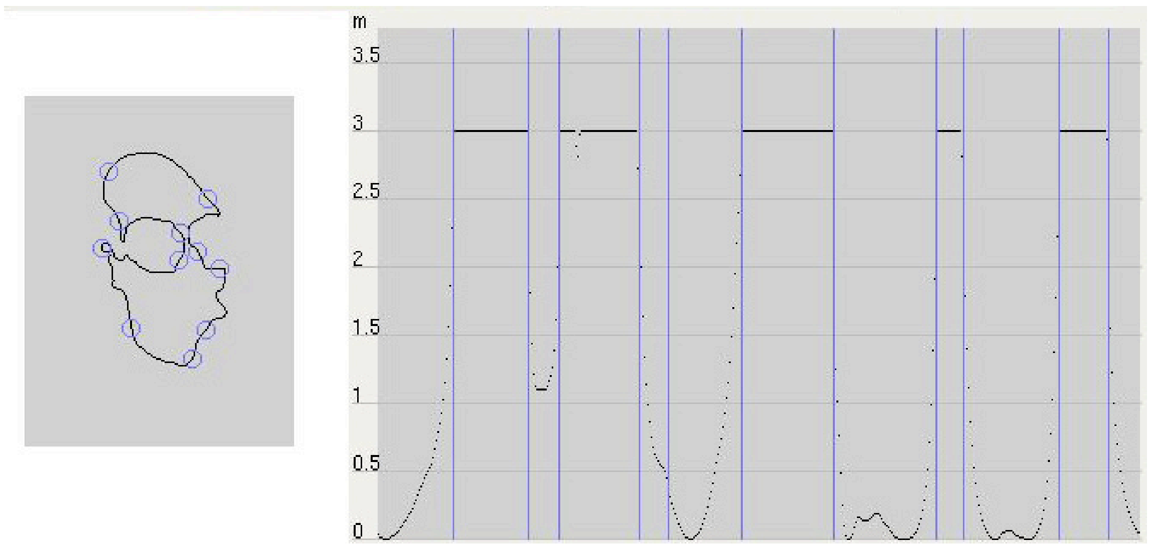


Figure 4-9: $_{s}$=3, $_{cc}$=3, =3, μ=1, =40

4.5.3 Comic head (big)

4 times the same shape with different parameters. $_{s}$ and $_{cc}$ are 3 for all shapes.

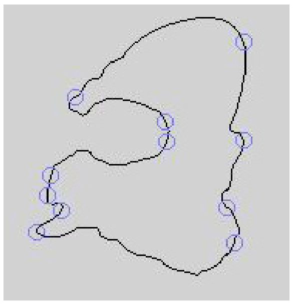


Figure 4-10: =5, μ=5, =40

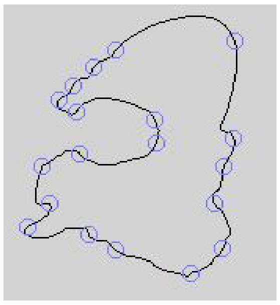


Figure 4-11: =5, μ=5, =4

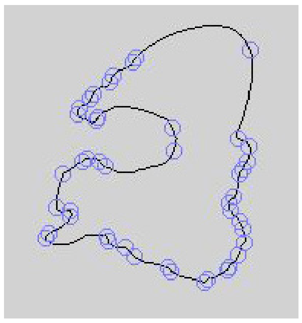


Figure 4-12: =5, μ=0, =4

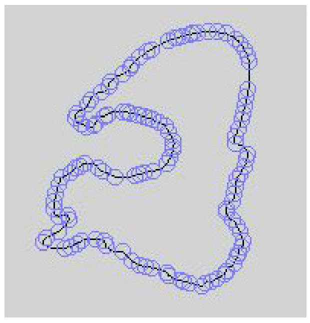


Figure 4-13: = , μ=0, =1

PART II

Optic flow

5 Optic flow

«The term "optic flow" refers to a visual phenomenon that you experience every day. Essentially, optic flow is the apparent visual motion that you experience as you move through the world. Suppose you are sitting in a car or a train, and are looking out the window. You see trees, the ground, buildings, etc., appear to move backwards. This motion is optic flow. This motion can also tell you how close you are to the different objects you see. Distant objects like clouds, and mountains move so slowly they appear still. The objects that are closer, such as buildings and trees, appear to move backwards, with the closer objects moving faster than the distant objects. Very close objects, such as grass or small signs by the road, move so fast they whiz right by you.» [17]

While this extract explained optic flow in a general way, the next one relates in to the artificial domain:

«The motion of individual pixels in a video is often called optic flow and is measured by attempting to find pixels in the next frame that correspond to a pixel in this (correspondence being measured by similarity in color, intensity, and texture). In principle, there is an optic flow vector at each pixel, forming a *motion field*. In practice, it is extremely hard to measure optic flow reliably at featureless pixels because they could correspond to pretty much anything.» [4]

For the biological vision, optic flow is of great importance. For the artificial vision, the sensing of optic flow from video stills has not yet been solved. Many approaches have been proposed so far. They can roughly be divided in [17]

- The feature-based approach
- The gradient-based approach
- The correlation-based approach

A very good description of optic flow is given in [17]. In this paper we propose a feature-based approach [chapter 8] as well as a gradient-based approach [chapter 5.1].

Optic flow has many applications especially in the sensor area. Steering and controlling of unmanned vehicles is a typical one, where reliability and real-time estimation are required. Algorithms need to be simple, so they can be rebuilt in hardware. The Facette approach tries to fulfill these criterions.

5.1 Facette

Facette is inspired by the facetted eyes of insects, therefore the name. Consider Figure 5-1 to get an impression on how optic flow in mosaic vision works. The RGB object proceeds from the lower left to the upper right corner. While moving, different *cells* become active over time.

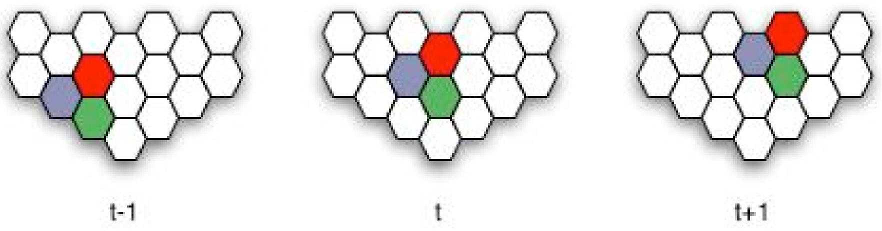


Figure 5-1: Mosaic vision

This activation history can be interpreted as the movement vector of the object and all these vectors together would form the optic flow.
This would work if the movement of the object was not faster then one cell per time step. If we choose a time step close to 0, this restriction would be fulfilled. The amount of frames and therefore the data to be evaluated would be immense. The visual system of humans registers a seamless flow at about 25 frames per second. With such a frame rate the motion between 2 frames will exceed 1 pixel by far.
On the other hand we could increase the size of the cells. If we use cells of the same size as the translation intensity of the object, we received a continuous activation history.
The drawback is now, of course, that slower motion will not be detected anymore.
Because motion is never homogene (an important fact for optic flow) there is no perfect cell size.
Facette addresses this problem by using cell pyramids [chapter 5.1.2].

5.1.1 The Cell

Because the algorithm operates in pixel space, a cell was chosen to be a square and not a hexagon like the 'cells' of the facetted eyes.
Hexagons have the advantage that every neighboring cell is equal distant, which is not the case for squares. Every square cell has 8 neighbors.

5.1.2 Cell pyramid

The cell pyramid is the core idea behind Facette. The pyramid consists of ℓ Facette layers (L), where the lowest layer will be labeled L_0 and the top-layer $L_{\ell\ \ 1}$. Each layer contains cells of different size. The cells in the lowest layer are the smallest; the ones in the top layer are the biggest.
The cells in every layer are arranged in a $N_k \quad M_k$ matrix. In the bottom layer, cells might correspond to a pixel in the source image or to a cluster of pixels. They could also represent features, but this approach is not discussed here. This does not matter for the pyramid and the cells in the bottom layer are considered to have unit size 1.

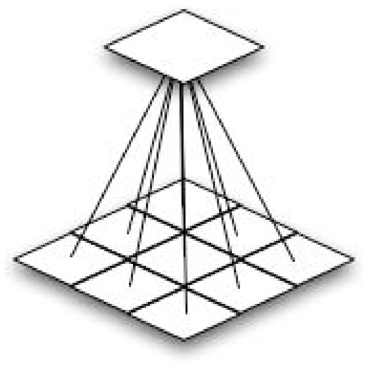


Figure 5-2: Cells are arranged hierarchically

Every higher layer is formed from the cells of its descendent layer in a way that every cell in layer k consists of 9 cells from layer $k \quad 1$.

$$L_k(x,y) \quad L_{k\ \ 1}(x+i+1, y+j+1) \tag{5.1}$$

where

$$\begin{aligned} i,j &\quad \{\text{-}1,0,1\} \\ x &\quad N_k \\ y &\quad M_k \end{aligned}$$

This has the advantage that every cell in layer k has a center cell and 8 surrounding cells in layer $k \quad 1$.

On the other hand, every cell in layer $k \quad 1$ contributes to 9 cells in layer k.

$$L_{k\ \ 1}(x,y) \quad L_k(x\text{-}1+i, y\text{-}1+j) \tag{5.2}$$

where

$$\begin{aligned} i,j &\quad \{\text{-}1,0,1\} \\ x &\quad \{2,..,N_{k\ \ 1} \quad 3\} \\ y &\quad \{2,..,M_{k\ \ 1} \quad 3\} \end{aligned}$$

This is important as seen in [chapter 5.1.3]. Equation (5.2) does not consider the cells at the borders. It would not be correct for these cells because the size of the layer decreases the more the higher the layer is (5.6). The special case at the border is ignored for the rest of the paper.

The value of the super-cell is the average value of these 9 sub-cells.

$$L_k(x,y) = \frac{\overset{1 \quad 1}{\underset{i=\ 1\, j=\ 1}{}} L_{k\ 1}(x+i+1, y+j+1)}{9} \tag{5.3}$$

Note that the indices x and y are shifted by 1 each. Alternatively, the formula can be written as

$$L(x_k, y_k) = \frac{\overset{1 \quad 1}{\underset{i=\ 1\, j=\ 1}{}} L(x_{k\ 1}+i, y_{k\ 1}+j)}{9} \tag{5.4}$$

where

$$\begin{aligned} x_k &= x_{k\ 1} \quad 1 \\ y_k &= y_{k\ 1} \quad 1 \end{aligned} \tag{5.5}$$

This is a direct effect from the pyramid structure. The dimension of the matrix at layer k is

$$N_k \quad M_k = (N_{k\ 1} \quad 2) \quad (M_{k\ 1} \quad 2) \tag{5.6}$$

The simplicity of equation (5.5) is one reason we decided that a super-cell needs to consist of 9 sub-cells. Every cell has exactly one center cell in every lower layer and is the center of a super-cell in every upper layer as demonstrated in Figure 5-3.

Theorems (5.1) and (5.2) lead to a structure of the pyramid as seen in Figure 5-3 and Figure 5-4.

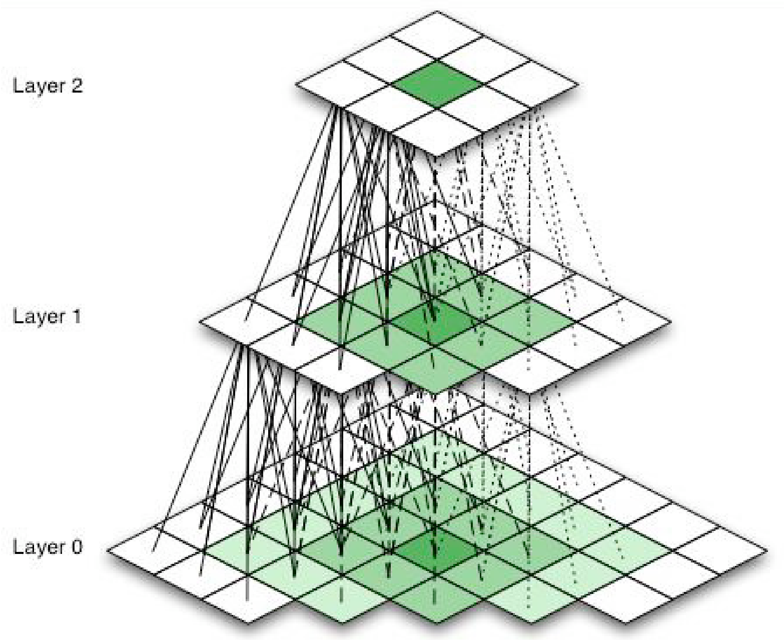


Figure 5-3: Pyramid structure (schematic)

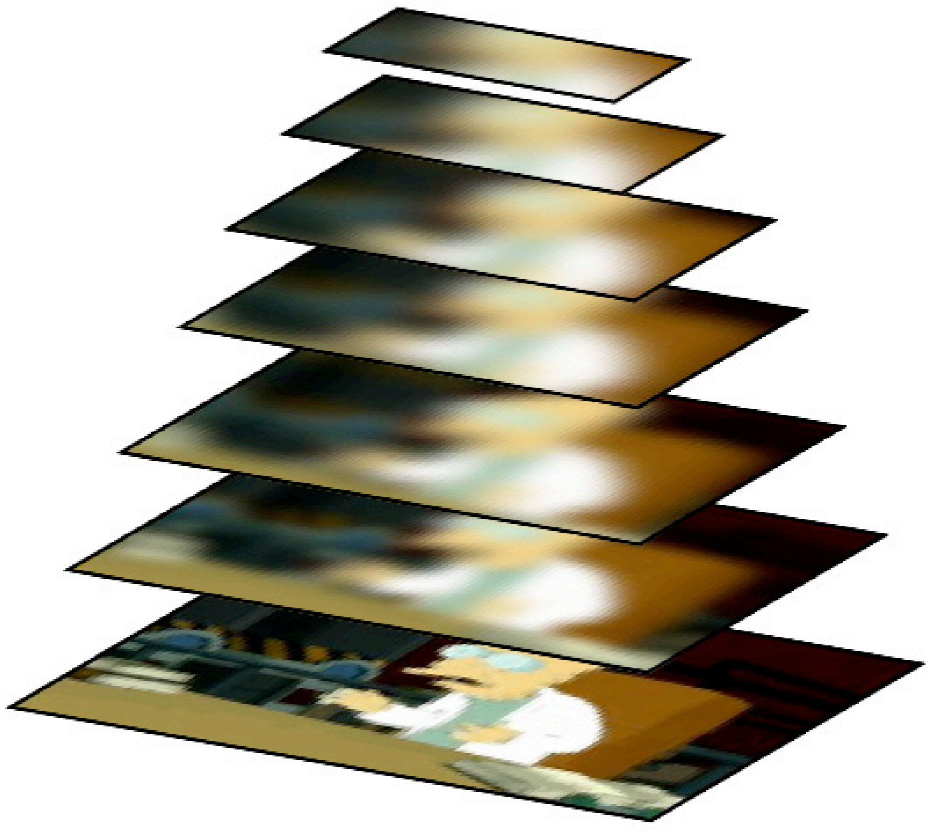


Figure 5-4: Example of a pyramid with 30 layers (every fifth layer is displayed)

As stated at the beginning of this chapter, every layer is responsible for sensing motion of a specific intensity. While the lowest level senses small motion based on one unit only, upper layers sense more intense motion of multiple units at a time.
This means that the upper layers do only sense motion of unit-groups, or swarms. The penetration effect [chapter 5.1.3] relaxes this restriction. How big the swarm needs to be to be tracked correctly depends on the content of the image. In clutter-free images even single units can be sensed, but usually the swarm size depends on the amount of translation.

5.1.3 Penetration

The penetration describes how the influence of a single unit in layer k is in the upper layers. Figure 5-5 shows the penetration schematically.

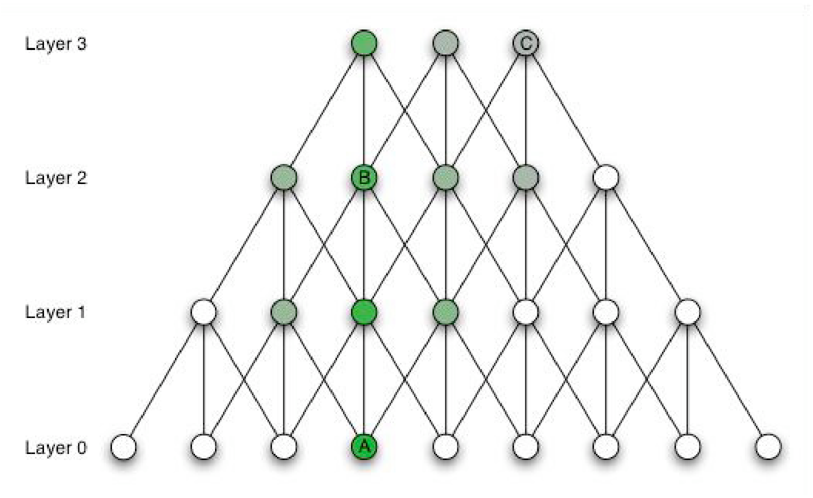


Figure 5-5: Penetration of a cell in its upper 3 layers

Theorem (5.2) states that every unit influences 9 units in its adjacent upper layer. The penetration can be written as a recursive formula

$$_{k+1} = \left(\sqrt{\quad_k} + 2\right)^2 \tag{5.7}$$

Formula (5.7) states how many cells are influenced, but not how strong this influence is. The amount of influence decreases proportional to the distance of the two cells (influencer and influencee) in the pyramid.

$$(x_k|x_0, y_k|y_0) = k \quad |x_k \quad x_0| \quad |y_k \quad y_0| \tag{5.8}$$

$$\frac{\sum_{i=\ 1}^{1}\sum_{j=\ 1}^{1}\left(x_{k\ 1}+i \mid x_0, y_{k\ 1}+j \mid y_0\right)}{9}$$

$$(x_k \mid x_0, y_k \mid y_0) = \quad 1,\ \text{if} \quad (x_k \left| x_0, y_k \right| y_0) = 0 \qquad (5.9)$$

$$0,\ \text{if} \quad (x_k \left| x_0, y_k \right| y_0) < 0$$

$($

According to (5.9) the influence of cell A on cells B and C in Figure 5-5 is $\left(B \mid A\right) = \frac{1}{9}A$ and $\left(C \mid A\right) = \frac{19}{729}A$, resp.

As one can see, the influence decreases faster towards the periphery while remaining higher in the epicenter.

5.1.4 Setting up the pyramid

How many layers does the pyramid need to consist of? This depends on the intensity of the motion to be captured. The cells of the top layer need to consist of at least as many unit cells as the maximum intensity of the optic flow. This leads to the equation

$$\ell = \mathit{floor}\left(0.5(\quad +2)\right) \qquad (5.10)$$

or to avoid the floor function

$$\ell = \frac{+2 \qquad \bmod 2}{2} \qquad (5.11)$$

For bigger the formula converges to

$$\ell \quad \frac{+1.5}{2} \qquad (5.12)$$

5.1.5 Optic flow detection

Currently the optic flow is detected by comparing the Facette pyramids ($_t$, $_{t+1}$) of frames t and $t+1$, resp. The comparison process is fairly simple. Starting at the top layer, we compare every cell in $_{t+1}$ to the corresponding cell and its 8 neighbors in $_t$. Corresponding means the cells are at the same position in the layer matrices. The cell with the highest correlation is selected. The tuple of this cells form a motion vector, where the cell $L_t(x_{\ell\;1}, y_{\ell\;1})$ in $_t$ is the source and the cell $L_{t+1}(x_{\ell\;1}, y_{\ell\;1})$ in $_{t+1}$ is the destination of the vector.

Next, we sink in both pyramids one level, retaining the motion vector. Sinking is performed by applying equation (5.5). Now we compare the source cell to the destination cell and its 8 neighbors in this layer. This is done recursively until the bottom layer is reached. The resulting motion vectors in the bottom layer form the optic flow.

The whole process is shown in Figure 5-6.

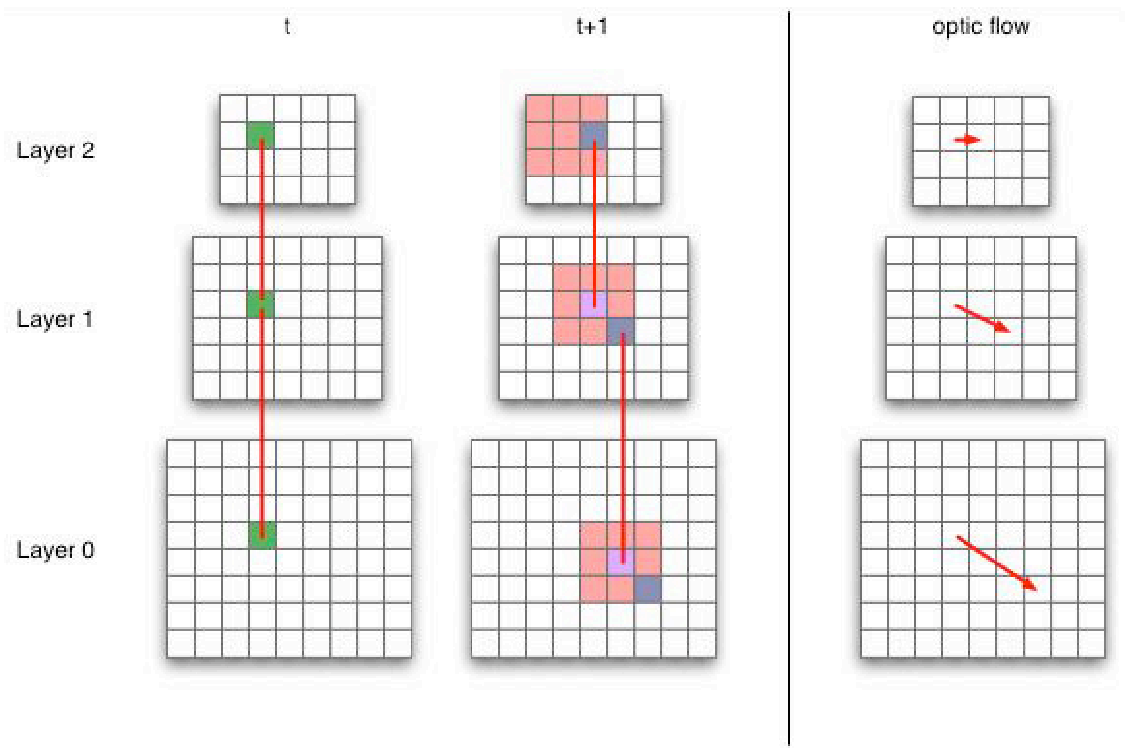


Figure 5-6: Optic flow estimation within the pyramid layers

The green cell remains static throughout the whole process. It is the source cell in pyramid $_t$. The red cells in pyramid $_{t+1}$ mark the possibly correlated cells. The blue cell is the cell with the highest correlation.
The arrows in leftmost fields mark the calculated motion vectors.

With every step we restrict the possible correlating unit cells because we restrict the possible correlating cells in layer $k\;\;1$ to 9 cells. These are the 9 sub-cells the super-cell in layer k consists of. We need to compare the cell in $_t$ to these cells only, which is exactly what we wanted. This is the second reason we defined a super-cell to consist of 9 sub-cells.

5.1.6 Performance

The performance of Facette depends on the intensity of the motion to be detected and the size of the frames only. In contrast to the feature-based approach [chapter 8] it does not depend on the content. This provides a static runtime, which is essential to many real-time applications.

The amount of comparisons which need to be calculated are given by

$$O = 9\ell\left(N_{\ell \quad 1} M_{\ell \quad 1}\right) \tag{5.13}$$

By applying equation (5.11) we get

$$O = \frac{9(\quad +2 \qquad \bmod 2)\left(N_{\ell \quad 1} M_{\ell \quad 1}\right)}{2} \tag{5.14}$$

and for bigger

$$O \quad \frac{9(\quad +1.5)\left(N_{\ell \quad 1} M_{\ell \quad 1}\right)}{2} \tag{5.15}$$

Experiments support this equation. On our test computer [chapter 12.2] we benchmarked frames with ℓ from 4 to 20 and $N \quad M$ from 32x22 up to 500x470. The relation

$$\bar{\ } = \frac{\text{Total time } (t)}{\text{expected } O} \tag{5.16}$$

remained almost constant at a value of 0.5 µs with a standard deviation of 0.02 µs.

Given a frame rate of 5 fps and a translation of 20% of the frame size, we can calculate the maximum frame size for which real-time optic flow estimation on this computer is possible.

The maximum total tracking time per frame is

$$t = \frac{1}{5}\ [\mathrm{s}] \tag{5.17}$$

Reforming equations (5.16) and (5.13) gives us

$$O = \overset{t}{=} = 9\ell\left(N_{\ell \ \ 1} M_{\ell \ \ 1}\right) \tag{5.18}$$

By applying equations (5.6) and (5.12) formula (5.18) becomes

$$\begin{aligned}
\overset{t}{=} &= 9 \ \ \frac{\ \ +1.5}{2}\Big(\big((N\,) \quad)\ 1.5\big)\big(M\,) \quad)\ 1.5\big)\Big) \\
&= 9 \ \ \frac{2 \ \ +3}{4}\Big(\quad \frac{2N\,)\ 2 \ \ )\ 3}{2}\Big(\quad \frac{2M\,)\ 2 \ \ )\ 3}{2}\Big(\Big(\\
&= \frac{9}{8} \ \ \frac{2 \ \ +3}{2}\Big(\big(2N\,)\ 2 \ \ )\ 3\big)\big(2M\,)\ 2 \ \ )\ 3\big)
\end{aligned} \tag{5.19}$$

Assuming an aspect ratio of 4:3, M can be rewritten as

$$M = \frac{3}{4}N \tag{5.20}$$

and can be expressed as

$$= \frac{1}{5}N \tag{5.21}$$

By applying equations (5.21) and (5.20) to (5.19) we get

$$\begin{aligned}
\overset{t}{=} &= \frac{9}{8} \ \ \frac{2N+15}{10} \quad 2N\,(\ \frac{2}{5}N\,(\ \ 3 \quad \frac{3}{2}N\,(\ \frac{2}{5}N\,(\ \ 3 \\
\frac{8000}{9}\overset{t}{=} &= (2N+15)(16N\,(\ \ 30)(11N\,(\ \ 30)
\end{aligned} \tag{5.22}$$

This is a polynom of order 3. We solved it numerically for our test computer and received

$$N \quad 100$$

which leads to a frame format of

$$100 \quad 75$$

We tested the algorithm with this frame size and got results as expected around 200 ms.

This might not seem a big format, but as Facette can deal well with downsampled images, it can in fact handle quite big frame sizes. If we use a downsampling factor of 5, which still delivers good optic flow detection, we are able to track frames of size

$$500 \quad 375$$

in real-time. By applying some optimizations [chapter 5.1.7] this format can be increased even more.

5.1.7 Optimization

Two factors in (5.13) can be modified to get better performance. Either the amount of layers ℓ or the size of the matrix $N \quad M$ can be reduced.

5.1.7.1 *Reducing* ℓ

Experiments showed that most of the frames do not contain movement at maximum intensity. Most of the time the motion intensity is a lot lower.
A better approach therefore would be not to start in the top layer. Instead the algorithm could first examine a lower layer. If this layer showed no optic flow, chances are high that the upper layers would not show motion as well. We can effectively prune away a part of the pyramid. In this case we could save additional computation time by building the pyramid iteratively. First we build all the layers up to this key-layer. If we don't detect optical flow, the rest of the pyramid needs not be built. Otherwise we build the pyramid up to the next of these key-levels, and so on.
How many of these key-layers to choose and at which intensity has not yet been examined.

5.1.7.2 *Reducing* $N \quad M$

Reducing $N \quad M$ is the same as reducing the amount of image data to be processed. The parts of the image, which do not show motion do not need to be processed. These can be avoided with *saliency maps*. These maps (mostly binary) define regions of interest. For example the center of the image or the moving regions.

To detect which parts of the image are in motion we need only to subtract frame t from frame $t+1$. This reliably produces a saliency map of the moving regions.

5.1.8 Neuronal approach

Figure 5-5 reminds somehow of a *MLP (multilayer perceptron)* neuronal network. In fact, Facette could be realized as a neuronal network similar to MLP's. The bottom layer of the pyramid would form the perception layer (retina cells), the top layer corresponds to the output layer and the remaining layers to the hidden layers.
This approach would have some interesting advantages towards the classic implementation. Shape preference, blending of undesired parts and direct matching with spiking neurons (no need to build the second pyramid) are just a few.
The neuronal approach remains a topic of research.

5.1.9 Discussion

This chapter described a new approach to optic flow estimation. Its main advantage is its simplicity, which allows it to be rebuilt in hardware, but the concept itself could be expanded to other capabilities [chapter 5.1.8].
Tests have showed good results both in performance and detection, but not perfect ones. Because it is a gradient-based approach, it fails when the gradients are not distinct enough. This is a well-known problem [19]. A small percentage of these wrongly detected flow vectors could be pruned using post-processing methods like trajector pruning [chapter 8.4.2] or a derivate of the median filter [chapter 1.2].
Definitely more research is required to evaluate Facette thoroughly.

5.1.10 Examples

All examples are made up of three kinds of images.

- Optic flow
- Motion field
- Source-sink field

5.1.10.1 Optic flow

The first category shows the detected optic flow. The red vectors show the direction of the flow, starting at the square.

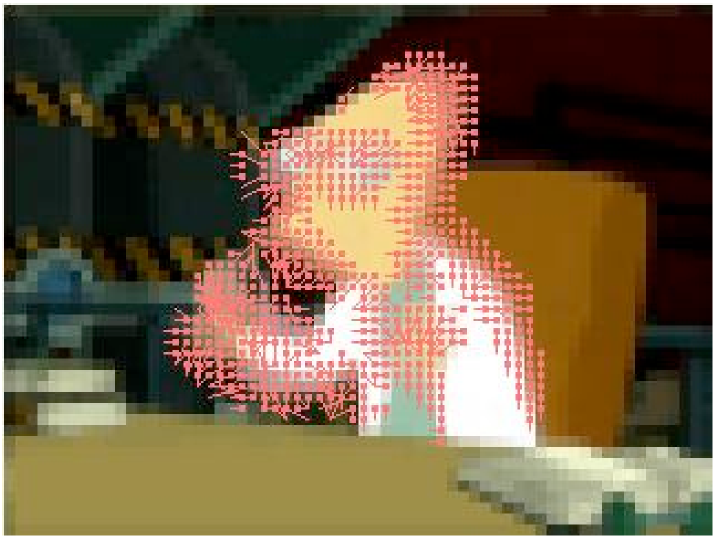


Figure 5-7: optic flow

5.1.10.2 Motion field

The second type shows the detected motion fields. These are fields of clustered motion vectors. The vectors are clustered with *k-means* [1] using their size and angles as the correlation measurement.

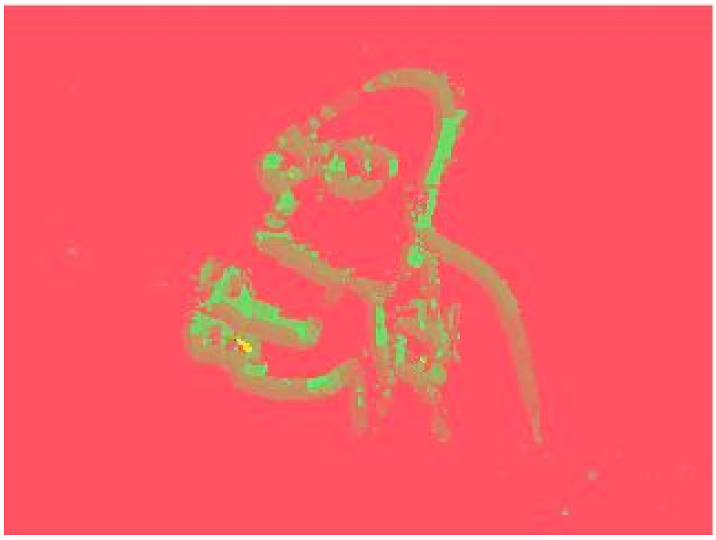


Figure 5-8: motion field

5.1.10.3 Source-sink field

The third class of images display so called source-sink-fields. The white cells state that two or more motion vectors point to this field. This is an indication for an occlusion, because cells from frame $t \quad 1$ shift over cells from frame t for which the algorithm does not find any correlating cell.
The black cells are cells where no motion vector points at. These are the source of the motion.
The light grey cells are in motion and indicate a moving swarm (a collection of cells with same motion intensity).
Finally, the dark grey cells indicate static cells.

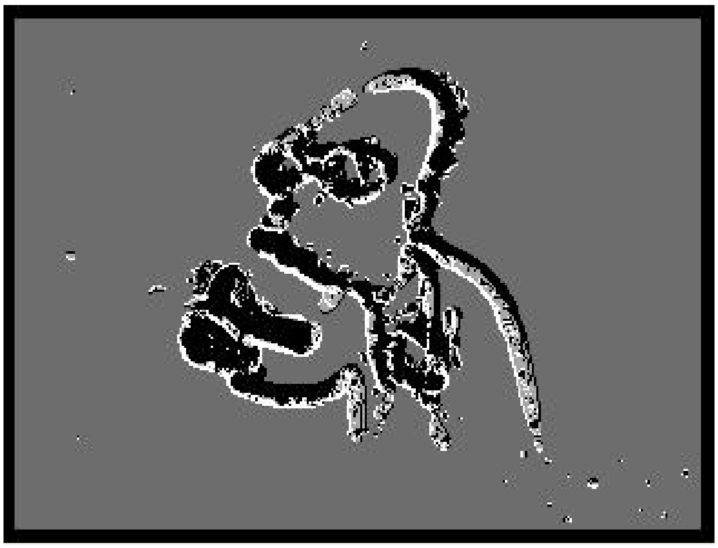


Figure 5-9: Source-sink field

5.1.10.4 Long house

This is one of the classical images used in artificial vision. It was taken from Berkeley [38]. The original image (510x480) was downsampled by a factor of 5 to (102x96). For we used 10, which lead to a total estimation time of 151 ms.

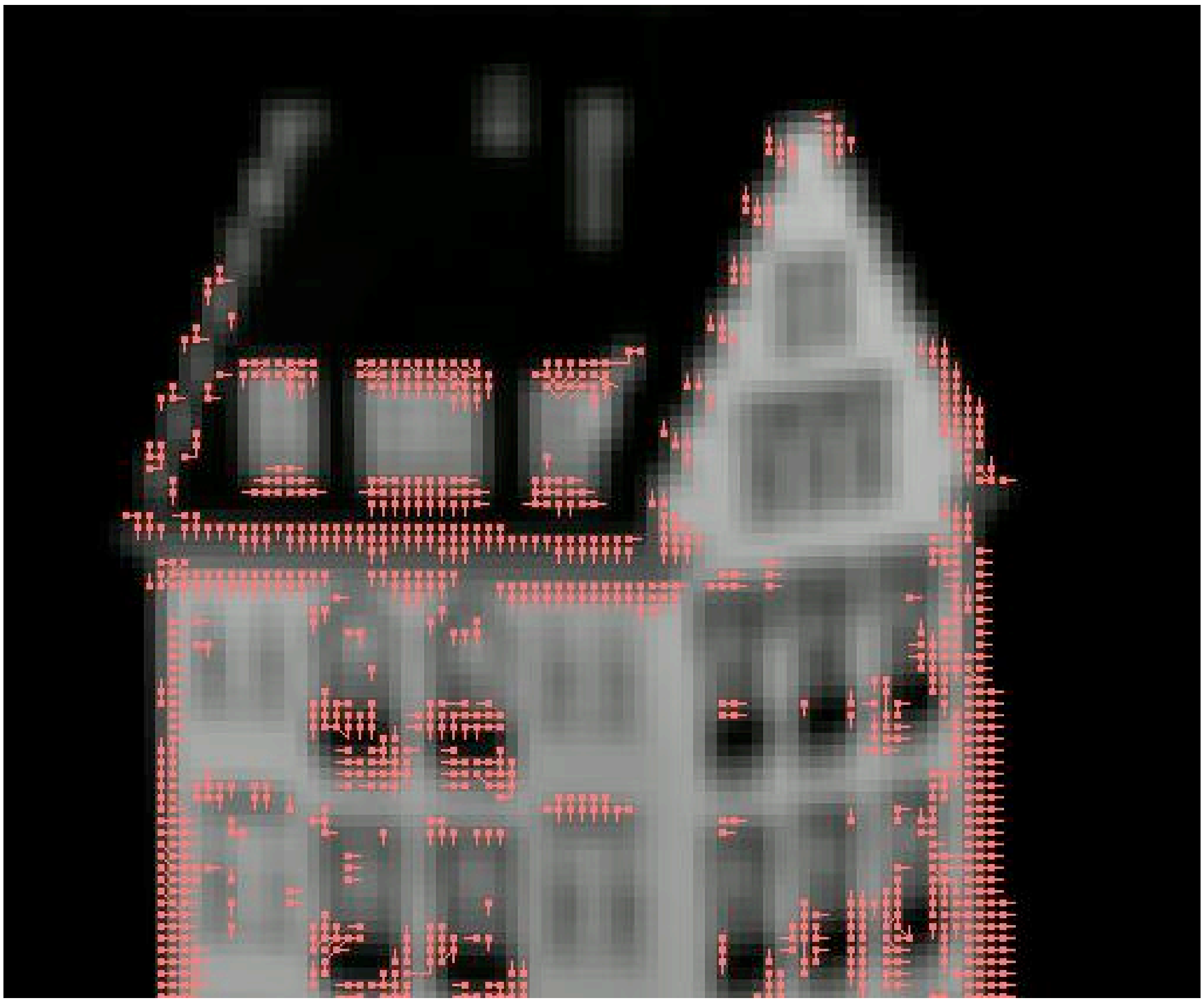


Figure 5-10: Optic flow. Note the stronger motion intensity at the bottom because the house rotates around a pole near to the roof.

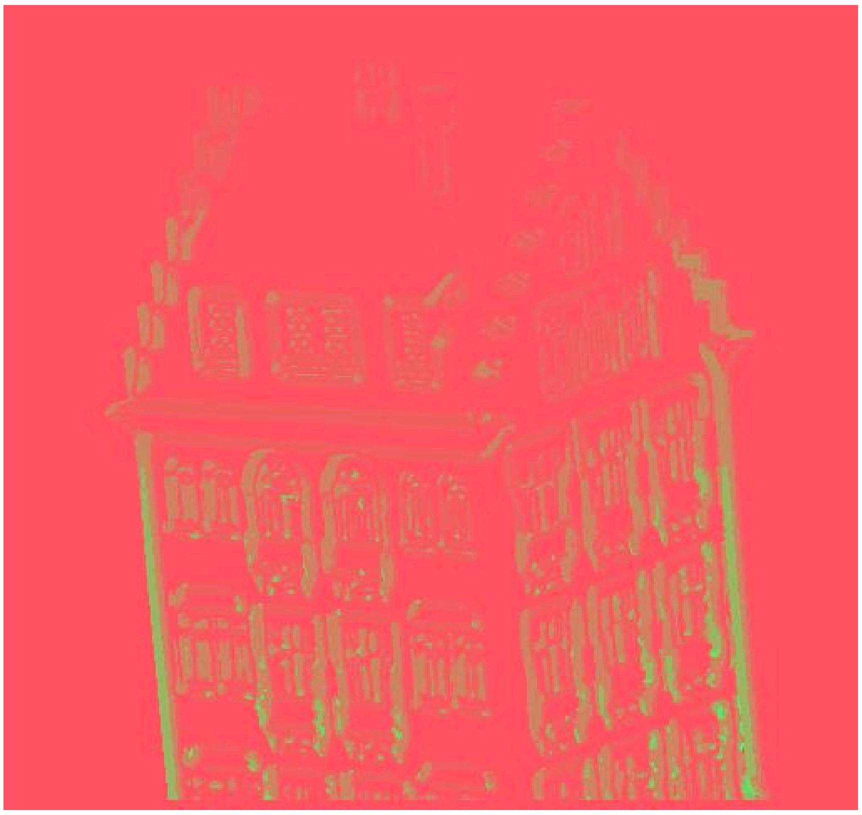


Figure 5-11: Motion field. The stronger motion seen in Figure 5-10 finds its representation in the bright green fields, which as well indicate high intensity.

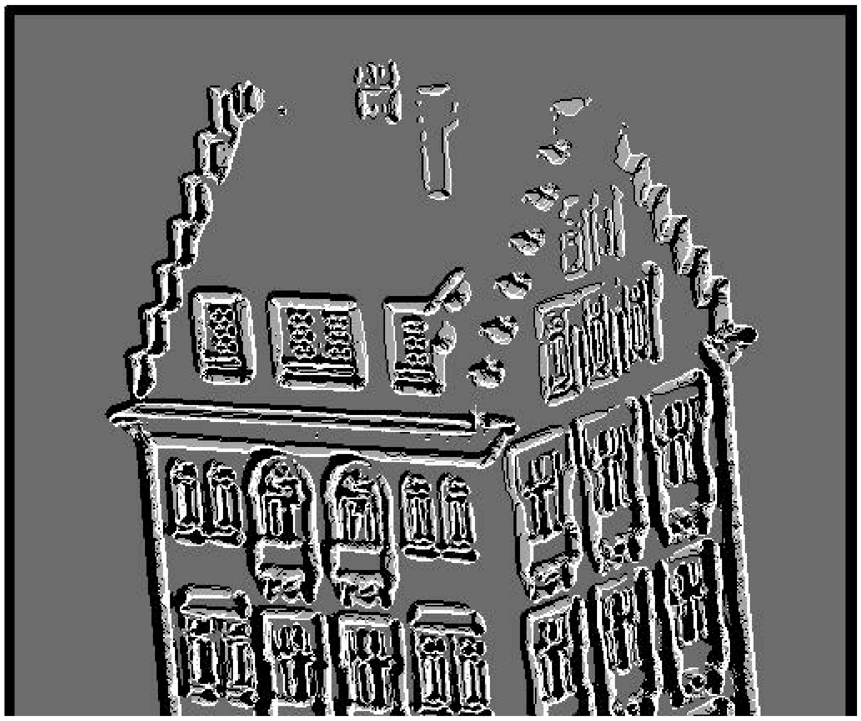
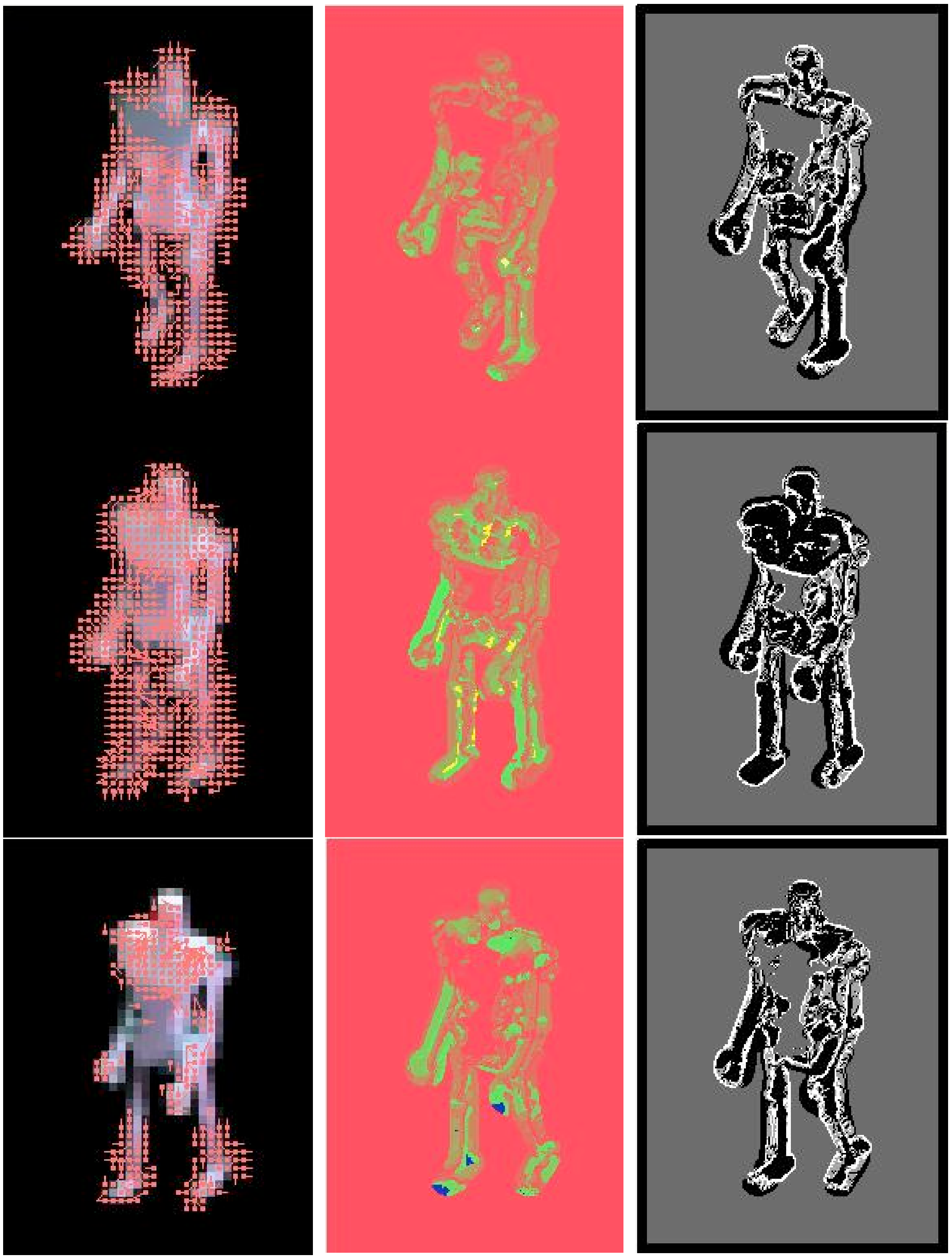
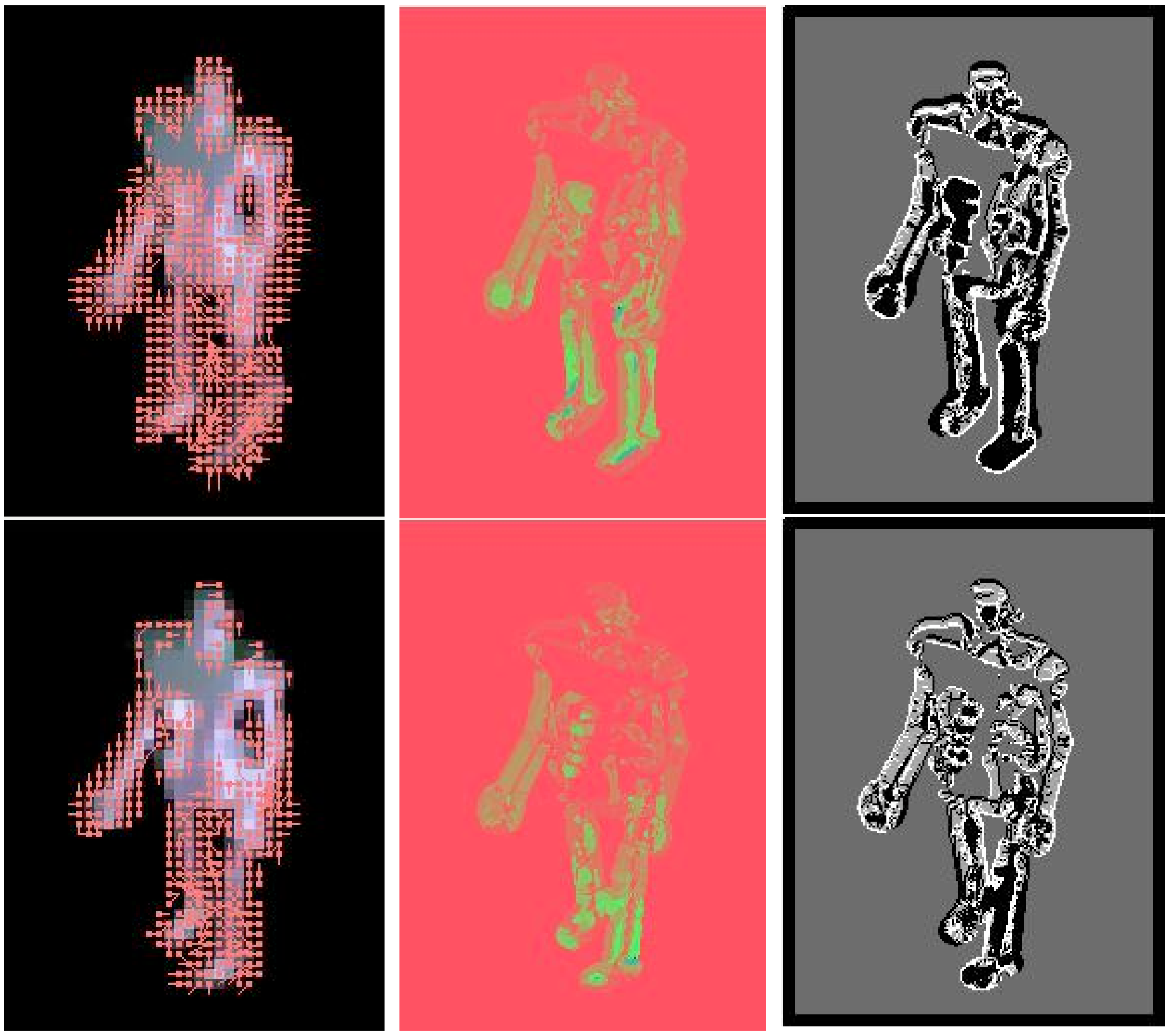


Figure 5-12:Source-sink field. Again the stronger motion at the bottom leaves its tracks.

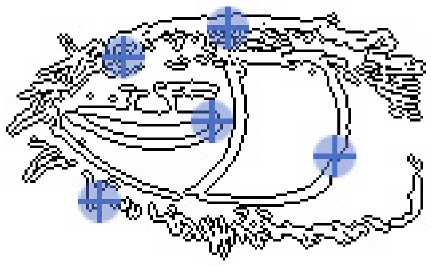
5.1.10.5 WALKING ROBOT

The following sequence shows a walking robot and the resulting optic flow, again with the associated motion fields.
The frames (190x255) were downsampled by a factor of 5 to 38x51. was set to 10 once again. The optic flow was calculated at 22 ms per frame.
Note how well the white cells reflect the shape of the robot.





PART III



Features

6 Feature detectors

Features are regions in an image with high entropy. They need to be easy detectable and distinctable enough so we can determine what object the feature(s) belong(s) to.
The visiual system of humans makes extensive use of features. When looking at an object we look for distinct points. The more we find the better we recognize the object.

Finding good features is still a topic of intense research [7][8][9][10][21][27]. In this work we integrated three different feature detectors. The aim was to inspect the detectors in a comparable environment for their performance. The implemented feature detectors are:

- Chain code feature detector
- Harris corners feature detector
- PCA SIFT feature detector

We will first briefly describe their functionality and then compare their performance.

6.1 Chain code feature detector

The chain code feature detector bases on chain codes [chapter 4.1] introduced in this work. It focuses on the detection of peaks in shapes.

The detector first segments the image using a region-growing algorithm [chapter 2.1] and produces an outline of the areas using an edge tracer [chapter 3].
Finally, the shapes are analyzed using the chain code generation algorithm [chapter 4.1] to produce the features.

While the detection performs well, the segmentation process produces unreliable areas, which in turn lead to unstable features.
Also the detector produces almost every feature twice, because the segmentation produces areas, which share their edges with one or more other areas. In this way, every edge is traced and encoded twice, except the edges at the borders. But these edges should not be encoded anyway, as they are most likely to exist only because the image captures only a part of the world.
This is a subject of enhancement. Preliminary edge pruning could enhance the performance.

The following charts show that most of the computation time is used by the segmentation process in order to create the segments. Instead of using a region-growing algorithm, one could use an edge detector, like the Canny edge detector [11]. This could reduce the time expenses and stabilize the feature detection process.

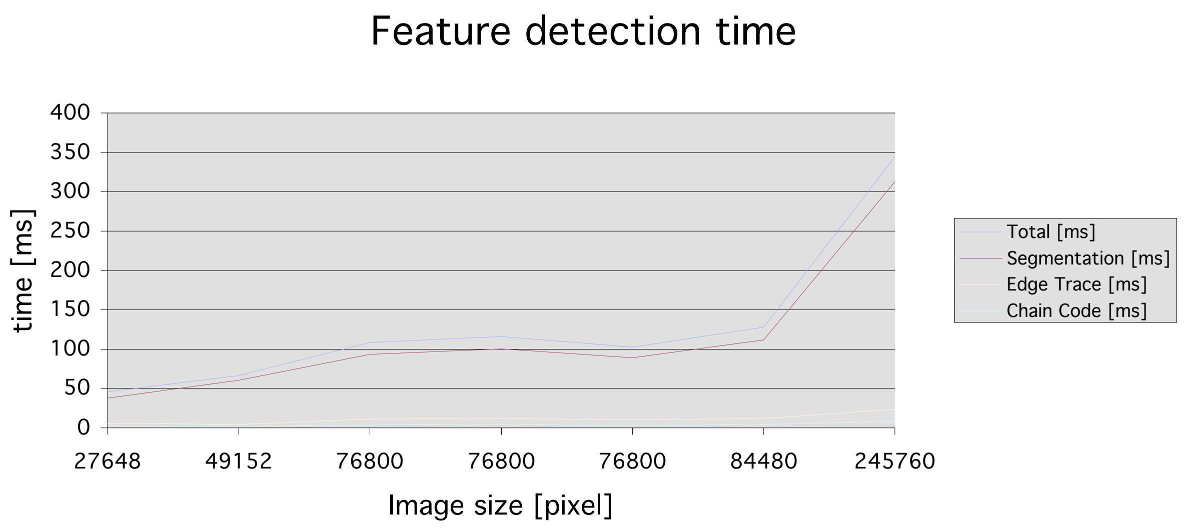


Figure 6-1: Image sizes from 192x144 up to 512x480 (avg. over 50 frames each)

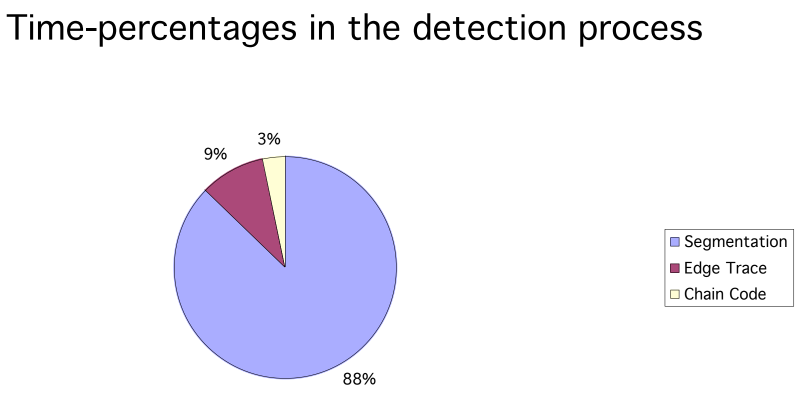


Figure 6-2: The segmentation process uses almost 90% of the time used in the detection process

6.2 Harris corners feature detector

Harris corners is the most famous corner detector. For details please refer to the literature [5]. The following extract was taken from [6].

«Consider the following matrix

$$M = \begin{matrix} \frac{I}{x}^{2} & \frac{I}{x}\frac{I}{y} \\ \frac{I}{x}\frac{I}{y} & \frac{I}{y}^{2} \end{matrix} \qquad (6.1)$$

where $I(x,y)$ is the grey level intensity. If at a certain point the two eigenvalues of the matrix M are large, then a small motion in any direction will cause an important change of grey level. This indicates that the point is a corner. The corner response function is given by:

$$R = \det M \quad k(traceM)^2 \qquad (6.2)$$

where k is a parameter set to 0.04 (a suggestion of Harris). Corners are defined as local maxima of the cornerness function. Sub-pixel precision is achieved through a quadratic approximation of the neighborhood of the local maxima.»

6.3 PCA SIFT feature detector

SIFT stands for Scale Invariant Feature Transform and was invented by David G. Lowe [7][8][9]. PCA SIFT is an improvement of the SIFT technology proposed by Yan Ke and Rahul Sukthankar [10].

We did not use the PCA SIFT feature descriptors but only the scale-invariant feature detector. We integrated directly the source code from Yan Ke and Rahul Sukthankar, with their courtesy.
As we did not need scale invariant features for the tracking task (change in scale is assumed to be small) we modified the code to provide better performance [7][8][9][10]. The changes are mainly:

- Build only 1 DoG (difference of Gauss) octave
- Don't double the image size
- Don't calculate keypoint orientation
- Don't calculate feature descriptors

This assures comparable testing conditions.

Compared to the other detectors used in this work, the SIFT detector finds features not only at edges but also inside of areas.

6.4 Comparison

6.4.1 Setup

All edge detection algorithms have been tested under the same conditions [chapter 12.2] to produce comparable results. The results are averaged over 50 frames in each movie file. The files have been chosen randomly to avoid preferred conditions for one of the algorithms.
They have been evaluated in terms of their performance and the amount of produced features. While the performance is easy to compare, the produced features depend highly on the chosen parameters. In addition, we cannot define how many features are optimal per image. Other classification numbers, like the feature distribution would be more appropriate. But even these cannot state directly whether the algorithm is good or not. Therefore, the provided values are used to demonstrate how different the three algorithms perform on different images.

6.4.2 Performance

The performance measurements are in milliseconds.

Image size	Chain Code	Harris Corners	PCA SIFT
27648 (192x144)	45.92	8.41	647.09
49152 (192x256)	66.27	17.12	1176.43
76800 (320x240)	108.49	27.36	1173.42
76800 (320x240)	116.25	24.37	1172.29
76800 (320x240)	102.66	28.27	1782.03
84480 (352x240)	127.72	28.36	1987.99
245760 (512x480)	344.60	74.10	5771.83

It can easily be seen, that the SIFT algorithm is outperformed by the others by far.

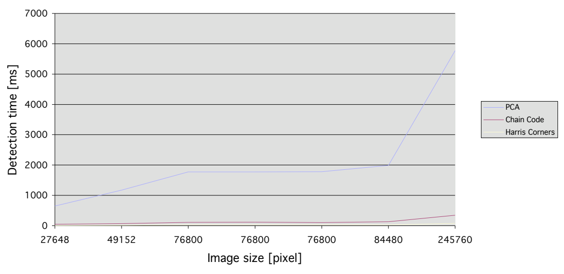


Figure 6-3: Performance benchmark

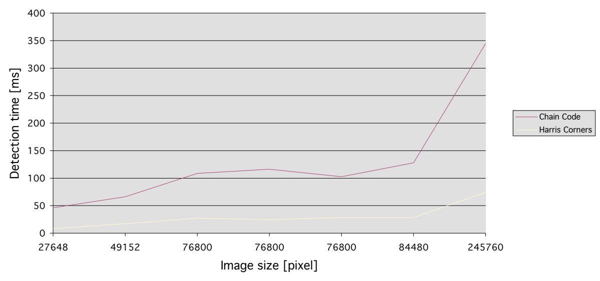


Figure 6-4: Harris Corners and Chain Code

Harris Corners detector is in average 70 times faster than the PCA SIFT scale-invariant feature detector and 4 times faster than the Chain Code detector, resp.
This is the main reason we focused on Harris Corners in this work.

6.4.3 Feature extraction

Image size	Chain Code[1]	Harris Corners	PCA SIFT[2]
27648 (192x144)	70	53	59
49152 (192x256)	34	113	44
76800 (320x240)	147	170	108
76800 (320x240)	172	82	75
76800 (320x240)	141	207	118
84480 (352x240)	167	79	32
245760 (512x480)	333	113	262

The following graph shows that the amount of features extracted depends strongly on the size of the image.
In addition the amount of detected features per detector varies. This because the detectors extract features based on different criteria. None is, of course, content invariant. This could mean that we should combine one or more feature detectors to provide more reliable feature extraction.

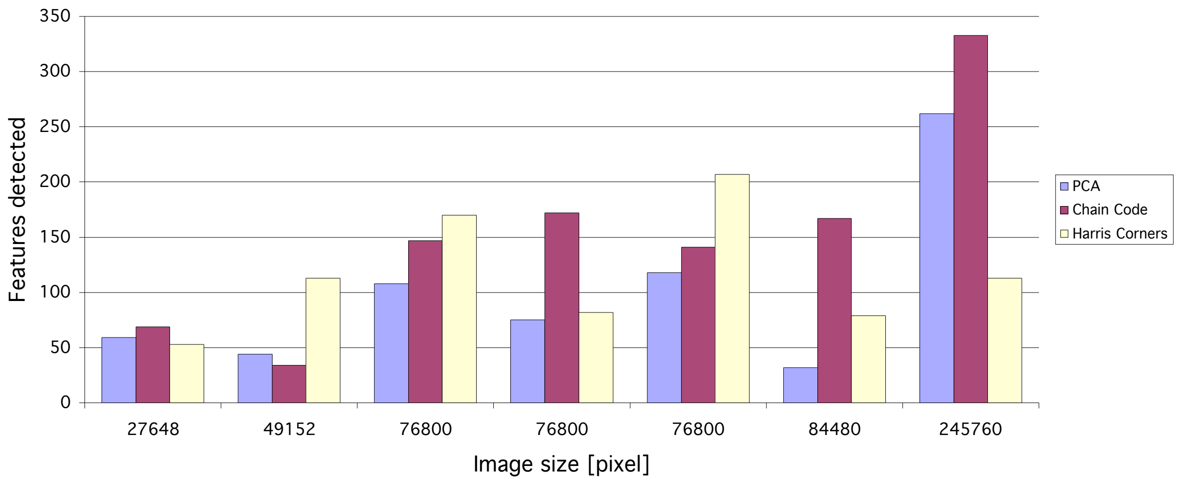


Figure 6-5: The amount of features detected varies depending on the content of the image

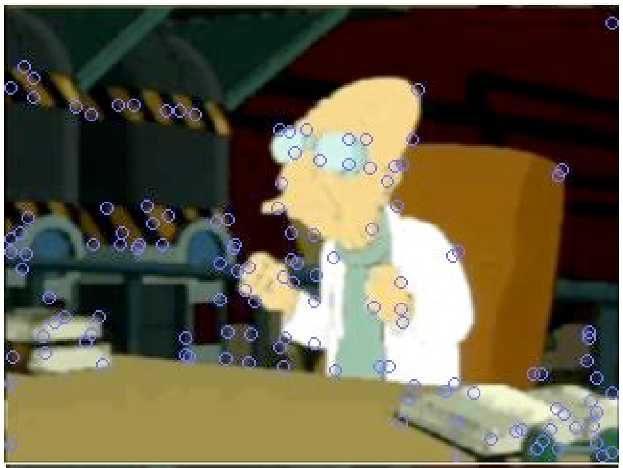
[1] The effective amount of unique features is assumed to be lower [chapter 6.1]
[2] When operating on double sized images, more features would have been detected [10]

6.5 EXAMPLES

The following examples show some images and their extracted features. In addition the time taken is mentioned.

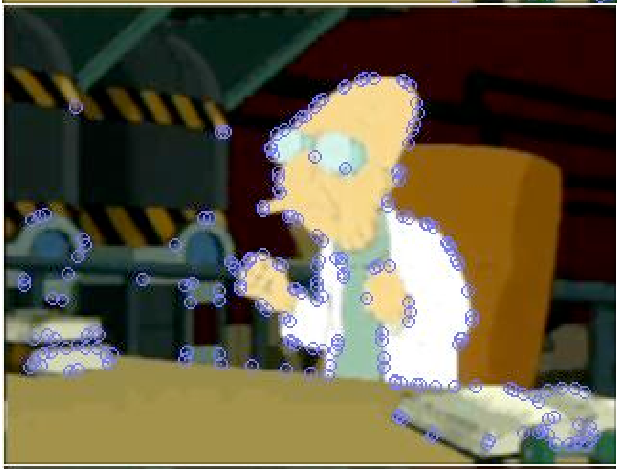
6.5.1 PROF. FARNSWORTH

Comic style, 320 x 240, 24-bit RGB color image



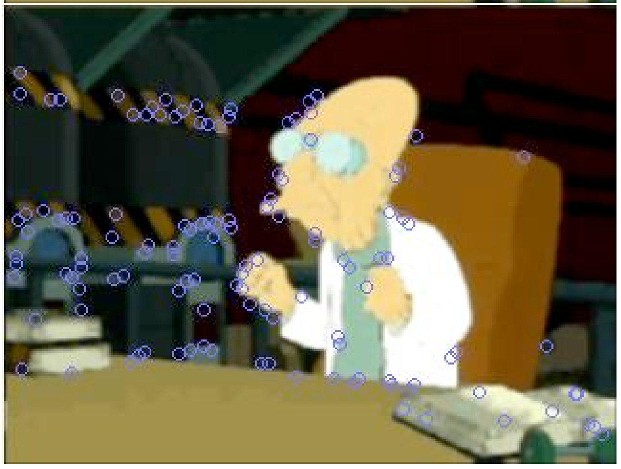
Chain Code

Time taken: 106.43 ms
Features detected: 130



Harris Corners

Time taken: 87.13 ms
Features detected: 198

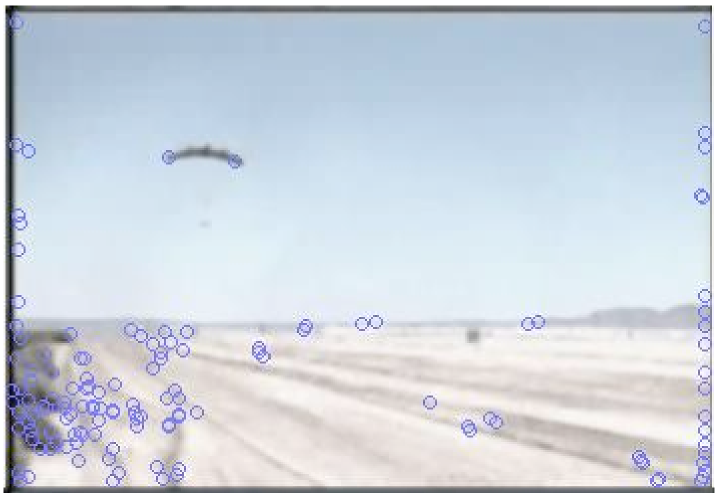


PCA SIFT

Time taken: 1818.93 ms
Features detected: 123

6.5.2 Parachute

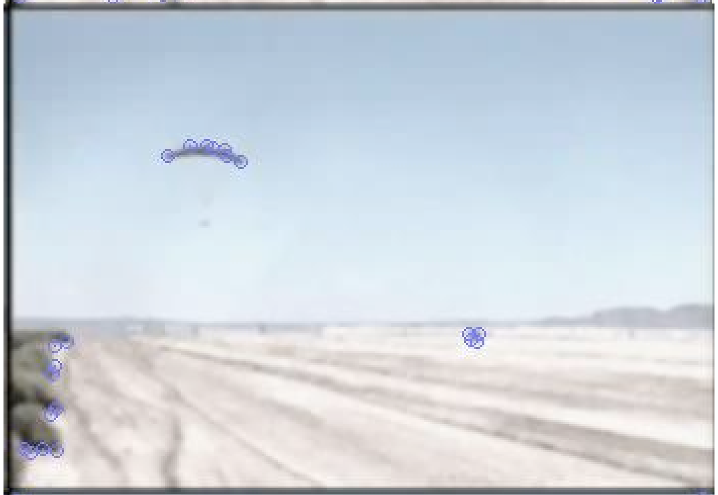
Real world, 352 x 240, 24-bit RGB color image



Chain Code

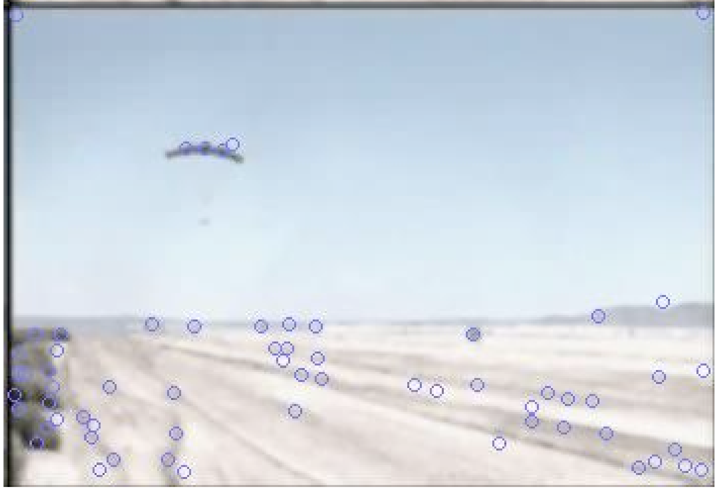
Time taken: 147.36 ms
Features detected: 132

Many features were detected at the edge of the image. This is a flaw of the segmentation process.



Harris Corners

Time taken: 24.49 ms
Features detected: 21



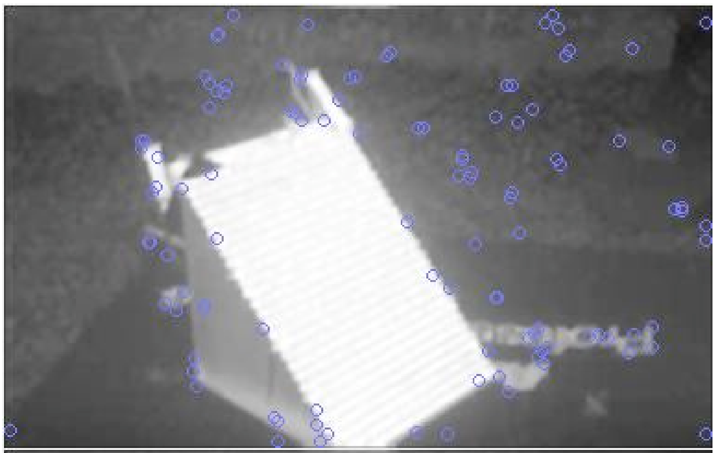
PCA SIFT

Time taken: 1996.75 ms
Features detected: 63

Many features have been detected on the plain. This is optimal for optical flow estimation.

6.5.3 Toy Scale

Lab picture, 384 x 240, 8-bit grayscale image



Chain Code

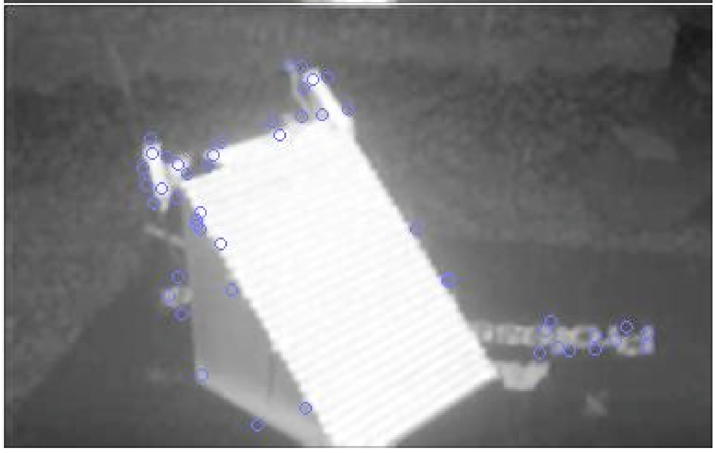
Time taken: 138.18 ms
Features detected: 110

Many features have been detected in the background. The segmentation process oversegments smooth gradients.

Harris Corners

Time taken: 29.80 ms
Features detected: 126

Features are almost only detected at the edge of the scale because of the strong contrast.



PCA SIFT

Time taken: 2173.47 ms
Features detected: 49

6.6 Conclusion

This paper compared three feature detectors for their performance. Harris Corners outperforms the others by far, but has its flaws when operating on images with less contrast.

PCA-SIFT scale-invariant feature detector produces features not only at the edges. This is a major advantage in some images. Especially on low contrast images feature extraction is more stable. The algorithm showed to be way too slow for (real-time) video evaluation.

The Chain Code feature detector produces unstable features because of the underlying image segmentation process. A better segmentation or edge detection could provide a massive performance gain as well as stable features.

There is no perfect feature detector. Further experiments should combine different approaches to provide better results. The drawback then is the higher performance requirement. In our future work we used mainly the Harris corners feature detector.

7 Feature descriptors

The features found by the feature detectors [chapter 6] need to be encoded in order to be useful for the tracking process [chapter 8].
It showed, that the selection of appropriate feature descriptors is crucial for success.
Again, many different types of feature descriptors were proposed in the last few years [9][10][2]. They can be classified in respect to their tolerance towards

- orientation
- scale
- illumination and color changes
- projective distortion
- performance (generation, matching)

7.1 SIFT

Recently David G. Lowe proposed an interesting approach on feature descriptors [7][8][9]. SIFT features need to be detected by a scale invariant feature detector. As seen in [chapter 6] the performance of such detectors is rather poor.
The following extract is a compact description of the SIFT descriptor taken from [10].

«The standard keypoint descriptor used by SIFT is created by sampling the magnitudes and orientations of the image gradient in the patch around the keypoint, and building smoothed orientation histograms to capture the important aspects of the patch. A 4x4 array of histograms, each with 8 orientation bins, captures the rough spatial structure of the patch. This 128-element vector is then normalized to unit length and thresholded to remove elements with small values.
The standard SIFT keypoint descriptor representation is noteworthy in several respects:

1) The representation is carefully designed to avoid problems due to boundary effects – smooth changes in location, orientation and scale do not cause radical changes in the feature vector.
2) It is fairly compact, expressing the patch of pixels using a 128 element vector.
3) While not explicitly invariant to affine transformations, the representation is surprisingly resilient to deformations such as those caused by perspective effects.

On the other hand, the construction of the standard SIFT feature vector is complicated and the choices behind its specific design (as given in [9]) are not clear.»

7.2 PCA SIFT

Yan Ke and Rahul Sukthankar proposed an enhanced version of the SIFT technology [chapter 7.1]. Their feature descriptor has been proved to be «theoretically simpler, more compact, faster and more accurate than the standard SIFT descriptor» [10].
Still the calculation of the descriptors is slow compared to other methods [chapter 7.3].

7.3 Intensity patch

This feature descriptor is very fast to be calculated and matched. On the other hand it is almost not tolerant to scale, orientation, illumination and affine transformations at all.

The feature descriptor is created by sampling the neighborhood of the feature. This patch forms the feature descriptor. We adapted the formula found in [2]

$$FD = \bigcup_{j=\quad N}^{N} \bigcup_{i=\quad N}^{N} I(x \quad j, y \quad i) \quad \bar{I} \tag{7.1}$$

to

$$FD' = \bigcup_{i=\quad N}^{N} \bigcup_{j=\quad N}^{N} I(x + j, y + i) \tag{7.2}$$

because it is simpler to calculate and performs comparable when illumination changes are small.
The patch forms a $(2N+1) \quad (2N+1)$ matrix.

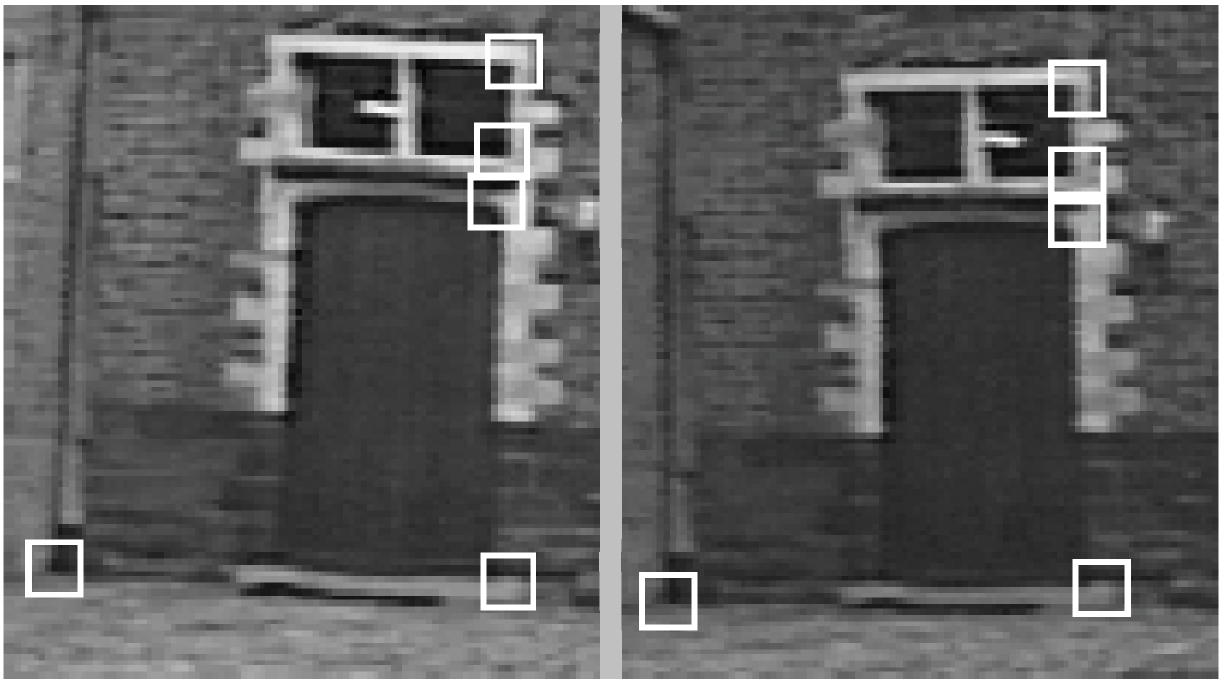


Figure 7-1: Correlated features from two different images. Image from [2].

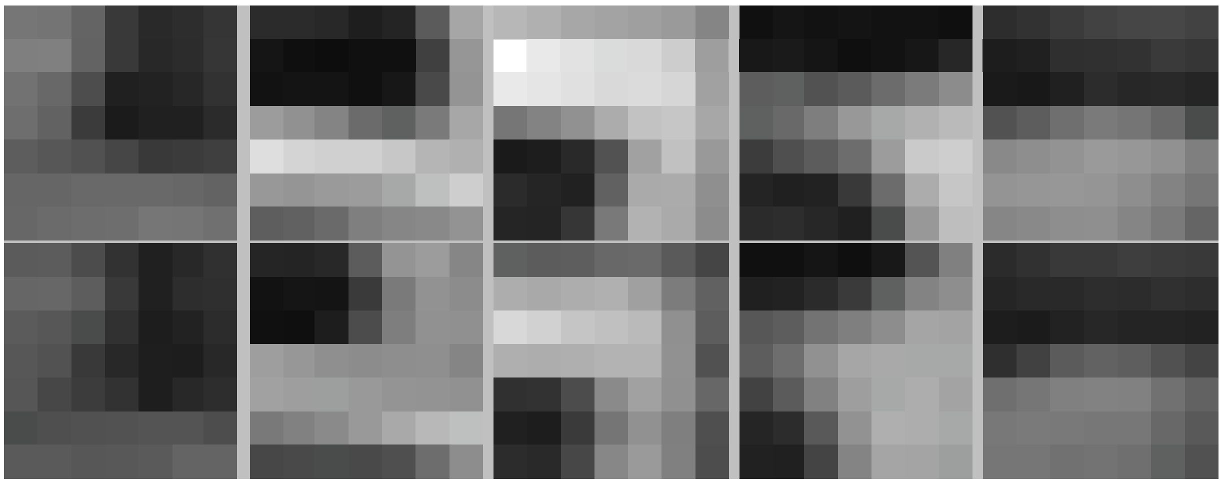


Figure 7-2: Extracted feature descriptors from Figure 7-1. N=3. Image from [2].

The matching of the feature descriptors is equal simple to its generation. Again we modified the formula found in [2]

$$C = \mathop{}_{i=\ N}^{N}\mathop{}_{j=\ N}^{N}\left(I\left(x\quad j,y\quad i\right)\quad \bar{I}\right)\left(I'\left(x'\quad j,y'\quad i\right)\quad \bar{I'}\right) \tag{7.3}$$

because we wanted a probability distribution within [0,1], where 0 means the features match not at all and 1 stands for a complete match. We used a least squares approach:

$$\begin{aligned} p_{FD} &= 1 \quad \frac{\mathop{}_{i=\ N}^{N}\mathop{}_{j=\ N}^{N}\left(I\left(x\quad j,y\quad i\right)\quad I'\left(x'\quad j,y'\quad i\right)\right)^2}{(2N+1)^2} \\ &= 1 \quad \frac{\mathop{}_{i=0}^{2N+1}\mathop{}_{j=0}^{2N+1}\left(FD\left(j,i\right)\quad FD'\left(j,i\right)\right)^2}{(2N+1)^2} \end{aligned} \tag{7.4}$$

Note that FD and FD' both refer to (7.2).

7.4 Conclusion

Because we operate on continuous movie sequences we can assume that the changes in orientation, scale and perspective projection remain small between the images. Illumination could change massively, i.e. when someone turns off the lights.
On the other hand a fast evaluation of the motion sequences is desirable. Therefore we decided to use the simpler *Intensity patch* feature descriptor.

8 Feature tracking

The second approach to optic flow estimation [chapter 5] is to track special features and derive the optic flow from them.
Basically this means we need to find the corresponding feature in frame $t+1$ for every feature in frame t. Let's call these matched features a feature tuple.

$$F_T = (F_t, F_{t+1}) \tag{8.1}$$

Usually it is not possible to find a feature tuple for every feature. Noise, occlusion, changes in illumination and perspective projection lead to feature mismatch.
A stable feature tracker should be able to find all existing feature tuples, correctly identify new features and ignore redundant features. Such a matching process is mostly NP hard.

8.1 Distance measurement

As we consider continuous frames only, features might not move further than a given threshold   . Therefore we need to match F_{t+1} only to features F_t within a $(2\ \ +1)\ \ (2\ \ +1)$ window. The closer two features are, the higher the probability they belong to the same tuple. We use a Gaussian distribution to calculate this probability

$$p\ \ = e^{\frac{x^2+\ y^2}{2}} \tag{8.2}$$

8.2 Correlation measurement

The whole correlation of two features is given by

$$p = p\ \ p_{FD} \tag{8.3}$$

where p is the distance correlation (8.2) and p_{FD} is the feature descriptor correlation [chapter 7].

8.3 Correlating the features

We implemented several approaches to this problem. The simplest use only the similarity measurement (8.3), while more elaborate trackers use the feature distribution and neighbor likelihood as well.
Because the images from motion sequences change only a little between two frames, the latter showed not to perform better than the simple ones. Often they performed even worse, because the feature detection process misses features and redetects them in later frames. The lack of some features at one position in the image leads to completely different feature distribution. We will only discuss the simpler approach in this paper.

8.3.1 Simple cross correlation

Simple cross correlation tries to find for every feature F_{t+1} a feature F_t such that $c(F_t, F_{t+1})$ is maximal.
This might result in a sub-optimal overall match as seen in Figure 8-1.
Simple cross correlation matches according to the red lines because

$$c(A,B') = \max(c(A,A'),c(A,B'),c(B,B'),c(B,A'))$$

which finally leads to the matched tuples

$$\{(A,B'),(B,A')\}$$

The overall correlation would be higher if the matched tuples would be according to the green lines

$$\{(A,A'),(B,B')\}$$

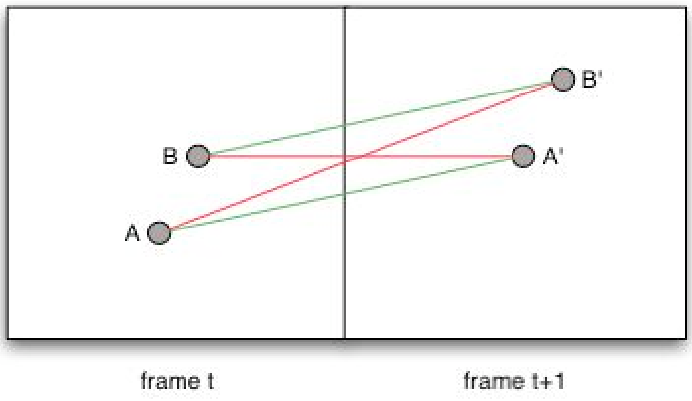


Figure 8-1: Feature mismatch

The *enhanced cross correlation* and *optimizing cross correlation* feature trackers try to avoid this phenomenon [chapters 8.3.2 and 8.3.3, resp.].

The performance of this simple tracker is rather good as seen in Figure 8-6 and Figure 8-7.

The runtime of the algorithm is $O(NM)$ because it tries to find a matching partner in F_t for every feature in F_{t+1}. N and M are the amount of features in F_t and F_{t+1}, resp.
This can be enhanced by matching to the features within a maximum translation window only [chapter 8.1].

8.3.2 Enhanced cross correlation

The problem in Figure 8-1 is that both A and B favor A'. Instead of assigning A' to B like the *simple cross correlation* does, this feature tracker tries to find the second best matches for A and B, resp.
Then it assigns the tuples such that the sum $c(F_t, F_{t+1}) + c(F_t', F_{t+1})$ is maximal. Because we consider two tuples at a time only, the probability is high that we will end up in a local maximum.
As seen in Figure 8-3 the result is sometimes even worse than for the simple cross correlation (Figure 8-2).

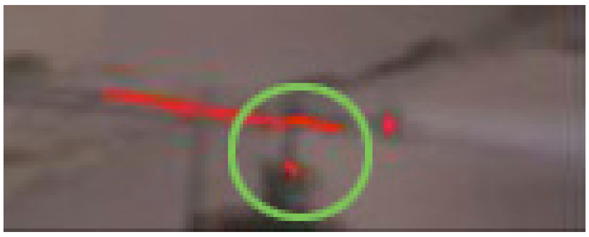


Figure 8-2: Simple cross correlation

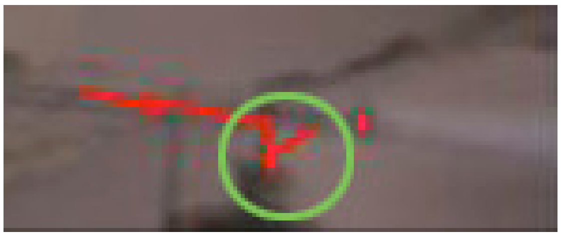


Figure 8-3: Enhanced cross correlation

8.3.3 Optimizing cross correlation

This tracker tries to maximize the overall correlation c_{tot} to find the optimal matches, optimal in terms of the matching function. As the name states, this is an optimization problem and as described in [1] such problems are best solved with local search strategies.
We use the *Hill-Climbing* algorithm, sometimes also called greedy local search because it grabs a good neighbor state without thinking ahead about where to go next. This algorithm has three well-known flaws (local maxima, ridges and plateaus), which may result in sub-optimal solutions.

Optimization problems are generally NP hard, such is this one. To speed up things, *optimizing cross correlation* first applies the *simple cross correlation* [chapter 8.3.1] to produce the initial correlation seed.
This reduces the runtime of the optimizing process to $O(\ \ (n!))$, where   is the re-matching factor. The best-case runtime is therefore $O(n!)$ while the worst case is $O(n^2(n!))$, which is unlikely to ever happen as the original seed was chosen heuristically.

Based on the correlation seed, it now searches for tuples $\{F_t, F_{t+1}\}$ and $\{F'_t, F'_{t+1}\}$ where $c(F_t, F'_{t+1}) + c(F'_t, F_{t+1}) > c(F_t, F_{t+1}) + c(F'_t, F'_{t+1})$ and swaps them. This algorithm converges to a global maximum with respect to its seed correlation.

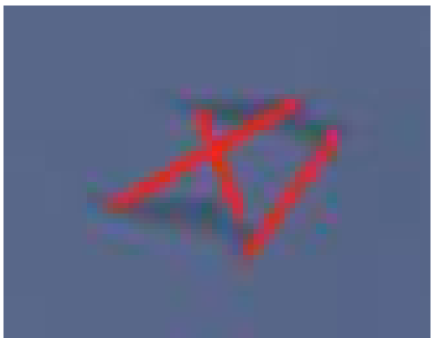


Figure 8-4: simple cross correlation

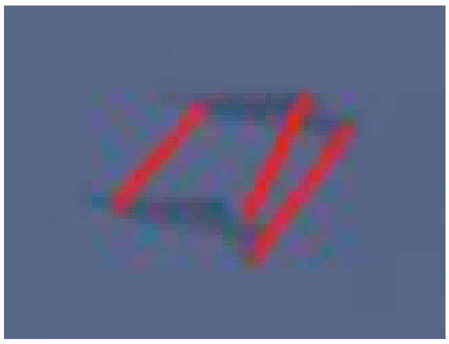


Figure 8-5: optimized cross correlation

Figure 8-5 shows how the *optimized cross correlation* solves the matching problem discussed in [chapter 8.3.1].

8.3.4 Benchmark

The following benchmark compares the performance of the *simple cross correlation* tracker [chapter 8.3.1] and the *optimizing cross correlation* tracker [chapter 8.3.3]. Please refer to [chapter 8.5] for a comparison of their tracking capabilities.

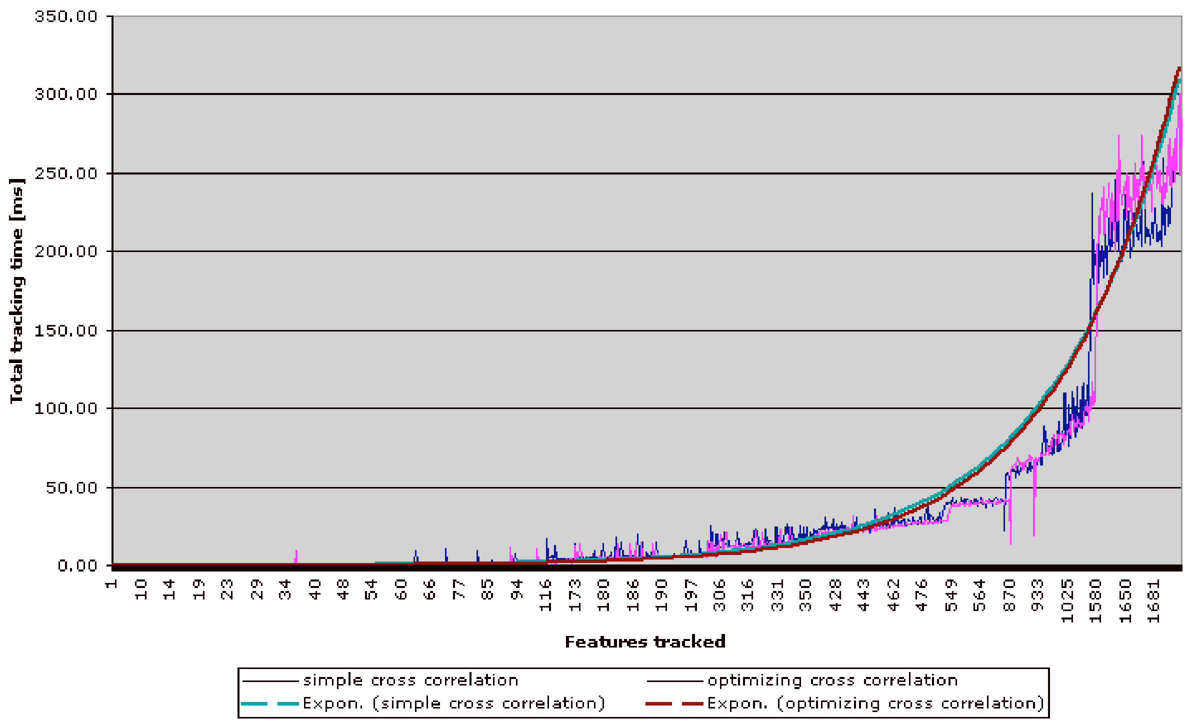


Figure 8-6: Total tracking time

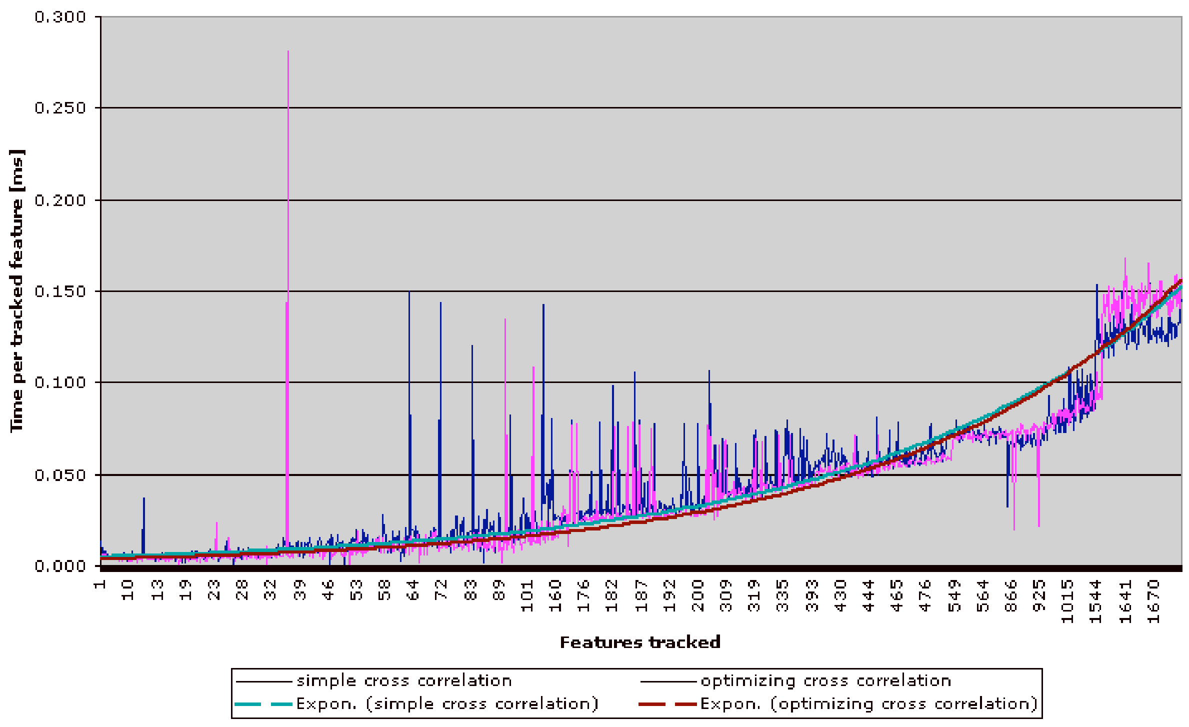


Figure 8-7: Tracking time per feature

It might be surprising, that both trackers perform equally well. This is because the algorithms use a different amount of memory. While the *simple cross correlation* uses no additional memory, the *optimizing cross correlation* uses an internal lookup matrix of the size $|F_t| \quad |F_{t+1}|$ to speed up the optimization process.

The spikes and the increasing trend in Figure 8-7 appear because of the feature re-matching. This happens when the correlations of the features are close to each other. The more distinct the features are, the less re-matching occurs.

8.4 ENHANCED TRACKING

In addition to solely correlate the features based on their similarity (8.3) we can enhance the result by post processing the tuples. A very famous approach is the Hough transform [4][16]. Despite its popularity we did not use the Hough transform. Instead we applied two different approaches, both much simpler and way faster than the Hough transform.

8.4.1 COLLAPSING FEATURES

The first one addresses the problem of inconsistent feature detection discussed in [chapter 6]. Instead of tracking all features we collapse features within a certain distance radius r_c to so-called superfeatures. A superfeature averages all the properties of its sub-features (Position, orientation, feature descriptor).
This has two positive effects: 1) the performance improves because we have fewer features to track and 2) these superfeatures tend to be more stable.
I.e. the detector detected 5 features in the first frame within this distance radius, but only 3 in the second frame. If we tracked all features, we would have a mismatch of 2 features. If we collapse these features to 1 each, we have one tracked tuple only.
Of course, r_c should not be too big, because otherwise important features would be lost and the superfeature would be misplaced.
To avoid collapsing distinct features we could restrict the algorithm to only collapse features with high feature descriptor correlation (p_{FD}).
One drawback of this method is the *flickering* of the tracked superfeatures, an effect which takes place because of the position averaging. This effect has its impact on the trajector pruning [chapter 8.4.2] and the graph stability [chapter 9.2.1].

For the following examples we set r_c to 10. Using Harris corners, this reduces the amount of superfeatures to about 50% of the originally detected features.

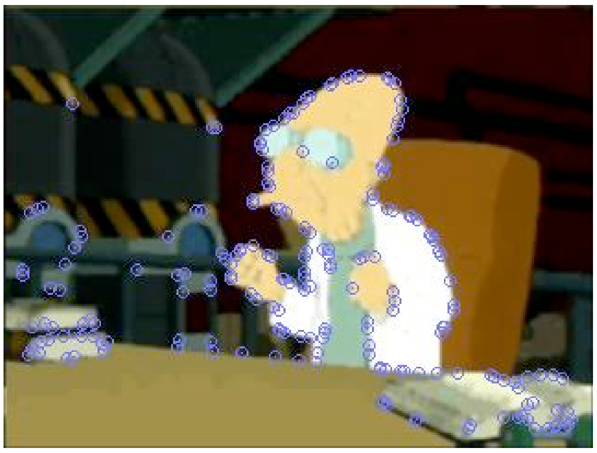


Figure 8-8: Detected features (198)

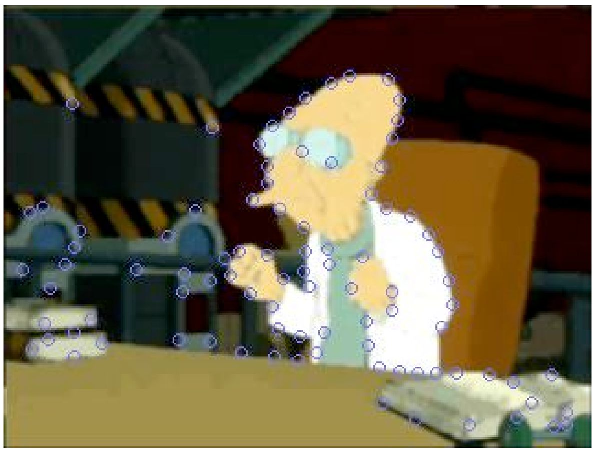


Figure 8-9: Remaining superfeatures (91)

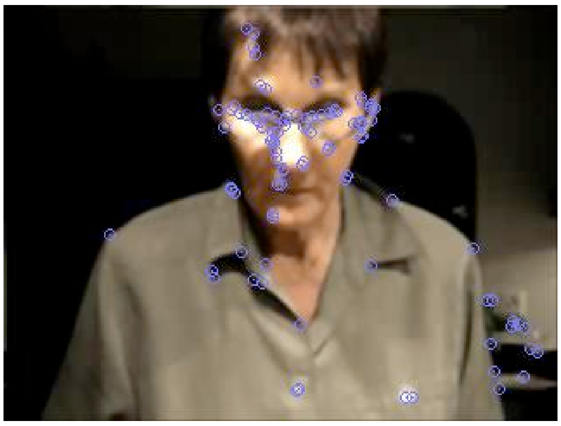


Figure 8-10: Detected features (90)

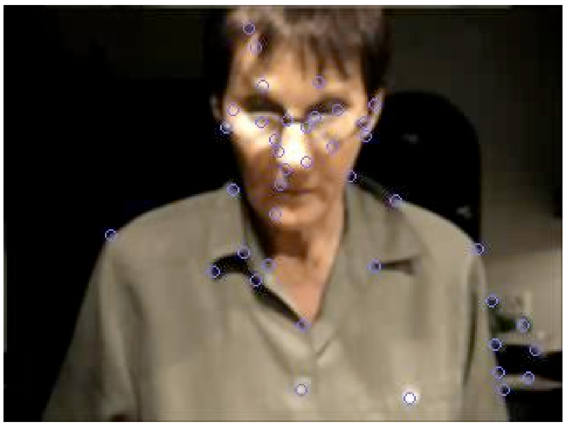


Figure 8-11: Remaining superfeatures (41)

8.4.2 Trajector pruning

The found tuples describe the translation of the features over time. The movement vectors

$$\vec{T} = \begin{matrix} F_t(x) & F_{t+1}(x) \\ F_t(y) & F_{t+1}(y) \end{matrix} \tag{8.4}$$

are called trajectors. The idea behind trajector pruning comes from the swarm theory [1]. The alignment axiom of the boids theory states that adjacent boids steer toward the average orientation of their neighbors.
We can turn around this axiom such that a boid only belongs to a swarm when his direction is not completely different from its neighbors.

It is a high probability that adjacent features belong to the same object. As the object moves, they should move too. The translation should conform with the object and therefore with the surrounding features.
Because features also have neighbors on different objects, they need to conform with a fraction of the neighborhood only.

To compare two trajectors we need once more a similarity measurement. Orientation and size of the trajectors should be compared. The output domain of the resulting function should lie within [0,1].

Size
The correlation of the two trajectors u, v in relation to their length is defined as Gaussian distribution. The output domain is defined within [0,1], where 1 states the vectors are of the same size and 0 implies they are not correlated at all.

$$p \quad = e^{\frac{(|u| \quad |v|)}{2})(} \tag{8.5}$$

Orientation
Again, the desired output should be defined within [0,1], where 0 states no correlation at all.

The output of the cosine would be maximal at an angle of 0° but it is distributed within [-1,1]. To enforce the desired output we can use the following cosine instead

$$p \quad = \cos^2 \quad \frac{}{2} \tag{8.6}$$

Using the fact that

$$\cos^2 \quad \frac{}{2} \quad = \frac{1+\cos(\quad)}{2} \tag{8.7}$$

the formula resolves to

$$p \quad = \frac{1+\cos(\quad)}{2} \tag{8.8}$$

Orientation of two vectors can be compared with the scalar product

$$\cos(\quad) = \frac{uv}{|u||v|} \tag{8.9}$$

We can now apply the scalar product (8.9) to (8.8) and form

$$\begin{aligned} p \quad &= \frac{1+\dfrac{uv}{|u||v|}}{2} \\ &= \frac{1}{2} + \frac{uv}{2|u||v|} \end{aligned} \tag{8.10}$$

The complete correlation function p is defined as the product of (8.5) and (8.10).

$$p = p \; p \qquad (8.11)$$

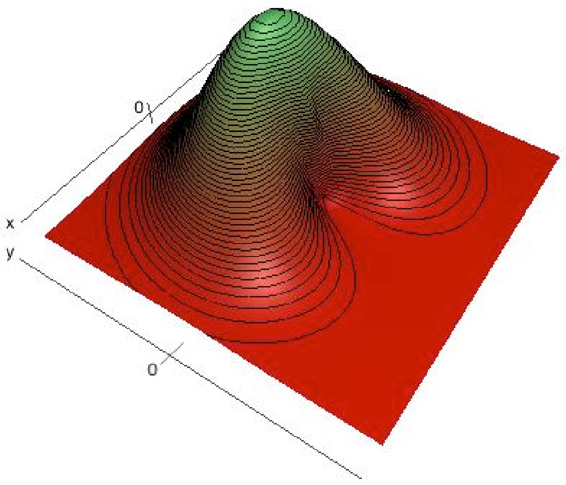


Figure 8-12: Correlation function

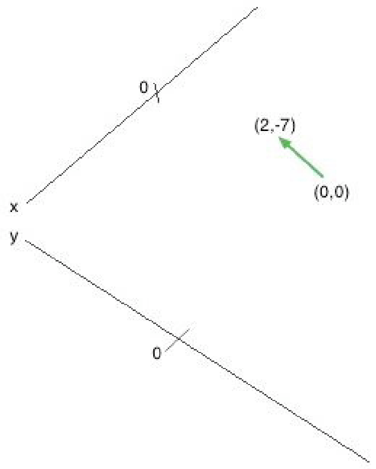


Figure 8-13: Seed trajector $\vec{T_o}$

Figure 8-12 shows the correlation field of trajector $\vec{T_0} = \begin{matrix} 2 \\ 7 \end{matrix}$ and trajectors $\vec{T}_i = \begin{matrix} x \\ y \end{matrix}$. The trajector is plotted in Figure 8-13 for convenience.

As seen in Figure 8-12 the correlation differs most close to (0,0). Small trajectors are not as stable as large ones due to the discrete nature of the image space. The *flickering* effect discussed in [chapter 8.4.1] aggravates this phenomenon. This means that small trajectors do not produce a stable trajector field and pruning is not applicable to them. We therefore restrict trajector pruning to trajectors of size bigger than a given threshold .

Experiments showed that trajector pruning effectively eliminates undesired feature tuples. The following images show tracked features without and with trajector pruning, resp. The correlation threshold was set to 0.5, the size threshold to 20 and the minimum amount of correlated neighbors was 40% of all surrounding features.

Orbit around the earth
Feature translation is little while the feature detection is not consistent. Mismatched features in Figure 8-14 were correctly removed with trajector pruning in Figure 8-15.

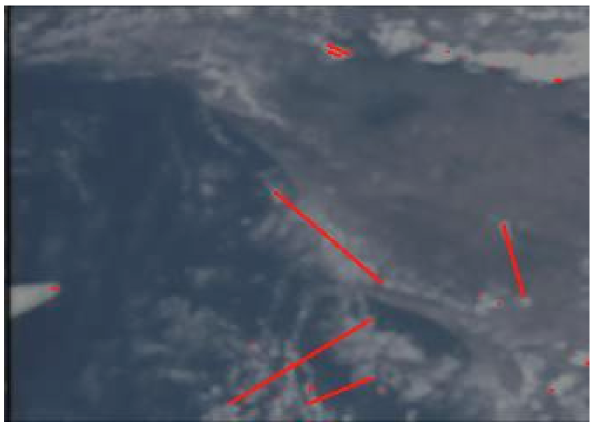


Figure 8-14: Feature inconsistency leads to mismatches.

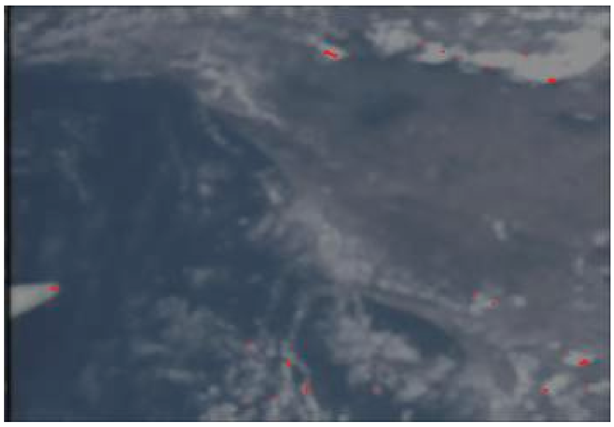


Figure 8-15: The same picture after trajector pruning.

Futurama crew
Figure 8-17 shows what happens when   is too big. Although all mismatched trajectors of size >   have been correctly removed, the small mismatches remain. These trajectors are a direct consequence of the flickering discussed in [chapter 8.4.1].

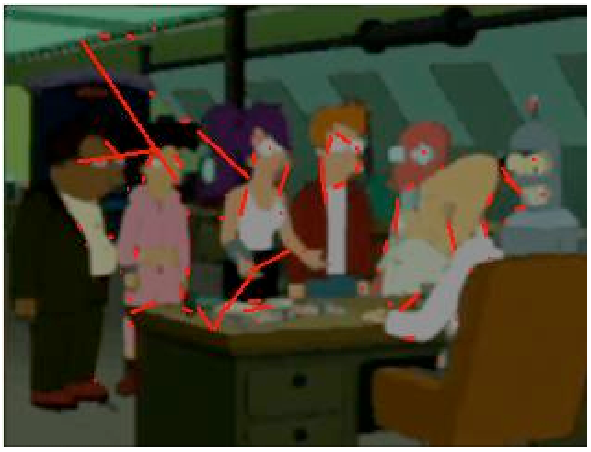


Figure 8-16: Single features were completely mismatched.

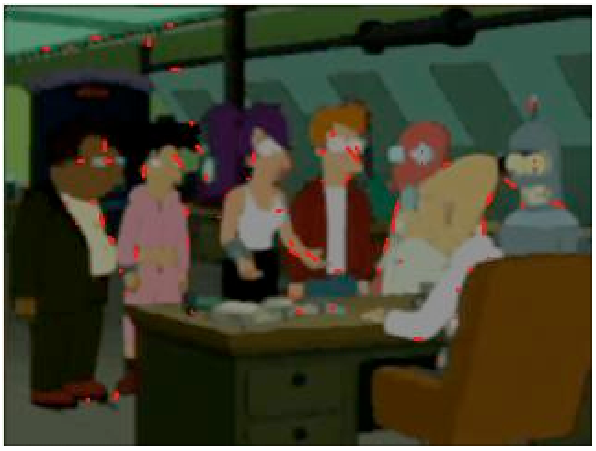


Figure 8-17: Features with size <   flicker even after trajector pruning.

8.4.3 Feature tracking with history

Up to now we considered features from two adjacent frames only. A different approach is to take into account all features detected so far, or at least over a longer period of time [36]. Instead of matching F_{t+1} to F_t only, we would then match it to $\bigcup_{i=0}^{t} F_i$.

This has one major advantage: features, which disappear temporarily, can be re-matched and this increases the stability of the matching process.
The downside is that such hidden features do not necessarily remain static. Some will definitely move while not visible, which leads to a positional uncertainty of the feature. This uncertainty is proportional to the time a feature could not be observed. And, of course, we need to track much more features.

We try to get the best of both worlds and to avoid the negative effects by applying a three-stage tracking process.

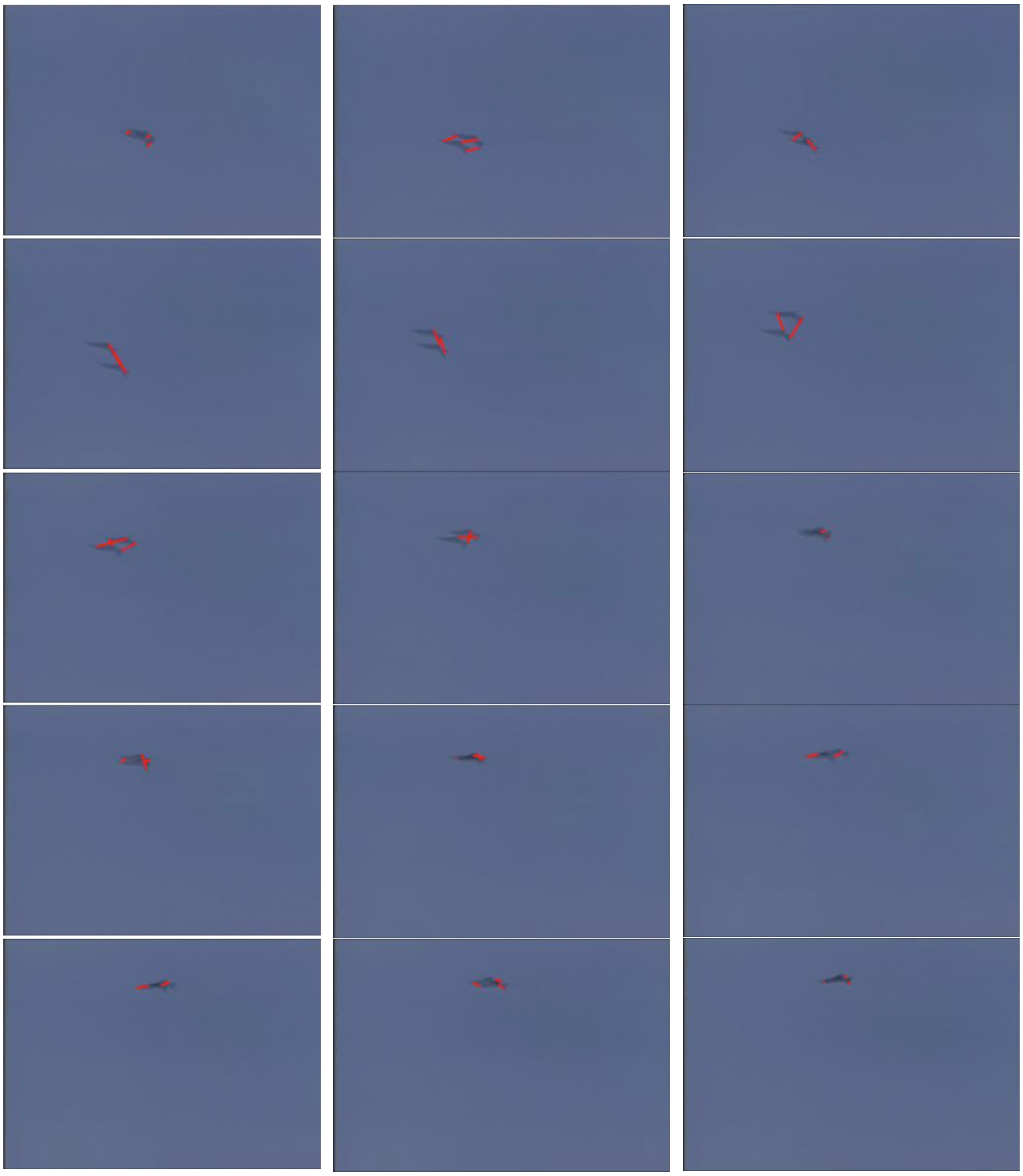
1) match F_{t+1} to F_t
2) estimate the position of the all features prior to F_t
3) match all unmatched features to $\bigcup_{i=0}^{t} F_i$

Steps 1 and 3 are performed with the same feature tracker. Step 2 is explained in [chapter 9.3].

8.5 *Examples*

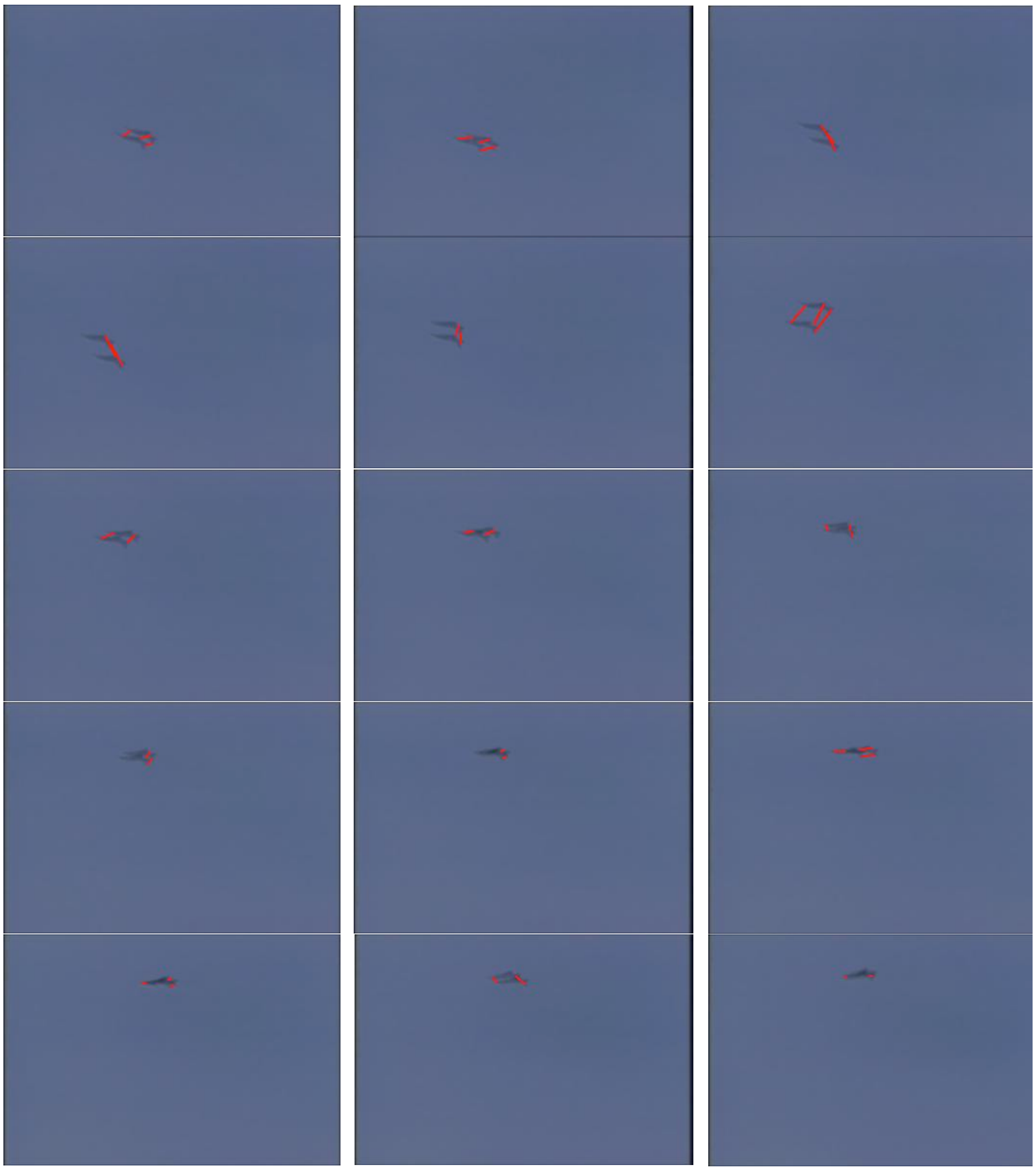
8.5.1 Flying aircraft (*simple cross correlation*)

The following 15 frames show a tracked aircraft. Only 1 to 3 features were detected per frame. Still the tracking performed well. Note that some frames have feature mismatches as discussed in [chapter 8.3.1].



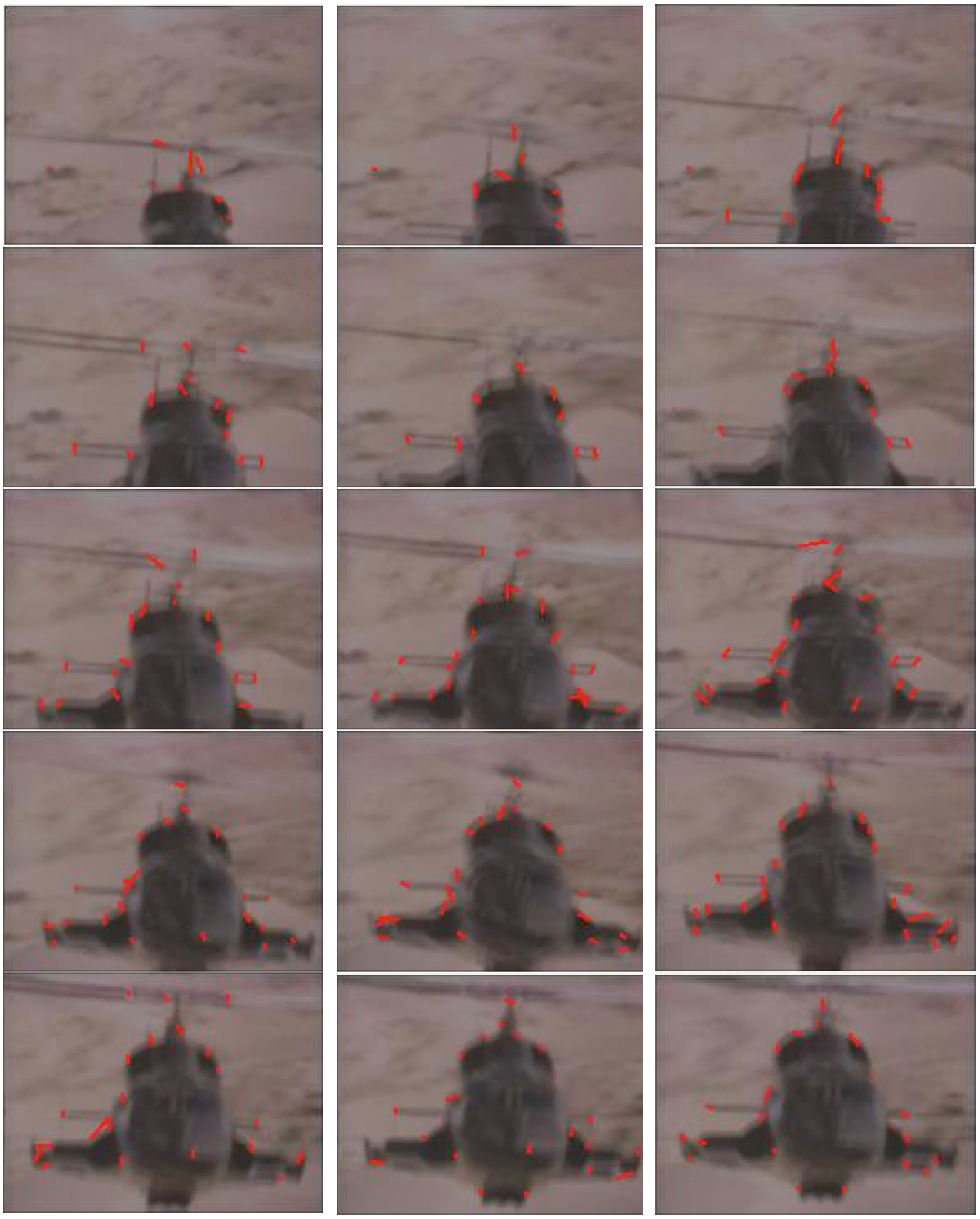
8.5.2 Flying aircraft *(optimizing cross correlation)*

The following 15 frames show the same sequence as [chapter 8.5.1], but this time tracked with the *optimizing cross correlation* instead. Note that all feature mismatches were correctly suppressed.



8.5.3 Airwolf (*simple cross correlation*)

Tracking more features is generally more stable. The following sequence shows an appearing helicopter and an increasing number of features.



9 Feature graph

One possibility to represent structure are graphs [31][23].

Feature graphs are a flexible way to do so because they store the features (nodes) as well as the relations between them (edges).
The following graph was designed to store features, which vary over time in their position and appearance so the global structure can be retained.
Furthermore, the graph can be used to segment objects because the behaviors of individual features can be related over time.

9.1 Nodes

Nodes are used to store the features. Every node represents one feature. A node has therefore a position in feature space as well as an associated feature descriptor [chapter 7].

As stated in [chapter 8.4.3], features are not redetected in every frame. Therefore a node may also represent 'invisible' features or even features outside of the image. As the original features are expected to be in motion even when not visible, the confidence about the position of the feature should decrease when the feature could not be redetected. This is achieved by decreasing the confidence of the feature as a whole [chapter 9.1.2].

9.1.1 Connecting the nodes

An important property of a graph is the connectivity of its nodes. We could connect every node to every other in the graph, but this would lead to a waste amount of edges. Another approach is to connect only nodes within a predefined window. This has the drawback, that if we have a dense distribution of the nodes, we will again have many edges, while a sparse distribution will result in no connectivity at all.
The chosen connection model is similar to the *k-nearest* clustering algorithm, because we connect the k nearest neighbors of a node. This results in a constant connectivity independent of the feature distribution.

9.1.2 Node pruning

In every time-step one of the following may happen

1) the feature could be redetected -> the confidence should rise
2) the feature was not re-detected -> the confidence should sink

This is achieved with two factors, the feature *aging* † and *anti-aging* * factors.

Nodes with a confidence lower than a given threshold are removed from the graph.

9.2 EDGES

Edges connect two nodes. This graph does use undirected edges only

$$F_i \quad F_j \qquad F_j \quad F_i \tag{9.1}$$

and self-loops are prohibited

$$\begin{aligned} &F_i \quad F_j \quad i \quad j \\ &i = j \quad F_i \diagup F_j \end{aligned} \tag{9.2}$$

An edge represents the correlation strength of the connected features. A strong correlation indicates both features belong to the same object. This correlation is a combination of the *spatial correlation* and the *temporal correlation* and therefore called *spatio-temporal correlation* p_{st}.

9.2.1 SPATIAL CORRELATION

If the distance between two features remains constant over time, this is an indication that the features belong to the same object. This is especially true, when the features are in motion. We could use the trajector correlation [chapter 8.4.2] as the spatial correlation, but this has three major drawbacks.

1) If two features separate slowly from each other, their trajector correlation will be high because we inspect two adjacent frames only. If the features separate over a longer period of time, their spatial correlation should decrease.
2) On the other hand, if two features move away from each other in one frame and approximate each other in the next, the trajector correlation is in both cases weak. But in fact the features have now the same relative positions as they had at the beginning. This behavior might be caused by *flickering* [chapter 8.4.1].
3) Hidden features do not have trajectors.

Because of these reasons we introduced the e*dge elasticity* .
The trajector correlation also has some positive effects. If two features have similar trajectors, their correlation should definitely rise. Note, that it should not decrease when this is not the case (2).

9.2.1.1 Edge elasticity

Let's think of an edge as a rubber band. As long as the rubber band is not stretched more than a given amount, it will return to its prior state without consequences. If it is overstretched it will eventually break [31].
Because of various factors described in prior chapters, features cannot be expected to have a consistent distance to each other. The edge elasticity {0,1} is tolerant towards these inconsistencies because it allows the features to move freely up to a given threshold

.

If this threshold is exceeded the edge elasticity is set to 0 (the rubber band breaks).

9.2.2 Temporal correlation

If two features are often detected at the same time, or consequently over a long period of time, they are somehow related. This is achieved by applying an anti-aging factor *
[chapter 9.1.2]
In contrast to the nodes, we do not age the edges. The reason is, that when a feature disappears e.g. because of rotation in space, it is not less connected to the features it was connected before. When a feature is not visible, we cannot conclude that it is less correlated to the visible ones.
Edge aging is done indirectly by the feature pruning process.

9.2.3 Initial correlation

When features are detected for the first time, they need to be correlated. One way is to apply a static initial correlation. A better approach would be to use a dynamic correlation. From the first frame on, we have some hints on how correlated the features are

1) trajectors
2) feature descriptors
3) position

Currently we use all these indicators to calculate the initial correlation. The first two are described in [chapter 8.4.2] and [chapter 7], resp. The positional correlation is explained in [chapter 9.2.3.1].

These three parameters form a linear combination to describe the initial correlation

$$p_I = d + \quad \frac{ap_T + bp_{FD} + cp}{a+b+c} \quad d \tag{9.3}$$

Equation (9.3) can be simplified to

$$p_I = d + \quad \left(ap_T + bp_{FD} + cp \qquad d\right) \tag{9.4}$$

when we normalize the parameters to

$$a + b + c = 1 \tag{9.5}$$

is a weighted learning factor [0,1], which determines how close p_I is to the static correlation d. When set to 1, the static correlation has no impact and when set to 0, the dynamic correlation disappears.

The values of the parameters $a, b, c, d,$ depend on the application and the content of the images. Currently we use the empirically determined values given in Table 1.

Parameter	Value
a	0.6
b	0.2
c	0.2
d	0.3
	0.7

Table 1: correlation parameters

These parameters prefer the trajector correlation.

9.2.3.1 POSITIONAL CORRELATION

The statement, that features are higher correlated if they are closer is not always correct, but holds in many cases. This is the reason why we introduced the positional correlation p . But what distance is 'close'? In some frames two features are far apart with a distance of a few pixels, in others they are considered to be close with a distance of 100 pixels or more. To address this relativity, we do not correlate the features based on a fixed distance but in relation to the remaining surrounding features.
The correlation is defined as

$$p \quad = 1 \quad \frac{\qquad_{\min}}{\qquad_{\max}} \tag{9.6}$$

where is the distance between the features, $_{\min}$ is the distance to the closest feature, $_{\max}$ the distance to the remotest feature and is an adjusting factor to assure a slower decrease of the correlation.

9.2.4 ADAPTING THE CORRELATION

At every frame the correlation of an edge might change. As stated before, only edges between two tracked features might be adapted, because we don't know about the other features.

Edges are adapted by calculating the initial correlation for the matched tuples and apply this in a weighted learning algorithm to the old edge correlation. The new correlation is then

$$p'_{t+1} = (p_t + \quad (p_I \quad p_t)) \tag{9.7}$$

In addition the correlation is multiplied by the anti-aging factor to reward temporal correlated features.

$$p_{t+1} = \min\left(\;{}^{*}p'_{t+1}, 1\right) \tag{9.8}$$

Figure 9-1 shows how the adaptation of the correlation works. Red lines indicate weak correlation, while green stands for a high correlation. At time t=0 all edges have about the same correlation. Now the red ball moves from left to right, while the square remains static. This leads to a higher correlation within the objects and a decreasing correlation between them. In frame 7 the correlation has segmented the objects and the edges with a low correlation can be removed.

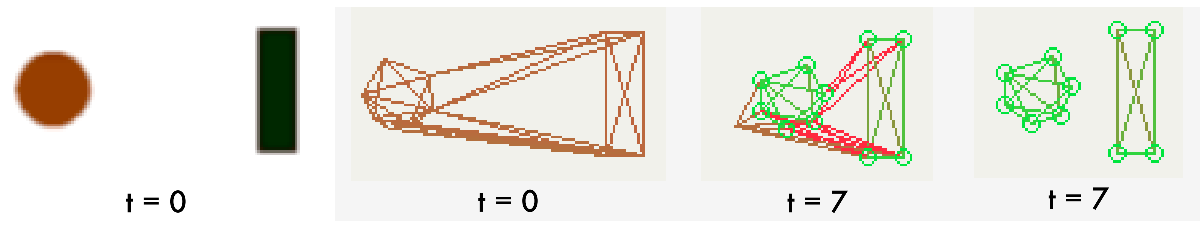


Figure 9-1: Adapting the correlation over time

9.3 GRAPH TRANSFORMATION

As mentioned in [chapter 8.4.3] the 'hidden' features do not remain static. They move in 'some' way. This motion can be estimated in two ways

1) build a model from the period the feature could be observed
2) transform the feature according to its surrounding features

9.3.1 LINEAR DYNAMIC MODELS

Linear dynamic models estimate the position of a point according to a motion model [4]. Common models are constant velocity or constant acceleration. The model is chosen and the parameters are calculated from the observed motion. This estimation holds as long as the feature does not change the model. For such moving objects, this approach provides reliable tracking capabilities even when the object is completely occluded.
Linear dynamic models are usually not built from single features because they behave to inconsistent. Instead motion fields are observed over some time period to estimate the motion model [36].
We do not use linear dynamic models in our work so far. Instead we applied the swarm model discussed in [chapter 9.3.2].

9.3.2 SWARM MODEL

The swarm model assumes that the points move within a swarm. The motion of the hidden features can therefore be derived from the motion of the visible features in the swarm. The problem in first place is to determine the membership of a feature.
This is done by evaluating the edge correlation described in [chapter 9.2]. The feature is then transformed according to a trajector, which is calculated by averaging the weighted trajectors of the neighboring features.

$$\vec{T} = \frac{\sum_{i=0}^{N} p_i \vec{T_i}}{\sum_{i=0}^{N} p_i} \qquad (9.9)$$

How the transformation works can be seen in Figure 9-2. While the plane moves out of the screen its shape is retained by the graph even beyond the border of the frame.

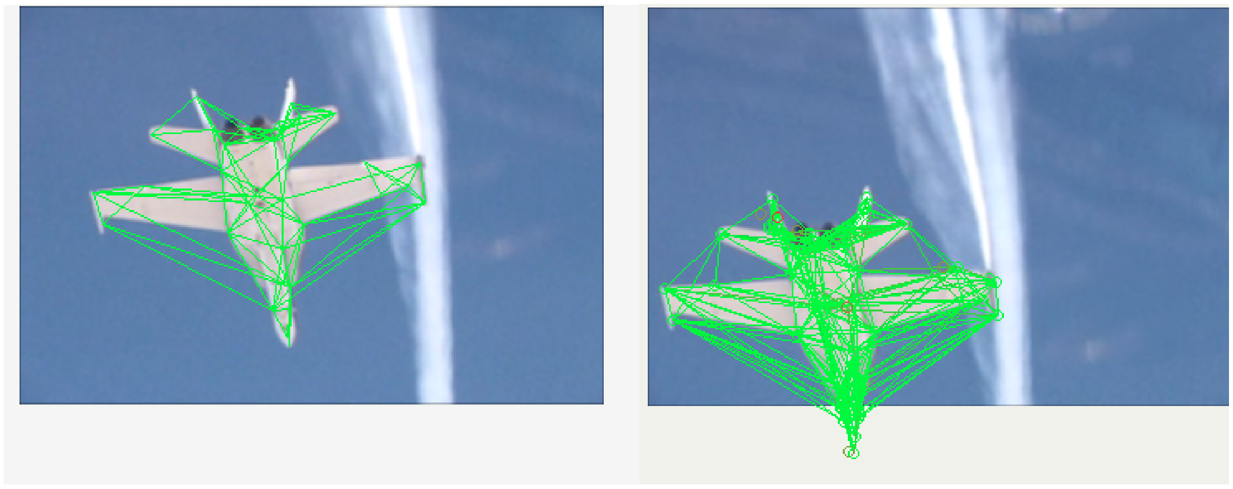


Figure 9-2: F/A-18 moving out of the screen. The feature graph retains the structure even beyond the border.

9.4 EXAMPLES

The plots of the graph contain lines (edges) and/or circles (nodes). Red lines state that the features are less correlated, while the green ones indicate a high correlation.
The color of the nodes indicates their confidence. Again, red color stands for low and green for high confidence.

9.4.1.1 SPACE SHUTTLE

The space shuttle is segmented against an easy background.

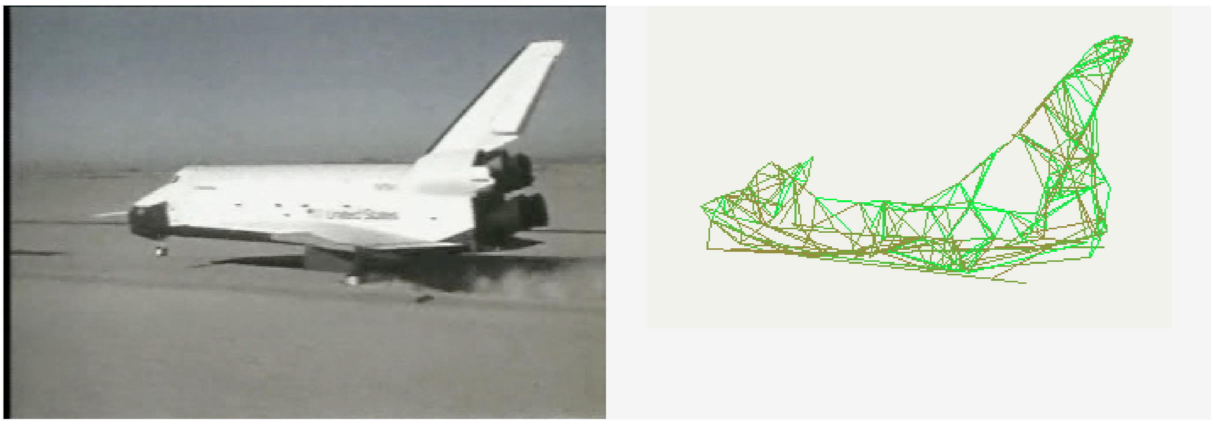


Figure 9-3: Space shuttle

9.4.1.2 FLYING AIRCRAFT

Note the edges at the tip of the plane show less correlation because the tip of the plane has a different color.

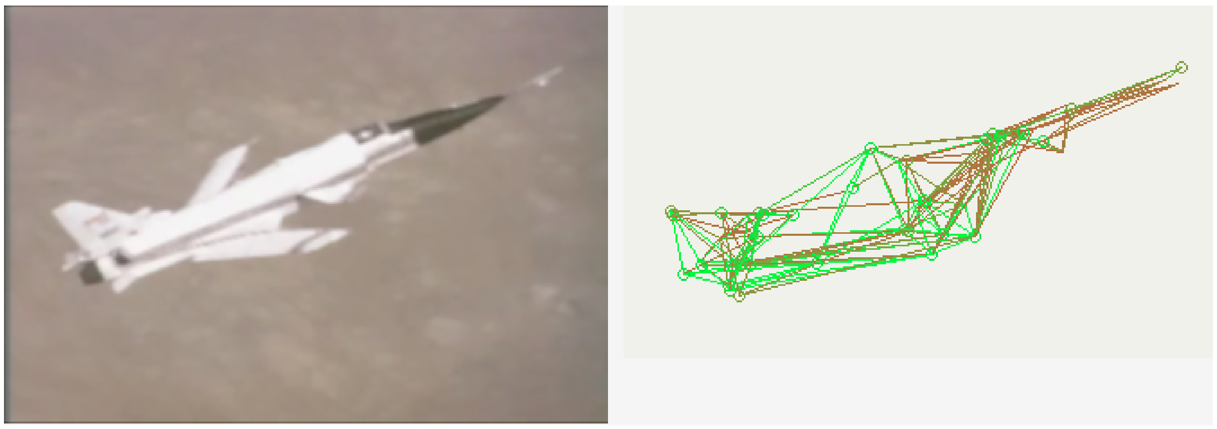


Figure 9-4: Flying aircraft

9.4.1.3 HUMAN

The clip shows a human moving in front of a fixed camera. The first graph plot shows the features retained in the graph and their confidence, while the second plot shows the feature correlations.

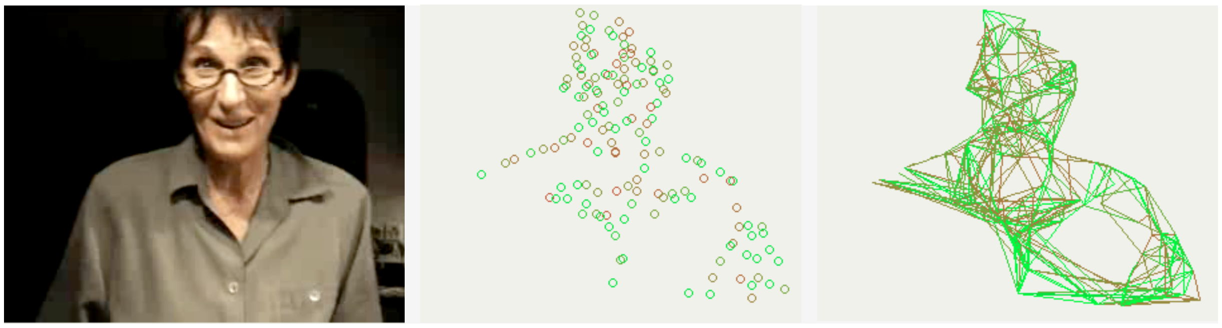


Figure 9-5: Human (Original, Features, Edges)

9.4.1.4 Ping-Pong ball

The following clip shows a Ping-Pong ball rotating around its z-axis. The first graph shows the lower correlation of the edges at the border of the ball, because the features either just appeared (left side) or disappeared (right side).
The second plot shows the graph at the same state but only the edges with maximum correlation are displayed.

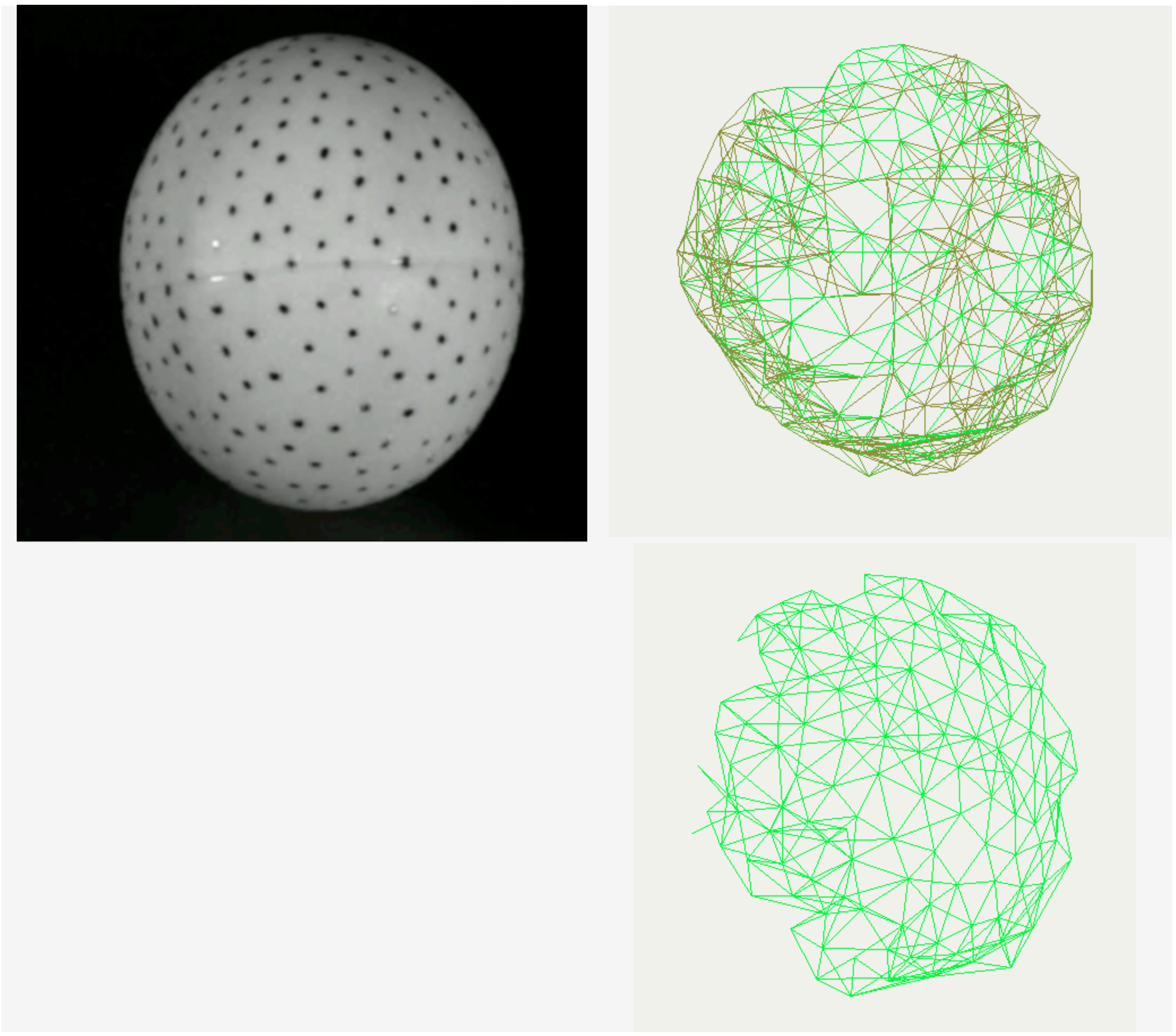
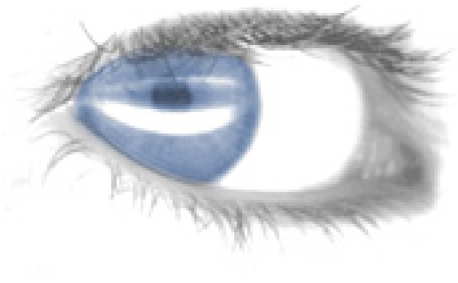


Figure 9-6: Ping-Pong ball

PART IV



Conclusion

10 A glimpse back...

Unsupervised object tracking

In this work we examined two different approaches to unsupervised object tracking. Both do not use any prior knowledge about the objects at all.

Gradient-based approach

Facette pyramid

The gradient-based approach described in [chapter 5] tries to estimate the optic flow by finding the corresponding pixels in two adjacent frames. It is considered a gradient-based approach, because the correspondence is found by comparing the color intensities of the pixels. The method we introduced uses so called Facette pyramids. These are multi-scale pyramids to track motions of different intensities. The approach is simple, stable and fast enough for real-time estimation. Because the amount of data to be evaluated does not depend on the content of the frames, the runtime of the algorithm is constant.

Feature-based approach

Feature detector

The feature-based approach evaluates the frames to find so-called features. These are regions in the image, which are somehow special. How *special* a region is, is determined by the feature detectors.
We implemented and compared three different feature detectors [chapter 6] and found the Harris corners feature detector to most suitable for our project.

Feature descriptor

Feature tracker

The detected regions are encoded by feature descriptors [chapter 7] in a way they can be compared later on.
The task to find corresponding features in two adjacent frames is called tracking. We introduced three different feature trackers [chapter 8], of which the *optimizing cross correlation* proved to be the best. We enhanced the trackers with some fast and efficient post-processing algorithms to receive a better result.

Feature graph

Finally, the detected structure is stored in a *feature graph* [chapter 9], which is tolerant to temporal inconsistencies in the feature detection. Furthermore it is able to transform unobserved features, which enables it to retain the structure of objects even beyond the border of the screen.

Object segmentation

Another benefit of a retained structure is the object segmentation. The graph is able to segment objects unsupervised, when the content of the image is only slightly cluttered.

Chain code

Besides these two approaches we also introduced a method to encode the shape of objects [chapter 4]. The method basically interprets the shape as the output of a function and finds exposed points in the first derivation. The calculation is fast and the representation is stable towards partial occlusion and noise. Furthermore the method could easily be enhanced to be completely scale and rotation invariant.

Region growing

Edge tracer

To generate the shapes for the chain code transformation, we proposed a fast region segmentation algorithm [chapter 2]. It was designed for simple color gradients (no texture) as found in cartoons. The segmented areas are traced by a simple and fast edge tracer as described in [chapter 3].

Image filtering

Finally we implemented and tested some image pre-processing filters [chapter 1] and found that a combination of median and Gaussian convolution produces reliable noise reduction.

11 ... WHILE STARING AHEAD

This chapter alone would fill books.
Artificial vision is still at its beginning. It is an enormous area and applications for a reliable system are easy to find. They range from image and video retrieval over augmented reality to autonomous vehicles and robots.

Even considering only our work presented in this paper, there is a lot left to do.

Gradient-based approach

The gradient-based approach should be investigated more in depth for its usability in a real-time environment.
An interesting approach would be to generate motion clusters like ASSET-2 [36] does and to track these over time.

Object segmentation

These could also be used for object segmentation or for video post processing.

Neuronal network

An implementation of Facette as a neuronal network would provide higher flexibility and interesting possibilities as described in [chapter 5].

Feature-based approach

The feature-based approach could be expanded by higher-level information, such as *linear dynamic models* or structure models, to cluster the features to objects.

A better approach would be to generate these models from the observations and to create an artificial representation of the world. Such a mental model would be a great step towards unsupervised learning and artificial intelligence.

Unsupervised learning

Occlusion

Occlusion is a problem not addressed so far by *First Light*. This is definitely a must for a reliable object tracking system and should be addressed in later releases.

Combination of the approaches

The combination of both approaches could improve the tracking a big deal. A good tracking system should use all information available. Features should be extracted and encoded based on color, texture and shape.

The visual system of humans does so. In the center of our view-field, we are able to detect and recognize various features and objects in different ways. In the periphery, we perceive low-level optic flow, for example when driving in a car.

3-D reconstruction of 2-D images

The reconstruction of a three-dimensional model from two-dimensional input images is another topic of intense research. The trajectors of the perceived features could be used to do so. Given seven trajectors we could estimate an affine matrix and deduce the third dimension of the points.

Chain code

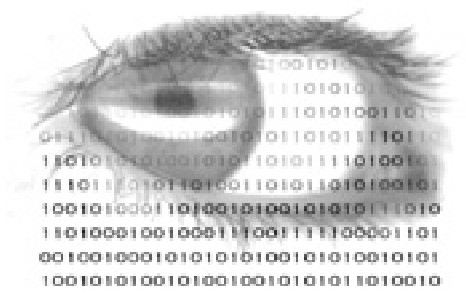
The chain code introduced in [chapter 4] is also worth further evaluation. It could be enhanced to be rotation and scale invariant using elastic graphs and multi-scale pyramids, resp.

Seeing means believing

As stated at the beginning of this chapter, we could continue with this enumeration for some more pages. There are many unsolved problems at every level, from early vision up to the storage and interpretation of the environment. We believe that the ability to see is inherently important for higher intelligence and that mankind would not be what it is without it. Seeing means believing.

Ask yourselves, which sense would you miss most?

PART V



Realization

12 Architecture

The following chapters describe the main concepts of the developed applications. We do not describe the classes in detail nor do we describe all the classes. *First Light* and *Facette* were designed to be a proof of concept. They are a patchwork of over 100 classes with a total of over 17'000 lines of code.
While it would be a waste of time to explain every piece, the basic concepts remain valid for later implementations of the approaches described in this paper.

12.1 LIBRARIES

First Light and *Facette* use the following libraries.

- ffmpeg
- Gandalf
- QT

All libraries are open source and available for MacOS X, Linux and Windows under the GPL (GNU public license).
These libraries have some more dependencies, which are not listed here. Please refer to their documentation for more information.

12.1.1 FFMPEG

ffmpeg is one of the fastest video encoding/decoding libraries available.
Both applications use the library to process the video data.

version	url
0.4.9-pre1	http://ffmpeg.sourceforge.net

12.1.2 GANDALF

Gandalf is a great library, which provides image processing routines as well as vision related algorithms. It is very fast because it is completely written in C and it is extensively documented.
First Light uses the Harris corner detection algorithm.

version	url
1.3.2	http://gandalf-library.sourceforge.net

12.1.3 QT

Qt is in our opinion the best GUI library for C++. It is simple enough to be learned in one day and powerful enough to create any GUI. Its documentation is very detailed and supported by many helpful examples.
Both applications use Qt for anything related to the GUI.

version	url
3.3.3	http://trolltech.com

12.2 Technical equipment

12.2.1 Hardware

Processor type	Intel Pentium 4
Clock speed	2.60 GHz
Memory	0.99 GB

12.2.2 Operating system

Linux Knoppix	
Kernel version	2.4.26

13 First Light

The application *First Light* implements the feature-based approach described in [chapters 6,7,8,9]. In addition the algorithms in [chapters 1,2,3,4] are implemented to support the approach.

13.1 OVERVIEW

The application consists of 4 stages

- Video Stage
- Feature Detection
- Feature Tracking
- Feature Graph

Especially the two center stages (Feature Detection and Feature Tracking) were designed to be modular, so we could test different feature detectors / trackers.

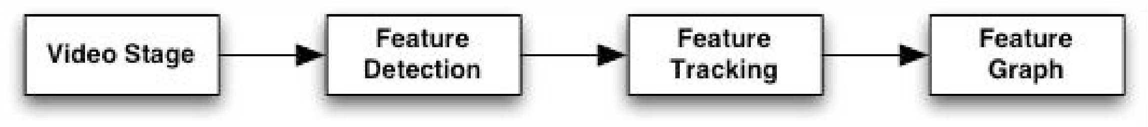


Figure 13-1: First Light consists of 4 stages

13.2 VIDEO STAGE

The video stage provides access to movie files of different formats. On one hand it is a wrapper class for the ffmpeg library [chapter 12.1.1] and on the other hand it provides additional image processing techniques [chapter 1].

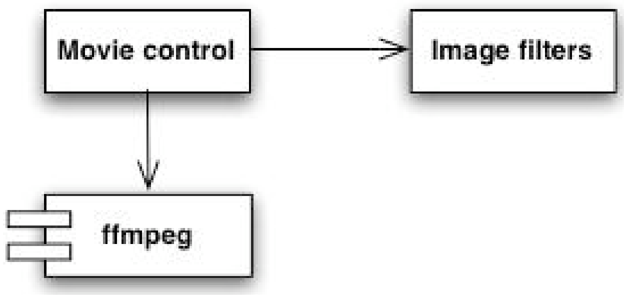


Figure 13-2: The Video Stage

Thanks to ffmpeg the application can process a variety of available video formats.

13.3 FEATURE DETECTION

The feature detection stage is completely modular and the modules can be exchanged at runtime.

13.3.1 FEATURE DETECTORS

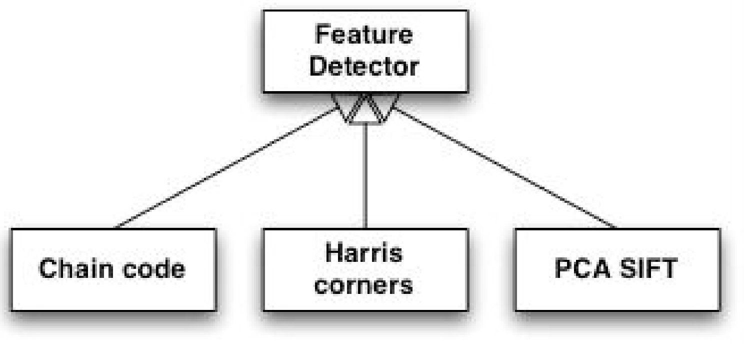


Figure 13-3: The feature detectors

The functionality of the feature detectors is described in [chapter 6].

13.3.1.1 CHAIN CODE FEATURE DETECTOR

The chain code feature detector uses the concepts described in [chapters 2,3,4].

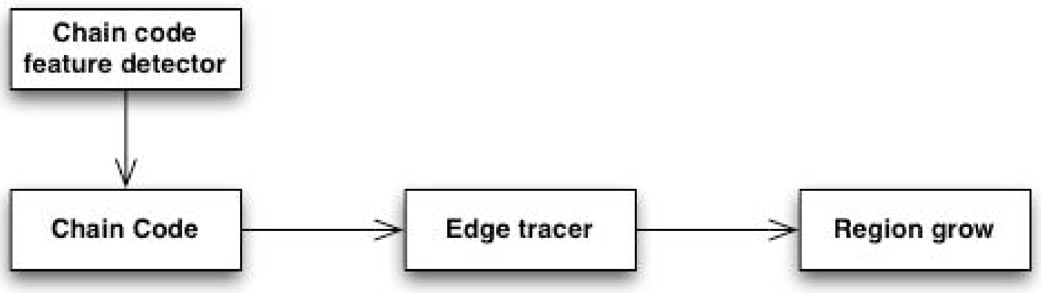


Figure 13-4: Chain code feature detector

13.3.1.2 HARRIS CORNERS FEATURE DETECTOR

The Harris corners feature detector [chapter 6.2] is a wrapper class for the Gandalf library [chapter 12.1.2].

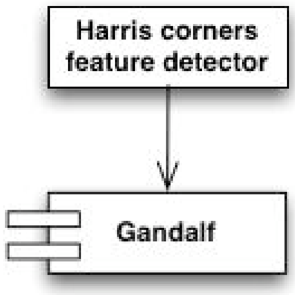


Figure 13-5: Harris corners feature detector

13.3.1.3 PCA SIFT feature detector

The PCA SIFT feature detector is a slightly modified version [chapter 6.2] of Yan Ke's and Rahul Sukthankars' original source code. They gave us the code for research purposes.

Figure 13-6: PCA SIFT feature descriptor

13.3.2 Feature descriptors

Theoretically, the system contains two feature descriptors [chapter 7]. But because the PCA SIFT descriptor needs a scale-invariant feature detector, it is not used in practice.

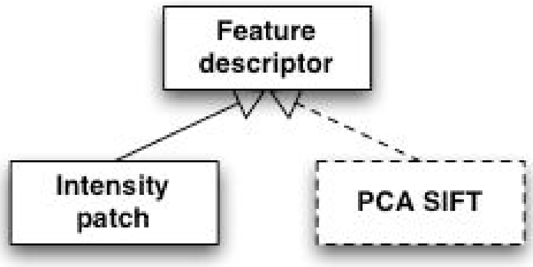


Figure 13-7: The feature descriptors

13.4 FEATURE TRACKING

Like the Feature Detection stage [chapter 13.3], the Feature Tracking stage is completely modular. The different tracking modules can be exchanged at runtime. The features are tracked using *trackable feature graphs* [chapter 13.4.2].

13.4.1 FEATURE TRACKERS

The functionality of the feature trackers is described in [chapter 8]. Note that the *enhanced feature tracker* is deprecated and not included in the system anymore. It is only mentioned to conform with [chapter 8].

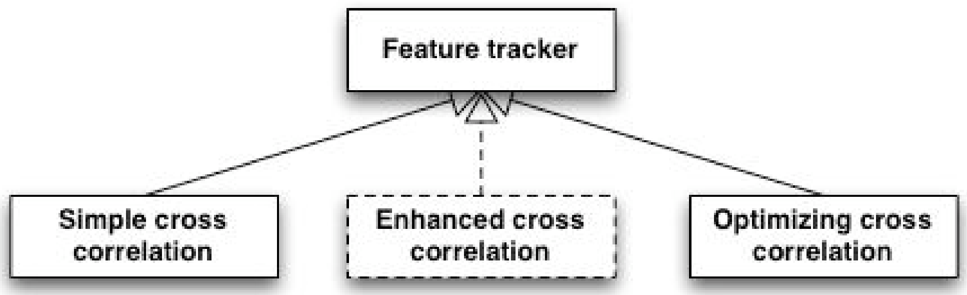


Figure 13-8: Feature trackers

13.4.2 TRACKABLE FEATURE GRAPH

Features are not tracked directly because the trackers need additional information the feature does not need to have. I.e. trajector and matched feature.
Therefore every feature is wrapped by a *trackable feature*, which provides these capabilities. These trackable features are organized in a *trackable feature graph* analog to the *feature graph* described in [chapter 13.5].

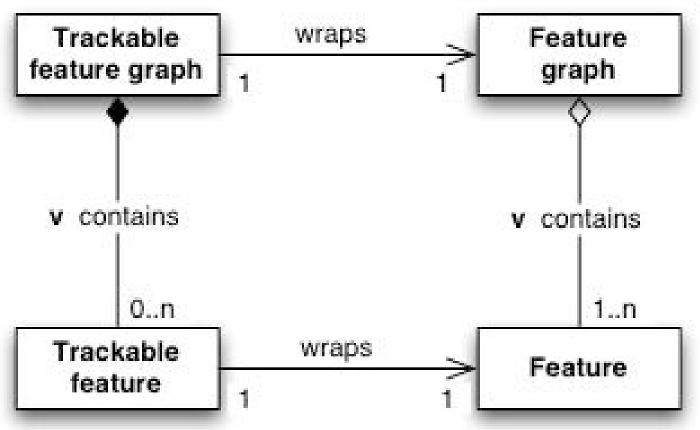


Figure 13-9: Trackable feature graph

Trackable features are linked using the same linking algorithms as the feature graph does [chapter 13.5].

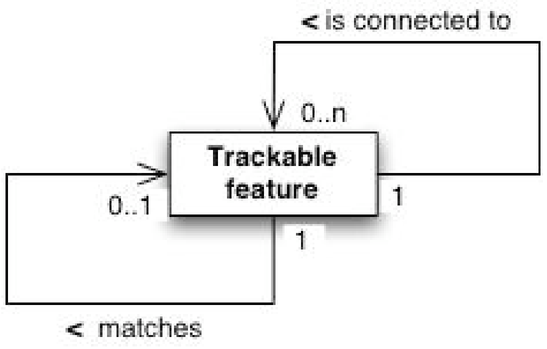


Figure 13-10: Trackable feature

13.5 Feature Graph

The feature graph consists mainly out of features (nodes) and their correlations (edges).

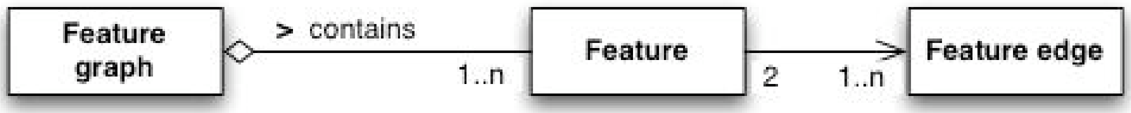


Figure 13-11: Feature graph

14 Facette

Facette is the implementation of the gradient-based optic flow estimation introduced in [chapter 5.1].
It uses the image filtering techniques described in [chapter 1] to pre-process the frames.

14.1 OVERVIEW

The application consists of three stages

- Video Stage
- Facette Stage
- Post-processing Stage

In contrast to *First Light* [chapter 13], only the last stage is modular. The first stage is exactly the same as for *First Light* and explained in [chapter 13.2].

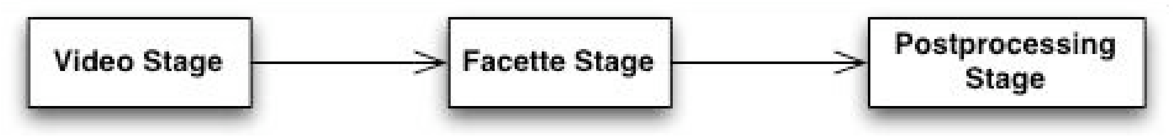


Figure 14-1: Facette consists of 3 stages.

14.2 FACETTE STAGE

The Facette Stage corresponds to a combination of the Feature Detection and Feature Tracking stages in *First Light*.
The stage is responsible to build the pyramids [chapter 5.1.2] and estimate the optic flow [chapter 5.1.5].

14.2.1 FACETTE PYRAMID

The Facette pyramid is realized as a template, because it has different applications. The *RGB Facette tracker* for example uses three integer pyramids, while the saliency pyramid is a Boolean pyramid.

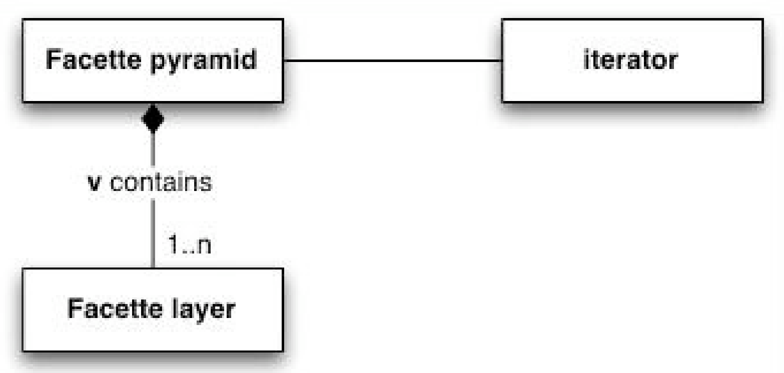


Figure 14-2: Facette pyramid

The iterator class is a convenient and powerful method to navigate inside the pyramid as well as inside a single layer.

14.2.2 FACETTE TRACKER

We implemented two different Facette trackers; one tracks in all three RGB channels, the other in the grayscale channel only.

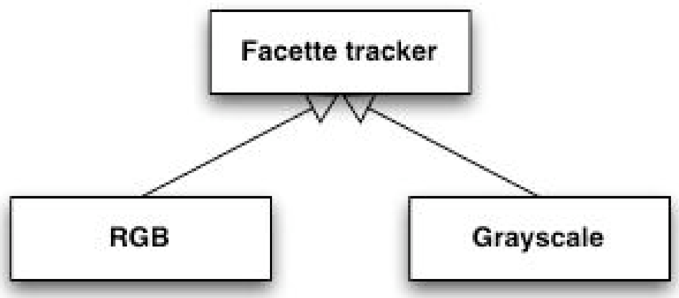


Figure 14-3: Facette tracker

14.3 POST-PROCESSING STAGE

The Post-processing Stage provides some algorithms to interpret the optic flow. The most important are:

- Motion field [5.10.1.2]
- Source-sink field [5.10.1.3]

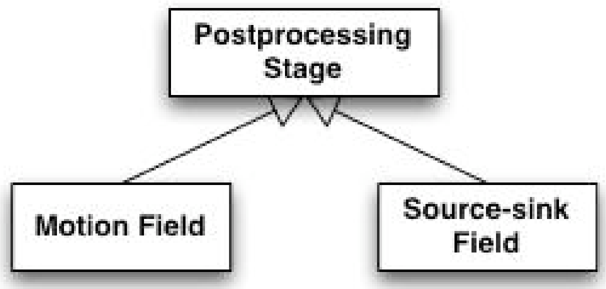
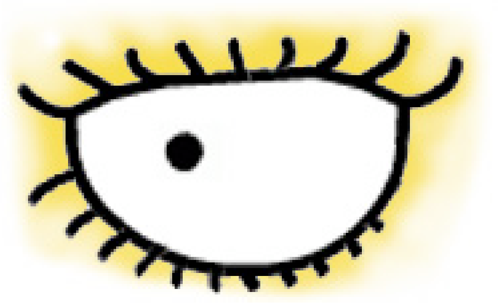


Figure 14-4: Post-processing Stage

APPENDIX



…

15 Credits

I want to thank

- **Prof. Dr. J. Joller**, for letting me choose such an interesting topic for my diploma thesis.

- **Yan Ke** and **Rahul Sukthankar,** for the source code of the PCA SIFT algorithm. It saved me a lot of time.

- **Sebastian de Castelberg** and **Markus Kinzler**, for their Linux support.

- **My parents**, for granting me the possibility to study.

- **All my friends here at school**, for stealing my precious time with pool-billard, which I loose always anyway.

- **And all my other friends**, for tolerating the lack of time I spent with them during these eight weeks.

- **Matt Groening**, for inventing *The Simpsons*.

- **Beer**, for being beer.

16 Table of figures

17 Bibliography

[1] Russel S., Norvig P., *"Artificial Intelligence – a modern approach"*, pp. 453, Second Edition, Pearson Education, Inc., 2003

[2] Pollefeys M., *"Tutorial on 3D Modeling from images"*, http://www.esat.kuleuven.ac.be/~pollefey/tutorial/node53.html, 2000

[3] Chandran S., Madheshiya K. K., „A Fast Segmentation Algorithm Revisited", Computer Science and Engg. Dept., 2002

[4] Forsyth D. A. and Ponce J., *"Computer Vision – a modern approach"*, Pearson Education, Inc., 2003

[5] Harris C. and Stephens M., *"A combined corner and edge detector"*, Fourth Alvey Vision Conference, pp.147-151, 1988

[6] Pollefeys M., *"Tutorial on 3D Modeling from images"*, http://www.esat.kuleuven.ac.be/~pollefey/tutorial/node51.html, 2000

[7] Lowe D. G., *"Object Recognition from Local Scale-Invariant Features"*, Proc. of the International Conference on Computer Vision, 1999

[8] Lowe D. G., *"Towards a Computational Model for Object Recognition in IT Cortex"*, First IEEE International Workshop on Biologically Motivated Computer Vision, 2000

[9] Lowe D. G., *"Distinctive Image Features from Scale-Invariant Keypoints"*, International Journal of Computer Vision, 2004

[10] Ke Y. and Sukthankar R., "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors", Computer Vision and Pattern Recognition, 2004

[11] Canny J., *"A Computational Approach to Edge Detection"*, 1983

[12] Asano, T., Chen, D.Z., Katoh, N., Tokuyama T., *"Polynomial-time solutions to image segmentation", pp.* 104-113., Proc. of the 7th Ann. SIAM-ACM Conference on Discrete Algorithms (Jan. 1996)

[13] Martin D., Fowlkes C., *"The Berkeley Segmentation Dataset and Benchmark"*, http://www.cs.berkeley.edu/projects/vision/grouping/segbench/, 2003

[14] J. Shi and J. Malik, "*Normalized Cuts and Image Segmentation*", *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2000.

[15] S. Wesolkowski, P. Fieguth, *„A probabilistic framework for image segmentation"*, IEEE ICIP, 2003

[16] Author unknown, *„The Hough transform"*, Massey University, Albany, http://cs-alb-pc3.massey.ac.nz/notes/59318/l11.html

[17] Barrows G., *„What is ‚Optic Flow'"*, http://www.centeye.com/pages/techres/opticflow.html

[18] Smith S. M., *„Finding Optic Flow"*, Oxford Centre for Functional Magnetic Resonance Imaging of the Brain, http://www.fmrib.ox.ac.uk/~steve/review/review/node1.html, 1997

[19] B.K.P. Horn and B.G. Schunk, *„Determining optical flow"*, Artificial Intelligence, 17:185–203, 1981

[20] Forina M. *et al.*, *„Minimum spanning tree: ordering edges to identify clustering structure"*, Facoltà di Farmacia, Università di Genova, 2004

[21] Park C. H., Park H, *„Nonlinear feature extraction based on centroids and kernel functions"*, University of Minnesota, 2003

[22] Paragios N., Tziritas G., *„Adaptive detection and localization of moving objects in image sequences"*, University of Crete, 1996

[23] Sanfeliu, A. *et al.*, *„Graph-based representations and techniques for image processing and analysis"*, Universitat Politècnica de Catalunya, Barcelona, 2000

[24] Palm C., *„Color texture classification by integrative Co-occurence matrices"*, Institute for Medical Informatics, Aachen University of Technology, 2003

[25] Mueller M. *et al*, *„Edge- and region-based segmentation technique for the extraction of large, man-made objects in high-resolution satellite imagery"*, GeoForschungsZentrum Potsdam, 2004

[26] Dougherty E. R., Brun M., „A probabilistic theory of clustering", Department of Electrical Engeneering, Texas, 2003

[27] Ferraro M. *et al.*, *„Entropy-based representation of image information"*, Dipartimento di Fisica Sperimentale, Università di Torino, 2002

[28] Ruberto C. D., *„Recognition of shapes by attributed skeletal graphs"*, Dipartimento di Matematica e Informatica, Università di Cogliari, 2003

[29] Toivanen, P. J. *et al.*, *„Edge detection in multispectral images using the self-organizing map"*, Laboratory of Information Processing, Lappeenranta University of Technology, 2003

[30] Shu H. Z. *et al.*, *„Moment-based methods for polygonal approximation of digitized curves"*, Departement of Biology and Medical Engeneering, China, 2000

[31] Triesch J., von der Malsburg C., *„Classification of hand postures against complex backgrounds using elastic graph matching"*, Departement of Cognitive Science, UC San Diego, 2002

[32] Banarse D. S. *et al.*, *„Analysis and application of a self-organising image recognition neural network"*, University of Wales, 1999

[33] Zhang D., Lu G., *„Review of shape representation and description techniques"*, Gippsland School of Computing and Info. Tech., Monash University, Australia, 2002

[34] Foucherot I. *et al.*, *„New methods for analysing colour texture based on the Karhunen-Loeve transform and quantification"*, Laboratoire d'Electronique Image, Université de Bourgogne, 2004

[35] Smith S.M., and Brady J.M., „A scene segmenter; visual tracking of moving vehicles", Engineering Applications of Artificial Intelligence, 7(2):191–204, 1994

[36] Smith S. et al., „ASSET 2: Real-Time Motion Segmentation and Object Tracking ", http://www.fmrib.ox.ac.uk/~steve/asset/, Fifth International Conference on Computer Vision, 1995

17.1 DATA ARCHIVES

[37] Dryden Research Aircraft Movie Collection, http://www.dfrc.nasa.gov/Gallery/Movie/index.html

[38] Berkeley Online Media Resources, http://www.lib.berkeley.edu/MRC/onlinemediamenu.html

[39] Internet Archive: Moving Image Archive, http://www.archive.org/movies/movies.php

[40] CNN Video Vault, http://www.cnn.com/video_vault/index.html

[41] Newsfilm Library at The University of South California, http://www.sc.edu/newsfilm/

[42] The Trailer Park, http://www.movie-trailers.com

[43] Google, http://www.google.com

18 Index