Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Tracking by Image Processing in a Real Time System

Examensarbete utfört i Bildbehandling
vid Tekniska högskolan i Linköping
av

Per Öberg

LiTH-ISY-EX-3374-2003

Linköping 2003
Tracking by Image Processing in a Real Time System

Examensarbete utfört i Bildbehandling vid Tekniska högskolan i Linköping av Per Öberg

LiTH-ISY-EX-3374-2003

Handledare: Klas Nordberg
            Per-Erik Forssén
            ISY, LIU

Examinator: Klas Nordberg
            ISY, LIU

Linköping, 1 March, 2003
Målföljning genom bildbehandling i ett realtidssystem

Tracking by Image Processing in a Real Time System

Per Öberg

Abstract

This master’s thesis develops an algorithm for tracking of cars robust enough to handle turning cars. It is implemented in the image processing environment Image Processing Application Programming Interface (ipapi) for use with the Witas project.

Firstly, algorithms, comparable with one currently used in the Witas-project, are studied. The focus is on how rotation, that originates from the turning of the cars, affects tracking performance. The algorithms studied all perform an exhaustive search over a region, close to the last known position of the object being tracked, to find a match. After this, an iterative algorithm, based on the idea that a car can only rotate, translate and change scale, is introduced. The algorithm estimates the parameters describing this rotation, translation, and change of scale, iteratively. The iterative process needs a initial parameter estimate that is accurate enough for the algorithm to converge. The developed algorithm is based on an earlier publication on the subject, however the mathematical description, and deduction, of it is taken one step further than in this publication.

The iterative algorithm used performs well under the assumption that the data used fulfills some basic criteria. These demands comprises: placement of camera, template size as well as how the parameters may vary between two observations. The iterative algorithm is also potentially faster than exhaustive search methods, because few iterations are needed when the parameters change slowly. Better initial parameters should improve stability and speed of convergation. Other suggestions that could give better performance is discussed, e.g., methods to better extract the target from the surroundings.

Keywords

Tracking, Rotating templates, Witas, Lucas-Kanade
Abstract

This master’s thesis develops an algorithm for tracking of cars robust enough to handle turning cars. It is implemented in the image processing environment *Image Processing Application Programming Interface (IPAPI)* for use with the WITAS project.

Firstly, algorithms, comparable with one currently used in the WITAS-project, are studied. The focus is on how rotation, that originates from the turning of the cars, affects tracking performance. The algorithms studied all perform an exhaustive search over a region, close to the last known position of the object being tracked, to find a match. After this, an iterative algorithm, based on the idea that a car can only rotate, translate and change scale, is introduced. The algorithm estimates the parameters describing this rotation, translation, and change of scale, iteratively. The iterative process needs an initial parameter estimate that is accurate enough for the algorithm to converge. The developed algorithm is based on an earlier publication on the subject, however the mathematical description, and deduction, of it is taken one step further than in this publication.

The iterative algorithm used performs well under the assumption that the data used fulfills some basic criteria. These demands comprises: placement of camera, template size as well as how the parameters may vary between two observations. The iterative algorithm is also potentially faster than exhaustive search methods, because few iterations are needed when the parameters change slowly. Better initial parameters should improve stability and speed of convergence. Other suggestions that could give better performance is discussed, e.g., methods to better extract the target from the surroundings.
Sammanfattning


Acknowledgements

I would like to thank everyone who have helped me with my thesis. Special thanks to my examiner, Klas Nordberg, and my supervisor, Per-Erik Forssén, who have both been very supporting. I would also like to thank everyone else at cvl, especially Johan Wiklund who, more or less, is in charge of the computer system, and have had to deal with my questions and problems. Finally everyone who read, and reread, my thesis over and over again coming with ideas and suggestions deserves some credit, thank you Gustaf Hendeby and Per-Erik. Finally Gustaf also deserves credit for the \LaTeX suggestions and \LaTeX hacking.

Thank you all!

Linköping, February 2003
Per Öberg
# Contents

I Introduction 1

1 Introduction 3
   1.1 Background 3
   1.2 Existing Systems 4
      1.2.1 IPAPI 4
      1.2.2 PAIRS 4
   1.3 Objectives 5
   1.4 Limitations 5
   1.5 Outline 6

II Theory 7

2 The Nature of a Continuous 3D World 9
   2.1 Camera Projections 9
   2.2 Occlusion Due to Rotation 12
   2.3 Continuous or Discrete 13

3 Basic Template Matching 17
   3.1 PAIRS 17
      3.1.1 Main Ideas 18
      3.1.2 Observations 18
   3.2 Normalized Cross Correlation 18
   3.3 Sum of Squared Differences 19
   3.4 Other Methods 19

4 Applied Template Matching 21
   4.1 Extended Lucas-Kanade Tracking 21
      4.1.1 Image Motion Model 21
      4.1.2 Image Motion Estimation 22
      4.1.3 Choosing Templates 25
      4.1.4 Problems and Solutions 25
      4.1.5 Summary 26
   4.2 Log-Polar Transform and CDD 27
Part I

Introduction
Chapter 1

Introduction

This chapter gives an overview of the background and objectives of this thesis. The background and already existing systems are described, and the objectives of this thesis are specified. Problems covered in this thesis are presented and discussed. Finally an outline of the thesis is presented.

1.1 Background

This thesis has been performed as a part of the Wallenberg laboratory on Information Technology and Autonomous Systems (WITAS) research project. The following quote from the Computer Vision Laboratory (CVL)-WITAS project home page [1] gives some idea about the major goals of the project. More information about the WITAS project can be found in [2] and on the WITAS home page [3].

The major goal is to demonstrate an airborne computer system which is able to make rational decisions about the continued operation of the aircraft, based on various sources of knowledge including pre-stored geographical knowledge, knowledge obtained from vision sensors, and knowledge communicated to it by radio.[1]

The WITAS on board system\(^1\) has a number of more or less separated parts. One of these parts is the vision system with the purpose of analyzing data coming from vision sensors, i.e., cameras. This system needs efficient algorithms for data analysis and the WITAS specific research at CVL focuses on this. One of the objectives of the vision system is to track moving objects on the ground, i.e., cars. When tracking a car, the car is first identified by the system and interesting features are extracted. The next step is to keep track of the car, and this is the problem discussed in this thesis. One of the limitations when designing an algorithm for this kind of tracking (see appendix A.1 for an explanation of the use of “tracking”) is the limited computational and memory resources of the system, and the fact that the system is working in “real time”. Apart from these resource

\(^1\)What is not mentioned here is that a helicopter was chosen as the WITAS platform.
limitations a number of problems may occur. The car can be subject to a number of different annoying phenomena such as occlusion, scaling, or more complex linear transformations, and other changes of features during the tracking.

More specifically, the car can be more or less occluded by other objects or it may turn and change its appearance, partly by occluding some parts of itself due to the nature of the 3D world, and partly by just rotating in 2D. The camera of the vision system can change distance to the car, hence changing the scale, and the car can be subjected to other, linear or non-linear, transformations, again related to the camera and the 3D world. Some of these problems are easily solved, and other are harder to handle.

In this thesis the problems associated with the rotation of the car is studied. An algorithm is implemented using the Image Processing Application Programming Interface (ipapi) system. IPAPI is a programming environment, amongst other things designed for easy management of image processing, described below. The existing tracking algorithm, based on the PAIRS matching algorithm, also described below, is replaced with a new one.

1.2 Existing Systems

During the work on the WITAS project a number of systems and algorithms has been developed. Most relevant for this thesis is the Image Processing Application Programming Interface (IPAPI) system and the PAIRS algorithm which are both presented below.

1.2.1 IPAPI

IPAPI is a WITAS-designed image processing system for easy image processing. More specifically IPAPI is a client-server system where a number of applications can connect to a IPAPI server. The applications instructs the server to build graphs consisting of nodes and all image processing is performed in these nodes. A more thorough description of IPAPI can be found in Section 7.1 and in [12].

1.2.2 PAIRS

As mentioned in Section 1.1 above the PAIRS algorithm can be used when tracking, e.g., cars. PAIRS is based on “Maximum Entropy Matching”, which together with the PAIRS algorithm is described in [4]. The reason that the algorithm is considered fast is that the speed critical parts is easily implemented using simple XOR-operations.

The PAIRS algorithm is covered in Section 3.1, but it deserves to be mentioned here since it is used in the WITAS project, and therefore is part of the already existing systems.
1.3 Objectives

The problem discussed in this thesis is the tracking of cars in the WITAS project. The objective in this thesis was to introduce a tracking algorithm for car tracking. The algorithm should work better than the existing PAIRS algorithm, and should be implemented in IPAPI. If necessary, the algorithm could use different methods for different situations. When expecting only translations a simple algorithm could be used and when expecting worse distortions, i.e., translation in combination with linear transformations and occlusion, a more advanced method could be used. One of the major problems with the PAIRS algorithm is that it loses track of the cars when the cars are turning. The new algorithm should therefore be designed to handle these situations better than the PAIRS algorithm.

1.4 Limitations

When discussing tracking of moving objects most people think of Kalman filters and related filter theory. Performing this kind of tracking is however based on a more or less accurate measure of the position of the object. In this master’s thesis the concept tracking will, however, mean this measuring of position. Little effort is put into the subject of predicting the position, but as concluded later using some sort of prediction could make the algorithms more stable and useful.
1.5 Outline

The thesis outline is described below.

- **Part I : Introduction**
  - **Chapter 1**: Introduction
    This chapter introduces the reader to the thesis. The background and objectives are specified, and existing systems are described.

- **Part II : Theory**
  - **Chapter 2**: The Nature of a Continuous 3D World
    This is where the reader gets introduced to problems related to how our world works, and how images enter a computer system.
  - **Chapter 3**: Basic Template Matching
    Theory for some basic template matching algorithms is introduced and discussed.
  - **Chapter 4**: Applied Template Matching
    Two more advanced matching algorithms are presented. First the Extended Lucas-Kanade Tracking (ELKT) and then the Chen-Defrise-Deconick (CDD) algorithm are introduced.

- **Part III : Results**
  - **Chapter 5**: Rotation Robustness
    In this chapter some of the algorithms introduced earlier are tested for robustness against rotation. The idea is to create a simulation of a case where the only disturbance is rotation.
  - **Chapter 6**: Tracking
    Here tests on real tracking are performed. The results are discussed and a final algorithm is presented.
  - **Chapter 8**: Conclusions and Future Work
    This chapter summarizes the results and conclusions obtained in the earlier chapters. Suggestions for future work are also made.
Part II

Theory
Chapter 2

The Nature of a Continuous 3D World

This chapter will bring forward some problems that are related to the way pictures enter the system. Some camera related issues are discussed as well as some more general problems with computer imaging.

2.1 Camera Projections

In an imaging system a camera takes photos of objects in front of it. When this is used for tracking it is essential to know how movements of the camera and movements on the ground affects the acquire image. For this a camera model is needed. One of the simpler camera models is the pinhole model which is described in [5]. The model is also illustrated in Figure 2.1 below. The coordinates of the object in the world is called world coordinates and the coordinates in the image plane is called image coordinates. Here an object with a visible point at \([x\ y\ z]^T\) in world coordinates is reproduced at \([x'\ y']^T\) in image coordinates. Since the two triangles in Figure 2.1 are similar we have

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\begin{bmatrix}
  1 \\
  z
\end{bmatrix}
= 
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
\begin{bmatrix}
  1 \\
  f
\end{bmatrix}
\]

which means that a point at \([x, y, z]^T\) is reproduced at \(\frac{f}{z} [x \ y]^T\) in image coordinates. Multiplying both sides of the equation above with \(z\) yields

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= 
\frac{f}{z} \begin{bmatrix}
  x \\
  y
\end{bmatrix}.
\]  

(2.1)

From this relationship, some interesting observations can be made. If the camera is placed looking at a flat surface parallel to the image plane of the camera, the \(\frac{1}{z}\) factor can be considered constant. When this is true all affine transformations, i.e., transformations that can be described with rotation, scaling and translation,
on the surface will correspond to the same affine transformations in the image plane. Even if the surface is not completely flat, $\frac{1}{z}$ will be almost constant as long as the surface is distant. If the surface is not parallel to the image plane of the camera, $\frac{1}{z}$ will change, but in a fairly decent manner. Inserting (2.1) in the plane equation $ax + by + z = d$ yields

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{f}{d - ax - by} \begin{bmatrix} x \\ y \end{bmatrix}.$$  

(2.2)

This shows that tilting the camera, using a fairly large $d$ (the plane far away from the camera), and fairly small $a$ and $b$ (small angles between image plane and surface) does not introduce noticeable non-linearities for small objects. If however the object was to be translated, the scaling of the object would change since $x$ and $y$ would change more than they do over the object itself. Normally objects on the ground only translate and rotate. If the angles between the image plane and the surface are sufficiently small it could be assumed that translation and rotation of a small object can be described by an affine transformation. If the image plane can not be assumed to be parallel to the surface, i.e., the tilting of the camera is substantial, there could however be some problems. The first problem is that when the object is translated in such a plane the scale changes differ over the object. The second is that the scaling changes with the rotation, as illustrated in Figure 2.2 below. If the objects in Figure 2.2 were correctly transformed the dashed lines would line up. ‘Case 1’ corresponds to the first problem mentioned above and ‘Case 2’ to the second.

The conclusion is thus that, if the surface is far away, and the objects on it are somewhat flat, all affine transformations of the objects about the surface can be described as affine transformations in the image plane.
Figure 2.2. An object on a tilted surface is subjected to a change of scale that is dependent on its coordinates when it translates or rotates. Assuming this can be described by affine transformations could lead to problems.
2.2 Occlusion Due to Rotation

Another problem related to the fact that we live in a three dimensional world is that objects may occlude themselves. This occlusion means that a part of an object is visible in one moment and not in another, making it difficult to track. This is illustrated in Figure 2.3, where a feature is a point in an image with a high information content.

![Illustration of how a car occludes parts of itself during rotation.](image)

When tracking a car in a image sequence some information on the car is needed so that the algorithm knows what to look for in the images. The easiest way is to cut out a sample of the car from the first image in the sequence. This sample is called template. When using templates for tracking one idea for a solution to this self occlusion problem is simply to detect the occlusion, and take a new sample of the car at the exact same position. It would then be possible to save the old template, and when the new template has gone bad try to match with the old one, hoping for that the disappearing of the new feature means that the old feature has become visible again. The weakness of this approach is obviously that if the feature fails to reappear the tracking could be lost with half a pixel each template switch. A solution to this would be to use more than one old template, thus always having some templates to choose between. If all goes well there will always be a way to figure out a good start point for the tracking algorithm.
2.3 Continuous or Discrete

This section is not linked to the 3D nature of the world in which we live, but its continuous behavior. Looking at the world at a macroscopic level gives the impression that it is more or less continuous. In computers however everything is discrete which leads to some problems. A computer stored image suffers from two of these problems. The first, and not very cumbersome, problem is that the light intensity values are quantized which leads to disturbances. This can however be thought of as noise and is not really a problem. The second problem is that a computer stored image is a collection of light intensity values from different points in the world without any information of what is in between. The intensity values in the different points are called samples, and the image is said to be sampled. If the world is a continuous function with a low enough frequency this is only a computational problem. If the frequency, however, is too high there will be aliasing problems. This can all be studied in [6] and [7], and is illustrated in Figure 2.4 below. Given that there is no aliasing the computational problem is thus what is left.
The Nature of a Continuous 3D World

(a) Original image

(b) Image is sampled with a sample frequency that is \( \frac{1}{15} \) of the original and then interpolated with bicubic interpolation (see below) to match the size of the original.

(c) Fourier transform of original.

(d) Fourier transform of sub-sampled image. Here the two peaks from the original image look like one peak at a lower frequency.

**Figure 2.4.** Aliasing occurs when the sample frequency is too low.
Sometimes it is nice to know what is in between two pixels in an image, or to re-sample the image as if it was taken with half a pixel displacement. This can be computed, or at least approximated, and there are a couple of ways of doing it. Assuming the frequency of the sampled function is low enough the best alternative would be to use the sinc function, \( \text{sinc} = \lim_{\xi \to \infty} \frac{\sin(\pi \xi)}{(\pi \xi)} \), to re-sample the image. This, however, cannot be done. For one the sinc function is not spatially limited and has to be truncated somewhere, secondly there is no need to be that exact and finally the nature rarely has a low enough frequency and a sharp edge interpolated with a sinc function would suffer badly from ringing phenomena. Instead attempts are made to approximate the sinc function with functions that are easier to calculate and give better results. Some of these are illustrated in Figure 2.5. In a one dimensional case the nearest neighbor uses only the closest sample, linear uses the two closest samples and bicubic uses the four closest. As seen in the figure, the bicubic interpolation method has the best correspondence to the sinc function. This does however not always mean that it gives the best results. If the interpolation method is used to compute one or two samples in between the original samples, linear interpolation gives good enough results. In this case bicubic interpolation can be worse than linear because it acts more like a low-pass filter than the linear interpolation does. This is because bicubic interpolation uses more neighboring samples than the linear interpolation. When interpolating many samples in between two samples the linear interpolation method will give diamond like artifacts which will be avoided with bicubic interpolation. The conclusion is thus that when interpolating a rotated, translated and somewhat scaled image linear interpolation is sufficient. In Figure 2.6 below some visual results from image re-samplings are presented.

![Figure 2.5](image-url)

**Figure 2.5.** Three different ways of approximating the sinc function. The sinc function is the dashed curve in the figures.
The Nature of a Continuous 3D World

Figure 2.6. Resulting images for different ways of interpolation images. The resulting images hold 256 times more samples than the original does.
Chapter 3

Basic Template Matching

This chapter will give the reader an overview of some basic methods for template matching. The PAIRS, NCC and SSD algorithms are introduced and discussed.

3.1 PAIRS

The PAIRS algorithm was developed in the WITAS project as an implementation of “Maximum Entropy Matching”. Below is a short overview of the algorithm and the idea behind it. For the interested reader a more complete documentation of the algorithm, and the ideas in “Maximum Entropy Matching”, can be found in [4].

An illustration of the PAIRS algorithm is provided in Figure 3.1 to the right. The PAIRS algorithm consists of three parts. First coordinate pairs are created. These pairs are then used to create a Template Bit set (TB) and an Image Region Bit set (IRB). The TB is pre-calculated, once and for all, as soon as the template is known, whereas the IRB is calculated for every position of the image region. The third, and last, part is the actual matching. Matching is done by comparing the TB with all the IRB of the image region. To do this it is possible to use an XOR-operation on the bit sets and then calculate the number of zeros in the result. The position with most zeros is then considered to be the match. As a certainty measure the number of zeros divided by the number of coordinate pairs can be used.

Figure 3.1. Illustration of the PAIRS algorithm.
3.1.1 Main Ideas

The following quote of [4, p. 4] illustrates some of the ideas of the PAIRS algorithm.

The approach is based on the following statements.

1. The less data we compare for each possible template, the faster this comparison will be.
2. The data we compare should have high entropy.
3. On average, less data needs to be compared to conclude two objects dissimilar than to conclude they are similar. This statement will be called the fast dissimilarity principle.
4. The data that we use for the comparison should be chosen so that similarity measurement will be distortion persistent.

The PAIRS algorithm is optimized for speed. The main argument to use it seems to be that it performs as well as other algorithms, but with less calculations.

3.1.2 Observations

Some observations can be made when studying the theory of the PAIRS algorithm.

1. In [4] the authors conclude that the PAIRS algorithm works best for template sizes of $16 \times 16$ and $32 \times 32$ pixels. Their idea about the reason for decreasing performance when using smaller templates is that the uniqueness of the template decreases when smaller templates are used. When using larger templates on the other hand they propose that any geometrical distortion has bigger effect on the outer pixels than when using small templates.

2. As for the number of coordinate pairs no recommendations are made in [4]. The performance of the algorithm clearly depends on the number of coordinate pairs. When using fewer coordinate pairs the performance decreases and when using more pairs the time consumption increases.

3. The selection of coordinate pairs can be critical. One idea is that this could be done randomly. This could however, in my opinion, make the method very unstable since there is no way of being certain that good coordinates are obtained. Another idea is that this should be done in an entropy maximizing way. This is however not tested in [4].

4. The similarity measure of the PAIRS algorithm is very discretized. This could cause problems when trying to decide the position of the template with sub-pixel resolution. This also makes the certainty measure very discretized.

3.2 Normalized Cross Correlation

A widely used matching algorithm is Normalized Cross Correlation (NCC). The idea is that when correlating an image and a template we get the highest correlation
measurement when placing the template in its correct position. The correlation function in NCC is

\[ r(x) = \frac{\sum_{y \in T(y)} T(y)I(y + x)}{\sqrt{\sum_{y \in T(y)} T^2(y) \sum_{y \in T(y)} I^2(y + x)}} \]  

(3.1)

where \( I(\cdot) \) is the image and \( T(\cdot) \) is the template. The correlation function, \( r(s) \), is hence a function depending on the displacement, \( s \). The task of finding the matching position is therefore \( s = \arg \max_x r(s) \). This is briefly described in [8, pp. 354–355]. The square root normalization used here is not necessarily the only usable normalization and can differ between different users.

### 3.3 Sum of Squared Differences

Another widely used matching algorithm is **Sum of Squared Differences (SSD)**. In SSD the approach is to minimize a distance measure between the template and the image. The lower the distance measure is the more likely they are to represent the same thing. The distance measure used in SSD is

\[ M(x) = \sum_{y \in T(y)} (I(x + y) - T(y))^2 \]  

(3.2)

where \( M(x) \) is the difference in position \( x \), \( I(x) \) is the image and \( T(y) \) is the template. The formula above can easily be implemented as a series of convolutions. The minimum of the match result, \( M(x) \), is calculated and this is used as matching position. A refinement of the method is described in [8, pp. 355–356] where an intelligent method of finding the minimum of the distance measure is used.

### 3.4 Other Methods

The above described matching algorithms can be more or less easily modified to get different results and different performance. A number of algorithms which correspond to the NCC algorithms are described in [9, pp. 77–85].
Basic Template Matching
Chapter 4

Applied Template Matching

In this chapter some more advanced methods for template matching are introduced. First a method introduced by J. Shi and C. Tomasi is described. After that the Log-Polar Transform and the Chen-Defrise-Deconick (CDD) algorithms are discussed.

4.1 Extended Lucas-Kanade Tracking

In this section a method proposed by J. Shi and C. Tomasi in [10] is introduced. It is an extension of a method proposed by B. D. Lucas and T. Kanade and will therefore, from now on, be called Extended Lucas-Kanade Tracking (ELKT). In the original version, tracking was performed assuming there is only a displacement, \( d \), between two images. In the extension, however, a method for checking the accuracy of the tracking, assuming an affine transformation with translation, rotation and scaling, is proposed. In a first attempt one may try to extend the tracker with prediction of all the parameters in the affine transformation, not only the ones corresponding to translation. This is, however, not recommended by the authors because it gives poor results. Instead it is suggested that the full affine model should only be used for detection of bad templates.

4.1.1 Image Motion Model

This section about the image motion model corresponds to [10, Sec. 2]. Most algorithms start with assumptions and models of what is actually happening, and so does this one. The ELKT is based on the assumption that most changes between two frames are caused by image motion and affine transformations or expressed in more mathematical terms,

\[ I_m(x, t + \tau) = I_m(\xi(x, t, \tau), t). \]  (4.1)

The equation shows how the image taken at time \( t + \tau \) can be obtained from the image at time \( t \), moving the points in it. The amount of motion, \( \xi(x) - x \),
is called the displacement of the point at \( x \), and \( \xi \) is called the transformation function. It could now be assumed that the changes of the displacement over a small template is small enough to be considered constant. This would however not be true since it would allow changes to add up during tracking over a sequence of images. Therefore an affine motion field is used,

\[
\xi = Dx + d
\]

\[
D = \begin{bmatrix}
  d_{xx} & d_{xy} \\
  d_{yx} & d_{yy}
\end{bmatrix}
\]

\[
d = \begin{bmatrix}
  d_x \\
  d_y
\end{bmatrix}
\]

Here the transformation function consists of the deformation matrix \( D \), and the translation, \( d \), of the center of the template. The image coordinates, \( x \), are measured with respect to the center of the template.

Let \( J \) and \( K \) be two samples of the image sequence \( I_m \) separated by \( \tau \), i.e., \( J = I_m(x, t_0) \) and \( K = I_m(x, t_0 + \tau) \). Using the affine model proposed above yields

\[
K(Dx + d) = J(x),
\]

which is used as the final motion model. Tracking is now equivalent to determining the six parameters

\[
[d_{xx} \; d_{yx} \; d_{xy} \; d_{yy} \; d_x \; d_y]
\]

that makes up \( D \) and \( d \) in (4.2).

### 4.1.2 Image Motion Estimation

This section is based on [10, Sec. 3] but some major modifications have been made to the proposed algorithm. In [10, Sec. 3] it is stated that the calculations can be iterated as when solving equations with Newton-Raphson’s method, but how to do this is left out. When solving (4.2) we cannot assume perfect correspondence between \( J(x) \) and \( K(Dx + d) \). This due to image noise and imperfections in the image motion model\(^1\). That is, some dissimilarity measure should be minimized. In EKLT the weighted sum of squares,

\[
\varepsilon = \iint_W [J(x) - K(Dx + d)]^2 \omega(x) \, dx.
\]

is chosen as the dissimilarity measure. Here \( \varepsilon \) denotes dissimilarity and \( \omega(x) \) is a weight function. The area of integration, \( W \), is a given feature window where \( \omega(x) \) is defined. \( \omega(x) \) can be constant but can also have other values decided by intuition, or even by some method that adapts it to support occlusion or other phenomena.

The natural approach to minimize the dissimilarity in equation (4.4) would be to differentiate the dissimilarity function with respect to the parameters in (4.3)

\(^1\)Considering the discussion in Chapter 2, the model is indeed not perfect.
and set the results to zero hoping for this to give a global minimum. This equation would however be non-linear, and therefore difficult to solve. One solution is to linearize the function with a truncated Taylor expansion with respect to some good estimate and solve the new system. If the model is a good model, and the estimate is good enough this will yield a better estimate. Iterating the procedure will then hopefully yield correct results.

Let \( D_0 \) and \( d_0 \) be approximations of \( D \) and \( d \) with the approximation errors \( D' \) and \( d' \). Gathering all parameters in \( D \) and \( d \) into \( \mu = [d_{xx} \ d_{yx} \ d_{xy} \ d_{yy} \ d_x \ d_y]^T \) gives, with the same notation as above, \( \mu = \mu_0 + \mu' \). Considering \( \xi \) as a function of both \( x \) and \( \mu \), and again using the notation above then yields

\[
\xi(x, \mu) = \xi(x, \mu_0) + \xi(x, \mu') \\
\xi(x) = \xi_0(x) + \xi'(x),
\]

where \( \xi(x) \) is identical to \( \xi(x, \mu) \) and so on. This can now be used to simplify the expression in (4.4). Starting with a Taylor expansion of \( K \) with respect to \( \xi_0(x) \) yields

\[
K(Dx + d) = K(\xi(x)) = K(\xi_0(x) + \xi'(x)) \approx K(\xi_0(x)) + \xi'(x)^T \cdot \nabla_{\xi_0} K(\xi_0(x)).
\]  

(4.5)

If \( \mu_0 \) is a good approximation of \( \mu \), \( K(\xi_0(x)) \approx J(x) \) which with the chain rule of differentiation yields

\[
\nabla_x J(x) \approx \nabla_x K(\xi_0(x)) = \nabla_x \xi_0(x) \cdot \nabla_{\xi_0} K(\xi_0(x)) = D_0^T \cdot \nabla_{\xi_0} K(\xi_0(x)).
\]  

(4.6)

Assuming it is possible to calculate \( D_0^{-1} \), (4.6) can be used to simplify (4.5) even more thus giving

\[
K(Dx + d) \approx K(\xi_0) + \xi'(x)^T \cdot (D_0^{-1})^T \cdot \nabla_x J(x),
\]

which with (4.4) gives the approximation

\[
\hat{\varepsilon} = \iint_W \left( J(x) - K(\xi_0(x)) - \xi'(x)^T \cdot (D_0^{-1})^T \cdot \nabla_x J(x) \right)^2 \omega(x) \, dx.
\]  

(4.7)

where \( \varepsilon \approx \hat{\varepsilon} \).
The idea is that minimizing the approximation in (4.7) will yield results correct enough to give a better $\mu$. As stated above this minimum can be calculated by setting the derivatives, with respect to the parameters in $\mu$, to zero. To simplify the notation of this two observations are made. First

$[D_0^{-1}]^{T}$ can be written as $\mu^{T} \cdot \Pi \cdot X(x)$ where

$\Pi = \begin{bmatrix} D_0^{-1} & 0 & 0 \\ 0 & D_0^{-1} & 0 \\ 0 & 0 & D_0^{-1} \end{bmatrix}^{T}$ and $X(x) = \begin{bmatrix} x \\ 0 \\ y \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$.

Secondly, $J(x) - K(\xi_0(x))$ can be written as the error image $E(x)$. This gives

$\tilde{\varepsilon} = \int_{W} \left( E(x) - \mu^{T} \cdot \Pi \cdot X(x) \cdot \nabla x J(x) \right)^{2} \omega(x) \, dx$.

which with $\Phi(x) = X(x) \cdot \nabla x J(x)$ becomes

$\tilde{\varepsilon} = \int_{W} \left( E(x) - \mu^{T} \cdot \Phi(x) \right)^{2} \omega(x) \, dx$. \hspace{1cm} (4.8)

Differentiating (4.8) with respect to the parameters in $\mu'$ and setting the result to zero yields

$\nabla_{\mu'} \tilde{\varepsilon} = 2 \cdot \int_{W} \left( E(x) - \mu^{T} \cdot \Pi \cdot \Phi(x) \right) \nabla_{\mu'} \left( \mu^{T} \cdot \Pi \cdot \Phi(x) \right) \, dx = 0 \implies$

$\int_{W} E(x) \cdot \Pi \cdot \Phi(x) \, dx = \int_{W} \left( \mu^{T} \cdot \Pi \cdot \Phi(x) \right)^{T} \cdot \Pi \cdot \Phi(x) \, dx =$

$= \int_{W} \Pi \cdot \Phi(x) \cdot \Phi(x)^{T} \cdot \Pi^{T} \cdot \mu' \, dx$. \hspace{1cm} (4.9)

Since images are discrete the double integration is really a double summation over the area of integration and can be written as product of matrices.

Let $W = \{x_1, \ldots, x_N\}$. Using

$E = \begin{bmatrix} E(x_1) \\ \vdots \\ E(x_N) \end{bmatrix}$ and $\Phi = \begin{bmatrix} \Phi_1(x_1) & \cdots & \Phi_1(x_N) \\ \vdots & \ddots & \vdots \\ \Phi_N(x_1) & \cdots & \Phi_N(x_N) \end{bmatrix}$

to simplify (4.9) yields

$\Pi \cdot \Phi \cdot E^{T} = \Pi \cdot \Phi \cdot \Phi^{T} \cdot \Pi^{T} \cdot \mu' \implies$

$\mu' = (\Pi^{T})^{-1} \cdot (\Phi \cdot \Phi)^{-1} \cdot \Phi \cdot E^{T}$. \hspace{1cm} (4.10)
As stated above the full affine model, where all parameters are determined, is not always preferable. Therefore solving (4.10) is not always necessary. Taking (4.9) and setting \( \mu = [0 \ 0 \ 0 \ d_x \ d_y] \) gives, in the same way as above, the less involved

\[
d' = D_0 \cdot (\nabla J \cdot \nabla J^T)^{-1} \cdot \nabla J \cdot E^T
\]

where

\[
\nabla J = \begin{bmatrix}
\frac{\partial J}{\partial x}(x_1) & \ldots & \frac{\partial J}{\partial x}(x_N) \\
\frac{\partial J}{\partial y}(x_1) & \ldots & \frac{\partial J}{\partial y}(x_N)
\end{bmatrix}.
\]

In a situation where the full affine model is not necessary, solving the translation only equation in (4.11) can thus be satisfactory. When tracking, it is critical to know what is interesting and what is not. Using the full affine model when only \( d \) is wanted will, according to [10], result in poor performance. However, when checking templates for occlusion and calculation of dissimilarity the full affine model is needed to get correct results. The strength in this method lies in its ability to detect poor templates without considering errors caused by affine transformation. If a template, by affine transformations, has changed too much for this method to yield correct results, some other method will have to be used to detect this since this method will not notice that performance decreases. Sometimes, however, it can be enough to inspect the magnitude of \( |D| \) and \( |d| \).

### 4.1.3 Choosing Templates

Shi and Tomasi also mention how to choose templates. Instead of choosing templates arbitrarily or by some intuitive method they state that a template should be chosen so that tracking performance is optimized. Tracking performance will be good if the symmetric matrix \( \nabla J \cdot \nabla J^T \) in (4.11) is well conditioned and the energy is above image noise level. \( \nabla J \cdot \nabla J^T \) is well conditioned when its eigenvalues are somewhat equal in magnitude. However, large eigenvalues means high energy and when the smaller eigenvalue is sufficiently large we usually have a well conditioned matrix. Therefore, to find the best templates, it is wise to chose templates by maximizing the smaller eigenvalues of \( \nabla J \cdot \nabla J^T \).

### 4.1.4 Problems and Solutions

In the earlier sections the mathematical aspects of ELKT have been discussed. Some things have however been left out. In a situation with bad luck, e.g., an initial approximation far away the correct solution, the iterative process could get stuck on a local minima. In a situation like this it could help to low-pass filter \( J \) and \( K \) before matching. This will sometimes give a good enough initial approximation to avoid the local minima when trying again on the original images. If however the initial estimate is totally wrong the method will not work at all no matter how many times the images are low-pass filtered. A good estimate in the sections above is thus a good enough estimate to be able to avoid any local minima on the image data used.
4.1.5 Summary

The highly mathematical approach in earlier sections may hide the different steps of the ELKT algorithm. To clarify, this section contains a summary of the algorithm.

1. Choose a template of the object that is going to be tracked, and set $D_0 = [0]$ and $d_0 = 0$.

2. Calculate the derivatives of the template.

3. Calculate $\Phi$ and $\nabla J$, and then $(\Phi \cdot \Phi^T)^{-1}\Phi$ and $(\nabla J \cdot \nabla J^T)^{-1}\nabla J$.

4. Use $D_0$ and $d_0$, i.e., $\mu_0$, to calculate a template estimate from the latest image, and use this to calculate the error image $E^T$.

5. If needed, calculate $(\Pi^T)^{-1}$.

6. Calculate $\mu'$ or just $d'$ using

$$
\mu' = (\Pi^T)^{-1} \cdot (\Phi \cdot \Phi^T)^{-1} \cdot \Phi \cdot E^T \quad \text{or}
$$

$$
d' = D_0 \cdot (\nabla J \cdot \nabla J^T)^{-1} \cdot \nabla J \cdot E^T
$$

7. Update $\mu$ with $\mu'$ or $d'$ and jump to 4.
4.2 Log-Polar Transform and CDD

The Log-Polar transform is basically a re-sampling that gives the possibility to transform rotation and scaling to 2D translation. The re-sampling functions are

\[
\begin{align*}
t & = \frac{1}{2} \ln \sqrt{x^2 + y^2} \\
\phi & = \text{arg}(x + iy)
\end{align*}
\]

\[
\begin{align*}
x &= e^{\lambda t} \cos(\phi) \\
y &= e^{\lambda t} \sin(\phi)
\end{align*}
\]

(4.12)

If a template is subjected only to rotation and scaling this re-sampling will make it easy to find this scaling and rotation. However, this method is highly intolerant of translation and will not behave well if the template is both translated and rotated. For a closer study of this method, read [9, pp. 87–90]. Combining this method with Fourier transformation will however allow us to get both translation, rotation and scaling invariants. This method is called the Chen-Defrise-Deconick (CDD) method and is also described in [9, pp. 87–90]. The approach is as follows:

1. Calculate the 2D Fourier transform of the image region and the zero padded template.
2. Re-sample the magnitude of the Fourier transforms to the Log-Polar space in (4.12).
3. Perform a 2D matching and calculate the scaling and the rotation.
4. Re-sample the original template with the scaling and the rotation parameters.
5. Perform another 2D-matching between the scaled and rotated template and the image region.

This will as mentioned before give us a rotation, scaling and translation invariant method. The keys are the following facts:

- The magnitude of the Fourier transform is invariant to translation.
- A scaling of a template corresponds to a scaling of the frequency axis.
- Rotation in the spatial domain corresponds to rotation in the Fourier domain.

All the statements above can be extracted from

\[
g(x) = f(A(x + b)) \implies G(u) = \frac{1}{|\det A|} e^{iu^Tb} F([A^T]^{-1}u).
\]

(4.13)

The properties of the Fourier transform can be studied in [11, p. 136]. The CDD method thus solves a lot of problems. The complexity, however, makes it very slow. A summation of the steps mentioned above gives that for a single match we have to use:

- 1+1 2D Fourier transforms
- 2 2D matchings
- 2+1 Re-samplings

This, at least today, is far too many operations for a real time system.
Part III

Results
Chapter 5

Rotation Robustness

In this chapter different methods are tested for their robustness against rotation. The idea was to create a simulation of a case where the only disturbance was rotation.

The images used for this tests are presented in Figure 5.1 and 5.2 below. The images are the two first images in image sequences from test flights with the BITAS helicopter. The image time stamp difference is 0.04 s (50 Hz interlaced) and the cars in the sequences drive slowly straight ahead in the beginning of the sequence, so the only difference between the images are the translation caused by the small movement of the car, or the camera, and the camera noise.

![Test images 1a and 1b](image)

Figure 5.1. Test images 1a and 1b used in the rotation robustness tests.
Figure 5.2. Test images 2a and b to 5a and b used in the rotation robustness tests.
5.1 PAIRS

This section describes tests performed on the pairs algorithm introduced in Section 3.1. The test setup and the test results are discussed.

5.1.1 Test Setup

In this test a template is taken from test image 1a and is matched with a region of test image 1b. As seen in Table 5.1 the template size is $17 \times 17$ pixels and the size of the image region is $41 \times 41$ pixels. The template is used both as is and in a rotated version. The rotation of the template is performed with linear interpolation and the rotation angle is $\pi/16$ rad. Test image 1b is rotated using bicubic interpolation in 401 steps between $-45$ and $45$ degrees with respect to the template rotation angle of the two templates. Then the rotated and non-rotated template are matched with a region of the rotated versions of test image 1b, and a certainty measure and a hit distance is recorded. As certainty measure the number of zeros in the match divided by the number of coordinate pairs, as mentioned in Section 3.1, is used. The hit distance is simply the distance between the position of the best match and the correct position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of template</td>
<td>$17 \times 17$ pixels</td>
</tr>
<tr>
<td>Size of image region</td>
<td>$41 \times 41$ pixels</td>
</tr>
<tr>
<td>Template rotation angles</td>
<td>$0$ and $\pi/16$ rad</td>
</tr>
<tr>
<td>Template interpolation method</td>
<td>Linear</td>
</tr>
<tr>
<td>Image rotation interval</td>
<td>$-45^\circ$ to $45^\circ$</td>
</tr>
<tr>
<td>Image interpolation method</td>
<td>Bicubic</td>
</tr>
<tr>
<td>Number of steps in rotation</td>
<td>401</td>
</tr>
</tbody>
</table>
5.1.2 Test Results

When testing the PAIRS algorithm a number of problems were discovered.

1. The certainty measure for the PAIRS algorithm is more or less useless. If there is a hit close to the correct position the certainty is quite high which is good. If there however is a really bad hit the certainty measure sometimes is as high as for a good hit.

2. The PAIRS algorithm has problems when trying to find a match, quite often a number of matches are discovered with a difference of just a few pixels. This is probably due to the fact that there is a very hard quantization of the similarity measure. Sometimes, however, hits are discovered in 2–8 of the neighboring positions but not in the position itself which is, put simply, very bad.

3. The selection of coordinate pairs with the randomizing function makes the algorithm very unstable. This has the result that the performance may change from time to time.

Due to these problems a number of extra steps were added to the test. To avoid the erratic behavior induced by the selection of the coordinate pairs the same test is performed 100 times, and the results are averaged. The problem with the bad certainty measure is partly solved by using a hit rate measure. A hit distance closer than two pixels away from the correct position is considered a hit, and the hit rate is defined as the average number of hits. This is however only possible if the correct position in test image 1b is well known. As for the problem of finding a match, the match positions are simply averaged to get a unique match. The test results can be found in Figure 5.3. The figures show the hit rate, and an averaged version of the hit rate, plotted against the rotation angle of the image relative to the rotation angle of the template, in the two test cases, i.e., $\phi = 0$ and $\phi = \pi/16$, (Figure 5.3(a) and 5.3(b)). They also show a comparison between the averaged hit-rates in the two cases (Figure 5.3(c)), and a comparison between the non-averaged version, in the case where the template rotation angle is zero, and a Gauss function (Figure 5.3(d)). The averaging kernel is Gaussian, with $\sigma = 1/4$, sampled so that 41 samples fit in the interval $[-1,1]$.

As for the interpretation of these results some things can be said. The PAIRS algorithm clearly performs as well on a linear interpolated template as on a non-rotated template. This means that if the rotation angle of a car is known it is easy to find it by using linear interpolation of the template. This is, however, useless information since it is, because of the poor certainty measure, not possible to find out what rotation angle the template really has. As noted before the PAIRS algorithm also suffers from erratic behavior which makes the situation even worse.
Figure 5.3. Hit rates for the PAIRS algorithm on rotated images.
5.2 Sum of Squared Differences

As for the pairs case, the SSD algorithm was tested for rotation robustness. The SSD algorithm is described in Section 3.3. These tests, and their results, are described and discussed here.

5.2.1 Test Setup

In this test image pairs 1 to 4 are used. As in the pairs case, templates are taken from test image 1a to 4a and matched with a region of test image 1b to 4b. However, this time five templates are chosen for each sequence generating twenty templates. This is shown in Table 5.2. It also shows that the template size is still 17×17 pixels and the size of the image region is 41×41 pixels. The template is used both as is and in a rotated version. The rotation of the template is performed with linear interpolation and the rotation angle is π/16 rad. Test image 1b to 4b are rotated using bicubic interpolation in 601 steps between −45 and 45 degrees with respect to the rotation angle of the two templates. As before a certainty measure and a hit distance is recorded. The certainty measure is 1 − average difference where the average difference is the square root of the dissimilarity measure divided by the number of pixels in the template. The images used are normalized so that their intensity values fits into the interval [0, 1], which means that the certainty measure also fits in the same interval. As with the pairs test the hit distance is the difference between the correct position and the estimated position.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of templates</td>
<td>5</td>
</tr>
<tr>
<td>Number of test sequences</td>
<td>4</td>
</tr>
<tr>
<td>Size of templates</td>
<td>17 × 17 pixels</td>
</tr>
<tr>
<td>Size of image regions</td>
<td>41 × 41 pixels</td>
</tr>
<tr>
<td>Template rotation angles</td>
<td>0 and π/16 rad</td>
</tr>
<tr>
<td>Template interpolation method</td>
<td>Linear</td>
</tr>
<tr>
<td>Image rotation interval</td>
<td>−45° to 45°</td>
</tr>
<tr>
<td>Image interpolation method</td>
<td>Bicubic</td>
</tr>
<tr>
<td>Number of steps in rotation</td>
<td>601</td>
</tr>
</tbody>
</table>
5.2 Sum of Squared Differences

5.2.2 Test Results

The results of this test are shown in Figure 5.4 and Figure 5.5 below. Figure 5.4 shows how SSD performs for a single template, and in Figure 5.5 all templates are shown together.

![Figure 5.4. Certainty and hit distance for SSD algorithm on one of the templates.](image)

(a) Certainties when $\phi = 0$.

(b) Hit distances when $\phi = 0$.

(c) Averaged certainties when $\phi = 0$ and $\pi/16$.

(d) Averaged hit distances when $\phi = 0$ and $\pi/16$.

When looking at a single template some interesting results can be noted. Figure 5.4(c) and Figure 5.4(d) shows that the SSD algorithm, as the PAIRS algorithm, performs as well on a linear interpolated template as on a non-rotated. Even if the certainty measure, and the hit distance, is noisy, as shown in Figure 5.4(a) and Figure 5.4(b), it shows that there is a strong correlation between the hit distance and the certainty measure, at least close to the true value. This means that it is possible to, for this single template, decide on a certainty measure threshold level, and use this to decide if something has happened. If the certainty measure is below the threshold the result is unlikely a good hit. It is then possible to try a linear interpolated version of the template and, if we do not have occlusion or some other phenomena, get a better certainty measure and thus a closer hit.
When studying results from more templates, as in Figure 5.5, it can be concluded that there still exists some sort of correlation between hit distances and certainty measure. However, it also shows that the certainty measure curves do not reach any given threshold level simultaneously. This means that it is not possible to decide on a global certainty measure threshold level. If this is done there will clearly be either some curves that never reach necessary certainty or some curves that allow way too much rotation to get a good hit distance. If there was a way to tell which kind of curve a template would follow without looking in to the future this could be used to set a template specific certainty threshold. This, however, is an unsolved problem.
5.3 Extended Lucas-Kanade Tracking

Finally the Extended Lucas-Kanade Tracking (ELKT) algorithm, which is described in Section 4.1, was tested for rotation robustness. In this section these tests are discussed.

5.3.1 Test Setup

This test had a slightly different setup compared to the pairs and SSD tests. Instead of, as before, rotating the templates looking at hit rate or dissimilarity measures, the rotation was calculated. First templates from test image 1a to 5a were matched with test image 1b to 5b using bicubic interpolation. Then test image 1b to 5b were re-sampled using bicubic interpolation, and templates were cut out. These new templates were thus as alike the ones from test image 1a to 5a as possible. Finally the templates from test image 1b to 5b were rotated, using bicubic interpolation, with 181 different angles between $-45$ and 45 degrees. The algorithm then had to try and estimate the rotation angle using linear interpolation. The procedure is illustrated in Figure 5.6 below.

![Figure 5.6](image)

*Figure 5.6.* Illustration of the test performed on the ELK algorithm.
5.3.2 Test Results

As for the results of this test some things can be concluded. In Figure 5.7(a) the estimated angles are plotted against the correct angles. Most of the lines in the graph never leave the line $y = x$, which shows that the algorithm, for most templates, was able to estimate correct values for rotation as high as about $\pm 45^\circ$. In one case it failed completely and in some cases it was able to estimate quite large angles before failing. Looking at Figure 5.7(b) through 5.7(d) it is possible to see that when failing, dissimilarity measure and difference in norm, $\|D_{Est} - D_{Real}\|$, goes quite high. The conclusion is thus that it would, in a tracking situation, be possible to detect this failure by looking at the dissimilarity measure. Another interesting feature is that when failing for small angles, which would normally be the case when failing in a tracking situation, the algorithm tries to re-scale the second test image into a line. This leads to very high values in $D_{Est}$, and can thus be detected. When failing for high rotation angles the failure is probably due to rotational symmetry of some sort, and when failing for small angles something has happened to the template itself. The template can be occluded, or it has over time just been affected by some non-affine transformation as discussed in Section 2.1. Trying to match an occluded, or non-affine transformed template with an affine transformed region will not only result in poor performance, but will thus lead the algorithm completely off track which could be a good thing since it is easy to detect.

![Graphs showing test results](image)

(a) Estimated angles for all templates.
(b) Estimated angle for one template.
(c) Dissimilarity measure for one template.
(d) $\|D_{Est} - D_{Real}\|$ for one template.

**Figure 5.7.** In most cases the ELK algorithm is able to estimate the rotation angle. When failing, however, this can be detected.
Chapter 6

Tracking

In this chapter some tests of real tracking are presented. Firstly tracking on computer generated data is discussed, and then authentic test sequences from test flights with the WITAS helicopter. The tracking methods used are based on the Extended Lucas-Kanade Tracking (ELKT) algorithm described in Section 4.1.

6.1 Computer Generated Data

To be able to evaluate the Extended Lucas-Kanade algorithm, and get a basic idea of its performance, it was tested on computer generated data where the demands of the ELKT algorithm were fulfilled, i.e., the car was only affected by affine transformations. In the WITAS project some computer generated test sequences are available and one of them was used for this test. The first image in the test sequence is shown in Figure 6.1(a) together with the path of the car being tracked. In this test templates of size $17 \times 17$ and $25 \times 25$ pixels are used. To make the algorithm more robust, and stable, some tests are performed during each iteration. These tests are supposed to give the algorithm enough information, to be able to tell whether it has succeeded, or if more advanced methods are necessary. The algorithm has to yield dissimilarity measures lower than three times the average dissimilarity, as well as reasonable values of $\|D_{k+1} - D_k\|$ and $|d_{k+1} - d_k|$ to be considered successful. The use of “reasonable values” aims for values that are not obviously wrong and the choice made here is not very elaborate. The demands on $\|D_{k+1} - D_k\|$ and $|d_{k+1} - d_k|$ are thus

$$\|D_{k+1} - D_k\| < 1 \quad \text{and} \quad |d_{k+1} - d_k| < \text{Template size}.$$ 

If the algorithm fails in its first attempt, another attempt is made with a low-pass filtered version of the template and the image, as suggested in Section 4.1.4. The low-pass filter kernel used is a two-dimensional version of $[1 \quad 2 \quad 1] / 4$. After the low-pass filter step the algorithm returns to the original image and tries again using the result from the low-pass step as initial parameter estimate. If this also fails the image is low-pass filtered twice and another attempt is made, this time on
the two low-pass filtered images and the original, updating the parameter estimate after each step. In a final attempt the algorithm tries a template of size $25 \times 25$ pixels. The parameters for the tracking test are summarized in Table 6.1, and the steps of the tracking algorithm are listed below.

The different steps of the tracking algorithm:

1. Run the translation only, and then the full version of the ELKT algorithm on the original image using the smaller template size. If successful, jump to 8.
2. Run the translation only version of the ELKT algorithm on a low-passed image using the small template size.
3. Run the translation only, and then the full version of the ELKT algorithm on the original image again, using the result from 2 as initial parameter estimate. If successful, jump to 8.
4. Run the translation only version of the ELKT algorithm on a twice low-pass filtered image using the small template size.
5. Run the translation only version of the ELKT algorithm on the single low-passed image again, using the result from 4 as initial parameter estimate.
6. Run the translation only, followed by the full version of the ELKT algorithm on the original image again, using the result from the 5 as initial parameter estimate. If successful, jump to 8.
7. Run the translation only, followed by the full version of the ELKT algorithm on the original image with the larger template size.
8. Update positions and start over with a new image.

Table 6.1. Parameters for the tracking algorithm used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template sizes</td>
<td>$17 \times 17$ and $25 \times 25$ pixels</td>
</tr>
<tr>
<td>One dimensional low-pass kernel</td>
<td>$[1 \ 2 \ 1]/4$</td>
</tr>
<tr>
<td>Requirement for $|D_{k+1} - D_k|$</td>
<td>$&lt; 1$</td>
</tr>
<tr>
<td>Requirement for $</td>
<td>d_{k+1} - d_k</td>
</tr>
</tbody>
</table>

In the image sequence the car is visible during 190 frames and the algorithm is able to track it from the beginning to the end. Some things deserves, however, to be mentioned about the tracking. Until frame 163 everything works without altering the size of the template. During the turn the algorithm tries to match the front of the car with the roof which is detected and fixed with one or two low-pass filterings. In frame 163 however the algorithm fails to find the car without changing the template size. In Figure 6.1 below the dissimilarity and $\|D_{Est}\|$ together with some images for the algorithm are presented. As seen in the figures the algorithm re-scales the current image to a line when trying to match the
image and the template. This failure can be detected both by looking at the dissimilarity measure and by observing that \( \|D_{k+1} - D_k\| \geq 1 \). \( \|D_{k+1} - D_k\| \) can be obtained by convolution of \( \|D_{Est}\| \) with \([1 \ -1]\). Figure 6.2 shows the same images as Figure 6.1(b) but with the larger template size. After frame 163 the algorithm has to use the larger template size for every frame to succeed. If the algorithm only fails in one single frame, or a couple of frames, the reason for the failure could be that the car is close to the edge of the road in these frames, or that the surrounding road has some strange feature there. When failing for all frames like it does, the reason has to be that after frame 163 the car can no longer be described as transformed by an affine transformation, and this does not affect a larger template as severely since it holds more information. The template size can not be much larger than the car itself and if the algorithm stops working using the largest possible template size a new template has to be chosen. Preferably, when the position of the car is well known, i.e., the last frame the algorithm succeeded for. The WITAS-system is, however, able to find the car again using a classification algorithm and the only reason not to use such a classification algorithm for tracking is that it would be computationally cumbersome. This means that the algorithm may loose the car as long as the loss is detected and reported.
(a) Path of the car in the test sequence. (b) Moment of failure. Estimated transformation applied on the current image before and after algorithm (upper left and upper right). Original template (lower right) as well as a close up view of the car in its position (lower left).

(c) Dissimilarity goes high on failure. (d) $\|D_{E,t}\|$ goes high on failure.

Figure 6.1. Performance of ELKT on computer generated data.
6.1 Computer Generated Data

(a) Estimated transformation applied to current image before algorithm.
(b) Estimated transformation applied to current image after algorithm.
(c) Close up view of car in its position.
(d) Original template.

Figure 6.2. Performance of ELKT on computer generated data.
6.2 Authentic Test Sequences

In this section tests performed on the ELKT algorithm with authentic test-sequences are discussed. The test sequences for these tests are the same as the ones used in Chapter 5, where only the first two images of the sequences were used. First, attempts using five templates chosen as discussed in Section 4.1.3 were made. The idea was that it was possible to track these different templates and calculate some average transformation. During the initial testing, however, it became evident that only tracking these features, corresponding to templates chosen as suggested, was difficult. Instead, another approach was tested where the template covered the whole car. Now the idea was that when tracking features, e.g., corners, the algorithm could slip off because there are so many other structures that looks like the feature. Since the algorithm is free to transform the original images with all kinds of affine transformations, it can easily find another target that is similar enough after transforming it. When tracking the whole car, however, it is less likely to find other “car-looking” objects in the image. The following sections present the results from these two types of tracking.

6.2.1 Feature Tracking

As mentioned above, tracking of features, i.e., templates chosen as suggested in Section 4.1.3, seems difficult. The problem appears to be that there just is not enough information in a single feature to make tracking robust. It is probably possible to use multiple templates, combining the estimated parameters to some better estimate. This could, however, require advanced filtering techniques to work. To start with, some basic tests using multiple independent templates in a sequence were performed. In these tests five templates are chosen and tracked individually until it becomes evident that they are lost. The algorithm used is the same as in Section 6.1, but with the addition that if the dissimilarity is larger after using the full affine ELKT algorithm than after using the translation only ELKT algorithm the translation only result is used. In Figure 6.3 below the initial template positions and the path of the car for one test case is shown. In Figure 6.4 the results from the same test case are presented.

Most templates for this test case can only be tracked for a small number of frames before the algorithm loses them. The template covering the upper right corner of the car, is even reported as matching the corner of the shadow a couple of times. None of the templates lasts until after the turn but the template covering the rear window of the car lasts until just before the turn. The low performance may be due to the nature of the data used. In Section 2.1 it is recommended that the camera is far away from the object and that the image plane of the camera is parallel to the surface being filmed. In the test sequences used for the testing, however, the camera is not very far away from the object and the tilting of the camera are for most sequences substantial. This has the effect that trying to track features on a such a car is very difficult since the movements of the car, or the camera, can make the feature transform in a non-affine way. Another problem is that the different features of the car are subjected to different non-
affine transformations. Trying to estimate some average transformation could, in a situation like this, be really difficult, and the usefulness of such an average is questionable. Another problem with tracking such features is that, in a case where the camera is as far away as recommended in Section 2.1, all features become very small. The need for tracking features on a car can thus be questioned and the conclusion is thus, that the entire car should be tracked with a single template.
(a) First template is lost after 17 frames. (b) Dissimilarity measure of the template that is lost after 17 frames.

(c) Second template is lost after 21 frames. (d) Dissimilarity measure of the template that is lost after 21 frames.

(e) Third template is lost after 26 frames. (f) Dissimilarity measure of the template that is lost after 26 frames.

(g) Forth template is lost after 52 frames. (h) Dissimilarity measure of the template that is lost after 52 frames.

(i) Last template is lost after 248 frames. (j) Dissimilarity measure of the template that is lost after 248 frames.

Figure 6.4. Tracking performance for one of the test sequences. The crosses shows the estimated positions for the different templates. The dotted crosses are the ones that are lost in that frame. The average dissimilarities are the sum of all the dissimilarities divided by the number of frames.
6.2 Authentic Test Sequences

6.2.2 Tracking the Entire Car

Tracking features of a car can be questioned. Instead attempts with templates covering the entire car were made. In these tests only one attempt with the translation only ELKT and full affine ELKT algorithm are made. No failure detection is used, and hence no low-pass filtering. Since no failure detection is used the algorithm looses the car before the end of the sequence in most cases. The reason not to use failure detection is that it is interesting to see how the algorithm handles a situation where the car is almost lost. The answer is that when the initial estimate is somewhat good, and the template matches the present car using only affine transformations, the algorithm will succeed. If the algorithm looses the car completely, or starts tracking some other “car looking” objects related to the car, e.g., the shadow or the front of the car, it will, in most cases, not be able to find its way back. It will, however, hang on to the car as long as the part of the car, that is mistaken for the entire car, does not change its appearance. This can be useful because it will give the tracking system time to prepare a new template for tracking. If, however, the dissimilarity measure is used for failure detection this can cause problems. As suggested before, the dissimilarity measure can be averaged and then compared to the current dissimilarity measure to detect a failure, but if the dissimilarity is allowed to be high for a long period of time this will prevent the algorithm from detecting a total failure. This means that the algorithm will be able to track the car for a long period of time after the dissimilarity has started to raise, but that letting it do so will make it harder to find out when the car is completely lost. The natural solution is to stop calculating a new average after each frame as soon as the algorithm detects partial loss of the car. The test was performed on all five sequences used in Chapter 5. In Figure 6.5 the first image in each sequence, and the templates chosen for this test, are presented. The results from these tracking tests are discussed in the following sections.
Figure 6.5. Templates for the different sequences.
Test Sequence 1

For Sequence 1 the algorithm manages to follow the car almost to the end. The template used, and the first image in the sequence, are shown in Figure 6.5(a) and 6.5(b). The dissimilarity measure during the tracking is shown in Figure 6.6 together with the path of the car in this sequence. As mentioned before the algorithm manages to track the car throughout most of the sequence. During the turn, however, the car transforms in a non-affine way which causes problems for the algorithm. Until frame 270 the car is seen from behind and above. After frame 270 the car is more and more seen from the side and above. This is why the dissimilarity raises after frame 270 in Figure 6.6(b). The result is that the algorithm re-scales the images so that the template matches the lower parts of the side of the car instead of the roof. Near frame 320 the algorithm has rotated the template 90° and near frame 370 180°. Near frame 432 the car is completely lost and the algorithm starts tracking the grass on the side of the road. When looking in Figure 6.6(b) the start of the degradation, in frame 270, can be detected but the complete loss of the car in frame 432 is harder to detect. In this case it would be wise to change template somewhere near frame 270. To make the algorithm work better, data that better fulfills the demands in Section 2.1 should be used.

![Path of the car in Sequence 1](image1)

![Dissimilarity measure during tracking for Sequence 1](image2)

**Figure 6.6.** Path of car and dissimilarity measure during tracking for Sequence 1. The car starts turning somewhere near frame 200.
Test Sequence 2

The first image in Sequence 2 is shown in Figure 6.5(c). For Sequence 2 the algorithm manages to follow the car through the turn and into the parking lot. Near frame 180, however, the landing gear from the helicopter makes the algorithm loose track of the car. After that the algorithm never recovers, but the car is tracked for about 50 more frames before the algorithm looses it completely. If some occlusion detection algorithm, that prevents the algorithm from trying to match the edge of the landing gear with the edge of the car, was used, the performance would probably improve for Sequence 2. Another improvement could be to prevent the parameters to change rapidly by using some sort of filter for the initial estimate. The dissimilarity measure during tracking in Sequence 2 is shown, together with the path of the car, in Figure 6.7 below.

![Path of the car in Sequence 2](image1)

![Dissimilarity measure during tracking for Sequence 2](image2)

Figure 6.7. Path of car and dissimilarity measure during tracking for Sequence 2.
Test Sequence 3

The car in Sequence 3 travels in a straight line and passes three crossings, as seen in Figure 6.8. The car passes the first crossing during frame 120 through 340 and the second during frame 590 through 680. The template for this sequence is shown in Figure 6.5(f). The figure shows that the template contains some of the roadside and this has some effects on the tracking. When the car passes the two first crossings the grass from the roadside, and the edge between the road and the roadside, is no longer present. This can have the effect that tracking performance decreases when the car passes the crossings. This theory is supported by the fact that the dissimilarity measure has higher values for the frames where the car is passing the crossings. The reason for this is the way the ELKT algorithm works. The ELKT algorithm uses high weights for areas with high derivatives, i.e., edges. Another problem is that the ELKT algorithm uses higher weights in the outer parts of the template than the inner parts, for the non-translation parameters. The effect is that the edge in the roadside gets high weights because it is an edge and even higher weights because it is in the outer parts of the template. The solution for these problems would be to use a mask that prevents the algorithm from putting large weights on features originating from other objects than the car.

Figure 6.8. Path of car and dissimilarity measure during tracking in Sequence 3.
Test Sequence 4

The first image of this sequence is shown in Figure 6.5(g), and the template used for this sequence is shown in Figure 6.5(h). As seen in the figure this template suffers from the same problems as the template for test Sequence 3. The first problem is the line in the road just above the left corner of the car. Secondly there is the edge on the side of the road. In the sequence the car continuously passes the lines in the middle of the road. During the tracking of the car it can be noted that the algorithm tries to match the lines passing the car with the line in the template. This has the effect that near frame 180 the algorithm starts tracking the front of the car instead of the entire car. The image region is stretched in the direction of the car so that the entire car fits the front of the car. At frame 220 a manual adjustment was made to the tracking. The parameters were adjusted so that the algorithm itself could find the correct parameters. The improvement can be noted in Figure 6.9(b). When using this adjustment the algorithm manages to track the car almost to the end of the sequence. The path of the car in Sequence 4 can be seen in Figure 6.9(a).

Figure 6.9. Path of car and dissimilarity measure during tracking in Sequence 4.

(a) Path of the car in Sequence 4  (b) Dissimilarity measure during tracking for Sequence 4.
Test Sequence 5

For Sequence 5 the tracker manages to track the car almost to the end of the sequence. The problem appear to be the shadow of the car which can be seen in Figure 6.5(i) or 6.5(j). Somewhere near frame 110 the algorithm starts to track the shadow instead of the car. Near frame 200, when the driver of the car is driving “off road” as seen in Figure 6.10(a), the algorithm starts to track the side of the road instead of the shadow of the car. The dissimilarity measure during the tracking is shown in Figure 6.10(b). As in Sequence 3 and 4 the problem appear to be features outside the car and the conclusion is thus that some kind of mask should be used.

![Path of the car in Sequence 5](image)

![Dissimilarity measure during tracking for Sequence 5](image)

Figure 6.10. Path of car and dissimilarity measure during tracking in Sequence 5.
Chapter 7

Final Algorithm and Implementation

The final algorithm and its implementation in *Image Processing Application Programming Interface* (IPAPI) is discussed here. The final algorithm is based on the ELKT algorithm introduced in Chapter 4.

7.1 IPAPI

As shown in Figure 7.1 IPAPI is a client-server system where a number of application programs connect to an IPAPI server. Each application program sends requests to the IPAPI server to build an application graph, consisting of nodes. It then sends a request to execute it. All computations are thus performed by the IPAPI server. The execution is performed by executing the nodes one by one in a predetermined order. Each node has a number of I/O ports which can be connected with I/O ports of other nodes, or with the application program. When executing the node, data is read from the Inports, and the result is written to the Outports. The most important type of port is the buffer which has the ability to store 2D images. Before the first execution of a graph, memory for the buffers must be allocated. This task is performed by an allocation manager which first propagates the buffer sizes through the graph, and then allocates memory. The allocation manager also propagates loop parameters through the graph. The loop parameters are used as start and stop conditions.

![Figure 7.1. The relationship between the IPAPI server and the IPAPI application.](image-url)
when looping over the 2D buffer data. The use of loop parameters has the effect that no more data is used than absolutely necessary.

The iPAPI system thus solves the problem with start/stop indices, and sizes which makes life easier for a node programmer. Another great benefit of the iPAPI system is that it is easy to connect different imaging operations, thus easily creating all sorts of different compound imaging operations.

As shown in Figure 7.2 the iPAPI system is a subsystem controlled by a main system. The iPAPI system is given a task by the main system and then reports back to it via a object handling database called DOR (Dynamic Object Repository). The DOR can be used by the main system and by iPAPI as well as other subsystems. In this way information about interesting objects localized by the vision sensors can be made available to the whole system, thus relieving the main system. Objects in the DOR can be updated by any subsystem so that the DOR always contains all the current knowledge about the situation on the ground. The different parts, i.e., subsystems, of the WITAS system can easily be replaced by simulations so that it is possible to test the different parts of the system independently.

Altogether, this gives iPAPI the possibility to get images from a, authentic or simulated, vision sensor system that is connected to the WITAS system. A more thorough description of iPAPI can be found in [12].
7.2 Algorithm and Testing

The different steps of the Extended Lucas-Kanade Tracking (ELKT) algorithm, from Section 4.1 is repeated here:

1. Chose a template of the object that is going to be tracked, and set $D_0 = [0]$ and $d_0 = 0$.
2. Calculate the derivatives of the template.
3. Calculate $\Phi$ and $\nabla J$, and then $(\Phi \cdot \Phi^T)^{-1}\Phi$ and $(\nabla J \cdot \nabla J^T)^{-1}\nabla J$.
4. Use $D_0$ and $d_0$, i.e., $\mu_0$, to calculate a template estimate from the latest image, and use this to calculate the error image $E^T$.
5. If needed, calculate $(\Pi^T)^{-1}$.
6. Calculate $\mu'$ or just $d'$ using
   \[
   \mu' = (\Pi^T)^{-1} \cdot (\Phi \cdot \Phi^T)^{-1} \cdot \Phi \cdot E^T \quad \text{or} \quad d' = D_0 \cdot (\nabla J \cdot \nabla J^T)^{-1} \cdot \nabla J \cdot E^T
   \]
7. Update $\mu$ with $\mu'$ or $d'$ and jump to 4.

These steps are used in a somewhat more advanced way in the final algorithm below:

1. Prepare data as suggested in 1-3 above.
2. Run the iterative steps of the translation only version of the ELKT algorithm on the original image.
3. Run the full affine version of the ELKT iteration step and chose the results with the lowest dissimilarity.
4. Compare the dissimilarity measure with the average dissimilarity and check if $|d_{k+1} - d_k|$ and $|D_{k+1} - D_k|$ have reasonable values, i.e., are not too big, e.g., as in Section 6.1. If everything is well the algorithm has succeeded hence skip to 8.
5. Try translation only ELKT iteration on low-pass filtered version of the template and image to get a better initial estimate for 2.
6. Possibly repeat low-pass filtering procedure more than once.
7. If all low-pass filterings fail, select a new template.
8. Update positions and start over with a new image at 2.
During the initial testing the ELKT algorithm was implemented in MATLAB®. The reason for this is that MATLAB® is very easy to use. The efficiency of that implementation is however not very good. The reason is that the interpolation functions in MATLAB® has no optimization for affine transformations, and that all mesh data has to be pre-generated, which takes a lot of time. During a profile on the MATLAB® implementation it was noted that most time was spent on reading images from disk and on the function meshgrid, which is used by the interpolation functions. Since IPAPI is more complex to use IPAPI was not used for the initial testing of the algorithm. The final version of the algorithm has however been implemented and tested in IPAPI, but no speed test has been performed. The reason for this is, as mentioned in Chapter 8, that such a speed test would not be useful at this point. However, when simply running the ELKT algorithm and the PAIRS algorithm and comparing the results ELKT performs much better. It is somewhat faster and it manages to track a template much longer than PAIRS algorithm before updating it. The sequences tested are the same as in Section 6.2.2.
Chapter 8

Conclusions and Future Work

In this chapter the final results of the thesis are presented. First the final algorithm is described and discussed. Then conclusions are drawn and finally some suggestions about future work are included.

8.1 Current Status

The final algorithm implemented in *Image Processing Application Programming Interface* (IPAPI) is described in Section 7.2 and works as follows:

1. Prepare data for the algorithm.
2. Run the translation only version of the *Extended Lucas-Kanade Tracking* (ELKT) algorithm.
3. Run the full affine version of the ELKT algorithm and choose the results with the lowest dissimilarity measure.
4. Compare the dissimilarity measure with the average dissimilarity and check if $|d_{k+1} - d_k|$ and $|D_{k+1} - D_k|$ have reasonable values, i.e., are not too big, e.g., as in Section 6.1. If everything is well the algorithm has succeeded hence skip to 8.
5. Try the translation only version of the ELKT algorithm on a low-pass filtered version of the template and image to get a better initial estimate for 2.
6. Possibly repeat low-pass filtering procedure more than once.
7. If all low-pass filterings fail, select a new template.
8. Update positions and start over with the next frame.
Conclusions and Future Work

If the data for the algorithm does not fulfill the demands from Section 2.1 a new template should be selected after a pre-decided number of frames to avoid losing the car. The template should cover the entire car and at the same time not include too many features from the road, or roadside.

Given that the template does not contain features originating from outside the car and the demands in Section 2.1 are somewhat fulfilled the algorithm suggested will work. The algorithm is implemented in ipapi, but no studies of how fast the algorithm is have been performed. Firstly little effort has been spent to optimize the ipapi code, and a speed test of non-optimized code says little about the speed of the algorithm. Secondly when simulating the camera with image sequences from a computer disk most of the time used for tracking is spent reading images from disk. Thirdly the ipapi implementation of the pairs algorithm is not optimized according to the suggestions in [4]. Comparing unoptimized implementations of elkt and pairs algorithms does not give much information on how fast they really are. Another problem is that the algorithm will probably be faster if the initial parameter estimates are good. Testing the algorithm as it is will thus not say much about the speed of an algorithm that uses some kind of prediction of the parameters. Since such improvements are suggested below, speed tests are also left for the future.

However, as mention in Section 7.2 the elkt algorithm has been compared with the pairs algorithm by simply running them both and comparing how often they change templates and how fast they seem. When doing this the elkt algorithm is somewhat faster than the pairs algorithm. The pairs algorithm has to change template quite often but the elkt algorithm does not change template more than a couple of times for every sequence. The sequences tested are the same as in Section 6.2.2.

8.2 Conclusions

The first conclusion is that using methods such as pairs or raw SSD, where an exhaustive search over a template region is performed, is not necessary. Such methods have problems finding rotated templates, and if an exhaustive search was to be performed for different template positions and template rotations with different angles, the computational burden substantially increases. Secondly the elkt algorithm works when used correctly. The templates should cover the entire car and at the same time not contain too many features from the road or other things not belonging to the car. This could be solved with a mask that tells the algorithm which parts of the error image, in the elkt algorithm, that are important. The data used for the tracking should fulfill the demands in Section 2.1. Finally, the elkt algorithm will probably work better and converge faster if the initial estimates of the parameters are good. Therefore some filtering technique such as Kalman filtering of the parameters should probably be used.
8.3 Future Work

The tracking method suggested in this thesis is far from complete. A number of improvements can probably be made. Some interesting topics are listed below:

- How do changes in illumination affect the algorithm?
- Is it possible to be certain that the algorithm is still tracking the car?
  - How does brightness affect the dissimilarity measure?
  - Which filtering technique is the best when looking at the dissimilarity measures?
  - Is there some way of telling if the dissimilarity is too high?
- Could Kalman filtering be used to improve the initial estimate?
- When using the full affine ELKT algorithm it sometimes get completely off track even if the initial estimate is good. Why is this, and is it possible to avoid?
- Sometimes templates cannot cover the entire car without covering features from the road, or other features not belonging to the car. Can this be solved with a mask that weights down these features, and improve the performance of the ELKT algorithm?
- How fast is the algorithm? Does it match the speed of exhaustive search methods?

During the end of this thesis two interesting master’s theses were discovered, i.e., [13] and [14]. In these theses a tracking method very similar to the ELKT tracking algorithm is used for tracking objects on the ground from airborne systems. Studying these theses is probably a good start for future work. Another interesting publication on the subject is [15] where an ELKT-like algorithm is discussed. Instead of only using the difference measure from SSD as in ELKT, the correlation measure in NCC is combined with the difference measure in SSD to create an alternative algorithm.
Bibliography


65


Appendix A

Notation

Used notation is listed below

A.1 Terminology

**Tracking** In this thesis the use of the word “tracking” can be misleading. Tracking is used where most people would use the term *Template matching*. In tracking theory some measure of position is performed before the actual tracking takes place. When using the word tracking this measurement of position is thus what is aimed at.

**Feature** A feature is, e.g., a point on a car with high information content. In most cases this corresponds to points with high derivatives.

**Aliasing** occurs when the sample frequency is to low. Aliasing is the reason that carriage wheels in old westerns sometimes moves backwards.

**Sampling** is the process of making a continuous signal discrete. The simplest form of sampling is to register the values of a continuous one dimensional signal, \( f(t) \), for different values of \( t \), i.e., \( t \in \{t_1, t_2, t_3, \ldots, t_n\} \)

**Samples** The values registered during the sampling, i.e., \( f(t_i) \), are called samples.

**Sample frequency** is defined as \( \frac{1}{\tau} \), where \( \tau \) is the period between two samples, i.e., \( \tau = t_{i+1} - t_i \).

**Kernel** A kernel is a description of a filter. The filter is applied using convolution between the signal and the kernel.

**Affine transformation** is a transformation that can be described with rotation, scaling and translation.

**Template** A template is an image that contain the object that is being tracked.
A.2 Acronyms

- CDD: Chen-Defrise-Deconick
- CVL: Computer Vision Laboratory
- DOR: Dynamic Object Repository
- ELKT: Extended Lucas-Kanade Tracking
- IPAPI: Image Processing Application Programming Interface
- IRB: Image Region Bitset
- NCC: Normalized Cross Correlation
- SSD: Sum of Squared Differences
- TB: Template Bitset
- WITAS: Wallenberg laboratory on Information Technology and Autonomous Systems

A.3 Operators and Functions

\( \nabla \)-Operator \( \nabla f(t) = \text{grad } f(t) \) is defined as \( \sum_i e_i \frac{\delta}{\delta t} f(t) \). Using this definition, the chain rule of differentiation, for \( h = f(g(t)) \), becomes

\[
\frac{\partial h}{\partial t} = \nabla t g \cdot \nabla g f = \begin{pmatrix}
\frac{\delta f_1}{\delta t_1} & \ldots & \frac{\delta f_1}{\delta t_1} \\
\vdots & \ddots & \vdots \\
\frac{\delta f_n}{\delta t_1} & \ldots & \frac{\delta f_n}{\delta t_1}
\end{pmatrix} \cdot \begin{pmatrix}
\frac{\delta f_1}{\delta g_1} & \ldots & \frac{\delta f_1}{\delta g_1} \\
\vdots & \ddots & \vdots \\
\frac{\delta f_n}{\delta g_1} & \ldots & \frac{\delta f_n}{\delta g_1}
\end{pmatrix},
\]

which is the notation used.

\( \text{sinc} \) is defined as: \( \text{sinc}(x) = \lim_{\xi \to x} \frac{\sin(\pi \xi)}{\pi \xi} \)
A.4 Variables

- $\xi$: Correct transformation function. $\xi = D\mathbf{x} + \mathbf{d}$.
- $\xi_0$: Initial estimate of transformation function.
- $\xi'$: Difference between initial estimate and correct transformation function.
- $D$: Deformation matrix of $\xi$.
- $D_0$: Initial estimate of deformation matrix.
- $D'$: Difference between correct deformation matrix and initial estimate.
- $d$: Translation in $\xi$.
- $d_0$: Initial estimate of translation.
- $d'$: Difference between correct translation and initial estimate.
- $J(x)$: Initial image obtained at $t = t_0$.
- $K(x)$: Image obtained at $t = t_0 + \tau$.
- $\varepsilon$: Difference measure.
- $\tilde{\varepsilon}$: Estimate of difference measure.

\[
\Pi = \begin{bmatrix} D_0^{-1} & 0 & 0 \\ 0 & D_0^{-1} & 0 \\ 0 & 0 & D_0^{-1} \end{bmatrix}^T.
\]

\[
\mathbf{X}(\mathbf{x}) = \begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix}^T.
\]

\[
W = \{x_1, \ldots, x_N\}.
\]

\[
\Phi(x) = \mathbf{X}(\mathbf{x}) \cdot \nabla_x J(\mathbf{x}).
\]

\[
\Phi = \begin{bmatrix} \Phi_1(x_1) & \ldots & \Phi_1(x_N) \\ \vdots & \ddots & \vdots \\ \Phi_0(x_1) & \ldots & \Phi_0(x_N) \end{bmatrix}.
\]

\[
E(x) = \left[ E(x_1) \ldots E(x_N) \right].
\]

\[
\nabla J = \begin{bmatrix} \frac{\partial J}{\partial x_1} & \ldots & \frac{\partial J}{\partial x_N} \\ \frac{\partial J}{\partial y_1} & \ldots & \frac{\partial J}{\partial y_N} \end{bmatrix}.
\]
På svenska

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera åktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnad som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet — or its possible replacement — for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Per Öberg