Compact Representations and Multi-cue Integration for Robotics

Robert Söderberg
Compact Representations and Multi-cue Integration for Robotics

© 2005 Robert Söderberg

Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
Sweden

ISBN 91-85299-36-7
I SS N 0280-7971
To Maria
Abstract

This thesis presents methods useful in a bin picking application, such as detection and representation of local features, pose estimation and multi-cue integration.

The scene tensor is a representation of multiple line or edge segments and was first introduced by Nordberg in [30]. A method for estimating scene tensors from gray-scale images is presented. The method is based on orientation tensors, where the scene tensor can be estimated by correlations of the elements in the orientation tensor with a number of 1D filters. Mechanisms for analyzing the scene tensor are described and an algorithm for detecting interest points and estimating feature parameters is presented. It is shown that the algorithm works on a wide spectrum of images with good result.

Representations that are invariant with respect to a set of transformations are useful in many applications, such as pose estimation, tracking and wide baseline stereo. The scene tensor itself is not invariant and three different methods for implementing an invariant representation based on the scene tensor is presented. One is based on a non-linear transformation of the scene tensor and is invariant to perspective transformations. Two versions of a tensor doublet is presented, which is based on a geometry of two interest points and is invariant to translation, rotation and scaling. The tensor doublet is used in a framework for view centered pose estimation of 3D objects. It is shown that the pose estimation algorithm has good performance even though the object is occluded and has a different scale compared to the training situation.

An industrial implementation of a bin picking application have to cope with several different types of objects. All pose estimation algorithms use some kind of model and there is yet no model that can cope with all kinds of situations and objects. This thesis presents a method for integrating cues from several pose estimation algorithms for increasing the system stability. It is also shown that the same framework can also be used for increasing the accuracy of the system by using cues from several views of the object. An extensive test with several different objects, lighting conditions and backgrounds shows that multi-cue integration makes the system more robust and increases the accuracy.

Finally, a system for bin picking is presented, built from the previous parts of this thesis. An eye in hand setup is used with a standard industrial robot arm. It is shown that the system works for real bin-picking situations with a positioning error below 1 mm and an orientation error below 1° for most of the different situations.
Acknowledgements

I’d like to thank several people for their help, input, feedback and for supporting me with motivation. This thesis could never have been written without the help from the following people.

Maria, for her love and encouragement, especially in those situations when I’ve lost my faith. Thank you, for being my best friend.

My family, for supporting me and trying to introduce some kind of sense into my mind.

All the people at the Computer Vision laboratory, for being so patient with me and my questions, for sharing their ideas and implementations with me, and for being such good friends.

Professor Gösta Granlund, who, first of all, gave me the opportunity to come to the Computer Vision Laboratory to work and study, and also introducing me to an interesting area of research.

Klas Nordberg for being a really good supervisor.

Fredrik Vilkstén, well… what can I say. It is always nice to have a really good friend that you can trust when going through the different states of a life; heaven and hell.

Anders Moe, for having patience with a restless soul like mine and always be ready to join me in a new adventure. For helping me when needed and for all the interesting discussions.

Björn Johansson for his willingness to spend hours and hours to explain and discuss different aspects of this topic.

Per-Erik Forssén for his willingness to share his knowledge in this field.

Johan Wiklund, for keeping the computers happy.

Erik Jonsson and Johan Skoglund for proof-reading the manuscript.

All the people in the VISATEC project.

The European tax payers, for basically funding my research within the VISATEC project.

The robot company Neuronics [29], for letting me use the robot image on the front cover.

And last but not least, I’d like to thank all my friends for their support and encouragement.
# Contents

1 Introduction
   1.1 Motivation ......................................................... 1
   1.2 Thesis overview ................................................... 2
   1.3 Contributions ................................................... 3
   1.4 Notations ....................................................... 4

2 Representation and Detection of Line and Edge Features .............................. 5
   2.1 Introduction ..................................................... 5
   2.2 The scene tensor representation ................................ 6
      2.2.1 The rank of the scene tensor ............................. 10
   2.3 How to estimate the scene tensor ............................... 14
   2.4 Analyzing the scene tensor .................................... 17
      2.4.1 The scene tensor norm .................................... 17
      2.4.2 The rank measurement .................................... 19
      2.4.3 Detection of interest points .............................. 21
      2.4.4 Extracting information from the scene tensor ......... 25
   2.5 A repeatability test ............................................ 29
      2.5.1 Introduction ................................................. 29
      2.5.2 Interest point detectors .................................. 30
      2.5.3 Experimental setup ....................................... 31
      2.5.4 Experimental results ..................................... 36
   2.6 Conclusions and discussions .................................. 42

3 Invariant Representations .................................................. 43
   3.1 Introduction ..................................................... 43
   3.2 Coefficients of the characteristic polynomial ................ 44
   3.3 Tensor doublet 1 ................................................. 45
   3.4 Tensor doublet 2 ................................................ 46
   3.5 Discussions and conclusions ................................... 50

4 Pose Estimation of Three-Dimensional Objects ...................................... 53
   4.1 Introduction ..................................................... 53
   4.2 Compact and invariant representation of local image data ...... 54
   4.3 Mapping from the representation to object state parameters .... 54
   4.4 Results ......................................................... 55
4.5 Conclusions and discussions ........................................... 58

5 Multi-cue Integration ....................................................... 59
  5.1 Introduction ............................................................... 59
  5.2 Mean-shift clustering .................................................... 59
  5.3 Multi-cue integration for robustness .............................. 60
  5.4 Multi-cue integration for accuracy ............................... 62

6 VISATEC Demonstrator .................................................. 69
  6.1 Introduction ............................................................... 70
  6.2 Assumptions .............................................................. 70
  6.3 System description ..................................................... 70
    6.3.1 Demonstrator hardware ......................................... 71
    6.3.2 Calibration ....................................................... 72
    6.3.3 Pose estimation algorithms .................................... 73
    6.3.4 How to train the system ....................................... 73
    6.3.5 Random movement in the reference plane ................. 75
    6.3.6 Purposeful movement ......................................... 76

7 Demonstrator Experiments ............................................. 79
  7.1 Introduction ............................................................... 79
  7.2 Experimental setup .................................................. 79
    7.2.1 Variables .......................................................... 79
    7.2.2 Parameters and training ....................................... 81
    7.2.3 Measured quantities ........................................... 81
  7.3 Results from experiments with ground truth ................. 84
    7.3.1 Guide to results ................................................ 84
    7.3.2 Conclusions ..................................................... 99
  7.4 Results from experiments without ground truth ............ 101

8 Summary ................................................................. 103
  8.1 Summary and discussions ........................................... 103
  8.2 Future work ........................................................... 104

Bibliography ............................................................... 105
Chapter 1

Introduction

1.1 Motivation

Industrial automation of today demands very dynamic automation systems, since the geometry of the products changes faster than before. As a consequence, old systems where the objects are placed in fixtures will not be sufficient in the future. Instead we need more advanced procedures that can find the pose of objects, guide the robot towards the objects and finally grasp them.

The work presented in this thesis has been performed within the VISATEC project [42]. The goal of this project is to develop a system useful for industrial automation and especially for the holy grail of automation; bin picking. The bin picking application requires an implementation of a cognitive vision system. In [16], it is stated that the purpose of a cognitive system is to produce a response to appropriate percepts. In the bin picking application, this response is a physical action, where input from the camera generates a movement of the robot arm. It is important that the representation of the visual percepts is more invariant and compact than the original iconic image, otherwise will the amount of information explode. It is also stated that a cognitive system has to be extendable, meaning that the system has to be able to acquire and store new information about the environment. In the case of bin picking, the environment can be new objects, but also a new type of manipulator added to the robot arm which extends the physical range for the system.

All these properties were in mind when the members of the VISATEC started to design the VISATEC demonstrator. The system has a classic hierarchical computing structure [15, 4], where computer vision algorithms were designed for different levels of vision; low-level, medium-level and high-level vision.

- Low-level vision usually contains representation and detection of low-level features, which are local and simple features such as line and edge segments, corners, crossings, curvature, etc. The main problem is to implement a stable detection process and find representations that are applicable to the different environments.
• Medium-level vision is about joining different low-level features to get a more descriptive feature that can uniquely describe the state of an object.

• High-level vision is harder to define. Some would say that it contains mechanisms for managing symbols useful for reasoning and decision making. We believe that high-level vision is a level where the system has an interpretation of the environment, e.g. the type objects it contains and the state of each object. This high-level information can then be used for executing a task, such as grasping an object.

This thesis will step by step describe the VISATEC demonstrator, where the journey will visit all three different levels of vision, although the visit at the higher level will be a little bit shorter.

1.2 Thesis overview

Chapter 2: The scene tensor is an representation of multiple line or edge segments. This results in a representation that is able to detect and distinguish between, e.g. line endings, corners, crossings, etc. This chapter starts with a theoretic description of the scene tensor, which was first presented by Nordberg in [30]. Methods for estimating scene tensors from gray-scale images and detection of interest points are described together with experimental results.

Chapter 3: Invariant representations are useful for many applications, such as pose estimation, wide baseline stereo and tracking. The scene tensor is not invariant itself and this chapter describes three different implementations of an invariant representation based on the scene tensor. The aspect of the different implementations are discussed and recommendations for different application areas are given.

Chapter 4: The goal for the VISATEC demonstrator is to grasp objects from a bin. One of the most essential parts of such a system is estimation of an objects pose. This chapter presents a statistical approach to pose estimation based on matching and clustering, where invariant representations of local features vote on the pose of the object.

Chapter 5: An industrial system that should cope with a large number of different objects can not be based on only one type of feature. In this chapter, we present an algorithm for fusing cues from several pose estimation algorithms and from several different views of the object. We call it multi-cue integration between algorithms and temporal multi-cue integration.

Chapter 6: As stated in section 1.1, the main purpose of the VISATEC project is to build a system useful for a bin picking task. This chapter is a description of the VISATEC demonstrator, where an eye-in-hand system is used together with an industrial robot. Visual servoing, based on the output from the multi-cue integration, is used for guiding the robot arm towards the object.
Chapter 7: The last chapter describes an extensive test of the VISATEC demonstrator, where the experimental setup is described and the results are presented and discussed. The experiments are both performed in a synthetic environment and in an industrial real environment.

1.3 Contributions

Chapter 2: The scene tensor representation was first introduced by Nordberg in [30]. This is mainly a theoretical presentation and it is always a big step from theory to a working implementation. The contributions are an efficient implementation of an estimation method and methods for analyzing the scene tensor, such as detecting points of interest and estimating feature parameters.

Chapter 3: The invariant representation in section 3.2 is based on the work in [30]. The two versions of tensor doublets are considered new, although some of the inspiration is from Granlund and Moe [17].

Chapter 4: The framework for pose estimation, based on matching and clustering is not new [23, 22, 11]. The combination of this framework and the tensor doublets is considered new.

Chapter 5: A lot of work has been done in the area of multi-cue integration. Most of the algorithms are based on integration of low-level features and the applications are usually tracking. We believe that multi-cue integration between several pose estimation algorithms and several views is new. The work in this chapter has been performed in collaboration with Fredrik Vikstén.

Chapter 6: Several parts of the VISATEC demonstrator are not new, such as visual servoing and camera calibration. However, putting it together to a complete system is considered new. The work in this chapter has been performed in collaboration with Fredrik Vikstén and the partners in the VISATEC project.

Chapter 7: It is not much to say about this chapter, since it describes experiments and results. However, such an extensive test of three dimensional pose estimation algorithms with ground truth has probably never been done. The work in this chapter has been performed in collaboration with Fredrik Vikstén and the partners in the VISATEC project.
1.4 Notations

The mathematical notations used in this thesis should resemble those most commonly used in the engineering community. There might anyway be some differences, and thus this section has been added to avoid confusion.

The following notations are used for mathematical entities:

- $s$: Scalars (lowercase letters in italics)
- $u$: Vectors (lowercase letters in boldface)
- $C$: Matrices (uppercase letters in boldface)
- $s(x)$: Functions (lowercase letters)

The following notations are used for mathematical operations:

- $A^T$: Matrix and vector transpose
- $\langle x \, | \, y \rangle$: The scalar product
- $|z|$: Absolute value of real or complex numbers
- $\|z\|$: Matrix or vector norm
- $(s \ast f_k)(x)$: Correlation
- $A \otimes B$: Tensor product
- $A \odot B$: Element-wise multiplication
- $\text{vec}(A)$: Conversion of a matrix to a vector by stacking the columns
Chapter 2

Representation and Detection of Line and Edge Features

2.1 Introduction

Detection of image features which are of higher complexity than lines or edges, such as corners and crossings, can be done by using so-called points of interest detectors [28, 1, 18, 37, 22]. The downside of these methods is that the spatial localization of the corresponding feature is not very accurate and they are not very selective in the sense that they are likely to be sensitive also to noise, point patterns, etc. Furthermore, even though some methods use various representations for describing what phenomenon is detected, these are not rich enough to distinguish between a corner and a T-junction, or describe the opening angle of a corner.

In [30, 32, 31, 8] the scene tensor representation is described, which is a representation of multiple line segments in terms of their orientations and positions within the local region. Representations of multiple orientations is not new [21, 6], but they usually lack information about position and extension of the line or edge segments in the local region. The scene tensor is a step further and contains a complete description of each segment in the region and the result is an representation of corners, crossings, etc., which both distinguishes between the different cases and describes the parameters of the segments, figure 2.1 and 2.2. This representation will be more selective than for example the Harris operator [18] and therefore be less sensitive to noise and point patterns in the sense of selecting interest points.

This chapter will start with a description of the scene tensor representation and section 2.3 will explain how to estimate the scene tensor from gray-scale images. Section 2.4 shows that the scene tensor can be used for detecting interest points and also for extracting parameters for different types of features, such as opening angle and orientation of a corner. Section 2.5 shows some experimental results, where a benchmark test is done for a couple of interest points detectors together with the scene tensor.
2.2 The scene tensor representation

How do we find a representation that is able to represent multiple line or edge segments? Well, in the spirit of engineering we first have to agree upon a list of specifications before we go into the mathematics.

As stated in the introduction, we seek a representation that can distinguish between corners, crossings and T-junctions, figure 2.1 and 2.2. This is only possible if each line or edge segment\(^1\) is represented by its orientation, position and extension. It is not enough with a representation of multiple orientations, since then it is impossible to distinguish between a corner and a crossing without further processing. It should also be based on some well known representation, where the orientation tensor \([15, 9, 3]\) is the obvious choice. It should also be easy to find out if the representation contains one, two or three segments. In the case of the orientation tensor this is done by estimating the rank of the tensor, where the rank is close to one if it is estimated from a local region containing only one segment and full rank if it contains two or more segments. This property is used by the well known Harris corner detector \([18]\). In the same manner, the scene tensor representation will have a rank that is equal to the number of segments it contains.

The difference from the 2D orientation tensor is that the possible ranks are not only rank one and full rank, but rank one, two and three. It should also be easy to add new segments to the representation, preferably by a linear operation such as superposition. The following item list summarizes the different specifications.

- The scene tensor representation should be able to represent multiple segments, where each segment is represented by its orientation, position and extension.

\(^1\)In the continuation we will refer a line or edge segment to just a segment.
2.2 The scene tensor representation

- It should be based on the orientation tensor, since years of research on estimating local orientation can be reused.
- The number of segments should be equal to the rank of the tensor.
- New segments should be added by superposition.

How do we find such a representation? Let’s start with the orientation tensor

\[ T = A \hat{n} \hat{n}^T \]  

(2.1)

where \( \hat{n} \) is the normal of the segment and \( A > 0 \) is related to the strength or contrast of the edge or line. It is clear that the orientation tensor will not fulfill the specifications in the list. In the case of two parallel lines, figure 2.2 to the right, the tensor for the upper segment \( T(x_1) \) and the lower segment \( T(x_2) \) will be linearly dependent, \( T(x_1) \propto T(x_2) \). \( x_1 \) is any point on the upper segment and \( x_2 \) is any point on the lower segment. Due to this dependency, the superposition \( S \) of the two tensors will therefore not be of rank two.

\[ S = T(x_1) + T(x_2) \]  

(2.2)

Except from the fact that the orientation tensor unambiguously only can represent one orientation, the main problem is that the orientation tensor contains only the orientation of the segments and not the position of the segments. If a coordinate system is introduced, then the orientation vector \( \hat{n} \) can be extended to a vector \( l \) as

\[ l = |l| \hat{n}, \quad |l| = x^T \hat{n} \]  

(2.3)

where \( x \) is the position of any point on the segment, figure 2.3. Observe that \( l \) both contains the orientation and the orthogonal distance from origo to the segment. A new tensor can then be formed:

\[ T' = All^T = |l|^2 A \hat{n} \hat{n}^T \]  

(2.4)

As seen in equation 2.4, this representation has the problem that the contrast of the segment \( A \) and the distance to segment \( |l| \) are mixed together and it is impossible to retrieve the distance again if \( A \) is unknown. The new tensor also still has the problem of being linearly dependent in the case of parallel lines. These problems can be solved by mapping the vector \( l \) to a projective space. A projective space is a space that is invariant to all linear homogeneous transformations and a scaling of a vector in a projective space will therefore not affect the vector itself: \( l \sim \alpha l \) where \( \alpha \neq 0 \in \mathbb{R} \). Such a mapping is performed by using a homogeneous representation of \( l \), for example

\[ l^H = (\begin{array}{c} \alpha x_1 x_2 \end{array}) \]

(2.5)

If \( x \) is also mapped to homogeneous coordinates

\[ x^H = (\begin{array}{c} 1 & x_1 & x_2 \end{array}) \]  

(2.6)
then

$$x_H^T l^H = 0$$ \hspace{1cm} (2.7)

is fulfilled iff $x_H$ is a point on the segment represented by $l^H$. The improved representation is defined as

$$S_{02}(x) = A(x)l^H(x) \otimes l^H(x)$$ \hspace{1cm} (2.8)

where $\otimes$ represents the tensor product, which is implemented by an outer product. Notice that $S_{02}$ is a function of $x$, and that it is a symmetric $3 \times 3$ matrix, i.e. it is an element of a 6-dimensional vector space. Notice also that $l^H(x)$ is constant for all points $x$ on the segment and consequently $S_{02}(x)$ will also be constant for the points on the segment except for variations in $A$. From equation 2.7, we get that all points $x_{H,i}$ lying on the segment will belong to the null space $N(S_{02})$ of $S_{02}$, and consequently $l^H$ will belong to the orthogonal complement to the null space $N^T(S_{02})$.

Up to now, our representation contains the segment’s orientation and orthogonal distance to the origin of the coordinate system, but it lacks the information on absolute position and extension. This can be implemented by adding the center of gravity and covariance matrix of the segment to the representation.

First, start with a point $x$ on the segment, figure 2.3. To get rid of problems with ambiguities, $x$ will be mapped to a projective space, equation 2.6. Then form a new tensor $S_{20}$ by an outer product between $x_H$ and itself.

$$S_{20}(x) = x_H \otimes x_H$$ \hspace{1cm} (2.9)
2.2 The scene tensor representation

Observe that this tensor can be estimated in each point on the segment and a summation of these tensors over all points $x \in \Gamma$ on the segment will result in

$$S_{20} = \sum_{x \in \Gamma} S_{20}(x) = \sum_{x \in \Gamma} \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix} = n \begin{pmatrix} 1 & \mathbf{r}^T \\ \mathbf{r} & \Sigma + \mathbf{r}\mathbf{r}^T \end{pmatrix}$$ (2.10)

where $\mathbf{r}$ is the center of gravity of the segment, $\Sigma$ is the covariance of the points from the segment and $n$ is the number of points on the segment. Observe that the mapping of $x$ to a projective space makes it possible to calculate the center of gravity and the covariance by normalizing with the number of points $n$.

It is impossible to calculate $S_{20}$, because it is not known which points that belong to the segment or not. This can be solved by combining $S_{20}$ with $S_{02}$ from equation 2.8. $A$ in equation 2.8 is the confidence measurement for the statement of segment or not and is only non zero if $x$ is a point close to a segment. Therefore, the confidence $A$ will sort out the points that belong to the segment. The two tensors $S_{20}$ and $S_{02}$ can be fused into the same representation by first reshaping the tensors to $6 \times 1$ vectors followed by an outer product between the vectors.

$$S_{22}(x) = S_{20}(x) \otimes S_{02}(x)$$ (2.11)

Observe that this tensor is dependent on $x$. Assume a local region $\Pi$ which contains one segment. Since $S_{02}(x)$ is constant for all points on a segment except for variations in $A$, a superposition of all $S_{22}(x)$ in the local region $\Pi$ will result in:

$$S_{22} = \sum_{x \in \Pi} A(x)S_{20}(x) \otimes S_{02}(x)$$

$$= \left[ \sum_{x \in \Pi} A(x)S_{20}(x) \right] \otimes S_{02} \approx S_{20} \otimes S_{02}$$ (2.12)

As is seen in equation 2.12, $S_{02}$ can be factorized from the sum, which then breaks down to a sum of all $S_{20}(x)$ times the confidence $A(x)$. Due to the confidence, this sum will be an approximation of $S_{20}$ in equation 2.10.

For a local region containing multiple segments, the tensor $S_{02}(x)$ will still be constant on a segment, but different for each segment. Equation 2.12 will then result in

$$S_{22} = \sum_{k} S_{20,k} \otimes S_{02,k}$$ (2.13)

where $k$ represents segment $k$. Since the sum of $S_{22}(x)$ over a local region containing multiple segments breaks down to a sum over the segments $k$, we have shown that it is easy to add new segments to the representation by superposition.

Now we have a representation that contains the orientation, center of gravity and covariance matrix for each segment in a local region. The representation is not complete until it is proven that the stored information can be retrieved from the tensor, which will be discussed in section 2.4.4. In the following sections, the rank of the scene tensor will be discussed.
2.2.1 The rank of the scene tensor

The rank of the scene tensor is a very important property, since it can be used for selecting interesting tensors and can also be used as a point of interest detector in a similar manner as e.g. the Harris operator [18]. In [30], the rank of the scene tensor is discussed and a proof that the number of segments is equal to the rank of the tensor is presented. It is also shown that this property only holds for up to three segments, which means that there are configurations of four segments in a local region which produce a scene tensor with rank three. In this section we will briefly discuss the rank of the scene tensor and also introduce two new parameters $x_0$ and $l_0$.

The rank we refer to is the matrix rank, since the scene tensor is a $9 \times 9$ matrix. The matrix rank corresponds to the number of linearly independent rows or columns in the matrix and when the structure is a sum of outer products between two different vectors

$$M = \sum_i v_i q_i^T$$

the rank of the matrix $M$ is the minimum number of linearly independent vectors in the sets $\{v_i\}$ and $\{q_i\}$. Since the scene tensor has the same structure, equation 2.13, as the matrix $M$ the rank will be the minimum number of linearly independent vectors in the sets $\{S_{20,k}\}$ and $\{S_{02,k}\}$.

The rank of a matrix can be measured by a Singular Value Decomposition (SVD), where a rank one matrix will have one non-zero singular value and vice versa. However, in practice no singular values are non-zero and the rank is therefore a continuous variable, which is dependent on the angle between the vectors in the sets $\{S_{20,k}\}$ and $\{S_{02,k}\}$. This is a necessary property, since nothing is discrete in images and for example two parallel segments will more and more resemble one segment if the distance between them is decreased. There is no distinct point where the two segments becomes one, but instead a continuous transition. A rank measurement will be discussed in section 2.4.2, which is useful in an algorithm for detecting points of interest.

In this chapter we will continue with discussing general aspects on the rank of the scene tensor for the case of one, two and three segments.

One segment

For a local region $\Pi$ with one segment, the sum in equation 2.12 will result in:

$$S_{22} = S_{20} \otimes S_{02}$$

It is obvious that this matrix will have rank one, since it is a sum of only one outer product. On the other hand, this sum could result in a higher rank tensor dependent on the thickness of the segment. Lets have a closer look at $S_{02}(x)$

$$S_{02}(x) = A(x)I^H(x) \otimes I^H(x) = A(x) \begin{pmatrix} |l|^2 & -|l| \hat{n} & -|l| \hat{n}^T \\ -|l| \hat{n} & \hat{n}^T & \hat{n} \end{pmatrix}$$

(2.16)
2.2 The scene tensor representation

Figure 2.4: One segment can be seen as a number of parallel segments. Especially if the segment is thick.

Since the orthogonal distance $|l|$ to the segment only affects some of the elements in $S_{02}$, the orthogonal distance $|l|$ will therefore not only scale $S_{02}$ but also change the direction of $S_{02}$. This is important in the case of distinguishing two parallel segments, but can also be a problem for a thick segment. A thick segment can be seen as a number of parallel segments lying close to each other, figure 2.4. Due to the dependency of $|l|$, $S_{02}$ might be linearly independent for the different parallel segments if the distance between them is large enough. The result is that a local region containing one thick segment may produce a scene tensor with higher rank than one. This problem can be reduced if a new parameter $l_0$ is introduced. This parameter is used for adjusting the influence of $|l|$ by a scaling: $|l|/l_0$. By inserting this in equation 2.16, we get

$$S_{02}(x) = A(x) \left( \begin{array}{c} |l|^2 \\ -\frac{|l|}{l_0} \hat{n} \\ -\frac{|l|}{l_0} \hat{n}^T \end{array} \right) \sim A(x) \left( \begin{array}{c} |l|^2 \\ -l_0 |l| \hat{n} \\ -l_0 |l| \hat{n}^T \end{array} \right)$$

(2.17)

and the new $I^H$ is

$$I^H(x) = (-|l| \hspace{0.5cm} l_0 n_1 \hspace{0.5cm} l_0 n_2)^T$$

(2.18)

Now, there is a possibility to adjust $l_0$ and get the desired behavior. $l_0$ should be adjusted so that a single segment results in a rank one tensor. If $l_0$ is too small, the tensor will have a higher rank than one and if $l_0$ is too large, a local region with two parallel segments will approximately produce a tensor with rank one. $l_0$ can be seen as an "resolution parameter", i.e. it defines the maximal width of a segment. In practice, no segment has a width of one pixel, but due to $l_0$ it is possible to adjust for this.
The behavior of $S_{20}$ is not only affected by the parameter $l_0$, but also by the distance to the segment. As stated above, $l_0$ should be adjusted so that a scene tensor estimated from a local region with one segment gets rank one. However, this is also dependent on the distance to the segment. A thick segment further away from the origin needs a lower value of $l_0$ to become rank one than a segment closer to the origin.

Two segments

Things get more complicated in the case of a local region with two independent segments. For this case equation 2.12 will result in

$$S_{22} = S_{20,1} \otimes S_{02,1} + S_{20,2} \otimes S_{02,2}$$  \hspace{1cm} (2.19)

In [30] there is a proof that the two sets $\{S_{20,1}, S_{20,2}\}$ and $\{S_{02,1}, S_{02,2}\}$ are linearly independent. The consequence is that a local region with two segments will produce a tensor of rank two. However, this is only the case for an ideal situation. In the case of two segments crossing each other in the origin and where each segment has its center of gravity in the origin, figure 2.5, the rank will be one if the angle between the segments is zero and rank two if the angle is $90^\circ$. The rank has to be a continuous function, decreasing from two to one when the angle is decreased from $90^\circ$ to $0^\circ$. It would be convenient to adjust the properties of this function and in [30] a new parameter $x_0$ is introduced to $x_H$ as

$$x_H = (x_0 \ x_1 \ x_2)^T$$  \hspace{1cm} (2.20)

and the new $S_{20}$ tensor will then be

$$S_{20} = \begin{pmatrix} x_0^2 & x_0\bar{x}^T \\ x_0\bar{x} & \Sigma + \bar{x}\bar{x}^T \end{pmatrix}$$  \hspace{1cm} (2.21)

From equation 2.21, it is difficult to say how $x_0$ will affect the rank of the tensor $S_{22}$. It is clear that the rank will decrease if the tensors $\{S_{20,1}, S_{20,2}\}$ become
2.2 The scene tensor representation

Figure 2.6: The angle $v$ between two $S_{20}$ tensors, created from each segment in figure 2.5. The angle is both dependent on the angle between the segments, $\alpha$, and the parameter $x_0$.

more similar. This similarity can be measured if the two tensors are reshaped to vectors and the angle between them is measured. A synthetic test for this has been done, where two tensors $S_{20,k}$, $k = \{1, 2\}$ have been estimated from the two segments in figure 2.5. The angle $\alpha$ between the segments and the parameter $x_0$ have been changed and the angle $v$ between the tensors is calculated. The result is illustrated in figure 2.6. As is seen in figure 2.6, the angle $v$ between the tensors decreases if $x_0$ is increased. This is not a desired behavior, since the angle should be close to $90^\circ$ when the angle between the segments is $90^\circ$. This is important if the rank of $S_{20}$ should be two. The conclusion is that the parameter $x_0$ should be close to zero. It cannot be equal to zero, since then it is impossible to compute the center of gravity $\mathbf{\mathcal{X}}$ and the covariance matrix $\Sigma$. In the experiments, $x_0$ is set to a value close to one.

In general, we can say that the parameter $x_0$ adjusts the relation between the influence of the center of gravity $\mathbf{\mathcal{X}}$ and covariance matrix $\Sigma$ for each segment. This is not completely true, since the part of $S_{02}$ that is not dependent on $x_0$ is a superposition between the covariance matrix and the center of gravity for each segment; $\Sigma + \mathbf{\mathcal{X}} \mathbf{\mathcal{X}}^T$. This makes things a bit more complicated, and honestly we have to say that the effect of $x_0$ is not completely understood.

Three segments

In [30] there is a proof that the two sets $\{S_{20,1}, S_{20,2}, S_{20,3}\}$ and $\{S_{02,1}, S_{02,2}, S_{02,3}\}$ are linearly independent for three independent segments. A local region with three
segments will therefore produce a tensor with rank three. It is also shown that some configurations of four segments will also produce a tensor with rank three. This makes the rank measurement for the case of three segments somewhat unreliable, since it is not clear if the tensor contains three or four segments. Therefore, this thesis will concentrate on tensors of rank two, where a lot of features are included such as corners, crossings, etc.

### 2.3 How to estimate the scene tensor

In section 2.2 the theory related to the scene tensor was discussed, but how can the scene tensor be estimated from real image data? As mentioned in the previous section, the scene tensor is an extension of the orientation tensor $\mathbf{T}$. Equation 2.8 in the previous section can be rewritten as

\[
\mathbf{S}_{02}(\mathbf{x}) = \mathbf{K}(\mathbf{x})\mathbf{T}(\mathbf{x})\mathbf{K}(\mathbf{x})^T \tag{2.22}
\]

\[
\mathbf{K}(\mathbf{x}) = \begin{pmatrix}
-x^T \\
l_0\mathbf{I}
\end{pmatrix} \tag{2.23}
\]

where the orientation tensor $\mathbf{T}$ is defined as

\[
\mathbf{T}(\mathbf{x}) = \begin{pmatrix}
a(x) & b(x) & c(x)
b(x) & c(x) & c(x)
c(x) & c(x) & c(x)
\end{pmatrix}
\]

Equation 2.12 will then result in

\[
\mathbf{S}_{22}(\mathbf{x}) = \mathbf{x}_H \otimes \mathbf{x}_H \otimes \mathbf{K}(\mathbf{x})\mathbf{T}(\mathbf{x})\mathbf{K}(\mathbf{x})^T \tag{2.25}
\]

The matrix $\mathbf{S}_{22}$ is a $9 \times 9$ matrix or a $6 \times 6$ matrix, if the symmetries are removed. By further investigation of equation 2.25, it turns out that each element in $\mathbf{S}_{22}$ is a bivariate polynomial in $\mathbf{x}$ with coefficients dependent on the elements in $\mathbf{T}$, table 2.1. The first column refers to index in a 4-dimensional matrix $\mathbf{S}_{22,i,j,k,l}$, where symmetries are found in the index pairs $i,j$ and $k,l$. This 4-dimensional matrix is the result if the tensor product $\otimes$ is implemented as an actual tensor product instead of “reshaping” to vectors followed by an outer product. The four index version is illustrated since it possible to see the symmetries and consequently reconstruct the full matrix. The second column illustrates corresponding indexes in the 2-dimensional matrix.

$\mathbf{S}_{22}(\mathbf{x})$ in equation 2.25 can be calculated in each point $\mathbf{x}$ in a local region $\Pi$ and the final $\mathbf{S}_{22}$ for the local region is then estimated as

\[
\mathbf{S}_{22} = \sum_{\mathbf{x} \in \Pi} g(\mathbf{x}) (\mathbf{x}_H \otimes \mathbf{x}_H) \otimes (\mathbf{K}(\mathbf{x})\mathbf{T}(\mathbf{x})\mathbf{K}(\mathbf{x})^T) \\
= \sum_{\mathbf{x} \in \Pi} g(\mathbf{x}) \mathbf{S}_{20}(\mathbf{x}) \otimes \mathbf{S}_{02}(\mathbf{x}) \tag{2.26}
\]

where $g(\mathbf{x})$ is a Gaussian weighting function used for localizing the tensor estimate. However, computing $\mathbf{S}_{22}(\mathbf{x})$ in each point and then perform a weighted summation
2.3 How to estimate the scene tensor

| $S_{221,1,1}$ | $S_{221,1}$ | $x_0^2 \left( a x_1^2 + 2 b x_1 x_2 + c x_2^2 \right)$ |
| $S_{221,1,1,2}$ | $S_{221,1,2}$ | $l_0 x_0^2 \left( -a x_1 - b x_2 \right)$ |
| $S_{221,1,1,3}$ | $S_{221,1,3}$ | $l_0 x_0^2 \left( -b x_1 - c x_2 \right)$ |
| $S_{221,1,2,2}$ | $S_{221,1,5}$ | $l_0^2 a x_0^2$ |
| $S_{221,1,2,3}$ | $S_{221,1,6}$ | $l_0^2 b x_0^2$ |
| $S_{221,1,3,3}$ | $S_{221,1,9}$ | $l_0^2 c x_0^2$ |
| $S_{221,2,1,1}$ | $S_{221,2,1}$ | $x_0 \left( a x_1^3 + 2 b x_1^2 x_2 + c x_1 x_2^3 \right)$ |
| $S_{221,2,1,1,2}$ | $S_{221,2,2}$ | $l_0 x_0 \left( -a x_1^2 - b x_1 x_2 \right)$ |
| $S_{221,2,1,1,3}$ | $S_{221,2,3}$ | $l_0 x_0 \left( -b x_1 x_2 - c x_1 x_2 \right)$ |
| $S_{221,2,2,2}$ | $S_{221,2,5}$ | $l_0^2 a x_0 x_1$ |
| $S_{221,2,2,2,2}$ | $S_{221,2,6}$ | $l_0^2 b x_0 x_1$ |
| $S_{221,2,3,3}$ | $S_{221,2,9}$ | $l_0^2 c x_0 x_1$ |
| $S_{221,2,2,1,1}$ | $S_{221,2,1}$ | $a x_1^4 + 2 b x_1^3 x_2 + c x_1^2 x_2^2$ |
| $S_{221,2,2,1,1,2}$ | $S_{221,2,2}$ | $l_0 \left( -a x_1^3 - b x_1^2 x_2 \right)$ |
| $S_{221,2,2,1,1,3}$ | $S_{221,2,3}$ | $l_0 \left( -b x_1^2 - c x_1 x_2 \right)$ |
| $S_{221,2,2,2,2}$ | $S_{221,2,5}$ | $l_0^2 a x_1 x_2$ |
| $S_{221,2,2,2,2,2}$ | $S_{221,2,6}$ | $l_0^2 b x_1 x_2$ |
| $S_{221,2,3,3}$ | $S_{221,2,9}$ | $l_0^2 c x_1 x_2$ |
| $S_{221,2,2,1,1,1}$ | $S_{221,2,1}$ | $a x_1^3 x_2 + 2 b x_1^2 x_2^3 + c x_1 x_2^3$ |
| $S_{221,2,2,1,1,1,2}$ | $S_{221,2,2}$ | $l_0 \left( -a x_1^2 x_2 - b x_1 x_2^3 \right)$ |
| $S_{221,2,2,1,1,1,3}$ | $S_{221,2,3}$ | $l_0 \left( -b x_1 x_2^3 - c x_1 x_2^3 \right)$ |
| $S_{221,2,2,2,2,2}$ | $S_{221,2,5}$ | $l_0^2 a x_2^3$ |
| $S_{221,2,2,2,2,2,2}$ | $S_{221,2,6}$ | $l_0^2 b x_2^3$ |
| $S_{221,2,3,3}$ | $S_{221,2,9}$ | $l_0^2 c x_2^3$ |

Table 2.1: The elements in $S_{22}$. Indexes in the 4-dimensional matrix is illustrated in the first column and indexes for the 2-dimensional matrix is illustrated in the second column.

is not an efficient way to estimate $S_{22}$. How can this be done more efficient? Let’s have a look at element $\{6, 2\}$ in $S_{22}$. By using table 2.1 and equation 2.26, an expression for element $\{6, 2\}$ in the pixel position $p = (p_1, p_2)^T$ can be derived

$$
S_{22,6,2}(p) = -l_0 \sum_{x \in \Omega} \left( g(x) x_1^2 x_2 A(p + x) + g(x) x_1 x_2 B(p + x) \right) 
$$

(2.27)
where $A$ and $B$ are matrices that constitutes of the elements $a$ and $b$ from each pixel. Equation 2.27 can be interpreted as a correlation of $A(p)$ and $B(p)$ with the filters $f_{gx_1x_2}$ and $f_{gx_1x_3}$

$$S_{22a_2}(p) = -l_0(f_{gx_1x_2} * A(p) + f_{gx_1x_3} * B(p))$$

(2.28)

where * is a correlation operator. Each term in the polynomial is a monomial in $x$ and since both the Gaussian weighting function and the monomials are Cartesian separable, equation 2.27 can be written as

$$S_{22a_2}(p) = -l_0(f_{gx_1} * (f_{gx_2} * A(p)) + f_{gx_1} * (f_{gx_2} * B(p)))$$

(2.29)

This is a typical situation, and each element in $S_{22}$ is a sum of monomials up to order four weighted with a weighting function, which all are Cartesian separable into 1D-filters. Some of these filters reappear and by reusing filter results we only need 47 1D-filter operations to estimate $S_{22}$ if the correlator structure in figure 2.7 is used. In figure 2.7 a circle represent the signal and each box represent one 1D-filter. In this structure both the separability and the possibility to reuse filter response is used to minimize the number of 1D-filters. The Gaussian weighting function is omitted from figure 2.7 to simplify the illustration. However, it should be applied in each correlation step.

Since an estimation of $S_{22}$ is implemented in terms of correlations on the elements in $T$, an estimation of the scene tensor will therefore start with estimating a dense field of orientation tensors. Which procedure for estimating the orientation tensor is not crucial, but might affect the end result. It is important that the confidence $A(x)$ is only nonzero sufficiently close to a segment. If not, some problems with the rank of the scene tensor will occur, which is discussed in section 2.2.1. In the next step, the images $a$, $b$ and $c$ is correlated as illustrated in figure 2.7. The
Figure 2.8: The confidence for the scene tensor should be highest when the segment is in the center of the local region and smoothly decrease when the distance increases.

result is 37 new images. In the last step, the scene tensor can be estimated in each pixel by using these 37 images and the table 2.1.

How sensitive is the estimation method of the scene tensor to noise? This is of course dependent on how sensitive the orientation tensor is to noise, since this is the in-data. However, estimation of orientation tensors is out of scope for this thesis, and a comparison between the most common methods can be found in [22].

In general, we can say that the estimation method of the scene tensor contains correlations of moment filters up to the fourth order. It is known that these filters are sensitive to noise, but this is reduced due to the Gaussian weighting function.

2.4 Analyzing the scene tensor

In the previous sections the theory related to the scene tensor was discussed, and a method on how to estimate the tensor was described. The following sections will describe how to select a scene tensor with some interesting property, for example tensors representing corners, section 2.4.3. When a certain scene tensor is selected, it is interesting to extract the information contained in the representation. This is described in section 2.4.4. We will start with the most fundamental part of a representation; a confidence measurement. A confidence measurement should describe if the representation contains anything interesting or not and for the scene tensor a certain kind of norm will be used, section 2.4.1.

2.4.1 The scene tensor norm

When a scene tensor is estimated from a local region, it is necessary to have a confidence of how well the representation can describe the local region. In the case of the scene tensor, a region with segments should give a high confidence and a region with simply noise should produce a low confidence. It is also important that the confidence measurement has a smooth behavior e.g. for a segment translated through the local region, figure 2.8, the confidence should be highest when the segment goes through the center of the region and smoothly decrease when the distance between the segment and the center of the region increases.
The scene tensor $S_{22}$ can be seen as a $9 \times 9$ matrix and the most obvious confidence measurement could be the matrix norm $\|S_{22}\|$. This norm can be seen as a function of the distance to the segment in the local region, and the problem is that this is a non-linear function. The norm will therefore not smoothly decrease when the distance between the segment and center of the region increases, but have a ringing effect where it goes up and down, figure 2.9.

A confidence measurement which will not have the problems with ringing effects is the norm of the weighted sum of the orientation tensors in the local region $\Pi$, i.e.

$$c = \left\| \sum_{x \in \Pi} g(x) T(x) \right\|$$  \hspace{1cm} (2.30)

As discussed earlier in section 2.3, an estimation of the scene tensor starts with estimating a dense field of orientation tensors. Then each element in the scene tensor is estimated by correlations of the elements in the orientation tensor with a number of filters. One of these filters is only the weighting function, and the result is the weighted sum of orientation tensors. This resulting tensor can be extracted from the elements of the scene tensor, and the norm of this tensor will give the confidence $c$ in equation 2.30.

The norm of the sum of orientation tensors are illustrated for two different images in figure 2.10 and figure 2.11. The norm is high on lines and edges and decreases smoothly when the distance to the segment increases. For some corners and line crossings the norm is lower, which is an effect of the orientation tensor. Since the orientation tensor is a representation of simple signals, the norm is lower on non-simple signals such as crossings and corners. Since the scene tensor confidence is the norm of a sum of orientation tensors, it is natural that the norm can be lower close to corners and crossings. However, since the local window for
the scene tensor is larger than for the orientation tensors, a number of orientation tensors computed from a simple signal will always be included in the sum.

### 2.4.2 The rank measurement

In section 2.2.1 it was discussed that the number of segments in the local region is equal to the rank of the scene tensor. This is an important property and can e.g. be used in an interest point detection algorithm. But before such an algorithm can be implemented, we have to find a rank measurement for the scene tensor.

The Singular Value Decomposition (SVD) is a standard method for measuring
the rank of a matrix and can be computed for the $9 \times 9$ matrix $S_{22}$. If there are exactly two non-zero singular values, then the rank is two, and vice versa. However, in practice all singular values are non-zero which means that we need some method for estimating a confidence for a certain rank. This can be done by taking the three largest singular values $\sigma_1, \sigma_2, \sigma_3$, and compute the following parameters:

\[
\begin{align*}
    c_1 &= \frac{9d - 4qt + t^3}{3d - 3qt + t^3} \\
    c_2 &= \frac{-9d + qt}{3d - 3qt + t^3} \\
    c_3 &= \frac{3d}{3d - 3qt + t^3}
\end{align*}
\]

\[t = \sigma_1 + \sigma_2 + \sigma_3 \quad d = \sigma_1 \sigma_2 \sigma_3 \quad q = \sigma_1 \sigma_2 + \sigma_2 \sigma_3 + \sigma_3 \sigma_1 \tag{2.31}\]

From this follows directly that $c_2 = 1$ if $\sigma_1 = \sigma_2 \neq 0$ and $\sigma_3 = 0$, and it vanishes if $S_{22}$ has rank one or if $\sigma_1 = \sigma_2 = \sigma_3 \neq 0$. The confidence measurement also has the property of summing up to one, $c_1 + c_2 + c_3 = 1$. Consequently, $c_k$ represents a measure of confidence for rank $k$.

The three confidence measurements are computed for two different test images, where one is a synthetic image and one is an image captured for a target object in the VISATEC project [42] with a standard camera. The result is illustrated in figure 2.12 and figure 2.13, where bright areas indicate high confidence of respective rank. The confidence measurement for the real image seem to react on both the object and the background. To understand this we have to look at the orientation tensor, which is the basis for the scene tensor. This can be done by plotting each tensor with a color that represents the orientation and the color strength representing the norm of the orientation tensor, figure 2.14. The estimation method used for the orientation tensor $T$ is

\[T = \int g(x) \nabla I \nabla I^T dx \tag{2.32}\]

where $g$ is a Gaussian weighting function and $\nabla I$ is the gradient. As is seen in figure 2.14 to the right, the orientation tensor has a high norm on the background. It might be possible to adjust this behavior with the parameters, but this is not done since it is interesting to see how the scene tensor reacts to this type of in data. The high confidence for rank one, two and three could depend on the appearance of the background structure. A closer look at the right image in figure 2.14 reveals that the structure looks like lines with different orientations. If the filter size for the scene tensor is small enough, this type of structure could result in high confidence for rank one, two and three. However, if the filter size is increased the
Figure 2.12: The three confidence measurements are calculated for scene tensors computed from the image in the upper left corner. The confidence measurement $c_1$ is illustrated in the upper right corner, $c_2$ is illustrated in the lower left corner and $c_3$ is illustrated in the lower right corner.

Confidence for rank one, two and three is not reduced. The only explanation for this behavior is then the low energy in these areas. Local areas with low energy results in a scene tensor with small and unstable singular values and the confidence measurements will then be a small number divided with a small number.

2.4.3 Detection of interest points

Given that we can compute $S_{22}$ at arbitrary points or even every point in an image, how can we use this data for image processing? One possible task is to
Figure 2.13: The three confidence measurements are calculated for scene tensors computed from the image in the upper left corner. The confidence measurement $c_1$ is illustrated in the upper right corner, $c_2$ is illustrated in the lower left corner and $c_3$ is illustrated in the lower right corner.

find points or regions which contains two independent segments, e.g. corners or junctions. There are simpler methods which can do this, e.g. [18], but the proposed representation allows us also to determine the parameters of a junction, e.g. the position and orientation of the corresponding segments, with high accuracy.

To start with, we want to find points which are characterized by being close to two segments. If we have estimated $S_{22}$ for an image point, this means that we want to know if $S_{22}$ has rank two. The confidence for rank two, equation 2.31, can be used for this purpose. However, this confidence measurement only considers the type of structure in the local region, but not the energy of the structure. The result is a confidence measurement that do not separate noise from lines and edges, figure 2.13. The problem with noise sensitive confidence measurements can be reduced by the tensor norm. In figure 2.11 we see that the norm is not as sensitive to noise as the confidence measurements and a multiplication with the norm will reduce the problem. The result is a confidence measurement that both considers the type of structure in the local region and the energy. The result for
2.4 Analyzing the scene tensor

Figure 2.14: An illustration of the orientation tensors computed from the two test images. The color represents the orientation and the color strength represents the norm.

Figure 2.15: The confidence of rank two multiplied with the tensor norm.

The confidence of rank two multiplied with the norm is illustrated in figure 2.15 for both the synthetic image and the real image. The result is not dramatical for the synthetic image and the only effect is more local responses from corners and crossings. For the real image almost all of the response from the background has vanished, due to the low energy in these areas.

To get more distinct peaks, the images in figure 2.15 can be convolved with a Laplace filter, which brings outs the peaks and suppresses lines, etc. The result is
Figure 2.16: Convolving the images in figure 2.15 with a Laplace filter results in more distinct peaks and a suppression of lines, etc.

illustrated in figure 2.16.

By searching e.g. local maxima in the images in figure 2.16 we can detect points of interest, \( \alpha \). Notice that the bright areas are always found inside the corners and with the maximal value further away from the corner point if the opening angle is decreased. This indicates that the rank measure is not only dependent on the number of segments but also on the distance to the corner or junction and the angle between the corresponding segments.

This is not a desired behavior of a points of interest detector, since the actual position of a feature should not depend on internal feature parameters. Many points of interest detectors have this behavior, e.g. the Harris detector [18]. In the case of the scene tensor there is a possible solution to this problem, since the parameters for each segment in the local region can be extracted, section 2.4.4. Assume that the \( \mathbf{m}_k^H \) vector is estimated for each segment. From equation 2.7, we get that all points \( \mathbf{x}_H \) on segment \( k \) is orthogonal to \( \mathbf{m}_k^H \). For two segments crossing each other there are only one common point \( \mathbf{m}_H \). \( \mathbf{m}_H \) is orthogonal to both \( \mathbf{m}_1^H \) and \( \mathbf{m}_2^H \), and can therefore be estimated by the cross-product.

\[
\mathbf{m}_H = \mathbf{m}_1^H \times \mathbf{m}_2^H
\]

(2.33)

The vector \( \mathbf{m}_H \) is in homogeneous coordinates, and by normalizing and mapping back to Cartesian coordinates we get a new vector \( \mathbf{m} \in \mathbb{R}^2 \). The position of the crossing \( \mathbf{p} \) can be calculated in pixels as

\[
\mathbf{p} = \mathbf{o} + \mathbf{m}
\]

(2.34)

Note that the actual position of the crossing is estimated with sub-pixel accuracy. The position estimate is also not dependent on small position variations of the
2.4 Analyzing the scene tensor

![Image](image.png)

Figure 2.17: The detected points of interest for the two test images.

local maximum, $o_i$, in the confidence map for rank two. The result is a points of interest detector for all types of features containing two segments, and the position of the crossing will be detected with sub-pixel accuracy. Features such as parallel lines and non parallel lines where the crossing is not situated inside the local region have to be taken into account. They could be used as features, but the position estimate gets more unstable when the segments get more parallel. These types of features can be filtered out by a threshold on the norm of $m$, which is the distance between the local origin and the position of the crossing.

The detected points of interest for the two test images are illustrated in figure 2.17. There are some unwanted points around the circles in the image to the right. This is a result from the confidence measurement of rank two, because a small segment of a circle can be seen as a corner with a very large opening angle. These points are not wanted since they are not repetitive, i.e. almost any point on the circle can be found by the detector and if one point is found the first time, another one can be found the next time.

### 2.4.4 Extracting information from the scene tensor

When a scene tensor is selected by a points of interest detector, it could be interesting to retrieve the information it contains. It is hard or even impossible to directly say anything from the elements in the $9 \times 9$ matrix, and the only way to get useful information is to extract the orientation for each segment $\hat{\mathbf{n}}$, the center of gravity for each segment $\mathbf{x}_k$ and the covariance matrix for each segment $\Sigma_k$ from the tensor. As described earlier, $S_{02}$ contains the orientation of the segment and $S_{20}$ contains the center of gravity and covariance matrix for the segment. Therefore, an analysis of the tensor will start with extracting $S_{20}$ and $S_{02}$ for each segment. The presentation will only handle the case of two segments in the local
region, since regions with a single segment are not so interesting, and regions with three segments can not unambiguously be represented by the scene tensor.

**Extracting S_{20} and S_{02}**

In the following discussion, both uppercase and lower case letters will be used for the same mathematical object. For example, $S_{20}$ is a matrix and $s_{20}$ is the matrix reshaped to a vector.

In the case of rank two, $S_{22}$ can be written as

$$S_{22} = S_{20,1} \otimes S_{02,1} + S_{20,2} \otimes S_{02,2}$$

(2.35)

Since the tensor product $\otimes$ is implemented by an outer product, the tensor $S_{22}$ is a $9 \times 9$ matrix. It is obvious that $\{s_{20,1}, s_{20,2}\}$ belongs to the range, $R(S_{22}) \in \mathbb{R}^9$, of the tensor and consequently $\{s_{02,1}, s_{02,2}\}$ will belong to the orthogonal complement of the null-space of $S_{22}$, denoted $N^\perp$. It is possible to estimate a basis for these two spaces by using an SVD (Singular Value Decomposition) of $S_{22}$,

$$S_{22} = \sum_i \sigma_i u_i v_i^T$$

(2.36)

where $\sigma_i$ is the singular value and $u_i$, $v_i$ are the left and right singular vectors. In the case of a rank two tensor the SVD will be

$$S_{22} = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$$

(2.37)

For a rank two tensor, the basis vectors $\{v_1, v_2\}$ will span $N^\perp(S_{22})$ of the tensor, and since $S_{02,k}$ belongs to that space, $S_{02,k}$ can be expressed as:

$$S_{02,k} = \alpha_k V_1 + (1 - \alpha_k) V_2$$

(2.38)

where $\alpha_k \in \mathbb{R}$ and $V_i$ is $v_i$ reshaped to a $3 \times 3$ matrix.

How do we find these $\alpha_k$? $S_{02}$ is an outer product of $1^d$ and is therefore a symmetric rank one matrix. Any symmetric matrix has $n$ real valued eigenvalues and for the case of $S_{02}$, $n$ is equal to three. When the matrix has rank one, it is only one eigenvalue, $\lambda_1$, that is nonzero and this is a property that can be used for estimating $\alpha_k$.

Consider the characteristic polynomial

$$\det (S - \lambda I) = 0$$

(2.39)

which in the case of a symmetric $3 \times 3$ matrix has three real roots $\lambda_1 \geq \lambda_2 \geq \lambda_3$. A factorization of equation 2.39 will then be

$$-\lambda^3 + (\lambda_1 + \lambda_2 + \lambda_3) \lambda^2 - (\lambda_1 \lambda_2 + \lambda_2 \lambda_3 + \lambda_1 \lambda_3) \lambda + \lambda_1 \lambda_2 \lambda_3 = 0$$

$$\Leftrightarrow -\lambda^3 + \text{Tr}(A) \lambda^2 - q \lambda + \det(A) = 0$$

(2.40)

The coefficient $q$

$$q = \lambda_1 \lambda_2 + \lambda_2 \lambda_3 + \lambda_1 \lambda_3$$

(2.41)
is especially interesting. If $S$ is positive semidefinite, then $q$ is zero iff $S$ is a rank one matrix. $S_{02}$ is positive semidefinite since it is Hermitian and has only one nonzero eigenvalue $\lambda_1$, which is positive.

\[
S_{02}^H = A \|H^T \|H^H = A \|H\|^2 \|H^H
\]

\[
\lambda_1 = A \|H\|^2 > 0
\]

(2.42)

$A$ is the confidence and is always nonnegative by definition.

It is obvious that $q$ can be calculated from the elements of $S$ and can therefore be used for estimating $\alpha_k$. An approach to estimate $\alpha_k$ is to calculate $q$ for the characteristic polynomial

\[
det(\alpha V_1 + (1 - \alpha) V_2 - \lambda I)
\]

(2.43)

and calculate $\alpha$ when $q$ vanish. $q$ is a second order polynomial in $\alpha$ and has two solutions $\alpha_1$ and $\alpha_2$ for $q = 0$. Since $q$ is zero iff $\alpha V_1 + (1 - \alpha) V_2$ has rank one, then $\alpha_1$ and $\alpha_2$ will result in two rank one matrices. $S_{02}$ for each segment can now be estimated by equation 2.38.

When $\alpha_k$ is estimated, it is possible to estimate $S_{20,k}$. Given the two roots $\alpha_1$ and $\alpha_2$ we can write

\[
(s_{02,1} s_{02,2}) = (v_1 v_2) A
\]

(2.44)

where

\[
A = \begin{pmatrix}
\alpha_1 & \alpha_2 \\
1 - \alpha_1 & 1 - \alpha_2
\end{pmatrix}
\]

(2.45)

In matrix notation we can rewrite equation 2.35 as

\[
S_{22} = (s_{20,1} s_{20,2}) (s_{02,1} s_{02,2})^T
\]

(2.46)

and substituting equation 2.44 into equation 2.46 then gives

\[
S_{22} = (s_{20,1} s_{20,2}) A^T (v_1 v_2)^T
\]

(2.47)

The SVD of $S_{22}$ in matrix notation is

\[
S_{22} = (u_1 u_2) \begin{pmatrix}
\sigma_1 & 0 \\
0 & \sigma_2
\end{pmatrix} (v_1 v_2)^T
\]

(2.48)

and by substituting equation 2.48 into equation 2.47 and solve for $s_{20,1}$ and $s_{20,2}$, we get

\[
(s_{20,1} s_{20,2}) = (u_1 u_2) \begin{pmatrix}
\sigma_1 & 0 \\
0 & \sigma_2
\end{pmatrix} (A^T)^{-1}
\]

(2.49)

When a method for extracting $S_{20}$ and $S_{02}$ is available it is possible to extract the orientation, center of gravity and covariance matrix for each segment. This will be discussed next.
Extracting the orientation and orthogonal distance for each segment

The orientation $\mathbf{n}_k$ and orthogonal distance $|\mathbf{l}_k|$ for each segment $k$ is easy to estimate once $S_{02,k}$ is calculated. $S_{02,k}$ is a rank one positive semidefinite matrix with the eigenvector $\mathbf{l}^H_k$. By an eigenvalue decomposition, $\mathbf{l}^H_k = \gamma \mathbf{l}^H_k$, can be estimated, where $\gamma$ is an unknown scaling factor. Since $\mathbf{l}^H_k$ is

$$\mathbf{l}^H_k = (-|\mathbf{l}_k| \quad l_0 n_{1,k} \quad l_0 n_{2,k})^T$$

the unknown scaling factor can be eliminated by the normalization ($k$ is omitted)

$$\mathbf{l}^H = -\text{sign}(\mathbf{l}^H_1) \frac{l_0}{|\mathbf{l}^H_{2:3}|} \mathbf{l}^H$$

Now it is easy to retrieve the orientation and orthogonal distance.

The orthogonal distance and the orientation is illustrated in figure 2.18 for two segments pointing from the crossing toward the local origin. All points that should be detected are detected, but at the center of the smaller square a local region with three segments is detected. This is somewhat strange since only regions with two segments is sought for. However, this is due to the locality of the scene tensor, since certain local regions will only contain two of the three segments. The result is a scene tensor of rank two.

Extracting the center of gravity and covariance matrix for each segment

The center of gravity $\mathbf{x}_k$ and covariance matrix $\Sigma_k$ for each segment $k$ can be estimated from the matrix $\hat{S}_{20,k} = \gamma S_{20,k}$, where $\gamma$ is an unknown scaling factor.
Consider the structure of $S_{20,k}$

$$S_{20,k} = \begin{pmatrix} x_0^2 & x_0 \mathbf{x}_k^T \\ x_0 \mathbf{x}_k & \Sigma_k + \mathbf{x}_k \mathbf{x}_k^T \end{pmatrix}$$

Due to the mapping of $x_H$ to homogeneous coordinates, we can eliminate the scaling factor by the normalization ($k$ is omitted)

$$S_{20} = \frac{x_0}{S_{20,1,1}} S_{20}$$

and the center of gravity and covariance matrix can easily be retrieved.

The center of gravity $\mathbf{x}$ and covariance matrix $\Sigma$ for each segment is illustrated in figure 2.19, where $\mathbf{x}$ is illustrated by a vector and $\Sigma$ is illustrated by an ellipse with the semiminor and semimajor axis’ lengths equal to the eigenvalues of the covariance matrix. The result is only illustrated for a subset of the test images in figure 2.18, since it is easier to see the result when the images are enlarged. It is still hard to see the vectors representing the center of gravity, but the ellipses are always plotted at the position of the center of gravity which may give a hunch. As is seen in the left plot in figure 2.19, it is possible to distinguish corners, crossings and T-crossings from each other by using the center of gravity and covariance matrix for each segment.

2.5 A repeatability test

2.5.1 Introduction

Several pose estimation algorithms are based on first detecting points of interest, e.g. [26, 27, 24, 39]. It is very important that this feature detection process is
stable under changes in scale, rotation, view and lightning. If the repeatability is low, i.e. if a set of features is detected on an object during the training process and only a small subset of these features is detected during the operation process, the system will likely fail with the pose estimation.

There are several different algorithms for feature detection and an evaluation of the most known algorithms is presented in [36], where it was found that the Harris operator [18] was the most stable one. We have performed a similar stability test to evaluate the scene tensor together with an algorithm for finding star patterns [22]. For reference the Harris operator has also been evaluated.

In the test we have used different test images, figure 2.20, and simulated changes in scale, rotation and view by computing the corresponding transformations in a computer. Transformations from an image to another image can be represented by homographies, which is useful for calculating the repeatability rate. A test to evaluate the stability under changes in lighting is also performed by using images captured from a static scene with different shutter speeds and images captured from a pile of objects with a moving light source. The repeatability is then computed between the transformed images and a reference image.

We will first describe the methods that are compared to the scene tensor approach, then explain some details of the experiment setup and finally present the results.

### 2.5.2 Interest point detectors

This section gives a short description of the methods compared to the scene tensor approach, section 2.4.3. Two versions of the Harris operator and a method for finding star patterns are considered. The parameters and thresholds for the methods are chosen such that they all are based on fairly the same region size and that they give approximately the same amount of points.

#### Harris, nms

The Harris function is computed as

\[ H = \det(T) - \alpha \text{Tr}(T), \tag{2.54} \]

where \( \alpha = 0.04 \) and \( T \) is the structure tensor

\[ T = \int g(x) \nabla I \nabla I^T dx. \tag{2.55} \]

\( g \) is a Gaussian window function with \( \sigma = 2 \). The image gradient is computed using differentiated Gaussian filters with \( \sigma = 1 \). Local maxima points are found by non-maximum suppression. All filters can be made separable.

#### Harris, subpixel

Same as the previous one, except that the local maxima points are found with subpixel accuracy. A second order polynomial is fitted to the Harris image around each of the maxima pixels in the previous method, and the local maxima of the polynomial function gives the subpixel position.
2.5 A repeatability test

Scene tensor

The method used here is the one described in section 2.4.3, where a dense field of orientation tensors are estimated from the gray-scale image. The orientation tensors are estimated by an outer product of gradient vectors, which is the same as for the Harris operator, equation 2.55. The same $\sigma$ is used for both the Gaussian window function and the differentiated Gaussian filters. A dense field of scene tensors are then estimated from the orientation tensors, where $\sigma = 2$ for the Gaussian window function in equation 2.26, $x_0 = 1$ and $l_0 = 50$. A confidence for rank two is then estimated for each scene tensor and multiplied with the scene tensor norm. This confidence image is then convolved with a Laplace filter and initial points of interest are detected by searching for local maxima. The final points of interest are then estimated by the crossing between the segments.

Star patterns

The method we use to find star-patterns is a combination of the ideas in [12, 20, 2]. This method is explained in detail in [24, 23, 22]. The basics are:

1. Compute the image gradient $\nabla I$. A differentiated Gaussian filter with $\sigma = 1$ is used (same as for the Harris methods).

2. Star patterns are found as local maxima to the function

$$ S_{\text{star}} = \int g(x) \left( \nabla I, x_{\perp} \right)^2 dx. \quad (2.56) $$

$g$ is a Gaussian window function with $\sigma = 2$. $S_{\text{star}}$ is made more selective by inhibition with a measure for simple signals. Local maxima points are then found by non-maximum suppression.

3. The point positions are improved by minimizing the circle pattern function

$$ S_{\text{circle}}(p) = \int g(x) \left( \nabla I, x - p \right)^2 dx \quad (2.57) $$

that are computed around each of the local maxima points.

The algorithm needs to compute a subset of monomes (or derivatives) up to the second order on the three images $I_x^2$, $I_y^2$, and $I_x I_y$. All filters can be made separable.

2.5.3 Experimental setup

Repeatability criterion

The repeatability criterion is the same as in [36]. We give a short summary here. Let $I_r$ denote the reference image and let $I_i$ denote an image that has been transformed. Let $\{x_r\}$ denote the interest points in reference image $I_r$, and let $\{x_i\}$ denote the interest points in the transformed image $i$. For two corresponding points in $x_r$ and $x_i$ in image $I_r$ and $I_i$ we have

$$ x_i = H_{ri} x_r, \quad (2.58) $$
where $H_{r,i}$ denotes the homography between the two images (the points is here represented in homogeneous coordinates). As in [36] we remove the points that do not lie in the common scene part of images $I_r$ and $I_i$. Let $R_i(\epsilon)$ denote the set of corresponding points pairs within $\epsilon$-distance, i.e.

$$R_i(\epsilon) = \{(x_r, x_i) \mid \text{dist}(H_{r,i}x_r, x_i) < \epsilon\}.$$  

The repeatability rate for image $I_i$ is defined as

$$r_i(\epsilon) = \frac{|R_i(\epsilon)|}{\min(n_r, n_i)},$$  

where $n_r = |\{x_r\}|$ and $n_i = |\{x_i\}|$ are the number of points detected in the common part of the two images. Note that $0 \leq r_i \leq 1$.

**Transformation of scale, rotation and view**

The homographies for the rotation, scale, and view transformations can be found in many text books, but we still include a short derivation for sake of completeness. The relation between a point $X = (X, Y, Z)^T$ in the 3D world and the corresponding point $x = (x, y)^T$ in the image is

$$\lambda \begin{pmatrix} x \\ 1 \end{pmatrix} = P \begin{pmatrix} X \\ 1 \end{pmatrix},$$  

where $P = K[R|t]$ is the camera (projection) matrix. The matrix $K$ contains the camera parameters. We assume the simple model

$$K = \begin{pmatrix} f & 0 & x_0 \\ 0 & 1 & 0 \end{pmatrix},$$  

where $f$ is the focal length and $x_0$ is the origin for the image coordinate system. The matrix $R$ and the vector $t$ defines the transformation of the 3D coordinate system. For the reference image we assume that the optical axis of the camera is orthogonal to the image in the 3D world and that the distance between the camera and the image is $d$. This gives $R = I$ and $t = 0$, and from (2.61) we then get

$$X = dK^{-1} \begin{pmatrix} x_r \\ 1 \end{pmatrix}.$$  

We now use (2.61) and (2.63) to compute a relation between the point $x_r$ in the reference image and a corresponding point $x$ in another image taken with the camera in a different position, i.e. for general choice of $R$ and $t$. The relation becomes

$$\lambda \begin{pmatrix} x \\ 1 \end{pmatrix} = K[R|t] \begin{pmatrix} X \\ 1 \end{pmatrix}$$

$$= KRX + Kt$$

$$= KRdK^{-1} \begin{pmatrix} x_r \\ 1 \end{pmatrix} + Kt$$  

$$= (dK[R]K^{-1} + [0|Kt]) \begin{pmatrix} x_r \\ 1 \end{pmatrix}.$$
and we identify the general homography between the reference image and a transformed image as
\[
H = d KRK^{-1} + [0|Kt] .
\]  
(2.65)

We get the following homographies for the special cases of rotation, scale, and view:

- Plane rotation:

\[
H = KRK^{-1} , \quad \text{where } R = \begin{pmatrix}
\cos \varphi & -\sin \varphi & 0 \\
\sin \varphi & \cos \varphi & 0 \\
0 & 0 & 1
\end{pmatrix} .
\]  
(2.66)

- Scale change:

\[
H = dI + [0|Kt] , \quad \text{where } t = \begin{pmatrix}
0 \\
0 \\
d'
\end{pmatrix} .
\]  
(2.67)

- Viewpoint change: Equation (2.65) where

\[
R = \begin{pmatrix}
1 & 0 & 0 \\
0 & \cos \varphi & -\sin \varphi \\
0 & \sin \varphi & \cos \varphi
\end{pmatrix} , \quad t = (I - R) \begin{pmatrix}
0 \\
0 \\
1
\end{pmatrix} .
\]  
(2.68)

**Data for the experiment**

The list below contains data for the experiments:

- Number of test images: 6, see figure 2.20.
- Rotation in the image plane: 18 images evenly spread between 0° and 180° ($\varphi = \frac{\pi k}{N - 1}$, $k = 0, 1, \ldots, N - 1$, $N = 18$). The first image is used as reference.
- Scale change: 9 images with a scale change (non-evenly spread) up to three times the original size ($d' = \frac{1}{2} - 1$, where \( c = 2 \left( \frac{k}{N - 1} \right)^2 + 1, k = 0, \ldots, N - 1, N = 9 \)). The first image is used as reference.
- View change: 21 images with a change in view between −45° and 45° ($\varphi = \frac{\pi k}{4N}$, $k = -N, \ldots, N$, $N = 10$). The middle image is used as reference.
- Two choices of $\epsilon$ is used, $\epsilon = 0.5$ and $\epsilon = 1.5$.

Figure 2.20 shows the test images that is used for the rotation, scale, and view transformations. They range from real images to synthetic images. Figure 2.21 shows an example of each of the transforms for one of the test images. The images has been expanded by zero padding before the transformation. This helps to avoid loss of points in the transformations (note however, that the padding is not enough for the scale transformation).
Figure 2.20: Test images for the rotation, scale, and view transformations.
Figure 2.21: An example of the transformations rotation, scale, and view.
Transformation of light

The test images for the light transformations are however not simulated. These images are taken of 3D scenes by a stationary camera, and either the camera shutter or the light source is changed. Three sequences were taken and shown in figure 2.22. In the first sequence we change the camera shutter, and the middle image is used as reference. The last two sequences are taken by changing the light source position, and the first image is used as reference in both cases. The scene is not planar and we therefore do not really have a ground truth in the last two cases. But we believe that the evaluation is still relevant since similar situations appear for example in object recognition applications, where the training data for an object is taken with different lightning conditions than the query data.

2.5.4 Experimental results

We will show the average results for all the test images. But we will also show the individual results for each test image to show that the result differs depending on the type of test image.

Comparison of the two Harris versions

From the results it was found that Harris with subpixel accuracy performed overall much better than Harris using only non-max suppression, figure 2.23 shows one
2.5 A repeatability test

Figure 2.23: Average results of the two Harris versions on the test images in figure 2.20 for the rotation, scale, and view transformations.

example. The difference is most obvious for $\epsilon$ smaller than 1 pixel, as would be expected. Because of this result we will only include the subpixel Harris from now on, to make the presentation less messy.

Rotation, scale, and view

Figure 2.24 contain the average results for all methods except Harris nms. The results are somewhat inconclusive, but if we examine each test image separately we see that subpixel-Harris performs best for natural images. The star-patterns and the four order tensors perform equally well or better than Harris on images that better resembles their models, i.e. straight lines and sharper corner points, especially for the scale transformation (figures 2.27 and 2.30).

Variation of illumination

The results on the light transformation sequences is shown in figure 2.31. We conclude that the differences are not significant between the different methods.
Figure 2.24: Average results on the test images in figure 2.20 for the rotation, scale, and view transformations.

Figure 2.25: Result on test image 1 for the rotation, scale, and view transformations.
2.5 A repeatability test

Figure 2.26: Result on test image 2 for the rotation, scale, and view transformations.

Figure 2.27: Result on test image 3 for the rotation, scale, and view transformations.
Figure 2.28: Result on test image 4 for the rotation, scale, and view transformations.

Figure 2.29: Result on test image 5 for the rotation, scale, and view transformations.
Figure 2.30: Result on test image 6 for the rotation, scale, and view transformations.

Figure 2.31: Results on the light change sequences in figure 2.22.
2.6 Conclusions and discussions

In this chapter the scene tensor representation has been discussed, which is an representation of multiple line or edge segments. It is shown that the representation can be estimated from standard orientation tensors by correlations with 47 1-dimensional filters. It is also shown that once the scene tensor is estimated it is possible select tensors by using a rank measurement and also extract parameters describing the line or edge segments.

These types of properties of the scene tensor allows us to implement a points of interest detector, which is essential in e.g. a pose estimation algorithm. The scene tensors ability to work as a points of interest detector have been tested by a repeatability test, where a comparison is done with the Harris operator [18] and a method for finding star patterns [22] The star-pattern method and the scene tensor assumes a bit more advanced models than the Harris operator and these methods has the best performance if their corresponding models fits the image content, as would be expected. If the model is less valid it seems that it is better to use a more crude model as in the Harris operator.

For real images and real objects it is hard to find a model that is always valid. Multiple line or edge segments which is the model in the scene tensor, is often not valid since real objects usually have a lot of curvature and the corners are often not sharp but more curved. Since the Harris operator usually works better for these kinds of images it might be a good idea to find points of interest with the Harris operator and represent the local region around these points with the scene tensor representation. However, one have to take into account that the detection method used by the scene tensor is not only a detection of interest points but also a method to find the tensor that best represents the local region. This will not be the case if the Harris operator selects the interest points.

So far, there is not an extended test for the scene tensor’s ability to work on images with noise. However, there has been a lot of tests on real images captured with standard cameras and these images contains both camera noise, different lighting conditions and shadows. Since this is the target environment for the scene tensor and since the performance still is good we believe that the lack of this noise test is not crucial.
Chapter 3

Invariant Representations

3.1 Introduction

If you hold an object in your hand and turn it, features that were on the backside of the object will appear. This is a typical situation for a child that plays with different toys and tries to discover and understand what happens if something is moved or rotated. The child tries to learn how an action effects the perceptual inputs [16, 14]. But what happens if the object is rotated in the image plane so that no new features are appearing on the object, for example a postcard on a table. If the post card is rotated, the image will rotate and appear as a new image but the information is still not changed. Do the child have to take into account each new image for each rotation angle and learn to recognize it as a post card? Probably not, since it will be an explosion of information if every possible projection of an object has to be learned.

It is the same situation for a pose estimation algorithm. Do we have to train a pose estimation system for all different object states? No, since there are certain types of object transformations that do not reveal new feature on the object, such as translation, rotation and scaling of the object. This is only approximately true since for example two parallel lines on an object will appear as one line if the distance to the object is increased or new features will appear if a thick object is translated on e.g. a table. However, in practice it is still possible to use so called invariant representations to decrease the amount of information from the percept side of a system. An invariant representation is a representation, which is invariant to a set of object transformations, i.e. the elements in the representation does not change when the object is transformed. For the case of pose estimation it is most interesting to be invariant to translation, rotation and scaling. Invariant representations are also interesting for other types algorithms, such as wide base line stereo and tracking.

The scene tensor itself is not invariant to these kinds of transformations and the following sections will describe three different ways to implement an invariant representation based on the scene tensor. Section 3.2 computes an invariant representation by an nonlinear mapping of the scene tensor to a set of invariants
parameters. Section 3.3 and section 3.4 combines grouping of interests points with the scene tensor theory to compute a set of invariant parameters. The idea of grouping interests points for estimating invariant representations is not new and has been investigated by several researches e.g. [17, 19, 33].

### 3.2 Coefficients of the characteristic polynomial

In [30] a method for computing a set of invariant parameters from the scene tensor is described, which is based on the coefficients of the characteristic polynomial. We will shortly discuss the results from the experiments, but first some of the details will be repeated from [30].

Let $\mathbf{R}$ be a linear transformation on a projective space. This transformation can both describe affine transformations and perspective projections. A scene tensor $S_{22}$ transformed by the transformation $\mathbf{R}$ can then be expressed as

$$
S_{22}' = \mathbf{R}_2 S_{22} \mathbf{R}_2^T
$$

$$
\hat{\mathbf{R}}_2 = \mathbf{G}_2 (\mathbf{R}_2^T) \mathbf{G}_2
$$

(3.1)

where $\mathbf{R}_2$ and $\mathbf{G}_2$ is linear transformations of $\mathbf{R}$ and $\mathbf{G}$. $\mathbf{G}$ defines the metric and is defined in [30].

Consider the characteristic polynomial of $S_{22}' \mathbf{G}_2$

$$
P(\lambda) = \det (S_{22}' \mathbf{G}_2 - \lambda \mathbf{I}) = \det (S_{22} \mathbf{G}_2 - \lambda \mathbf{I})
$$

(3.2)

From this follows then immediately that $P(\lambda)$ is invariant to transformations $\mathbf{R}$ which implies that, e.g. the coefficients of $P(\lambda)$ are invariant to $\mathbf{R}$. The coefficients are polynomial up to order 9 and the consequence is that the different coefficients will scale in different ways relative to scale changes in $S_{22}$. This is not wanted in an representation since different parameters will have a higher weight than others. In [30] a normalized characteristic polynomial $\hat{P}$ is introduced

$$
\hat{P}(\lambda) = P(\lambda \|S_{22} \mathbf{G}_2\|) = \det (S_{22} \mathbf{G}_2 - \lambda \|S_{22} \mathbf{G}_2\| \mathbf{I})
$$

(3.3)

where the norm is defined as

$$
\|\mathbf{S}\|^2 = \text{Tr}(\mathbf{S}\mathbf{S})
$$

(3.4)

The coefficients of $\hat{P}(\lambda)$ will now have the same scaling relative to $S_{22}$ and are still invariant to the transformation $\mathbf{R}$.

Can the coefficients the characteristic polynomial be used in an application? As stated in the introduction, it is important with invariant representations in a pose estimation application. However, it is also important that the representation is descriptive enough, i.e. that it can be a key to different views of an object. Some tests with these coefficients in a pose estimation system similar to the one in section 4 reveals that this kind of invariant representation is not descriptive enough. This is as expected, since the transformation $\mathbf{R}$ both contains affine transformations and perspective projections. In the case of a local region with two line segments, it is
not much left in the representation. Of course, the representation is invariant to translation, rotation and scaling, but it is also invariant to e.g. the opening angle of a corner.

Since it is a balance between invariance and descriptiveness in a pose estimation algorithm, it is important to only be invariant to a minimized set of transformations. For the case of a pose estimation algorithm, perspective projection is not as important as the other ones and can be rejected from this set.

In the introduction we mentioned wide base line stereo as an application for invariant representations. In these kinds of applications, the first step is to find stereo pairs, which are features from each image that are projections from the same point in the three dimensional space. These stereo pairs can be found by a matching algorithm, where a feature is connected with its match in the other image. For a stereo rig with two cameras, the demand on descriptiveness is not as high as for a pose estimation algorithm. For pose estimation, a feature has to be different from a significant part of features estimated from, e.g. up to two hundred to five hundred images and in a stereo application the number of images is only two. Therefore, could this kind of invariant representation be interesting for finding stereo pairs.

### 3.3 Tensor doublet 1

As stated in the introduction, the scene tensor described in section 2 is not an invariant representation, but a combination of the work by Granlund and Moe [17] and the scene tensor will result in a representation that is invariant to translation, rotation and scale. The idea is to calculate invariant parameters based on a geometry including two local regions with two line or edge segments. This representation is called a doublet or more precisely a tensor doublet [39], because the features are detected and represented by using the scene tensor.

From the feature detection process described in section 2.4.3 we get a list of tensors where each tensor is a description of a local region containing two line segments. Each segment is defined in terms of its position, extension and orientation. By extracting the line parameters from two of these tensors the tensor doublet illustrated in figure 3.1 can be computed. The four feature parameters $\alpha_1$, $\alpha_2$, $\beta_1$ and $\beta_2$ can be calculated from the line parameters, where $\alpha$ is the smallest angle between the line segments and $\beta$ is the orientation of each feature relative to the line connecting both features. These four parameters are invariant to both rotation, translation and scaling of the image. The position of each feature is defined by the intersection of the line segments and $\gamma$ is the distance between the features. The $\gamma$ parameter can not be used in an invariant representation since it is not invariant to scale, but it can be useful in the grouping process. It is more robust to use the intersection as the feature's position rather than using the result from the detection process, since that is dependent on contrast, lighting and even the angle between the line segments.

The process of grouping interest points is not an easy task and it is necessary to include some kind of perceptual grouping process in this step. The method
employed here is the same as the one presented in [17] and can be referred to the simplest type of perceptual grouping, where the rule for connecting two features is simply based on the distance $\gamma$ between the features. If the distance for a feature pair is between certain lower and an upper bounds, then the features are joined to build a doublet. Not all features in this region are connected, but only the $k$ closest. The maximal distance should be set to a value that minimizes the probability of a connection between features from the object and the background. A typical value is half the object size. The minimal distance should prevent connecting two tensors estimated from the same feature and the value should be based on the parameters used in the detection process.

The result in from the grouping process is illustrated in figure 3.2, where the features are illustrated with small line segments and the each tensor doublet is illustrated with a line connecting the two features. As is seen in the figure, features from the same object are connected, but there is also situations when features from different objects are connected.

### 3.4 Tensor doublet 2

The tensor doublet described above in section 3.3 has a simple and straightforward implementation. However, it does not use the full power of the scene tensor since a lot of the information is omitted, such as the center of gravity and the covariance matrix of each segment. Adding more information to a representation will increase the descriptiveness of the representation, which is important since a representation has to be descriptive relative to the training data set.

This way of calculating invariants is similar to the one described in section 3.3. The main idea is to introduce invariance by transforming the scene tensor to a certain reference rotation and scale, $\phi_{ref}$ and $\gamma_{ref}$. By doing so the scene tensor will have the same rotation and scale no matter how it was transformed from the beginning. The problem is to find a stable estimate of the rotation and the scale. This is where the doublet can be used again, figure 3.3, where the rotation of the doublet is defined by the angle $\phi$ and the scale is defined by the length $\gamma$. The reference rotation can then be defined as $\phi_{ref} = 0$ and the reference scale as
Figure 3.2: An illustration of the result from the grouping process. The features are illustrated with small line segments and the each tensor doublet is illustrated with a connecting line.

$\gamma_{\text{ref}} = 1$. Consequently, the tensor should then be rotated with $\phi$ and scaled with $1/\gamma$. The steps necessary for calculating an invariant tensor doublet is then:

- Calculate scene tensors tensors and points of interest, section 2.
- Group the tensors into tensor doublets and calculate the orientation $\phi$ and the scale $\gamma$, the upper image in figure 3.4. The grouping process is the same as the one in section 3.3.
- Rotate the coordinate system for each tensor so that the x-axis is parallel with the line connecting the two tensors, the middle image in figure 3.4
- Transform each tensors coordinate system to a common coordinate system preferably somewhere between the two tensors, the lower image in figure 3.4.
- Perform a summation of the two tensors into a tensor doublet.
- Scale the coordinate system by the scaling factor $1/\gamma$. 
Figure 3.3: Definition of the doublet rotation $\phi$ and the doublet length $\gamma$.

Figure 3.4: The steps required for computing a tensor doublet. The upper image is the start condition where the coordinate system and the Gauss window is illustrated. The first step, illustrated in the middle image, is to rotate the coordinate system for each tensor so that the $x$-axis is parallel to the line connecting the two tensors. The last step is to translate the coordinate system to a common point, sum the two tensors and scale according to the distance between the tensors.
3.4 Tensor doublet 2

The only thing that is left in the implementation is how to transform the coordinate system for a scene tensor.

As is described in [30] and section 2, the tensor representation is constructed by two different second order tensors, \( S_{20} \) and \( S_{02} \). \( S_{20} \) is constructed by an outer product between the vector \( x_H \) and itself:

\[
S_{20} = x_H \otimes x_H, \quad x_H = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}
\]  

(3.5)

\( S_{02} \) is on the other hand constructed by an outer product between the vector \( 1^H \) and itself:

\[
S_{02} = 1^H \otimes 1^H, \quad 1^H = \begin{pmatrix} -|l| \\ l_0 \hat{l}_1 \\ l_0 \hat{l}_2 \end{pmatrix}
\]  

(3.6)

In [30] it is postulated that if \( x_H \) is transformed with \( R \) then all vectors \( 1^H \) must be transformed with \( \hat{R} \) defined by

\[
\hat{R} = G^{-1} (R^T)^{-1} G, \quad G = \begin{pmatrix} l_0 & 0^T \\ 0 & x_0 I \end{pmatrix}
\]  

(3.7)

Then we get

\[
x'_H = Rx_H, \quad 1'^H = \hat{R}1^H
\]  

(3.8)

The transformations that rotate \( R_R \), scales \( R_s \) and translates \( R_t \) \( x_H \) with the rotation \( \phi \), the scaling \( 1/\gamma \) and the translation \( t^T = (t_1 \ t_2) \) is expressed as

\[
R_R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{pmatrix}
\]

\[
R_s = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \gamma & 0 \\ 0 & 0 & \gamma \end{pmatrix}
\]

\[
R_t = \begin{pmatrix} 1 & 0 & 0 \\ \frac{t_1}{x_0} & 1 & 0 \\ \frac{t_2}{x_0} & 0 & 1 \end{pmatrix}
\]  

(3.9)

And \( \hat{R} \) follows from equation 3.7. I have chosen to express the different transformations in different matrices, since they will be used in different stages in the algorithm.

The next step is to apply these transformations on the scene tensor \( S_{22} \). \( S_{22} \) is constructed by an tensor products between \( x_H \) and \( 1^H \):

\[
S_{22} = x_H \otimes x_H \otimes 1^H \otimes 1^H = \alpha_1 \otimes \alpha_2 \otimes \alpha_3 \otimes \alpha_4
\]  

(3.10)
Since the tensor product can be interpreted as all possible products between the elements in the different vectors, equation 3.10 can be rewritten as

\[ S_{22} (k) = \alpha_k \otimes \text{vec} \left( \prod_{p \neq k} \alpha_p \right)^T \]  

(3.11)

where \( \prod \) is the tensor product. The operator \( \text{vec} (M) \) produces a column vector from \( M \), where the elements are taken columnwise from \( M \). A full transformation of \( S_{22} \) is implemented by the following recursive formula

\[
S'_{22} = R S_{22} (1) \\
S''_{22} = R S_{22} (2) \\
S'''_{22} = \hat{R} S_{22} (3) \\
S''''_{22} = \hat{R} S_{22} (4) 
\]

(3.12)

Observe that \( S_{22} \) has to be reshaped back to a \( 9 \times 9 \) matrix before equation 3.11 is applied again.

The result from this algorithm is a scene tensor which is invariant to rotation and scale. It is estimated from a region with four segments and it is not completely investigated how the scene tensor behaves for such situations. However, it is clear that certain configurations of four segments will produce a scene tensor with rank three. For these situations the scene tensor will not fully represent the four segments, but instead represent an superposition of the information from the different segments.

### 3.5 Discussions and conclusions

In this chapter, three different invariant representations based on the scene tensor is described. The one in section 3.2, which is based on the coefficients in the characteristic polynomial can be estimated directly from the elements in the scene tensors and does therefore not require any grouping if interest points. The drawback of this representation is the lack of descriptiveness. The other two described in section 3.3 and section 3.4 is based on doublets of features. These representations is more descriptiv than the one in section 3.2, but requires a process of grouping features. This grouping process has some drawbacks, such as an increased number of features. If \( n \) features are found in an image then \( kn \) doublets will be computed from that image, if each feature is connected with the \( k \) closest features. A second drawback is the locality. If a feature is less local, then it is a higher risk that the feature will contain information from the background or an occluding object. The third drawback is the increased sensitivity to noise features. With noise features we mean features from the background or features occurring from changes in lighting conditions relative to some reference light. Invariant representations based on a single interest point \([26, 41]\), a so called singlet, is less sensitive to noise features. The following example will show that. If there are \( n_c \) correct
features and $n_e$ wrong features, then the probability for an incorrect singlet $p(s_e)$ and the probability for computing an incorrect doublet $p(d_e)$ is:

$$p(s_e) = \frac{n_e}{n_e + n_c}$$

$$p(d_e) = \frac{kn_e + kn_c n_e}{k(n_e + n_c)} = \frac{n_e + \frac{n_e n_c}{n_e + n_c}}{n_e + n_c}$$  \hspace{1cm} (3.13)$$

It is hard to see the difference between the different probabilities. Figure 3.5 illustrates the probabilities relative to the number of incorrect features $n_e$ estimated from the image. The number of correct features is fixed to $n_c = 50$, which is a realistic value for the scene tensor. $n_e$ is of course dependent on the characteristics of the objects in the image. The figure shows that the probability for an incorrect doublet is higher than for an incorrect singlet, as expected. The conclusion is that a singlet should be used if possible. This is only possible if most of the information from the image is included into the representation, otherwise will the representation not be descriptive enough. In the case of the scene tensor, we have to use doublets to compute both an invariant and descriptive feature.
Chapter 4

Pose Estimation of Three-Dimensional Objects

4.1 Introduction

Pose estimation of objects is of great interest in several industrial applications, especially in the unsolved bin picking problem. Industrial automation of today demands very dynamic automation systems since the geometry of the products changes faster than before. As a consequence, old systems where the objects are placed in fixtures, will not be sufficient in the future. Instead we need more advanced procedures that can find the pose of objects, guide the robot toward the objects and finally grasp them.

Over the years several algorithms have been developed for view centered pose estimation of objects based on local invariant features [17, 27, 35, 23, 38], where Lowe’s SIFT features [27] are considered state of the art. These features seem to deliver a very stable and accurate pose estimate, but the representation of the local feature is iconic. By using a model based approach to represent these local features, it is possible to have a more compact representation, and it is also possible to extract information about the local area which could be useful in a grouping process.

The approach to pose estimation proposed in this chapter uses the scene tensor in 2D, described in [30, 31] and section 2, as a basis for a set of invariant features. A tensor doublet, [39] and section 3, based on the information from the scene tensor is then used as the invariant representation of the local feature.

The tensor doublet only consists of four parameters which all are invariant to translation, and variations in orientation and scale. In comparison to the SIFT feature’s 128 invariant parameters where more than 50 percent is non zero, the tensor doublet is a very low dimensional feature vector. If the database contains a large number of feature vectors, the lower dimensionality of the feature vector will definitely speed up the rest of the pose estimation procedure.

When the tensor doublets are computed, the mapping from feature parameters
to object state parameters is implemented by a matching and clustering procedure. This chapter will start with short description of the used invariant representation, section 4.2. Section 4.3 will discuss the mapping from feature parameters to object state parameters and the last section will show some results.

4.2 **Compact and invariant representation of local image data**

An invariant representation is very useful for a pose estimation algorithm, since it allows a smaller training data set and therefore a faster training. In section 3, three different approaches to implement an invariant representation from the scene tensor is described. The approach based on extracting four invariant parameters from a tensor doublet, section 3.3, is used in this pose estimation algorithm. That approach is chosen since it generates very low dimensional feature vectors and will consequently speed up the matching procedure.

The method for computing invariant feature vectors starts with estimating a dense field of orientation tensors, where an outer product of gradient vectors is used and a Gaussian window function with $\sigma = 1$. A dense field of scene tensors are then estimated from the orientation tensors, section 2.3, where $\sigma = 2$ for the Gaussian window function in equation 2.26, $x_0 = 1$ and $l_0 = 50$. Then a grouping process starts with the interest points from the detection process in section 2.4.3. In the grouping process there are three parameters; each feature is connected with three other features, only features that are within 50 pixels radius will be connected with the current feature and features closer than 3 pixels will be rejected.

4.3 **Mapping from the representation to object state parameters**

In this approach to pose estimation we have used a matching and clustering procedure to perform the mapping from feature vectors to object state parameters, but it is also possible to use an associative network together with these types of features. The object state parameters are the two pose angles $\phi$ and $\theta$, figure 4.1, the scaling relative to the training view $s'/s$, the rotation in the image plane $\alpha$, and the translation $x, y$.

During training, images are taken from different views of the object using a rotation table. Tensor doublets are calculated for each image and stored in a database, called the prototype doublets in figure 4.2. A label containing the pose angles, $\phi$ and $\theta$, together with the positions for the interest points in the doublet is also stored for each prototype doublet. When a query image, or a test image, is presented to the system, tensor doublets are calculated. These doublets are referred to as query doublets in figure 4.2. Each of these query doublets is then matched with the prototype doublets and for each match a translation $t$, rotation $R$ and scaling $s$ of the object relative to the training image is calculated according
4.4 Results

Figure 4.1: Definition of the two pose angles

to

\[ p_q = t + sR p_p \]  \hspace{1cm} (4.1)

where \( p_q \) and \( p_p \) are the positions of one of the points in the doublet in the query image and in the prototype image, respectively. The transformations have 4 degrees of freedom in total, so one doublet is sufficient to compute the transformations.

All doublets computed from interest points on the objects will vote on the same object state parameters and will therefore cluster in the six dimensional space illustrated in figure 4.2. Of course, other points from the background or e.g. shadows on the object will also form doublets and vote for a object state. It is low probability that these votes cluster and therefore the “correct” votes will form the most legible cluster. This cluster can be found by a mean-shift filtering followed by a mean-shift clustering [5, 13]. A confidence measurement is then calculated for each cluster. This measure is the estimated probability for the cluster mean multiplied by the number of votes in the cluster.

4.4 Results

The system has been trained for the socket in figure 4.3. Images have been taken from different views of this object where \( \theta \) ranges from 0° to 40° and \( \phi \) ranges from 0° to 180°. The step between the training images is 10° for both the \( \phi \) and \( \theta \) variable.

The pose estimation system has been evaluated with the worst case images, meaning the images between the training images. The result is illustrated in figure 4.4. The MAE (mean absolute error) is 1.6° for the \( \theta \) variable and 1.8° for \( \phi \).

The bin picking problem discussed in the introduction often implies in practice that the objects are stacked in a pile. The pose estimation system was evaluated for such a situation with good performance, figure 4.5. Each one of the three objects can be found with good accuracy. The upper leftmost image illustrates the object state derived from the cluster with the highest confidence, the upper rightmost is the cluster with the second highest confidence and the lower leftmost
Figure 4.2: Overview of the query mode. The resulting output is an estimated pose, position, rotation, and scale of the object. KNN refers to the $k$ nearest neighbor method.

Figure 4.3: The object used in the pose estimation test.

is the cluster with the third highest confidence. The white mesh illustrating the object state is the norm of the gradient of the closest training view, which is scaled and translated according to the object state. The system also works with other objects in the background which is illustrated in the lower rightmost image in figure 4.5. Figure 4.6 illustrates the performance when the object has a different scale relative to the training images.

More results from using this pose estimation algorithm will be shown in chapter 7.
4.4 Results

Figure 4.4: Pose estimation test on the socket. The dotted line is the estimated pose and the solid line represents the ground truth.

Figure 4.5: Pose estimation of sockets in a pile and a socket with background. The three first images is actually the same image, where the first image illustrates the object state estimate with the highest confidence, the second image illustrates the object state with the second highest confidence and so on. Clearly the algorithm can detect several objects from one image.
Figure 4.6: Pose estimation with background and different scales.

4.5 Conclusions and discussions

The main difference between this view centered pose estimation algorithm and others [26, 41, 23] is the low dimension of the feature vector, which will speed up the matching procedure. Despite the low dimensional feature vector, the algorithm seems to work well for objects stacked in a pile, object with a cluttered background and objects with different scales. Pose estimation of objects stacked in a pile is especially interesting in industrial automation, for example the bin picking problem.

Since the scene tensor is an representation of multiple line or edge segments, the pose estimation algorithm will work best on objects with features in that category. E.g. objects with a lot of curvature will probably result in very unstable pose estimates.
Chapter 5

Multi-cue Integration

5.1 Introduction

It is very important for a pose estimation algorithm to be robust for situations such as different types of objects and lighting conditions. It is hard or even impossible to meet this goal with only one pose estimation algorithm, since different algorithms have good performance in some situations and bad in others. If several algorithms are used there is a higher probability that at least one succeeds in estimating the object state parameters under the current conditions. If several algorithms are running however, we have to solve the problem of how to integrate the multiple cues from the different algorithms. Best performance of a multi-cue integration system is reached if the local features that the different algorithms are based upon are complementary, e.g. different features such as lines, corners, ellipses or model free patches. In section 5.3 a framework for multi-cue integration for increasing the robustness is presented, where cues from different pose estimation algorithms are integrated using a clustering algorithm described in section 5.2.

Most of the multi-cue integration systems found in the literature, including the one described here, are using multi-cue integration between features or between different algorithms. This type of system seems to improve the robustness rather than increasing the accuracy. Several visual servoing systems use an eye-in-hand setup, where the camera is mounted on the manipulator. If the camera is moved such that the object is seen from different views and the movement is recorded with high accuracy, it should be possible to perform a multi-cue integration over time to improve the accuracy of the object state estimate. In section 5.4, a system for improving the accuracy is discussed, which is based on the same framework used for multi-cue integration between algorithms discussed in section 5.3.

5.2 Mean-shift clustering

A clustering algorithm is used for integrating the cues in the multi-cue integration algorithms explained in section 5.3 and section 5.4. The clustering has to be effi-
cient on multi-dimensional vectors and it should also be able to take corresponding confidence values into account. One algorithm that has these properties is the mean-shift clustering algorithm [13],[5] and [11].

Mean-shift clustering can be described to work in a number of steps.

- Estimate the probability density function for the source of a number of samples by convolving the sample space with a kernel function such as a Gaussian.
- Perform a gradient-descent search on the estimated PDF, starting in each sample, and thereby finding what peak in the PDF that this sample is closest to.
- Finally there are some functions that need to be performed to solve the problem of samples getting stuck of ridges in the PDF etc.

The clustering procedure that is implemented can also handle modular parameters, e.g. angles, see [10].

5.3 Multi-cue integration for robustness

Several systems for multi-cue integration are based on the use of different types of features as cues for getting a more robust system. The integration presented here is different since the cues are results from a number of pose estimation algorithms instead of features. The algorithmic multi-cue integration (AMC) is illustrated in figure 5.1, where the votes from each pose estimation algorithm are put into a voting space with corresponding confidence value and the integration is implemented by using a clustering algorithm. The center of the cluster with the highest confidence will be the result from the multi-cue integration. The confidence value is based on the confidence values of the votes in the cluster and how dense the cluster is. Observe that each pose estimation algorithm should generate several object state estimates, where the additional estimates could come from several objects or noise.

The clustering algorithm used in this type of integration has to be efficient on multi-dimensional vectors and has to take the corresponding confidence value into account. One such algorithm is the mean-shift clustering algorithm, section 5.2.

It is important that the resulting object state vectors from the pose estimation algorithms are unambiguous in the sense of comparing different object states. For example the Euler-angle representation is ambiguous in that sense, but rotation matrices and quaternions are not. We have chosen to work with quaternions because they are a more compact representation and will therefore reduce the computation time.

When algorithmic multi-cue integration is implemented as a clustering of pose estimate results it can behave in a number of different ways, depending on the situation.

- Single algorithm result.
5.3 Multi-cue integration for robustness

Figure 5.1: Ordinary case for multi-cue integration between algorithms

Figure 5.2: Multi-cue integration increasing robustness

- Robust mean between two or more algorithms.
- Algorithmic voting.

The single algorithm result will occur when all algorithms but one have failed to estimate the object state parameters. It is unlikely that the algorithms that have failed will form a cluster in the voting space and therefore the result from the algorithmic multi-cue integration will be the result from the algorithm with the highest confidence. The confidence values should be relative to the quality of the object state estimate and the successful algorithm will therefore have the highest confidence. Of course, the robustness is highly dependent on the confidence measurements for this type of situation.

In the case where two or more algorithms succeed in estimating the object state parameters, the votes from these algorithms will cluster in the voting space.
For this type of situation the result from AMC will be a robust mean of the votes contained in the cluster. A robust mean is here referred to a local peak in the estimated PDF.

In case of occluded objects or when only a very small part of the object is visible in the image we may have the case that the correct pose is ranked only as second or third choice by the confidence of the clusters. One goal of AMC is to make the system more robust to this type of problems. Therefore the system is designed such that two votes which are not the first choice of two algorithms but which are similar should be chosen before a single vote that by itself has a higher confidence, we call this algorithmic voting. The mean-shift clustering implementation uses the confidence of each guess as a weight and therefore gives us exactly the kind of behavior mentioned above. This case is illustrated in figure 5.2 where color-intensity signifies confidence.

Each of the pose cues from the different pose estimation algorithms should have a corresponding confidence measure $c$ which lies in the interval $[0, 1]$. This can usually be obtained by normalizing the confidence measures produced by individual algorithms in a suitable way.

### 5.4 Multi-cue integration for accuracy

Although the cameras of today have gone far in the sense of image quality, there is still some noise in the images. The amount of noise is of course dependent on the camera quality, but also on the camera settings such as gain and exposure time. To analyze how this noise influences the quality of two pose estimation algorithms, 40 images of an object were captured from the same camera position and the pose estimates from these images were compared with the ground truth. In this test we have used the tensor doublet and the log-polar algorithms, which both are based on matching and clustering. The error for the 40 images is plotted in figure 5.3, where the error is the angle $\beta$ between the quaternions that represent the ground truth $q_g$ and the pose estimate $q_p$.

\[
\beta = 2 \arccos \left( \frac{q_g \cdot q_p}{|q_g| |q_p|} \right) \tag{5.1}
\]

The output from the pose estimation does not vary when run on the exact same image and therefore must the variations be due to the image noise. The algorithms were evaluated on low-pass filtered 8-bit gray images with pixel-noise having a variance of just below 1 pixel-level\(^1\). Since the output from the pose estimation algorithms are robust means of cluster centers from the 6-dimensional voting space, see [41, 39, 23], the noise must introduce a shifting of these robust means.

In the experiment discussed above it is clear that there is some noise in the pose estimate result, and from other experiments on the data we also know that

---

\(^1\) Images in range 0-255 with usual values for black around 30 and the lighter areas of objects around 240.
this noise is approximately unbiased. It is therefore possible to decrease the noise by calculating a mean of a subset of the pose estimates. In figure 5.4 the pose estimate result from multi-cue integration between the tensor doublet and the log-polar algorithm is plotted together with the result from using both multi-cue integration between algorithms and with previous estimates. The log-polar algorithm is briefly discussed in section 6.3.3. The dashed curve is the result when the mean is calculated over three pose estimates and the solid curve is the result when the mean is calculated over five pose estimates. It is clear that both the mean and the standard deviation is decreased compared to only using multi-cue integration between algorithms. For example, the mean error has decreased with 42 percent and the standard deviation has decreased with 87 percent when averaging is made over five pose estimates (the solid curve) as compared to only using multi-cue integration between algorithms (dotted curve).

In a robotic system you usually don’t have a number of images from the same camera position, but during movement a number of images can be captured on the way toward the target view. The problem is that the camera has different positions for the different images and the pose estimates will therefore be different. In figure 5.5 two camera positions are shown; one where the object is close to the border of the camera image and one where the camera is centered over the object. In this case the position estimate of the object will be completely different and it is not possible to calculate a mean between the two different pose estimates. Instead it is possible to transform pose estimate from camera position one to camera position two.

However, it is not only the position that is incompatible for cue integration, the pose estimates are also wrong as can be seen in figure 5.6. This is due to the
Figure 5.4: Variation in pose estimate when previous estimates are integrated.

camera not looking directly at the object. The pose estimate says the object is rotated even though the object itself is orientated in the same way all the time. Before we can do anything else we have to correct the pose estimates for this effect. The correction in \( \theta \) and \( \phi \) angles is calculated as

\[
\theta_{\text{corr}} = \text{arg}(x_{\text{mm}} - iy_{\text{mm}}) \quad \phi_{\text{corr}} = \text{arg}(z_{\text{std}} + i\sqrt{x_{\text{mm}}^2 + y_{\text{mm}}^2}),
\]

where \( x_{\text{mm}} \) and \( y_{\text{mm}} \) are the pose estimates in position described in millimeters and \( z_{\text{std}} \) is the distance in millimeters to the plane through the object and parallel to the image plane. The pose estimates are corrected with these values by a transformation matrix. Figure 5.7 gives a feel for how large this effect can be in this kind of system.

In figure 5.8 an illustration of the different coordinate systems is seen; the world coordinate system, the camera coordinate system for two different camera positions and the object coordinate system. In this figure we have used the notation \( W T_{C_1} \), where \( W \) stands for the world coordinate system, \( C_1 \) stands for camera position number one and the full expression stands for camera position number one relative to the world coordinate system.

From the robot controller and the hand-eye calibration we get the transformations \( W T_{C_1} \) and \( W T_{C_2} \). The pose estimate \( C_1 T_{\text{Obj}} \) can then be transformed to camera position two by:

\[
C_2 T_{\text{Obj}}' = [W T_{C_2}]^{-1} W T_{C_1} C_1 T_{\text{Obj}}
\]

(5.2)

If the pose estimate is stored together with the current camera position for each step during movement toward the target view it is possible to perform multi-cue integration with previous estimates since we can transform them into the new
Figure 5.5: The setup for multi-cue integration between several views.

Figure 5.6: Errors in pose estimates due to relative camera-object position.

coordinate system. We call this kind of integration temporal multi-cue integration (TMC). For a certain camera position all previous pose estimates are transformed using equation 5.2. The confidence measurement for these pose estimates are then weighted with the weight $w_i$

$$ w_i = f^{(n-i)} $$  \hspace{1cm} (5.3)

where $i$ is the step number for the previous pose estimate, $n$ is the current step number and $f$ is the forget factor. All previous pose estimates are then used as cues in the same manner as described in section 5.3.
Figure 5.7: Differences in seen pose due to relative camera-object position.
Figure 5.8: The different coordinate systems for the setup used in multi-cue integration between several views.
Chapter 6

VISATEC Demonstrator
6.1 Introduction

Currently there are a number of vision-based products that solve the pick-and-place problem available on the commercial market. The most successful of these pick objects from a horizontally placed and well organized pallet or from a conveyor belt. Systems able to pick directly from a conveyor belt, where objects are randomly organized, use some kind of mechanic solution to get the objects segmented and non-occluded. There has also been developments towards solving the bin-picking problem, where a vision guided robot should be able to pick objects from a unorganized pile. Such systems commonly use some kind of range data, having an active sensor which is either stationary, mounted on an external axis to the manipulator or on the manipulator itself. The VISATEC demonstrator tries to go towards the goal of solving the bin picking problem without using expensive range sensors. The demonstrator should be seen as both some first steps in that direction and also as a testbed for the algorithms developed within the VISATEC project. The VISATEC demonstrator is implemented in an industrial environment at the laboratory of the industrial partner in the VISATEC consortium.

6.2 Assumptions

In the design of the VISATEC demonstrator a number of assumptions are made. This is due to the goal of the VISATEC project, which is not to build a working bin-picking prototype but rather to investigate the performance of the pose estimation algorithms and the multi-cue integration.

The assumptions are:

- One limitation of the algorithms used in the demonstrator is that they cannot estimate the distance between the camera and the object. Therefore all objects are located on a plane and 450 mm above that plane there is a second plane, the reference plane, in which the camera moves during the two first system states random movement in the reference plane and center the object in the image.

- It is assumed that the lighting conditions can be controlled, i.e., it is not subject to random changes over time.

- The optical axis of the camera and the z-axis of the robot tool coordinate system are parallel, figure 6.2.

- There are no obstacles during the robot movements.

6.3 System description

In this section, the basic setup of the VISATEC demonstrator and the strategy which is used to find and grasp an object is described. An overview of the components in the demonstrator can be seen in figure 6.1.
6.3 System description

![VISATEC system architecture diagram](image)

Figure 6.1: VISATEC system architecture

### 6.3.1 Demonstrator hardware

The demonstrator is an industrial robot with a camera attached to the manipulator arm. As a consequence of the project form, different software and hardware components have to be connected, and in short the configuration is as follows. There are two PCs, one running the robot communication software package and the other one running the VISATEC software package. Both PCs are connected by Ethernet. The connection between the PC with robot communication software and the robot controller is realized by a standard field-bus interface (INTERBUS). The PCs communicate via a proprietary TCP/IP based communication protocol. The camera is a gray scale camera and has a fixed lens. A gray scale camera requires less light and has a better SNR than a color camera. The use of a fixed lens implies that there is no automatic way to change zoom, the focus of the lens, or the focal length. This is a type of camera/lens configuration that is found in most of the industrial vision applications around the world. Rectification of the image to take care of lens distortions or other geometric distortions generated by the camera has been implemented.
6.3.2 Calibration

There are two calibrations of the system that need to be performed. First, we need a camera calibration that estimates the camera projection matrix and the lens distortion parameters. This calibration is made using a pin-hole camera model and by calculating the parameters explicitly by means of a calibration grid, see i.e. [43]. The estimated parameters are then used to rectify the images as mentioned in section 6.3.1.

Second, a hand-eye calibration is needed which estimates the position of the camera coordinate system relative to the robot tool coordinate system. Figure 6.2 shows a schematic drawing of how the camera is attached to the robot arm. The camera is the darker box mounted on the manipulator arm and its coordinate system is $F_C$. The coordinate systems $F_W$ and $F_{Tool}$ are predefined in the controller of the manipulator. The goal of the hand-eye calibration is to find the transformation that is marked $T_{Tool}T_C$ in the figure, i.e. the transformation between $F_{Tool}$ and $F_C$. This not an easy task and there are several procedures for estimating this transformation, which typically use a well defined calibration object [34].

However, in the VISATEC demonstrator it is assumed that the camera is attached to the robot in such a way that the optical axis is parallel to the tool axis, which reduces the transformation $T_{Tool}T_C$ to only a 3D translation. By removing the camera house it is possible to estimate this translation by an ordinary robot tool calibration. In this calibration procedure an operator moves the center of the camera chip toward a calibration pin. This procedure is repeated a number of times but with different starting positions of the robot. The 3D translation is then calculated by the robot controller box.

![Coordinate systems for robot](image)

Figure 6.2: Coordinate systems for robot
6.3 System description

The type of hand-eye calibration used in the VISATEC demonstrator is probably not the most exact one, but this is not a problem since the system uses visual servoing and view based pose estimation algorithms.

6.3.3 Pose estimation algorithms

The camera image is used as input to two algorithms for pose estimation. The following algorithms are implemented in the system, for a detailed description see the references.

- **2D scene tensor** which finds points of interest corresponding to 2D corners and estimates parameters of the corners to find rotation and scale invariant tensor doublets. See section 4 and [31], [39], [32] for more details. In the following, this algorithm is referred to as the Tensor algorithm.

- **Log-polar** which uses the standard Harris [18] point-of-interest detector combined with a log-polar transform of the corresponding local region to find rotation and scale invariant feature points. See [41, 40] for more details. In the following, this algorithm is referred to as the Log-polar algorithm.

6.3.4 How to train the system

The demonstrator system is trained by changing the position, orientation and pose of the camera relative to a specific object. The robot is given a reference position in its world coordinate system and an object is placed there. The object must during training be located on a smooth background such that no extra features that do not belong to the object are detected by the algorithms. The robot then moves the camera on a half-sphere around this reference point and collects projections from the object from different views. Due to physical limitations of the manipulator used in the demonstrator, only the top-most part of the sphere was sampled. The sampling should be made in such a way that the sampling points are as evenly distributed on the sphere as possible.

![Figure 6.3: Example of sphere sampling](image)

The sampling used in the demonstrator works in following way
1. Specify start and stop-values for $\phi$ and $\theta$ as well as number of steps to take in the $\phi$-direction.

2. Calculate the step-length in $\phi$-direction by $\phi_{steplength} = (\phi_{end} - \phi_{start})/\phi_{steps}$.

3. For each step in the $\phi$-direction calculate the part of the circumference of a horizontal circle and divide it with the step-length in $\phi$. This gives the number of steps in $\theta$ at the current $\phi$ by $\theta_{steps} = c/\phi_{steplength} + 1$, with $c = r(\theta_{end} - \theta_{start})/(2\pi)$ and $r = \sqrt{1 - \cos^2(\phi_{current})}$. Put a sample point at each step in $\theta$ and $\phi_{current}$.

The captured images together with the angles of the tilt of the robot tool are sent to the pose estimation algorithms, section 6.3.3, for processing. As a result of this processing, the algorithms build up their databases of features.

The advantage of using the manipulator as the tool for training, rather than a separate turntable, is that in every setup that uses such a system the robot is already available. In this way, the lighting conditions during training is as similar as possible to what they will be like when the system operates, thereby providing a more stable system.

After the build-up of the databases for the different pose estimation algorithms a target view must be calculated. The term target view is used to refer to the viewing direction which will give the most accurate estimate of the object state in a moving robotic system. The target view also makes it possible to specify a “blind” grasping movement in the robot controller without calibrating a tool in the controller. In a running system the target view should be used in the blind-grasping capacity. However, in the demonstrator it also serves a third purpose. We need to move somewhere besides the top view while still retaining ground-truth for the system evaluation, see section 7.2.3.

The estimation of object state is normally dependent on how many local features can be seen from each view and therefore there are one or more views that give the best estimate since these views contain many features which in addition are unique for that view. Initial testing showed that the choice of target view must be made such that it lies in the center between points where the half-sphere was sampled. Furthermore those sample points should be from views with a high mean and a small variance of the number of local features present in those views.

![Figure 6.4: Possible choices for a target view](image)

Figure 6.4: Possible choices for a target view
Looking at figure 6.4 we see three points marked with crosses and numbered 1-3 on the sampled half-sphere that could be selected as target view. The circles represent sampling points, i.e., training views. For each of them the following will happen:

1. Choosing a training view as the target view would give us a very exact pose estimate once we were in that position. The problem is reaching that position in a moving system. All positions that are close to the target view, within an imaginary circle around point 1 with a radius of about the distance between point 1 and 3, would also give us the pose estimate of sample point 1. This is due to the fact that we can not have pose votes on anything but the training views. Pose estimates with values other than those from the training views come from interpolation between votes from a number of different training views.

2. Having this position as target view is slightly better than in case 1 above. However, still a large number of votes is likely to fall onto the closest training view and the pose estimate would therefore be biased towards that pose.

3. Case 3 gives us a reasonable chance that the votes will be evenly distributed between neighboring training views and therefore can be interpolated well. This also means that if the robot positions itself slightly off this position, the pose estimate will be something else than the target view and therefore the robot can try to move towards the target view.

From this we conclude that the third option is the best way to choose the target view.

Remarks The target view could also be found after the training procedure by sampling the half-sphere again with somewhat shifted sampling positions and evaluating the trained system with those images. Then checking the confidence measurements would yield the target view. If the ground truth is available it could be even better to measure the error in the object state estimate and use this for detecting the optimal target view. This would however take much longer time and we would probably find a better target view by using the third alternative in figure 6.4.

It might be that different algorithms choose different target views for an object. There is no straightforward solution to this problem, but an easy and reasonable approach is to use the target view from the “best” algorithm. Which algorithm is best is decided on by measuring how small the error of the estimate is in those target views that are in the proposed set of target views.

6.3.5 Random movement in the reference plane

When starting the VISATEC demonstrator the system will enter the state random movement in the reference plane. In this state, the camera is always looking straight down and an exploration of the scene is performed with random movements of the camera in the reference plane. The random movement will continue until an object is found. If an exploration of the scene is used then the camera’s
field of view does not have to cover the whole scene. The result of this is that the
object will cover a larger part of the image and the pose estimation task will be
simpler due to more pixels on the object and less background.

After each random movement an image is captured and sent to the active
pose estimation algorithms. The estimates from all algorithms are integrated by
means of clustering (section 5.2) and when there is at least one cluster which has a
confidence $c$ larger than some threshold $t$, this is taken as an indication that there
is an object for which the system can estimate pose parameters and the system
will then enter the state **purposeful movement**. If there are several clusters with
$c > t$, the one with largest $c$ is used.

The assumption is that $t$ can be set to a value which allows us to assume that
an object for which $c > t$ is possible to grasp, e.g., is not occluded by other
objects. If several objects are placed in a pile, some of them are oriented such that
they can be grasped whereas others cannot. The system starts by removing the
top objects, that are possible to grasp, and eventually the other objects will also
become possible to grasp.

### 6.3.6 Purposeful movement

The system state **purposeful movement** has the goal to move the camera to the target
view $O_{obj} T_{CT}$, section 6.3.4, where the direction of the movements are based on
the object state estimates $C T_{obj}$. Refer to figure 6.5 for definition of transforma-
tions. When the target view is reached a geometrical relation between the object
and the camera is established and a possible gripping process can start. This
approach is identical to **position based visual servoing** [25, 7], with the assumption
that the distance to the object is known, section 6.2.

The purposeful movement contains of two different parts:

- Centering of the object in the camera image
- Purposeful movement on a half-sphere toward the target view

In the first part, the system will try to center the object in the camera image
by minimizing the distance, $d$, between the calibrated image center and the object
position. The object state estimate also includes an estimate of the objects ori-
etation, $\alpha$, in the image. The camera is also rotated such that $\alpha$ is minimized.
When the distance $d$ and the orientation $\alpha$ are below a certain threshold the pur-
poseful movement will enter the second and last part **purposeful movement on a half-sphere**. The purpose of first centering the object in the camera image and
then move on the half-sphere is that the view angle estimates often are worse when
the object is not centered. However, the position estimate is still of good quality
and by first centering the object in the image the risk of moving the camera to
completely wrong positions will be minimized.

When the object is centered, the system moves the camera on the half-sphere
towards the target position, $W T_{CT}$. The target position is dependent on the target
view, $O_{obj} T_{CT}$, and the object state estimates, $C T_{obj}$:

$$W T_{CT} = W T_{C}^{-1} C T_{obj} O_{obj} T_{CT}$$

(6.1)
For some views of the object it is possible to get bad object state estimates, but the target view is chosen such that it is especially good for the pose estimation algorithms, section 6.3.4. This characteristic will result in a converging system behavior, where the error between the current camera position and the target camera position decreases over the number of steps. When the camera movements are sufficiently small the purposeful movement is finished.

In section 6.2 it is mentioned that the distance between the camera and the object is known and fixed, however, this is not correct if an object is lying on top of another object. As a consequence, the object may not be centered in the image when the target view is reached. By using this object translation it is possible to calculate the distance error between the camera and the object and correct for it. This is however not implemented in the demonstrator.
Chapter 7

Demonstrator Experiments

7.1 Introduction

When the VISATEC demonstrator had been implemented as described in chapter 6, a series of tests were performed. The tests were made to evaluate the performance of the system and of the algorithms that were part of the system. Three different sets of experiments have been done. First we have real data with known ground truth, the second group uses real data but there is no ground truth and the third is on all synthetic data. This chapter starts by describing the experimental setup used during testing of the demonstrator. The chapter will then continue to describe results for the different types of data.

7.2 Experimental setup

A schematic of the experimental setup used can be seen in figure 7.1. The lights marked A and B in figure 7.1 are covered with a plastic cover for making the light more diffuse. The lights are placed very low so it gives a low-angle lighting. The low-angle lighting combined with having a number of small spotlights in the ceiling, not visible in 7.1, gives a somewhat harsh lighting environment since it casts shadows etc.

7.2.1 Variables

The properties of the test setup that have been changed during the testing phase include the

- object
- lighting
- background
- number and type of other objects present
Figure 7.1: Schematic of experimental setup

- level of occlusion of the object to be detected
- active pose estimation algorithm and state of multi-cue integration.

The objects that have been tested for are

- A plastic power socket with two outlets which is referred to as socket.
- A cast metal piece of an electrical motor which is referred to as bug.

It should be noted that the two objects are different in size as well as different in the type of internal structure. Even more important is perhaps the differences in surface properties such as reflectiveness and color. This suggest that the results will generalize to a large set of object types. The lighting conditions used during the tests are

- Both lights A and B in figure 7.1 turned on, referred to as light 1.
- Only light A in figure 7.1 turned on, referred to as light 2.

The first of the above is the light setting used during training. It should be emphasized that due to the low-angle property of the light, the second light condition is a significant change in lighting. The background has been changed between:

- A black velvet blanket, referred to as background 1. This is the same background that was used during training.
- A conveyor belt typically used in industrial transportation systems, background 2.
- A calibration plate with a mixture of very high contrast black and white areas, background 3.
7.2 Experimental setup

- A piece of a regular cardboard box with some printed plastic tape on it, background 4.

The combinations of pose estimation algorithms and multi-cue integration used during the test were

- Tensor-based pose estimation.
- Log-polar-based pose estimation.
- Both of the above with cue integration between them active.
- All of the above with temporal multi-cue integration active.

It should be emphasized that not all combinations of the above mentioned properties have been changed during the tests. A few examples of typical images can be seen in the figures 7.2 and 7.3.

7.2.2 Parameters and training

The pose estimation algorithms were trained by using 94 images for the socket and 177 for the bug. The objects were placed on background 1, i.e. the black velvet, during capture of the training images.

All algorithm-specific parameters were kept at the same values throughout all experiments. The settings for the multi-cue integration were also kept constant throughout the experiments. These two statements are very important since they indicate how stable the algorithms really are. It is always easier to get things to work well on a subset of the problem if it is allowed to tune the parameters of the algorithms and this was something we wanted to avoid doing.

The number of steps for each part of the purposeful movement described in section 6.3.6 was set to 7 for movement in a plane and to 6 for the movement on a half-sphere. This enables us to evaluate if the system converges or if it oscillates. In a real system a termination condition based on the predicted level of accuracy should instead be used. The start position was in most tests set to the same position, a position where only part of the object with known ground truth was visible. Having only a single start position was deemed to be sufficient since the system did not follow the same path all the time, as will be seen in the results later on. The forget factor for the temporal multi-cue reweighting of old cues was set to 0.7.

7.2.3 Measured quantities

As previously stated there were three groups of experiments performed, namely

- Real images with ground truth.
- Real images without ground truth.
- Synthetic images (with ground truth).
Figure 7.2: Example images captured during the experiments with ground truth.
7.2 Experimental setup

Object: bug  
Background: 2, 3, 4  
Light: 1, 2

Figure 7.3: Example images captured during experiments without ground truth.

The real images were all captured under circumstances varied as described in section 7.2.1 and exemplified in figures 7.2 and 7.3. For the test with synthetic images a model of a Tetra-pak milk box with texture mapping was used, a typical rendered view can be seen in figure 7.4. The reason for having not only one point with real images in the list above is that tests with a more complex background requires removing the object from its training position and thereby removing the ground truth. The ground truth comes from storing the robot positions used during training and comparing that with the current position in each step of running the demonstrator. Since the system tries to move such that the relative displacement in all object state parameters is zero it should move towards the learned position. Each experiment with ground truth was carried out at least 5 times.

In the forth-coming presentation of results we will use the following error measures as for the Cartesian position

$$
\epsilon_{pos} = \sqrt{(x_t - x_c)^2 + (y_t - y_c)^2 + (z_t - z_c)^2},
$$

(7.1)

and for angular position

$$
\epsilon_{ang} = 2 \cos^{-1}(\hat{q}_t^T \hat{q}_c),
$$

(7.2)

where we used subscript $t$ for training position, subscript $c$ for current position and $\hat{q}$ is a quaternion of the rotation in column vector format. The same error measures were used in the synthetic tests. In the synthetic tests, every position error is however in units instead of in millimeters since the artificial world was not converted to a physical unit.

The experiments without ground truth were carried out after the tests with ground truth and were evaluated qualitatively from the experience from the pre-
vious runs with ground truth. The outcome of those experiments could only be failure or success.

7.3 Results from experiments with ground truth

When examining this section the reader should remember that ground truth is only available if the object is not moved after training of the system as mentioned in section 7.2.3.

7.3.1 Guide to results

In the following figures we will see results for single algorithms; the tensor doublets (T) and the log-polar algorithm (LP) as well as results when using algorithmic multi-cue integration between these two algorithms (T+LP). In cases where the legend includes “+TMC” temporal multi-cue was active.

In figures containing 2 plots the left plot is of the mean angular error, equation 7.2, and the right plot shows the Cartesian error, equation 7.1.

In figures with 4 plots, left and right in the top row are the same as in figures with 2 plots, i.e. angular error and Cartesian error respectively. The second row shows the standard deviation for angular error and Cartesian error.

In the figure that has 3 plots the angular error for all separate runs in a set are plotted to give a view how the results from the runs are distributed.

In table 7.1 the number of completed test-runs not using temporal multi-cue integration is shown. It should be noted that completed does not necessarily mean successfully completed, only that the test run was completed with the full number of steps and therefore logged by the program. If a failure of some sort occurred during a test run it will be noted in those tables. The number of completed test-
runs made with temporal multi-cue integration active can be seen in table 7.2. In table 7.3 the number of completed test runs made in the synthetic environment can be seen. The same lighting was used in all synthetic experiments. The second row includes temporal multi-cue integration. Since initial synthetic tests showed that the system had exact repeatability on images with no noise, it was decided to instead vary the starting position to see the variation in the taken trajectory towards the target view. For each of the 18 tests per setup, 9 different starting positions were used, i.e. two runs per start position per setup were completed.

The structure of the plots is designed to show the difference between using TMC and not using TMC, as well as the difference between using AMC and only using single algorithms. The figures are divided into the following sets:

- Figure 7.5 to figure 7.8 each show the result for one pose estimation algorithm or AMC without TMC together with the result for the same algorithm when using TMC.
- Figure 7.18 to figure 7.20 shows the same thing as above, but for the synthetic milk box.
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Method</th>
<th>Tensor</th>
<th>Log-pol</th>
<th>Multi-cue</th>
<th>T+LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>milk / synthetic background 1 / no other objects</td>
<td>Light 1</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Light 1 (TMC)</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Tests made in synthetic environment.

- In figure 7.9 to figure 7.17 the result for each pose estimation algorithm is plotted against AMC.
- Figure 7.22 and figure 7.23 shows the same thing as above, but for the synthetic milk box.
- In figure 7.21 results for all test runs for each pose estimation algorithm and for AMC are plotted.

The purpose of the figures referred to in the first two points is to show the benefit of using TMC. The figures mentioned in the third and the fourth point illustrate the characteristics of using AMC compared to not using AMC. The purpose of the figure in the last point is to illustrate the variance for the different pose estimation algorithms and when using AMC.
Figure 7.5: A comparison between using and not using TMC for the tensor algorithm. The result is for the socket with lighting 1 and no other objects in the scene.
Figure 7.6: A comparison between using and not using TMC for the log-polar algorithm. The result is for the socket with lighting 1 and no other objects in the scene.
Figure 7.7: A comparison between using and not using TMC with AMC (i.e. T+LP). The result is for the socket with lighting 1 and no other objects in the scene.
Figure 7.8: A comparison between using and not using TMC with AMC (i.e. T+LP). The result is for the socket with lighting 2 and no other objects in the scene.
Figure 7.9: A comparison between the pose estimation algorithms and AMC (i.e. T+LP). The result is for the socket with lighting 1 and no other objects in the scene.

Figure 7.10: A comparison between the pose estimation algorithms and AMC (i.e. T+LP). The result is for the socket with lighting 2 and no other objects in the scene.

Figure 7.11: A comparison between the pose estimation algorithms and AMC (i.e. T+LP). The result is for the socket with lighting 1 and other objects present in the scene.
Figure 7.12: A comparison between the tensor estimation algorithm and AMC (i.e. $T + LP$). The result is for the socket with lighting 2 and other objects present in the scene.

Figure 7.13: A comparison between the pose estimation algorithms with TMC and AMC (i.e. $T + LP$) combined with TMC. The result is for the socket with lighting 1 and no other objects in the scene.

Figure 7.14: A comparison between the pose estimation algorithms with TMC and AMC (i.e. $T + LP$) combined with TMC. The result is for the bug with lighting 1 and no other objects in the scene.
Figure 7.15: A comparison between the pose estimation algorithms with TMC active. The result is for the bug with lighting 2 and no other objects in the scene.

Figure 7.16: A comparison between the pose estimation algorithms with TMC and AMC (i.e. T+LP) combined with TMC. The result is for the bug with lighting 1 and other objects present in the scene.

Figure 7.17: A comparison between the pose estimation algorithms with TMC and AMC (i.e. T+LP) combined with TMC. The result is for the bug with lighting 2 and other objects present in the scene.
Figure 7.18: Comparison between using and not using TMC on pose estimates from the tensor algorithm for the synthetic milk box.
Figure 7.19: Comparison between using and not using TMC on pose estimates from the log-polar algorithm for the synthetic milk box.
Figure 7.20: Comparison between using and not using TMC on pose estimates integrated with AMC (i.e. T+LP) for the synthetic milk box.
7.3 Results from experiments with ground truth

Figure 7.21: One plot per algorithm showing spread between runs with and without TMC active for the synthetic milk box.

Figure 7.22: Comparison between single algorithms and AMC (i.e. T + LP) for the synthetic milk box.
Figure 7.23: Comparison between single algorithms and AMC (i.e. T+LP) all using TMC for the synthetic milk box.
7.3 Results from experiments with ground truth

7.3.2 Conclusions

Conclusions for temporal multi-cue integration (TMC)

As stated in section 5.4, a system should be able to generate more accurate pose estimates if cues from several views of the observed object were used. In figure 7.5 to figure 7.8 and figure 7.18 to figure 7.20 we have plotted the result for each pose estimation algorithm together with the result for each algorithm using TMC. Both the mean accuracy and the standard deviation between the different plots are better with TMC than without TMC for all algorithms. From these very convincing results, we can draw the conclusion that using votes from several views will both increase the accuracy of the system and improve the repeatability of the system, i.e. a lower standard deviation for the different test runs. This conclusion has led to the decision that TMC should be used as a default setting for the rest of the tests.

In the synthetic test in figure 7.21 it can be seen that TMC works to stabilize the results and reduce the variation in the taken trajectories. How much of a low-pass character TMC has depends on the forget factor \( \gamma \), which is a temporal weight that adjust the influence of older pose estimates. In initial tests we tried a few different forget factors and \( \gamma = 0.7 \) seemed to be a quite good choice.

Although the results for TMC is very good it is important to understand that the performance of this algorithm is highly dependent on the 3D position estimate of the object. In these tests, the distance to the object (z-coordinate) is fixed and known by the system. It is therefore no error in the z-coordinate, which definitely improves the result. Another point that could be a problem is if one pose estimate during the purposeful movement toward the target view has large errors. When not using TMC this is either no problem or a very severe one, i.e. the system will either have a second chance in the next iteration assuming that the object is still in the image or it will fail completely if the robot moves such that the object is no longer in the next image. When using TMC the situation is somewhat different. The large pose estimation error will remain in memory and affect intermediate results and possibly also the end result. The effect of this error is reduced by the forget factor, since the influence of older pose estimates is reduced. However, most important is that this wrong estimate will not be part of the most prominent cluster once a few correct pose estimates have been made. Despite all these potential problems, we still believe that TMC significantly improves the result and should always be used when possible.

Conclusions for algorithmic multi-cue integration (AMC)

In section 5.3 it is stated that a system using algorithmic multi-cue integration should be more stable to changes in light, background and target object, since cues from several algorithms are used and integrated. One thing that we took extra care to monitor during the experiments was that the confidence measures for the different algorithms had the same scale. This is important since it can make all assumptions on the properties of AMC totally wrong due to an imbalance between the algorithms.
In figures 7.9 and 7.10 we see results for the socket with two different lighting conditions plotted. In these plots most of the differences are seen in the Cartesian error. The tensor-based algorithm shows a worse result for light 1 than for light 2 which is strange in itself but not relevant for an evaluation of AMC. The log-polar-based algorithm shows a worse result for light 2 than for light 1 which is expected, since a different lighting condition than the training situation is a harder problem. The interesting part of the results is that AMC works for both lighting conditions. It makes the results better or at least not worse than the worst algorithm as predicted in section 5.3. Therefore it seems like the statement that AMC should make the system more robust against light changes holds.

In figure 7.11 and figure 7.12 is the result for the socket with other objects in the scene illustrated. The only thing that varies between the figures is the lighting condition. For those two situations we can see that AMC introduces robustness to the system. For instance, when the tensor-based algorithm has a quite bad Cartesian error for light 1 the AMC result is still very good. Moreover, the log-polar based algorithm fails for light 2 but still the AMC result is stable. Thus we conclude that AMC makes the system more robust to failures by single algorithms, as stated in section 5.3.

In the last result for the socket, figure 7.13, is TMC used for both the single algorithms and for the case of multi-cue integration between algorithms (AMC). This figure illustrates that it is still useful to use AMC when TMC is used.

Figure 7.16 is especially interesting, since AMC significantly improves both the angular error and the Cartesian error. This could be the case, when the pose vote with the second highest confidence for each algorithm is the “correct” one and for the case of AMC these will cluster and be the end result. However, it is difficult to say if this situation has occurred or not since the different results comes from different test runs.

From the synthetic tests in figure 7.22 and figure 7.23 we can see that AMC is almost as good as the best algorithm. In a synthetic test the situation is as good as it can be for the different algorithms and therefore the impact of AMC is not significant. It is still interesting to see that AMC is not worse than the worst result.

The conclusion from all these tests is that AMC makes the system more robust to different objects, lighting conditions and other objects in the scene. We have cases where one algorithm fails and AMC still works. There are situations where both algorithms work, but still it seems like AMC improves the result or at least is close to the “best” algorithm.

Conclusions for the tensor algorithm

On a single algorithm level the tensor algorithm has been evaluated together with the log-pol algorithm. It is hard to say if one algorithm is better than the other one, since the performance is dependent on the actual parameter settings and the type of objects used in the test. What we can say is that both the log-pol algorithm and the tensor algorithm works most of the time and that the pose error is usually down to a couple of degrees and the position error is below three
7.4 Results from experiments without ground truth

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Method:</th>
<th>Tensor</th>
<th>Log-polar</th>
<th>Multi-cue</th>
</tr>
</thead>
<tbody>
<tr>
<td>bug / background 2 / other objects in scene</td>
<td>Light 1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Light 2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug / background 3 / other objects in scene</td>
<td>Light 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug / background 4 / other objects in scene</td>
<td>Light 2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bug / background 5 / other objects in scene</td>
<td>Light 2</td>
<td>1 (with failures)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Test without ground truth to test the robustness of the algorithms.

to two millimeters. For some situations is the position error below 0.5mm, which is considered as a extremely good position estimate. The performance of the tensor algorithm is of course dependent on the type of object, e.g. objects will a lot of curvature will probably produce unstable object state estimates. This is why multi-cue integration between different pose estimation algorithms is so powerful, since no single pose estimation algorithm can be optimal for all kinds of objects.

7.4 Results from experiments without ground truth

Some tests were also performed with the final demonstrator system in cases when we had no ground truth. Example images from test runs without ground truth can be seen in Figure 7.3. The number of completed test runs without ground truth can be seen in Table 7.4. The reason for there being so few runs with the tensor-based method invoked is that the current implementation has not been optimized for speed at all and those tests took too long time to run. Besides the completed test runs in Table 7.4 a large number of tests can be said to have been run. The reason for those not being in Table 7.1, Table 7.2 or Table 7.4 is mainly due to the fact that those tests were run during testing of the system implementation and therefore not completed. This section deals with experiences gained during all these test runs.

For the test runs in Table 7.4 the system was deemed to be working as it should. The methods seem to be able to cope with both changing background, clutter and illumination. Illumination changes are the ones that seem to cause the most problems. This is not so strange since changes in illumination affect most of the features in the image and adding other objects to the scene only adds new features or occludes old ones.

With the low-angle light in the demonstrator setup and with internally varying height of objects, rotation in the image plane apparently changes a lot in the image, e.g. the shadows that are cast. This kind of rotation seemed to be equal to a quite severe lighting change. This was what the system had the most difficulty with
during the tests. During some test runs the system failed completely, i.e. a single algorithm test run had the wrong result as a first guess, see section 5.3.

During these runs as well as during development runs we have many times observed the behavior that two non-first guesses group together during multi-cue integration to form the most prominent cluster and therefore stabilizing the system as discussed in section 5.3.

Finally as stated in section 5, the algorithms used for multi-cue integration should return several votes. These votes could come from the object of interest, other objects in the scene or even noise. If this would happen, then the result when using AMC could be much worse than the result for a single algorithm. It seems from our experience that there is a low probability that the votes from noise will cluster in the voting space and form a cluster with the highest confidence. We can say this since we have not to our knowledge observed such an effect during our experimentation with the system.
Chapter 8

Summary

8.1 Summary and discussions

This thesis describes methods useful for a bin picking application, where the core of such a system is stable and accurate pose estimation of objects. Of course, we also need intelligent grasping procedures and obstacle avoidance, but this is out of scope for this thesis.

A pose estimation algorithm needs a number of building stones. Probably the most important one is a descriptive and invariant representation of local features. This is a “moment 22” situation since increased invariance decreases the descriptive-ness and vice versa. In this thesis we have used the scene tensor to implement an invariant representation. Several tests have shown that the scene tensor can represent local regions with multiple segments and the main conclusion from these tests is that the scene tensor works well if the model fits the data. A model of multiple line segments contains a wide range of features, such as corners, crossings, etc. These features are often present in real images, but they are definitely not ideal. With ideal, we mean extremely sharp corners with no curvature and lines that is only one pixel wide. The scene tensor theory works best for these ideal situations, but we have shown that it is also successful for the non-ideal situations.

However, the scene tensor is not invariant in itself and by grouping interest points into tensor doublets, we can compute four rotation and scale invariant parameters. This is a low dimensional feature vector compared to e.g. the SIFT features [26] which has more than one hundred elements. The tensor doublet is combined with a matching and clustering algorithm and it is shown that stable and accurate pose estimates can be estimated despite occlusion of the object, different backgrounds and changes in lighting.

An industrial system for bin picking have to cope with several different types of objects and environments. We believe that no single pose estimation algorithm can solve such a difficult task and one solution is to use cues from several different algorithms. In this thesis we present a framework for multi-cue integration between different pose estimation algorithms and the experiments shows that the stability of the system is increased. The same framework can also be used for increasing
the accuracy of the pose estimates by integrating cues from several views of the object.

There are of course several problems left to solve until a generic solution to the bin picking problem is available. Several people believe that the problems will be solved by extremely accurate and stable methods for detecting and representing local features. A large part of the computer vision community have tried to find these extremely stable feature detection algorithms for over 20 years without any success. Of course several working algorithms have been produced, but we do not believe that this is the way to solve the problem. The lack of stability of low level operations can probably be solved on higher levels in the hierarchy. One example is the multi-cue algorithm but other examples could be feedback of information to the low-level operations for fine tuning parameters. We have to take the whole system into account and not only spend time on finding extremely stable and accurate low-level methods.

8.2 Future work

One of the most interesting parts to continue with is grouping of local features into descriptive and invariant representations. This can be done in many different ways, where different rules for grouping are defined such as two features are grouped if they have a line connecting them. In this thesis we have used a very simple grouping rule, where features that are close to each other are grouped together. This is a very limited way to solve the problem and a more generic solution is to use association. In a pose estimation algorithm we have methods for mapping from the percept space to the object state space. This framework can also be used in a grouping process. Instead of mapping directly to the object state space a local feature can map on the position of the feature it should be grouped with. The power in such a system is the ability to verify a statement of a feature position, since a feature has to be at that certain position. If not then the object is either occluded or the current feature is a noise feature.

The second thing is to further investigate the power of multi-cue integration. We think that simple and fast methods for detecting and representing a wide range of features combined with a framework for multi-cue integration can be a solution to a robust and accurate pose estimation algorithm.
Bibliography


