Examensarbete utfört i Reglerteknik
vid Tekniska högskolan vid Linköpings universitet
av
Hanna Nyqvist

LiTH-ISY-EX--12/4629--SE

Linköping 2012
Image Database for Pose Hypotheses Generation

Examensarbete utfört i Reglerteknik
vid Tekniska högskolan vid Linköpings universitet
av
Hanna Nyqvist

LiTH-ISY-EX--12/4629--SE

Handledare:  Tim Bailey
            ACFR, University of Sydney
Johan Dahlin
            ISY, Linköpings Tekniska Högskola
James Underwood
            ACFR, University of Sydney

Examinator:  Thomas Schön
            ISY, Linköpings Tekniska Högskola

Linköping, 19 september 2012
The presence of autonomous systems is becoming more and more common in today’s society. The contexts in which these kind of systems appear are numerous and the variations are large, from large and complex systems like autonomous mining platforms to smaller, more everyday useful systems like the self-guided vacuum cleaner. It is essential for a completely self-supported mobile robot placed in unknown, dynamic or unstructured environments to be able to localise itself and find its way through maps. This localisation problem is still not completely solved although the idea of completely autonomous systems arose in the human society centuries ago. Its complexity makes it a wide-spread field of research even in present days.

In this work, the localisation problem is approached with an appearance based method for place recognition. The objective is to develop an algorithm for fast pose hypotheses generation from a map. A database containing very low resolution images from urban environments is built and very short image retrieval times are made possible by application of image dimension reduction. The evaluation of the database shows that it has real time potential because a set of pose hypotheses can be generated in 3-25 hundreds of a second depending on the tuning of the database. The probability of finding a correct pose suggestion among the generated hypotheses is as high as 87%, even when only a few hypotheses are retrieved from the database.
Abstract

The presence of autonomous systems is becoming more and more common in today’s society. The contexts in which these kind of systems appear are numerous and the variations are large, from large and complex systems like autonomous mining platforms to smaller, more everyday useful systems like the self-guided vacuum cleaner. It is essential for a completely self-supported mobile robot placed in unknown, dynamic or unstructured environments to be able to localise itself and find its way through maps. This localisation problem is still not completely solved although the idea of completely autonomous systems arose in the human society centuries ago. Its complexity makes it a wide-spread field of research even in present days.

In this work, the localisation problem is approached with an appearance based method for place recognition. The objective is to develop an algorithm for fast pose hypotheses generation from a map. A database containing very low resolution images from urban environments is built and very short image retrieval times are made possible by application of image dimension reduction. The evaluation of the database shows that it has real time potential because a set of pose hypotheses can be generated in 3-25 hundreds of a second depending on the tuning of the database. The probability of finding a correct pose suggestion among the generated hypotheses is as high as 87%, even when only a few hypotheses are retrieved from the database.
Acknowledgments

"For every complex problem, there is a solution that is simple, neat, and wrong.”
- Henry Louis Mencken, 1880-1956

Thank you Tim and James for your firm but gentle guidance towards simple, neat but yet working solutions to my problems. Many thanks also to Johan for your positivism and for having patience to read and interpret all of my incoherent e-mails. Lastly, I would like to express my appreciations to Thomas for encouraging me to accept the challenge of bringing this work to pass in a foreign country. It has been an incomparable experience for me.

Linköping, August 2012
Hanna Nyqvist
# Contents

1 Introduction .............................. 1
   1.1 Background ................................ 1
   1.2 Contributions ............................ 4
   1.3 Outline .................................. 4

2 Searching among images from unsupervised environments ... 5
   2.1 Visual images ................................ 5
      2.1.1 Colour descriptors and their invariance properties .... 7
      2.1.2 Histogram equalisation ..................... 14
      2.1.3 Similarity measurements .................... 16
      2.1.4 Image aligning ......................... 17
   2.2 Fast nearest neighbours search ............... 19
      2.2.1 Principal Component Analysis .............. 21
      2.2.2 KD-tree ................................ 23

3 Image Database for Pose Hypotheses Generation .......... 27
   3.1 Image data sets ............................ 29
      3.1.1 Environments ........................... 30
      3.1.2 Discarded data .......................... 30
      3.1.3 Database and query set selection .......... 32
   3.2 Algorithm .................................. 34
      3.2.1 Image pre-processing ...................... 35
      3.2.2 PCA on image sets for dimension reduction ........ 37
      3.2.3 Creation of the database KD-tree ........... 39
      3.2.4 Image retrieval .......................... 40
      3.2.5 Design parameters ....................... 41

4 Experimental results ........................ 43
   4.1 Evaluation methods ......................... 43
      4.1.1 Ground truth ............................ 44
      4.1.2 Evaluation of the database search result ....... 44
   4.2 Pre-processing evaluation .................... 48
   4.3 Evaluation of the approximate search .......... 53
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 Evaluation of the complete algorithm</td>
<td>57</td>
</tr>
<tr>
<td>4.4.1 TSSD</td>
<td>59</td>
</tr>
<tr>
<td>4.4.2 True SSD compared to approximate SSD</td>
<td>62</td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>65</td>
</tr>
<tr>
<td>A Derivation of Principal Components Analysis</td>
<td>69</td>
</tr>
<tr>
<td>B The Lucas-Kanade algorithm</td>
<td>73</td>
</tr>
<tr>
<td>C Derivation of the Inverse Compositional image alignment algorithm</td>
<td>75</td>
</tr>
<tr>
<td>Bibliography</td>
<td>77</td>
</tr>
</tbody>
</table>
Autonomous systems are becoming more and more present in today’s society. They appear in many different contexts and shapes, from large and complex systems like autonomous mining platforms to smaller more everyday useful systems like the self-guided vacuum cleaner. The idea of using autonomous robots in the duty of man has ancient roots. One of the first known sketches of a human like robot was drawn by Leonardo Da Vinci in the 15th century. Even older references to autonomous devices can however be found in ancient Greek, Chinese and Egyptian mythology.

Necessary requirements for a mobile robot in unknown, dynamic or unstructured environments to be completely autonomous are the ability to localise itself and find its way through maps. This complex problem still distracts the minds of many researchers although the dream of completely autonomous systems has been present for centuries. In this report the localisation problem is approached with an appearance based method for place recognition.

1.1 Background

There are many different solutions for the localisation of mobile platforms, comprising state of the art algorithms in the literature and also available as commercial off-the-shelf systems. The variations in the sensors and algorithms used to gather and process data are large. Arguably, the most well known and commonly used sensor in localisation contexts is the GPS receiver. Localisation with GPS is based on triangulation from a number of satellite signals. Many navigation systems today relies completely on the GPS, but there are environments and situ-
ations where it can not be used, for example in tunnels, caves and indoors where satellite signals can not reach. Another example is the urban environment where tall buildings could be blocking the signals and make a GPS pose estimate highly unreliable.

Other motion sensing sensors, such as accelerometers and gyroscopes, are also often used for localisation and tracking. The current position relative to a starting position can be estimated with these kind of sensors without the need of any external references. However, the responses from these sensors need to be integrated and small measurement errors therefore gets progressively larger. This is called integration drift and makes motion sensing sensors unsuitable for localisation in large scale areas.

Nowadays a common approach is to merge the mapping and the localisation problems and solve them simultaneously. This is often referred to as Simultaneously Localisation And Mapping, or in short SLAM, and the references [Durrant-Whyte and Bailey, 2006a,b] gives a short overview of the SLAM problem and how it can be solved. The issues mentioned above can be avoided if SLAM solutions are used. Measurements from different types of sensors are merged into one single pose estimate where the accuracy in the different sensors has been taken into account. This eliminates the issue with sensors which does not work properly in some environments. Also, landmarks are identified and compared with the internal map for loop closure detection\(^1\). A correctly identified loop closure means that the robot can identify a place where it has already been. The last pose estimate for this place can then be used to decrease the cumulative estimation error due to sensor drift. Good algorithms for re-arrival detection are therefore important since it could make mapping and localisation in large scale areas, where estimation drift is a big problem, more reliable.

SLAM algorithms involves interpretation of the sensor responses by an observation model. Many observation models are based on the assumption that the world can be described by basic geometry but this assumption does not hold in all environments. However, the computational power and data storage ability have grown large in recent days and this has enabled new opportunities. Providing accurate measurement models is now not the only available solution. Algorithms which operate on raw data are also an option and an example of such an algorithm is Pose-SLAM. With this algorithm entire scans from a sensor are stored and compared to each other, avoiding the need to model the specific content of the sensor measurements or how the sub-components of the environment gave rise to the data.

\(^1\)Detection of re-arrivals to already visited places
However, SLAM algorithms comparing and matching raw data scans does often involve data aligning. For example, let’s say that the environment is observed with some kind of sensor. It could i.e. be a RADAR or LIDAR sensor. Let’s also say that the sampling is done often, so that there is an overlap between two adjacent environmental scans. The spatial displacement between two such overlapping scans can be determined by data aligning. This spatial information can tell how far the sensor has moved since the last observation, enabling localisation or map creation.

There exist several methods for matching data based on the assumption that the data are overlapping. The Iterative Closest Points algorithm [Zhang, 1992] is one example of a data aligning strategy for point cloud data where an initial guess of the displacement has to be given. If no initial guess can be made then Spectral Registration [Chen et al., 1994] aligning is another option where the only required knowledge is the existence of an overlap. However, sometimes there might arise situations where there is no knowledge at all about the overlap or where an overlapping scan needs to be found before aligning can be done. An example of such a situation is when a robot is put in an arbitrary location without being given any knowledge about its pose. This is often called the kidnapped robot problem [Engelson and McDermott, 1992]. Another situation where any data aligning strategy would be highly insufficient is when an autonomous system suffers from a localisation failure. The pose estimate after a localisation failure could be inaccurate or even completely wrong. A fast way of identifying the correct region of interest in a map without actually performing any data aligning could be very useful in these kinds of situations.

Another, rather new, approach to the localisation and mapping problem is the appearance based approach. Visual cameras are information rich sensors which can capture many features from the surroundings, like for example textures and colours, which may be hard to model. Intense research in the appearance based localisation and mapping field was triggered by the increasing ubiquity and quality of cameras in the beginning of the 21th century. Nowadays, since the computers became fast enough and their memories large enough to handle massive image databases, the use of appearance based methods has started to become more and more common.

Several algorithms where visual images are used in SLAM or localisation contexts have been proposed. A rather successful one is the FAB-map algorithm [Cummins and Newman, 2008, 2010] which uses a bag-of-words description of collected images. This algorithm is based on probability theory and networks. The system is firstly trained to recognise salient details, words, in images and these salient details are then extracted and compared online. Other papers, where the use of visual images in mapping and localisation contexts are described, are [Little et al., 2005], [Benson et al., 2005] and [Cole et al., 2006].
One thing that all the above mentioned appearance based methods have in common is the dependency of the ability to successfully extract reliable features from the images. Scale invariant SIFT-features [Lowe, 1999] are most commonly used but Local Histograms [Guillamet and Vitrih, 2000] and Harris Corners [Harris and Stephens, 1988] occur in the literature as well. Appearance based localisation and mapping methods excluding this type of feature extraction have not been explored to the same extent. However, Fergus et al. [2008] shows that object and scene recognition without feature extraction is possible. Their article describes an image database where entire images are compared for the purpose of image classification. The question raised by this is if the simple nearest neighbour image database look-up approach taken in this article is adaptable for the purpose of place recognition as well.

1.2 Contributions

The objective of this work is to develop an algorithm which can be used in localisation and mapping contexts for fast localisation in a map when no current pose estimate is available. An appearance based method for place recognition is developed and the concept of comparing images as whole entities instead of extracting features is explored with the purpose of broadening the field of research. A database containing low resolution images is built and image dimension reduction enables very short image retrieval times. The database is able to generate a satisfying set of pose hypotheses from a query image in 3-25 hundreds of a second depending on the desired hit rate.

1.3 Outline

The subsequent chapter describes different representations of visual images and their properties and also how a nearest neighbour search among images can be made more time efficient. Some algorithms that are used later on are described in more detail. Chapter 3 presents the proposed algorithm for appearance based place recognition. The experimental methodology is described in Chapter 4 together with the results from the database evaluation. Finally, future work and conclusions are presented in Chapter 5.
2

Searching among images from unsupervised environments

This chapter explains some different representations of visual images and how one can measure the similarity between such. It also describes methods which can be used to speed up image database look-ups.

To be able to completely understand the image retrieval algorithm suggested later on in this paper, it is essential to have some knowledge about the different components out of which it is built. A presentation and discussion about issues which have to be taken into account when developing an image database for real time scene recognition in uncontrollable environments are presented in this chapter. Previous methods for dealing with these problems are discussed and methods that have already been explored or developed by others and applied in this work are described in more detail.

2.1 Visual images

Visual images referred to in this work are digital interpretations of the light emitted from our surroundings captured by a camera. A common way to represent a visual image, $I(x)$, is with a two dimensional matrix of pixels where each pixel is assigned some values. $x = (x, y)$ denotes the index of a pixel in an image. The length of the colour vector assigned to each pixel can vary depending on which colour description that is used but the values are always an interpretation of the intensity and colour of the light captured by the camera sensors. An illustration of this representation can be seen in Figure 2.1.
Visual cameras are information rich sensors which can capture many features, e.g. textures, colours and shapes. This makes these kind of sensors interesting and attractive for use in contexts such as object and scene recognition or localisation and mapping. Nevertheless, appearance based methods introduce new issues. According to a reasoning by Beeson and Kuipers [2002] there are two main difficulties that arises when dealing with appearance based scene recognition. The first problem is that images showing the same scene and captured from the exact same spot may differ from each other. The differences can be caused by for example changes in weather conditions, time of day, illumination conditions or moving objects occluding the view. This phenomena goes under the name of image fluctuations. Secondly, two different viewpoints may give rise to very similar images. This is in turn referred to as image aliasing. It is very likely that a camera can capture details from the environment which could be used to distinguish similar frames thanks to the richness of the sensor. However, this richness also makes it likely to capture noise and dynamic changes. It is therefore most likely that image fluctuations will be the biggest problem when trying to perform place recognition in uncontrollable environments.

Pre-processing of images in such a way that their representations are as invariant to dynamic changes in the surroundings as possible is very useful for dealing with some image fluctuation problems. The pre-processing can for example involve an image transformation to a more suitable colour description or a normalization according to some well-selected image property. Unfortunately, dynamic objects occluding a scene can not be compensated for with any of these pre-processing methods.

A common approach to deal with occlusions is instead to extract subsets of pixels, which are considered to be more salient or informative than others, from the images. Image noise can be excluded if these subsets are chosen in a good way. The subsets are often called features and there are several methods for extracting features from an image, for example the Harris Corners [Harris and Stephens, 1988] or the SIFT [Lowe, 1999] methods. There are also a wealth of algorithms which can be used for matching features to determine image similarity, for example RANSAC [Bolles and Fischler, 1986]. Feature matching algorithms can perform the matching in such a way that the inner relative positions between features in an image are preserved while neglecting the absolute positions within the image frame. Some feature matching algorithms, for example RANSAC, also allows the positions of the features to be warped consistently according to some homography respecting the assumption of a rigid 3D structure of the world. Feature matching can thus be made invariant to changes in viewpoint leading to image transforms such as rotation, translation and scaling. Also, noisy points which can not be warped by such an allowable warp can be detected and rejected.
2.1 Visual images

The rotational invariance enabled by feature extraction is a favourable attribute, but there are two main reasons why this still is not the approach taken in this thesis. Firstly, finding good features in an image could be very difficult. Feature detection is often based on finding changes in images, e.g. edges\(^1\) or corners\(^2\). Dynamic image regions, such as the border of a shadow or the roof of a red car in contrast to blue sky, are therefore easily mistaken for reliable features. This implies that feature based scene recognition algorithms might not be quite as insensitive to dynamic changes as hoped for. Secondly, the concept of visual feature based localisation and mapping has already been carefully explored by many others. Often one algorithm succeeds where another fails and it is therefore important to have several different algorithms producing comparable results. It is for example possible to make a SLAM algorithm more robust by running several different approaches in parallel.

This work takes a different, much simpler approach then extracting features. Here images are considered as whole entities instead. The expectation is that similarities between two images captured from approximately the same viewpoint will turn out to overwhelm the dissimilarities due to dynamic environments.

The remainder of this section is dedicated to discussions regarding making the image representation less sensitive to dynamic changes. Also, how to define an image similarity measurement without involving feature extraction is described in the end of this section.

2.1.1 Colour descriptors and their invariance properties

In order to robustly reason about the content in images, it is required to have descriptions of the data that mainly depend on what was photographed. However, fragile and transient properties such as illumination and the viewpoint from which a snapshot is captured have great influence on the resulting image. This can be a big issue when working with image recognition in environments where these circumstances can not be controlled. Luckily are some image representations less sensitive to these kind of disturbances. The colour description used to interpret the camera sensor response should therefore be analysed and carefully chosen.

There are several different commonly used ways to model the light and assign values to the pixels in an image. The most well known method to describe colours is perhaps the RGB colour description, where each pixel is associated with three values representing the the amount of red, green and blue colors in the incident light. This is illustrated in Figure 2.1. However, there are others presented in

\(^1\) An edge is a border in an image where the brightness is changing rapidly or has discontinuities.
\(^2\) A corner is an intersection of two edges.
literature [Gevers et al., 2010] [Gevers and Smeulders, 2001] that might be more suitable for scene recognition purposes. Some of them are described in this section.

**Figure 2.1:** An illustration of how an RGB image with \(n \times m\) pixels can be represented.

### RGB

The RGB colour model is an additive model based on human perception of colours. A wide range of colours can be reproduced by adding different amounts of the three primary colors red, green and blue as can be seen in Figure 2.2. Each pixel in a RGB image is therefore assigned three values representing the amounts from each of these colour channels, \(C\), according to (2.1). It is common that all pixel values are limited to be within the range \([0, 1]\) or \([0, 255]\). White light corresponds to maximum value (1 or 255) in all the three colour channels while zero in all three channels corresponds to black.

\[
I_{RGB}(x, C) = \begin{cases} 
\text{amount of red in the light incident on pixel } x, & C = R \\
\text{amount of green in the light incident on pixel } x, & C = G \\
\text{amount of blue in the light incident on pixel } x, & C = B 
\end{cases}
\]

(2.1)

The RGB colour model is often used in contexts such as sensing and displaying of images in electrical systems because of its relation to the human perceptual system. Unfortunately this colour description is not very suitable for image recognition since it is not invariant to changes such as the intensity or the colour of
the illumination as well as the viewpoint from which the image was captured. Images of the same scene, captured at the same place could vary a lot when using the RGB colour model and this could bring a lot of trouble if not performing some kind of pre-processing before image comparison to obtain more favourable invariance properties.

\[ I_{\text{gray}}(x) = \sum_{c \in \{R,G,B\}} \lambda_c I_{\text{RGB}}(x, c) \] (2.2)

**Figure 2.2:** An illustration of how the three primary colours red, green and blue can be added together to form new colours. This image is used under Creative Commons with permission from Wikimedia Foundation [Commons, 2012-06-15b].

**Gray-scale**

To avoid the complications imposed by a three dimensional colour space described above, one commonly used approach is to compress the channels into a single dimensional grey scale value. For the gray-scale colour descriptor, unlike the RGB descriptor, each image pixel is associated with only one value. The maximum value allowed corresponds to white whilst zero corresponds to black and everything in between represents different tones of gray. How to convert a three dimensional RGB image into a gray-scale image is a dimensional reduction problem and many solutions has been proposed, for example by Gooch et al. [2005] where an optimization problem is solved to preserve as many details in an image as possible. A very simple way to achieve this dimension reduction is by computing a weighted sum of the red, green and blue colour channels according to
The weights can vary and depends on the choice of primaries for the RGB colour model but typical values, which are also used in this work, are

\[
\begin{align*}
\lambda_R &= 0.2989 \\
\lambda_G &= 0.5870 \\
\lambda_B &= 0.1140
\end{align*}
\]  

(2.3)

These weights are calibrated to mirror the human perception of colours. The human eye is best at detecting green light and worst at detecting blue. Green light will for us appear to be brighter than blue light with the same intensity and the green channel is therefore given a larger weight. The transformation from RGB to gray is not unambiguous and two pixels that differ in the RGB space might be assigned the same gray-scale value.

**Normalized RGB - rgb**

A way to make an RGB image less sensitive to the lighting conditions under which it was captured is to normalize the pixels with their intensity. The obtained descriptor will be referred to as the rgb descriptor.

The *intensity* of a RGB pixel \( x \) is defined as the sum of the red, green and blue values for this pixel according to

\[
A_{\text{RGB}}(x) = I_{\text{RGB}}(x, R) + I_{\text{RGB}}(x, G) + I_{\text{RGB}}(x, B)
\]

(2.4)

The normalized RGB image, \( I_{\text{rgb}}(x, C) \), is then calculated from a RGB image as done in (2.5). One of the three colour channels is redundant after the normalization since the sum of the three channels in an rgb image always sums up to one.

\[
I_{\text{rgb}}(x, C) = \frac{I_{\text{RGB}}(x, C)}{A_{\text{RGB}}(x)}, \quad C \in \{R, G, B\}
\]

(2.5)

**HSB**

*HSB* stands for hue, saturation, brightness. It is a cylindrical model where each possible colour from the visual spectra is associated with a coordinate on a cylinder as illustrated in Figure 2.3.

*Hue* denotes the angle around the central axis of this colour cylinder and correspond to the wavelength within the visible light spectrum for which the energy is greatest. More commonly, think of the hue as how similar a colour is to any of the four unique hues red, yellow, green and blue.
Saturation is the denotation of the distance from the central axis and it is an interpretation of the bandwidth of the light, or in other words how pure a colour is. High saturation implies light which consists of only a few number of dominant wavelengths.

![HSB cylindrical coordinate representation](image)

**Figure 2.3:** An illustration of the HSB cylindrical coordinate representation. This image is used under Creative Commons with permission from Wikimedia Foundation [Commons, 2012-06-15a].

Colours found along the central axis of the cylinder have zero saturation and ranges from black to white. The distance along this central axis is denoted as brightness and is an interpretation of intensity of the light. It corresponds to the intensity of the of coloured light relative the intensity of a similarly illuminated white light.

A RGB image can easily be converted into a HSB image if introducing the following auxiliary variables

\[
\begin{align*}
M(x) &= \max_{C \in \{R,G,B\}} (I_{RGB}(x, C)) \\
m(x) &= \min_{C \in \{R,G,B\}} (I_{RGB}(x, C)) \\
\Delta(x) &= M(x) - m(x)
\end{align*}
\]

Hue, \(H\), saturation, \(S\), and brightness, \(B\), are then computed according to equations (2.7) - (2.9) respectively.
\[ H(x) = 60^\circ \cdot \begin{cases} 
\text{undefined}, & \text{if } \Delta(x) = 0 \\
\frac{I_{RGB}(x,G) - I_{RGB}(x,R)}{\Delta(x)} \mod 6, & \text{if } I_{RGB}(x, R) = M(x) \\
2 + \frac{I_{RGB}(x,R) - I_{RGB}(x,B)}{\Delta(x)}, & \text{if } I_{RGB}(x, G) = M(x) \\
4 + \frac{I_{RGB}(x,R) - I_{RGB}(x,G)}{\Delta(x)}, & \text{if } I_{RGB}(x, B) = M(x) 
\end{cases} \]
\[(2.7)\]

\[ S(x) = \begin{cases} 
0, & \text{if } \Delta(x) = 0 \\
\frac{\Delta(x)}{M(x)}, & \text{otherwise} 
\end{cases} \]
\[(2.8)\]

\[ B(x) = M(x) \]
\[(2.9)\]

It is necessary to perform a modulo operation when computing the hue if the angle is to stay within the range of \([0, 360]\) degrees. It can be seen that hue is undefined when the saturation value equals zero. This implies that the hue property becomes unstable close to the gray-scale axis. It is derived by van de Weijer and Schmid [2006] that the uncertainty of the hue is inversely proportional to the saturation and a method of weighting hue with saturation by multiplying them together is therefore suggested. Another method to deal with this instability issue is proposed by Sural et al. [2002] where saturation is used as a threshold to determine when it is more appropriate to associate a pixel with its brightness property than its hue property.

**Invariance properties**

The colour models described above are not all equally suitable to use in place recognition contexts due to their varying sensitivity to the circumstances under which an image is captured. Luckily, some colour models are more robust to this kind of disturbances than others and these may be better choices for image recognition purposes.

Invariance properties for the colour models described in this section are derived by Gevers and Smeulders [2001] where the authors use the dichromatic camera model proposed by Shafer [1985]. The light reflected from an object is modelled as consisting of two components. One component is the ray reflected by the surface of the object. The second component arises from the fact that some of the incident light will penetrate through the surface and be scattered and absorbed by the colourants in the object before it eventually reflects back through the surface again. This is illustrated in Figure 2.4a. This reflectance model allows for analysis of inhomogeneous materials which is not possible under the common assumption of a visual surface being a plane. The surface and body reflectances are given by \(c_s(\lambda)\) and \(c_b(\lambda)\) respectively where \(\lambda\) is the wavelength of the light.
2.1 Visual images

(a) The reflected light is divided into two rays. One corresponds to interface reflection and the other corresponds to body reflection. On a macroscopic level the surface would have appeared smooth and the macroscopic reflectance direction therefore differs from the true interface direction.

(b) Close-up of Figure 2.4a. Definition of three geometrically dependent vectors which could have influence over the camera sensor response.

Figure 2.4: An illustration of the reflectance model.

The camera model allows the red, green and blue camera sensors to have different spectral sensitivity given by \( f_R(\lambda), f_G(\lambda) \) and \( f_B(\lambda) \). The spectral power density of the incident light is denoted by \( e(\lambda) \). The sensor response \( c \) of an infinitesimal section of an object can, with this notation, be expressed as

\[
c = m_b(n, s) \int_{\lambda} f_C(\lambda)e(\lambda)c_b(\lambda)d\lambda + m_s(n, s, v) \int_{\lambda} f_C(\lambda)e(\lambda)c_s(\lambda)d\lambda, \tag{2.10}
\]

where \( c \) is the amount of incident light with colour \( C \in \{R, G, B\} \). \( n, s \) and \( v \) are the surface normal, the direction of the illumination source, and the viewing direction respectively, as defined in Figure 2.4b. An interpretation of the expression is that each of the two rays reflected from an object can be divided into two parts. The first part, corresponding the integrals in the equation, is the relative spectral power density of each ray. This part only depends on the wavelengths of the light source and does not at all depend on any geometrical factors. The
The invariance properties in Table 2.1 are derived simply by combining (2.10) above with the previous mentioned transformations, (2.7) - (2.9), from RGB into other colour representations. One can see that hue is the only image representation invariant of all properties in Table 2.1.

<table>
<thead>
<tr>
<th></th>
<th>Intensity</th>
<th>RGB</th>
<th>Normalized RGB</th>
<th>Saturation</th>
<th>Hue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Viewing direction</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Surface orientation</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Highlights</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
<tr>
<td>Illumination direction</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Illumination intensity</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Illumination colour</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>x</td>
</tr>
</tbody>
</table>

*Table 2.1: Overview of invariance properties of various image representations/properties. x denotes invariance and - denotes sensitivity for the colour model to the condition.*

### 2.1.2 Histogram equalisation

Histogram equalisation is another way to pre-process an image to be able to reason more robustly about its content. It is a method which advantageously can be used together with image comparison algorithms because of its effectiveness in image detail enhancement. Histogram equalisation increases contrasts in images by finding a transform that redistributes an image histogram in such a way that it becomes more flat. The objective for the histogram equalisation algorithm is to transform the image so that the entire spectrum of possible pixel values are occupied. The process is as follows.

Let \( I(x) \) be a discrete image with only one channel, for example a gray-scale image or an intensity image. In this context the meaning of discrete is an image where the values of the pixel only can take some discrete values \( L_{\min} \leq i \leq L_{\max} \). Furthermore, let \( n_i \) be the number of occurrences of pixels in the image taking the value \( i \). If the image has in total \( n_{\text{pix}} \) pixels then the probability of an occurrence of a pixel with value \( i \) can be computed as in (2.11). In fact this probability equals the histogram of the image normalized to \([0, 1]\). The corresponding cumulative distribution function \( F_I \) can be calculated according to (2.12).
2.1 Visual images

\[ p_I(i) = \frac{n_i}{n_{\text{pix}}} \quad (2.11) \]

\[ F_I(i) = \sum_{j=0}^{i} p_I(j) \quad (2.12) \]

The objective is now to find a transformation \( \tilde{I}(x) = T_{\text{HE}}(I(x)) \) so that the new image has a linear cumulative distribution function, that is (2.13) should hold for some constant \( K \) where

\[ F_{\tilde{I}}(i) = iK. \quad (2.13) \]

This can be achieved simply by choosing \( T_{\text{HE}}(I(x)) = F_I(I(x)) \). This is a map into the range \([0, 1]\). If the values are to be mapped onto the original range then the transformation has to be slightly modified according to

\[ \tilde{I}(x) = T_{\text{HE}}(I(x)) = F_I(I(x))(L_{\text{max}} - L_{\text{min}}) + L_{\text{min}}. \quad (2.14) \]

Note that this histogram equalisation does not result in an image with a completely flat histogram. The heights of the staples in the original histogram are unchanged, but the staples are spread more apart so that a larger part of the possible pixel value space is occupied, as illustrated in Figure 2.5.

\[ \begin{array}{c}
\text{Normalized histogram - dark image} \\
\text{Cumulative histogram - dark image} \\
\end{array} \quad \rightarrow \quad \begin{array}{c}
\text{Normalized histogram - histogram equalization} \\
\text{Cumulative histogram} \\
\end{array} \]

\[ \begin{array}{c}
\text{Normalized histogram - dark image} \\
\text{Cumulative histogram - dark image} \\
\end{array} \quad \rightarrow \quad \begin{array}{c}
\text{Normalized histogram} \\
\text{Cumulative histogram} \\
\end{array} \]

**Figure 2.5:** An illustration of histogram equalisation where a dark image is transformed so that its new cumulative histogram becomes linear. The transformation lightens up some of the dark pixels so that a larger part of the possible pixel values is occupied.
2.1.3 Similarity measurements

A measure of similarity has to be defined to be able to compare images. Since local feature extraction is to be avoided in this work, the similarity measurement should be comparing images as whole entities rather than comparing selected parts of them. There are many methods presented in literature to achieve this. One commonly used strategy seems to be to summarize some image attribute into a global feature, for example an image histogram, before comparison. Majumdar et al. [2003] describes and compares some histogram similarity measurements. However, spatial connections between pixels are neglected when creating this type of global feature and important information might be lost. A very simple and basic measurement, which does not neglect inner structures of images, is the sum of squared distances (SSD) measurement. The definition of the SSD can be seen in (2.15).

A comparison between some similarity measurements, which all take inner image structures into account, is done by di Gesù and Starovoitov [1999]. The authors of this article states that there are other ways of measuring similarity than the SSD which perform better. There are also distance metrics such as the Earth Movers Distance [Guibas et al., 2000, 1998] or the IMED [Feng et al., 2005] which claims to be better for image similarity comparisons. The pixel-wise squared distances summation is however still considered to be a better choice for this work because of its simplicity. It is used by Fergus et al. [2008] with positive results and the properties of the SSD also makes it easy to combine with data structures for more efficient database searches (see Section 2.2) unlike some of the other suggestions.

A query image and its corresponding database image may be captured from slightly different viewpoints and the query image may therefore be a slightly transformed version of its corresponding database match. The SSD involves pixel-wise comparisons and a database look-up strategy based on this measurement could therefore be sensitive to these kinds of transformations. A modified SSD, called TSSD, is hence considered as well. The TSSD is computed as in (2.16). The query image is transformed before the SSD is computed and the transformation consists in an image alignment to the database image. How an appropriate image aligning transform can be found is described in the next section, Section 2.1.4.

\[
\text{SSD}(I_Q(x, C), I_{DB}(x, C)) = \sum_{x,C} [I_Q(x, C) - I_{DB}(x, C)]^2 \quad (2.15)
\]

\[
\text{TSSD}(I_Q(x, C), I_{DB}(x, C)) = \sum_{x,C} [T(I_Q(x, C)) - I_{DB}(x, C)]^2 \quad (2.16)
\]
2.1.4 Image aligning

Image aligning involves the art of finding and applying image warps so that two images matches better with each other according to some criteria. This enables more precise comparison of images with almost the same content but captured from slightly different viewpoints and could therefore be useful in this work.

The Lucas-Kanade (LK) algorithm is the ancestor of image alignment algorithms and deals with the problem of aligning the image $I(x)$ to a template image $T(x)$ where $x = (x, y)^T$ is a vector containing the pixel coordinates. The objective of the algorithm is to minimize the sum of squared errors between the two images by finding a warp $\tilde{x} = W(x, p)$ such that the expression in (2.17) below is minimized with respect to the warp parameter vector $p = (p_1, ..., p_{nw})$.

$$\min_p \sum_x [I(W(x, p)) - T(x)]^2$$  \hspace{1cm} (2.17)

Appendix B contains the steps of the LK algorithm. It is an iterative algorithm where an approximate estimate of the parameter vector $p$ is assumed to be known. The image alignment problem stated above is then, in each iteration, solved with respect to a small estimation deviation $\Delta p$ followed by an additive update of the current parameter estimate.

A Hessian $H = \sum_x \left[ \frac{\partial W}{\partial p} \right]^T \left[ \frac{\partial W}{\partial p} \right]$ has to be re-evaluated in each iteration of the LK algorithm. This means great computational costs but other, cheaper alternatives to the LK algorithm has been derived in literature. Two of the alternatives are the Inverse Additive (IA) and the Inverse Compositional (IC) algorithms described in the same article as their precursor. These two algorithms results, according to the authors, in image warps equivalent to warps obtained from LK, but they outperform the original in terms of computational efficiency. The authors also state that the computational costs of the IC and IA algorithms are almost the same. However, the IC algorithm is much more intuitive to derive than its additive counterpart and it is therefore a more reasonable choice for most applications according to the authors.

**Inverse Compositional algorithm**

The Inverse Compositional algorithm is an iterative algorithm just as the LK algorithm. The expression in (2.18) is minimized in each iteration. With this approach, the image alignment problem is solved by iteratively computing a small incremental warp $W(x, \Delta p)$ rather than an additive parameter update $\Delta p$ as done in the LK update. The warp update is no longer additive but must for the IC algorithm be a composition between $W(x, p)$ and $W(x, \Delta p)$. Also, the roles of the template $T(x)$ and the image $I(x)$ are inverted compared to the LK algorithm and this will in the end lead to an algorithm where the Hessian $H$ does not need to
be re-evaluated every iteration but can be pre-computed instead. These inverted roles also imply that the small incremental warp $W(x, \Delta p)$ obtained after each iteration has to be inverted before updating the warp parameters. The complete warp update is done according to (2.19).

$$\min_{\Delta p} \sum_x [T(W(x, \Delta p)) - I(W(x, p))]^2$$  \hspace{1cm} (2.18)

$$W(x, p) \leftarrow W(x, p) \circ W(x, \Delta p)^{-1} = W(W(x, \Delta p)^{-1}, p)$$  \hspace{1cm} (2.19)

A solution to the IC image alignment problem is derived in appendix C and it leads to the following algorithm.

**Algorithm 1 Inverse Compositional**

(* Algorithm for aligning images *)

**Require:** Template image $T(x)$, Image $I(x)$, Initial warp parameters guess $p_{\text{guess}}$, Warp $W(x, p)$

1: Compute the gradient, $\nabla T$, of the template image $T(x)$
2: Compute the Jacobian, $\frac{\partial W}{\partial p}(x, p)$, of the warp and evaluate it at $(x, 0)$
3: Compute the steepest descendant images $\nabla T \frac{\partial W}{\partial p}$
4: Compute the Hessian, $H = \sum_x \left[ \nabla T \frac{\partial W}{\partial p} \right]^T \left[ \nabla T \frac{\partial W}{\partial p} \right]$
5: $p \leftarrow p_{\text{guess}}$
6: while alignment not good enough do
7: Compute $I(W(x, p))$
8: Compute the error image $I(W(x, p)) - T(x)$
9: Compute $\sum_x \left[ \nabla T \frac{\partial W}{\partial p} \right]^T [I(W(x, p)) - T(x)]$
10: Perform the warp update $W(x, p) \leftarrow W(x, p) \circ W(x, \Delta p)^{-1}$
11: end while

**Modification to increase robustness for the Inverse Compositional algorithm**

In the derivation of the Inverse Compositional algorithm it is assumed that an estimate of the warp parameters exists. This makes the algorithm sensitive to the initial guess of the parameter vector $p$. If this guess differs too much from the true warp, then the algorithm may end up in a local minimum instead of the true global and the image alignment will not be reliable. The algorithm could be made more robust if only translation updates are considered during the first few iteration. That is, if the first few iterations are dedicated to find a better guess for $p$ then the algorithm won’t be as sensitive to the initial guess.
Thus, the IC algorithm with this modification is applied in this work whenever image aligning is used because of its robustness and speed compared to similar algorithms and also since it does not involve any feature extraction.

2.2 Fast nearest neighbours search

Exhaustive search among all images in a database is not an option when building a real time database image retrieval system for localisation and mapping purposes. This is due to the fact that the exhaustive search time will be linearly increasing when new places are being explored and new images are added to the database. This is not a desirable attribute since images with rather dense spatial distribution from large areas are expected to be stored in the database. With a look-up time that is linearly dependent on the number of items in the database the system would soon become overloaded and any real time performance would only be achievable for small databases. Some kind of indexing or efficient search structure is essential if the system will be able to deal with any real time demands.

Two data structures that can be used for efficient data retrieval and which frequently appear in literature are binary trees and hash tables. Hash tables use a function to map a one- or multi-dimensional key to a value and then store the item in a bucket corresponding to this value. This process is illustrated in Figure 2.6a. A binary search tree is a tree structure where each node contains only one key and also has only two branches. The branching is done so that the left branch of a parent node only contains keys smaller than the key in the parent node. The right branch does in turn only contain keys with greater values as can be seen in Figure 2.6b. The originally proposed binary tree could only handle one dimensional keys but there are now versions, for example the KD-tree [Moore, 1991] [Bentley et al., 1977], where multi-dimensional keys are no longer a problem.

A hash table often outperforms a binary tree in case of look-up time expenditure if the hash function is chosen in an appropriate way. However, how to determine an appropriate hash function is highly dependent on the data. Images from several different environments are used in this work. This means that the distribution of the image data might be difficult to determine beforehand. Hence, it is much harder to design an efficient hash function than an efficient KD-tree. Also, the purpose of using a more efficient data structure is to speed up a $k$ nearest neighbour search. A binary tree provides efficient and rather simple algorithms for $k$ nearest neighbour search whilst $k$ nearest neighbour search within a hash table is a slightly more complex process. The KD-tree is therefore considered to be the better option based on these two arguments.
20 Searching among images from unsupervised environments

(a) The hash table: Illustration of how a key $x$ with several dimensions can be inserted. A hash function projects the key onto a scalar. Which of the $N$ buckets to put the key into is then determined by applying the modulo $N$ operator on this scalar.

(b) The binary tree: Illustration of how one dimensional keys are stored in the tree structure.

Figure 2.6: Two data structures commonly used to speed up database searches.
Even if the KD-tree is a search structure which in theory can be used to speed up a neighbour searches in large multidimensional data, it still suffers from a condition referred to as the *curse of dimensionality*. For very large dimensional data the search algorithm tends to be less efficient and the more the dimensionality grows the more the efficiency will resemble a pure linear exhaustive search. This is however an issue for nearest neighbours search algorithms in general, not only for nearest neighbour search in a KD-tree. The problem will not disappear with some other choice of data search structure and the decision to use the KD-tree will therefore not be affected. Yet, some kind of dimension reduction is necessary if fast data retrieval for very high dimensional data, such as visual images, is required.

One way of achieving dimension reduction for visual images is to extract features as discussed in the beginning of this chapter. Feature extraction is however to be avoided in this work and another method has to be sought. A dimension reduction method that tries to approximate the whole content of an image but with fewer variables is better suited. It is also important that differences between images in the original data are still present in the reduced data. Otherwise a nearest neighbour search to a query image in the reduced database will be almost completely insignificant. There exist many such methods, from linear methods like *Principal Components Analysis*, *Factor Analysis*, or *Independent Component Analysis* to non-linear such as *Random Projection* or a non-linear version of Principal Components Analysis. All of these methods together with some more are further described by Fodor [2002]. In [Fergus et al., 2008], the paper from which much of the inspiration to this work comes, is the linear Principal Component Analysis (PCA) method used successfully. Thus, PCA is used also in this work.

### 2.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a method for analysing data with many variables simultaneously and can be used in many different contexts. The goal of the PCA can for example be simplification, modelling, outlier detection, dimensionality reduction, classification or variable selection. Here it will be used for data reduction. The idea behind PCA is to express a data matrix consisting of observations from a set of variables in a way such that similarities and differences in the data are highlighted. This is done by finding a set of orthogonal vectors \( \mathbf{p}_i \), called principal components, onto which the data can be projected, i.e. by finding a meaningful change of basis. To make the transformation meaningful the principal components can not be just any set of orthogonal vectors but are defined in such a way that the first principal component accounts for the largest amount of variability in the data set as possible, i.e. the variance when the original data are projected onto this component is as large as is possible. Each sequent component has in turn as large variance as is possible under the constraint that it should be orthogonal to all the previous components.
Let the data that are to be analysed consist of \( m \) observations of \( n \) different variables. Form a data observation matrix \( \mathbf{X} \) out of the observations

\[
\mathbf{X} = \begin{pmatrix}
  x_{\text{obs}1}^{\text{var}1} & x_{\text{obs}1}^{\text{var}2} & \cdots & x_{\text{obs}1}^{\text{var}n} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{\text{obs}m}^{\text{var}1} & x_{\text{obs}m}^{\text{var}2} & \cdots & x_{\text{obs}m}^{\text{var}n}
\end{pmatrix},
\]  

(2.20)

where each column corresponds to a variable and each row to an observation.

If the data matrix has zero mean then PCA can be mathematically formulated according to (2.21) - (2.23). The first principal component is obtained by solving

\[
\mathbf{p}_1 = \arg \max_{\|\mathbf{p}\|=1} \text{Var} \{\mathbf{Xp}\} = \arg \max_{\|\mathbf{p}\|=1} \mathbb{E} \{ \mathbf{p}^T \mathbf{X}^T \mathbf{X} \mathbf{p} \}. 
\]  

(2.21)

The problem for the remaining components has to be stated slightly different for the orthogonality criteria to hold. For the \( i \):th component, first subtract the information which can be explained with help of \( \mathbf{p}_1, \ldots, \mathbf{p}_{i-1} \) from the data matrix

\[
\tilde{\mathbf{X}} = \mathbf{X} - \sum_{k=1}^{i-1} \mathbf{Xp}_k \mathbf{p}_k^T.
\]  

(2.22)

Then solve the same problem as when determine the first principal component but with the modified data matrix according to

\[
\mathbf{p}_i = \arg \max_{\|\mathbf{p}\|=1} \text{Var} \{\tilde{\mathbf{X}}p\} = \arg \max_{\|\mathbf{p}\|=1} \mathbb{E} \{ \mathbf{p}^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \mathbf{p} \}. 
\]  

(2.23)

A derivation of a solution to the PCA problem can be further explored in appendix A. There it is derived that the principal components can be found by computing the eigenvectors to the covariance matrix \( \mathbf{X}^T \mathbf{X} \)

\[
\begin{cases}
  \{\lambda_1, \lambda_2, \cdots, \lambda_n\} = \text{eigval}(\mathbf{X}^T \mathbf{X}) \\
  \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_n\} = \text{eigvec}(\mathbf{X}^T \mathbf{X})
\end{cases}, \quad \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.
\]  

(2.24)

The eigenvalues corresponds to the amount of the original information the corresponding principal component can reflect and they should therefore be sorted in decreasing order as done in (2.24). Since the first principal components are more informative then the last ones the dimension reduction, from the original
2.2 Fast nearest neighbours search

$n$ dimensional space to a lower $m$ dimensional space, with the least information loss will be a projection of the data onto only the first $m$ principal components.

### 2.2.2 KD-tree

Search algorithms related to the KD-tree structure can be very effective and this tree is therefore often used in contexts where large amounts of data are searched, sorted or traversed in some way.

A KD-tree is a generalization of the binary search tree where the keys are not limited to scalars, but can be $k$ dimensional vectors. A node in the tree can be seen as a hyperplane dividing the key space. Keys that lie on one side of this hyperplane will end up in the left branch of the splitting node while keys on the other side of the plane will be in the right branch. In the originally proposed KD-tree the splitting dimension was simply determined by cycling through all dimensions in order, i.e. the splitting was first performed by splitting along the first dimension then the second and so on. Other suggestions for how to determine this in a more efficient way have arisen over the years. One example is [Bentley et al., 1977] where the discriminating dimension is proposed to be the dimension for which the keys have the largest variance in values.

Except for determining the dimension along which to insert the splitting hyperplane one must also determine a discriminating value for this hyperplane to be able to decide whether to insert nodes into the left or the right branch. The efficiency of a binary tree is highly dependent on its depth\(^3\). A balanced tree\(^4\) is therefore desirable. To ensure that a KD-tree is balanced, the median along the dimension which is to be split can be used as partitioning value. This method was also proposed by Bentley et al. [1977]. Choosing the median implies that each subtree in the KD-tree will have an equal number of nodes in its left and right branches and hence the tree will be balanced. However, other methods for determination of the discriminating value of a node have been suggested. Some of them have been described and implemented by Mount [2006].

The leaf nodes\(^5\) of a KD-tree are often called buckets and contains small subsets of the original data. These subsets are mutually exclusive because of the way the tree is branched. How big these buckets should be can not be determined in general, but depends completely of the size of the data set and the application. An illustration of how a KD-tree can be build can be seen in Figure 2.7.

---

\(^3\)The depth of a tree is the number of nodes from the topmost to the bottommost node.

\(^4\)A tree is balanced if for each node in the tree it holds that the number of nodes in all its branches are the same.

\(^5\)A leaf node is a node without branches.
Searching among images from unsupervised environments

(a) All keys are sorted along the first dimension and the median, 6, is found. The key corresponding to the median is chosen to be in the top node of the tree.

(b) The remaining keys are divided into two groups. Keys with value in first dimension less then the median in will be in the left branch. The splitting dimension on level 2 in the tree will be the second dimension. Starting with the left branch, the next key to be inserted will be chosen according to the median along the second dimension.

(c) The nodes are again divided into two groups. The splitting dimension on level three is now again the first. There is only one key in the left branch and this will be put into the leaf node bucket.

(d) The right branch has also only one key left. Now backtrace to a tree level with a non sorted branch, level 2 in this case. Repeat the process until there are no keys left.

(e) The complete tree when all keys are inserted.

(f) An illustration of how the 2-dimensional space has been split by hyperplanes.

Figure 2.7: Creation of a KD-tree with 2-dimensional keys. The splitting dimension is changed in each level of the tree, starting with the first dimension. The node to be inserted into the tree is determined by the median of the splitting dimension. The buckets in the leaf nodes can only contain one key.
2.2 Fast nearest neighbours search

Nearest neighbour search algorithm

The structure of the KD-tree enables efficient nearest neighbour search, where only a subset of the tree nodes has to be examined. The nearest neighbour search is a process where nodes in the tree are traversed recursively. The partitioning of the nodes defines upper and lower limits on the the keys in the right and left subtrees respectively. For each node that is to be searched, the limits from this node and its ancestors define two cells in the k-dimensional key subspace. These cells are subspaces in which all the keys in the left and right branches respectively are to be found.

If the node under investigation is a leaf node, all the containing keys in the bucket are examined. If the node under investigation is not a leaf node one of its branches might be possible to exclude completely from the search by performing a so called ball-within-bounds test. Let’s pretend that the $m$ nearest neighbours to a query record are to be found. Then a ball-within-bounds test fails if a ball, centered in the query record and with a radius equal to the distance to the $m^{th}$ closest neighbour found so far, does not overlap with the cell under investigation. If the test fails, none of the keys in this cell can be closer to the query then the current $m^{th}$ nearest neighbour and the branch does not need to be examined. This is illustrated in Figure 2.8. Thus, the search algorithm recursively examines nodes in branches which passes the ball-within-bounds test. A more detailed description of ball-within-bounds test and the nearest neighbours search algorithm is given in Appendix 1 and Appendix 2 of [Bentley et al., 1977].
Figure 2.8: A ball-within-bounds test. The figure shows the subspaces of the KD-tree created in Figure 2.7. The red dot represents query image and its one nearest neighbour is to be found. So far the best match is marked with a blue dot but the black dots are yet to be searched. A circle with radius corresponding to the distance of the current best match is centered in the query image. The circle does not cross the lower left subspace in the image which means that this region can be neglected from further searching.
The basic idea behind the image retrieval algorithm is to find matches in the database with the smallest SSD distances to the query image as illustrated in Figure 3.1. The hypothesis is that a neighbour in pose with high probability can be found among the first $k$ nearest SSD neighbours. Thus, the ability of the whole process to use image matches to assist with localisation depends on the existence of a significant correlation between image similarity, as measured in this case by SSD, and similarity in pose, as measured by Euclidean distance. I.e. it is required that low SSD implies low pose separation. Investigation of the correlation strength is one of the contributions of this work.

**Figure 3.1:** Illustration of the fundamental idea behind the image retrieval algorithm. The query image is compared to all images with the SSD as similarity measure.
Since the algorithm is to be used in localisation contexts the whole history of visited places must be stored in the database. This implies a large number of images. Also, visual images representations are high dimensional. An exhaustive approach like this will therefore be very computationally demanding and too slow when the computational power is limited.

The image retrieval process must be sufficiently fast if the database is to be truly useful in real time applications. The algorithm in Figure 3.2 is therefore proposed instead of the basic algorithm. The main difference is the introduction of image dimension reduction. How the dimension reduction should be performed is determined in a training phase of the algorithm. Lower dimension will of course make an exhaustive search faster, but also enable the use of any of the more efficient data search structures described in Section 2.2.2. However, dimension reduction means loss of information and a nearest neighbour search in a lower dimensional space will only give approximately the same result as a search in the full image space. The approximate nearest SSD neighbours from the database might therefore still need to be sorted by their true SSDs before being returned to the user. Nevertheless, full image SSD comparison is now only required on a small subset of the whole database as determined by the lower dimensional neighbourhood. This results in significantly faster computation times with comparable accuracy.

![Diagram](image)

**Figure 3.2:** Illustration of the algorithm suggested in this work. A set of training images (T) is used during a training stage to learn a dimension reduction transform. Approximations of the query (Q) and database (D) images are then computed with this transform before they are compared with the SSD similarity measurement.

A look-up in the database returns \(k\) different database images. These images can be seen as hypotheses. For example, if the database images also have GPS tags, then one would have some hypotheses about the actual geographical position. Another possible scenario is that the images in the database have topological graph relationships, which can also be reasoned about using the obtained hypotheses. However, to be able to interpret the hypotheses and compute a single estimate of
the current location, this database algorithm has to be plugged into some localisation back-end but this is out of the scope for this work.

This image retrieval algorithm was implemented and evaluated in the numerical computing environment MATLAB™. However, some of the more time critical parts of the algorithm were implemented in the programming languages C and C++ environment using so-called MEX-file (MATLAB™ executable file) interfaces to increase the performance of the algorithm.

A more detailed presentation of the algorithm is given later in this chapter but the data used during the development process are described first.

### 3.1 Image data sets

The image data sets used in the development process were captured by a pano-spheric ladybug camera. Each sample generated six images, each with resolution $1232 \times 1616$ pixels, from six cameras pointing in different directions. Five of the images were generated by cameras aligned in the vertical plane while one of them was pointing straight upwards towards the sky as can be seen in Figure 3.3. Not all of the 6 individual cameras were used in this work. The reasons for discarding some of the data are described later in this chapter.

![Figure 3.3: The camera setup. Only images captured with camera 2-5 were used.](image)

All images were captured in urban environments from the back of a car driving on the roads in the heart of a city. GPS coordinates were recorded together with the images.
3.1.1 Environments

Two different data sets were used when creating a test database and selecting query images to use during the algorithm development process. The first image set was collected in a park and the other was collected from a busy business district at noon. The images from the business district contained a large number of moving objects, such as cars and pedestrians, which made it suitable for testing the database look-up algorithm’s sensitivity towards image fluctuations due to dynamics in the environment. Images from the park data set did not contain as many moving objects as the previous set, though there were some present. These images were on the other hand very similar in colours and did not have very many salient details (from a human perspective) since they mostly contained different kinds of vegetation like trees and fields. This set of images could therefore be used for evaluating the algorithm’s sensitivity towards image aliasing. Examples of images from the two different sets of data can be seen in Figure 3.4a and 3.4b. The uncertainty in the obtained GPS coordinates was unfortunately often too large to be of any significant use. This was especially the case when images were collected in the business district since the satellite signals were blocked by tall buildings.

A third data set, also collected in the streets of an urban environment, was used as well. These images were only used when analysing the training phase of the algorithm to determine the response of the dimension reduction learning to different training sets. There was no pose overlap at all between this set and any of the other two sets mentioned above but the sets still contained images captured from similar environments. An example image from this third data set can be seen in Figure 3.4c.

3.1.2 Discarded data

The images collected with the camera pointing towards the sky had very few details making them stand out from other sky images. They turned also, not surprisingly, out to contain lots of saturated pixels and were occluded by noise due to different positions of the sun, clouds and the weather. Thus, these images were considered not to be useful and were therefore rejected.

The hopes were to obtain a rotational invariant method that could recognise a place independent of the rotation of the camera equipment. This was possible thanks to the 360 degree field of view the ladybug camera provided. However, the sensing equipment had a camera pointing forwards in the driving direction but no one pointing backwards. The front images would therefore have no correspondence in the database if driving back on a road in the opposite direction as the first time and this might have made it more difficult to obtain a rotational invariant place recognition algorithm. Also, a large part of the front images were occluded by the car pulling the camera equipment. Hence, the front images were rejected as well and only images from the camera number 2-5 in Figure 3.3 and 3.4 were left to work with.
3.1 Image data sets

(a) An example image from the data set captured in the park environment.

(b) An example image from the data set disjunctive from the park and the busy business districts data sets.

(c) An example image from the data set captured in the streets of an urban environment.

Figure 3.4: Three examples of samples from the ladybug camera. Six individual images are captured in each sample.
This set-up was not an ideal arrangement since the view from these four cameras still differed depending on which direction of the road the car was driving. To obtain true rotational invariance the four cameras should be placed with equal angular displacements. This is further clarified in Figure 3.5. Such a ladybug camera was however not available. Therefore, testing was performed according to the asymmetric configuration shown in Figure 3.3 on the assumption that this would not adversely affect the performance of the algorithm. Experimentation was performed to explicitly test this assumption and the results are shown in Section 4.4.

In [Fergus et al., 2008] it is established that humans can perform object and scene recognition in images with good performance even when the resolution of images are as low as $32 \times 32$ or even $16 \times 16$ pixels in some cases. Similar performance for computers is also shown, which is a good result since low resolution images occupies less memory. The paper shows that objects and scenes can be classified also by a computer using only low resolution representation of images. However, images were only classified and no tests were done to see if the exact same object or location could be identified.

For this report tests have been done to show if the method used in this paper can be used also for exact place identification. This would then imply that the algorithm can be used in localisation contexts and that larger areas could be explored and mapped without risking filling the computer memory up with large images. All collected images were therefore reduced to $32 \times 24$ pixels. All in all, image strips such as the one in Figure 3.6 were what was left to work with.

### 3.1.3 Database and query set selection

The two sets of images mentioned in Section 3.1 above, collected in a park environment and a busy city center, were used to create one single test database and to pick out two sets of query images. In this way the test database contained images from two very different environments, which very well could be the case if this image retrieval algorithm is to be used in large area explorations.

The GPS data were used to identify regions where the vehicle with the camera equipment had passed by more than once. All images from regions with only one passing were saved in the database while images from regions with several passings were divided into the database set and query set respectively. All images from the first passing were saved in the database while frames from the remaining passings were used as query images. Some of the images that should have been put into the query set according to this partitioning were still kept in the database instead. The reasons for this were both to make the database size as large as possible but also to make sure that as many potential observation cases as possible were represented in the database.
3.1 Image data sets

(a) Going up a street the actual camera setup used captures images of the green triangles while the other set-up captures the blue triangles.

(b) Going down the same street we can see that the actual camera setup used can not capture the same image as before. The second camera setup does however still capture the blue triangles.

Figure 3.5: An illustration of what happens when an image is captured from the same place but with the ladybug camera rotated 180 degrees. Two different camera set-ups are shown, the one actually used and one that might have been more optimal.
The query images were then in turn divided into two groups. The first partition only contained query images captured when driving in the same direction along the roads as doing when capturing the corresponding database image. This set will from now on be referred to as the same direction query set. The second partition only contained query images captured while driving in the opposite direction compared to the corresponding database image and will be referred to as the opposite direction query set. This enabled a separate analysis of the rotation invariance of the algorithm. The sizes of the query sets together with the database size after partitioning all images can be seen in Table 3.1.

<table>
<thead>
<tr>
<th></th>
<th>Database</th>
<th>Query - same direction</th>
<th>Query - opposite direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total nr of images</td>
<td>8120 x 4</td>
<td>552 x 4</td>
<td>539 x 4</td>
</tr>
<tr>
<td>% from park set</td>
<td>19%</td>
<td>34%</td>
<td>0%</td>
</tr>
<tr>
<td>% from central city set</td>
<td>81%</td>
<td>66%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 3.1: Overview of the sizes of the database and query sets respectively.

### 3.2 Algorithm

Pseudo code for the image retrieving process can be seen in Algorithm 2. The implementations of each step in this algorithm are described more in detail below in Section 3.2.1-3.2.3.
Algorithm 2 A Database with Tiny Images for Pose Hypotheses Generation

(* Algorithm for generating pose hypotheses *)

**Require:** Set of training images $T$,  
Set of database images $D$,  
Set of query images $Q$

1: %% Training

2: PRE-PROCESS($T$)
3: DIM_REDUCTION(images) := PCA($T$)

4: %% Create database

5: PRE-PROCESS($D$)
6: $\hat{D}$ := DIM_REDUCTION($D$)
7: CREATE_KD-TREE($\hat{D}$)

8: %% Hypotheses generation - image retrieval

9: for all $q \in Q$ do
10: PRE-PROCESS($q$)
11: $\hat{q}$ := DIM_REDUCTION($q$)
12: $I_{knn}$ := SEARCH_KD-TREE($\hat{q}, k$) % Indexes to $k$ nearest neighbours by SSD in low dimensional space
13: $H$ := SORT_BY_SSD($q, D[I_{knn}]$) % Pose hypotheses in form of database images together with associated SSD:s to the queries
14: return $H$
15: end for

3.2.1 Image pre-processing

Training, database and query images are all pre-processed in the same way. The starting point of the pre-processing is an image strip such as described in Section 3.1. This image strip is then split into the four separate images of which it is built. The motivation for this is that the view in some of the individual images might be clear even if others are occluded by for example dynamic objects. Considering each ladybug sample as four individual observations of a place instead of only one increases the chance of getting a correct place identification on at least one of the cameras. Another reason to split the image strips into four independent observations is because the camera order no longer matters. This means that rotational invariant place recognition should be possible.
The next step in the process is to transform each of the four foundation images into a colour description suitable for image comparisons. Several transformations were tested since the best method of doing this could not be known in advance. However, all methods transformed an original RGB image with three channels into a one channel representation. This was due to memory restrictions in implementation software and also since it simplified the implementations of some of the algorithms. All pre-processing transformations that were tested are presented in this section.

1. **RGB normalized with total intensity**

The intensity image $A_{RGB}(x)$ is calculated according to (2.4). The RGB image is then normalized as done in (3.1) - (3.2) before converting it into a gray-scale image according to Section 2.1.1.

$$A_{tot}^{RGB} = \sum_x A_{RGB}(x)$$  \hspace{1cm} (3.1)

$$I_{norm}(x, C) = \frac{I_{RGB}(x, C)}{A_{tot}^{RGB}}$$  \hspace{1cm} (3.2)

2. **RGB to rgb to gray-scale**

The RGB image is first normalized with its intensity according to Section 2.1.1. Then after this normalization, i.e. when the image is transformed into an rgb image with the denotation used in Section 2.1.1, a gray-scale conversion is performed as described in Section 2.1.1.

3. **Hue**

The hue of the RGB image, $H(x)$, is computed according to (2.7).

4. **Saturation**

The saturation of the RGB image, $S(x)$, is computed according to (2.8).

5. **Hue weighted with saturation**

Firstly, the hue and saturation of the RGB image are computed according to (2.7) - (2.8). The second step is then to make the hue property more stable by weighting it with the saturation according to one of the methods discussed in Section 2.1.1. This weighting is done as in (3.3).

$$I_{HS}(x) = H(x)S(x)$$  \hspace{1cm} (3.3)
6. Histogram equalization on RGB intensity image before gray-scale conversion

Histogram equalization can not be performed on the red, green and blue channels of an RGB image independently without changing the relative proportions of the channels. The relative proportions are important to preserve since RGB is an additive descriptor and the proportions therefore determines the actual colors in the image. Hence, histogram equalization is instead performed on the intensity image $A_{RGB}(x)$ (computed according to (2.4)) to preserve all colours in the images. Histogram equalization is described in Section 2.1.2 and the result is a more desirable intensity profile $A_{HE}^{RGB}(x)$. The original RGB image is therefore modified according to (3.4) below, so that its new intensity profile matches this more desirable one. Lastly, a gray-scale conversion is performed according to (2.2) - (2.3).

$$I_{HE}^{RGB}(x, C) = I_{RGB}(x, C) \frac{A_{HE}^{RGB}(x)}{A_{RGB}(x)}$$ (3.4)

7. Histogram equalization on HSB brightness channel before gray-scale conversion

The brightness in the HSB colour description is not related to the actual colour tints in an image. The RGB image is therefore transformed into a HSB description and histogram equalization, as described in Section 2.1.2, is performed only on the B channel before converting the image back into the RGB colour model. This preserves the colours in an image just as the histogram equalization method described above. A gray-scale conversion according to (2.2) - (2.3) then follows after the histogram equalization process.

3.2.2 PCA on image sets for dimension reduction

The objective of the algorithm is to find the images in the database that are closest to a query image. The true closest neighbours can however not be found fast enough due to the high dimensionality of images and an approximate search is done in a lower dimensional space. Principal Component Analysis, as described in Section 2.2.1, is used in the training phase of the algorithm to find an image transformation into this lower space.

After pre-processing, the training image data set consists of $n_T$ images with only one intensity channel. Let these training images be denoted by $I_{T_n}(x)$ where $n = 1, 2, ..., n_T$. Each image is represented as a matrix with $r$ rows and $c$ columns. A single data matrix representation, $X_{T_r}$, has to be formed out of all this data. This is done by first transforming each image in the training set from its image matrix representation into a vector with $n_{pix} = r \times c$ elements according to
\[ \mathbf{I}_{T_n}^{\text{vec}} = \text{vec}\left( \mathbf{I}_{T_n} \right). \] (3.5)

It is necessary for the data to be mean centered before a principal component analysis can be performed. Otherwise it can not be guaranteed that the first basis vector corresponds to the largest variance in the data as desired. If a mean subtraction is neglected, the first principal component might also correspond to a part of the mean besides the variance. Therefore, the mean of the data is computed as in (3.6) and the single matrix containing information from all training images can then be formed according to (3.7).

\[ \mu = \frac{1}{n_T} \sum_{n=1}^{n_T} \mathbf{I}_{T_n}^{\text{vec}} \] (3.6)

\[ \mathbf{X}_T = \begin{pmatrix} \mathbf{I}_{T_1}^{\text{vec}} - \mu \\ \mathbf{I}_{T_2}^{\text{vec}} - \mu \\ \vdots \\ \mathbf{I}_{T_{n_T}}^{\text{vec}} - \mu \end{pmatrix} \] (3.7)

The eigenvectors of the covariance matrix of the data are then found by doing a singular value decomposition on this data matrix. Singular value decomposition is a matrix factorisation

\[ \mathbf{X}_T = \mathbf{U} \mathbf{S} \mathbf{V}^T, \] (3.8)

where \( \mathbf{U} \) and \( \mathbf{V} \) are orthonormal matrices and \( \mathbf{S} \) is a diagonal matrix with only non negative values. The column vectors \( \mathbf{v}_i \), where \( i = 1, \ldots, n_{\text{pix}} \), of

\[ \mathbf{V} = \begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_{n_{\text{pix}}} \end{pmatrix} \] (3.9)

are the eigenvectors to the data correlation matrix. The square root of the eigenvalue \( \lambda_i \) can be found in

\[ \mathbf{S} = \begin{pmatrix} \sqrt{\lambda_1} & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda_2} & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \sqrt{\lambda_{n_{\text{pix}}}} \end{pmatrix} \] (3.10)

in the same column \( i \) as its corresponding eigenvector.
3.2 Algorithm

The principal components are identified by sorting the eigenvectors in $V$ in such a way that the first principal component, $p_1$, equals the eigenvector with the highest eigenvalue, the second, $p_2$, equals the eigenvector with the second highest eigenvalue and so on. A dimension reduction transformation matrix, $P$, is then formed by concatenating the $n_{PC}$ first principal components into a matrix according to (3.11). An arbitrary image with the same number of pixels as the training images, $I(x)$, can then be approximated in a lower dimensional space by applying the transformation in (3.12).

$$P = \begin{pmatrix} p_1 & p_2 & \cdots & p_{n_{PC}} \end{pmatrix}$$  \hspace{1cm} (3.11)

$$\hat{I} = P(\text{vec}(I(x)) - \mu)$$  \hspace{1cm} (3.12)

Note that the approximated image $\hat{I}$ is no longer a matrix but a vector with $n_{PC}$ elements. The length of this vector depends only on the number of principal components used to form the image dimension reduction transformation. The more components used the better the approximation becomes. How many principal components to use must be a balance between the desired accuracy and the final dimension of the approximation. This could not be determined beforehand but had to be evaluated through tests where both database hit rate and image retrieval time could be considered.

3.2.3 Creation of the database KD-tree

All $n_{DB}$ pre-processed database images, $I_{DB_n}(x)$ where $n = 1, 2, \ldots, n_{DB}$, are transformed into a lower space using the transformation in (3.12) obtained during the training phase. Denote such a low dimension approximation of a database image with $\hat{I}_{DB_n}$. Each $\hat{I}_{DB_n}$ is a $n_{PC}$ dimensional vector. These vectors are used as keys when a KD-tree is created. The KD-tree then enables a fast nearest neighbours search process.

The KD-tree was implemented with the nearest neighbour search library ANN\textsuperscript{1} [Mount and Arya, 2012-04-26]. The library came with some settings to determine how to insert the splitting hyperplane when a tree is created. However, in the library manual by Mount [2006] the authors made a suggestion of a method which, according to their experience, worked best in most cases. This method, the sliding midpoint rule, was hence chosen as the splitting principle. The bucket size of the tree was set to one.

According to the sliding midpoint rule a subspace cell in the KD-tree is always split up orthogonal to its longest side. The splitting hyperplane should primarily go through the midpoint of this longest side. However, such a split could result in all points in the cell being on the same side of the hyperplane. The midpoint is in such a case slided along the longest side towards the closest points until finally one point ends up in the empty subcell.

\textsuperscript{1}Version 1.1.2, Release Date 27th of January 2010
3.2.4 Image retrieval

A pre-processed query image, $\hat{I}_{Q_n}$ where $n = 1, 2, \ldots, n_Q$, is transformed with (3.12) into the same low-dimensional space as the database images. Let $\hat{I}_{Q_n}$ denote such a low dimension approximation of a query image. The true SSD between the query $n$ and a database image $m$ is computed according to (2.15) but is in this space approximated with

$$\hat{\text{SSD}} = \|\hat{I}_{Q_n} - \hat{I}_{DB_m}\|^2.$$ (3.13)

The $k$ approximate closest neighbours to the query image according to this measurement are found in the database KD-tree with help of the nearest neighbour search algorithm described in Section 2.2.2.

This neighbour search will not give the exact same result as a search in the full image space due to information loss during the dimension reduction. Small distances in the full space will still be small in the lower space but the exact rank of the closest neighbours may be permuted. It is therefore likely that the number of approximate neighbours which need to be found are larger then the actual number of true SSD neighbours that are of interest. That is, $k$ can be seen as a design parameter and how to best tune it had to be evaluated.

The very last step in the algorithm is to re-sort the approximate neighbours according to their true SSDs to the query. This is done since it is retrieval of the closest neighbours in the full image space, not the approximate closest neighbours, that is desired. In other words, the true SSDs between the database images obtained from the approximate search and the query image are computed to undo the neighbours permutation before the result is returned to the user. Alternatively, this last step could be to sort the approximate neighbours according to their true TSSDs instead. If the SSD or the TSSD gives the most satisfying result is evaluated in Section 4.4, in addition to verifying whether the dimension reduction is sufficiently accurate to avoid subsequent sorting altogether.

The TSSD involves image aligning with the Inverse Compositional algorithm in Section 2.1.4. The values corresponding to a pixel $x$ in the query image are warped onto another pixel $\tilde{x} = W(x, p)$. $p = (p_1, p_2, \ldots, p_{n_W})$ are the warp parameters and the algorithm chooses them in such a way that the error between the query image and a database image is minimized. The Inverse Compositional algorithm is only compatible with sets of warps $\tilde{x} = W(x, p)$ that form a group.
The two most obvious image transformations present in the data sets used in this work were translation and scaling but other such as for example small rotations did also occur. A warp which could compensate for these kinds of transformations was therefore needed. The set of affine warps in (3.14) was chosen. An affine warp corresponds to a linear warp followed by a translation. Image transformations such as translation, scaling, contraction or dilation, reflection, rotation and shear can be expressed with this warp.

$$\tilde{x} = W(x, p) = \begin{pmatrix} 1 + p_1 & p_3 & p_5 \\ p_2 & 1 + p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ 1 \end{pmatrix}$$  \hspace{1cm} (3.14)

### 3.2.5 Design parameters

A summary of the design parameters and their values can be seen in Table 3.2.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>One of the, in Section 3.2.1 mentioned, seven different pre-processing modes</td>
<td>Evaluated in Section 4.2</td>
</tr>
<tr>
<td>Number of principal components used for dimension reduction</td>
<td>Evaluated in Section 4.3</td>
</tr>
<tr>
<td>KD-tree: Splitting criteria</td>
<td>Sliding midpoint rule</td>
</tr>
<tr>
<td>KD-tree: Bucket size</td>
<td>1</td>
</tr>
<tr>
<td>KD-tree: Exact or approximate neighbours search</td>
<td>Exact</td>
</tr>
<tr>
<td>Number of approximate closest neighbours to search for</td>
<td>Evaluated in Section 4.3</td>
</tr>
<tr>
<td>SSD/TSSD/nothing in last sorting step</td>
<td>Evaluated in Section 4.4</td>
</tr>
<tr>
<td>Image alignment warp in IC algorithm</td>
<td>Affine warp</td>
</tr>
</tbody>
</table>

*Table 3.2: Design parameters*
This chapter describes the experimental methodology used in this work to evaluate the algorithm. Results from the experiments together with related analyses are presented as well.

There were basically three things which needed to be evaluated. Firstly, the correlation between the SSD image similarity measurement and pose from which images are captured had to be investigated. Secondly, the values of some of the design parameters in Table 3.2 had to be determined. Also, when all the design parameters were determined the definitive database efficiency could be estimated.

### 4.1 Evaluation methods

Two different ground truths were needed for the evaluation, pose references and SSD neighbourhoods to all query images. These ground truths are further described in this section together with an explanation of how the hypotheses obtained from a database search are interpreted in this work.
4.1.1 Ground truth

Two different ground truths were used when evaluating the performance of the suggested image database algorithm. One connects each query image with the database image which is closest in pose. This will be referred to as the true closest pose neighbours to the queries. The other consists in the SSD measurements from each query image to all database images when no dimension reduction is performed but after the pre-processing step. This will be referred to as the true closest SSDs.

True closest pose neighbours

The GPS data obtained together with the captured images were sometimes very poor, as mentioned in Section 3.1. Therefore, instead of using the obtained GPS positions to compute the pose ground truth it had to be done manually simply by viewing all images. It was often very difficult to determine one single image in the database closest in pose to a query image. However, the manual labeling was done much more easily if the span of database images connected to each query image was increased to three. The loss in pose accuracy was considered to be sufficiently small since the images were captured with a small average separation in meters. The pose ground truth for each query image was thus chosen as three corresponding images from the database. This ground truth was used to evaluate the efficiency of the complete database image retrieval algorithm.

True closest SSD/TSSD neighbours

The true closest SSD/TSSD neighbours were more easily determined then the true pose neighbours. They were directly calculable from the data, by using exhaustive computation of all query-database image pairs. Recall that the direct calculation of SSD/TSSD is feasible as per Figure 3.1, but too slow for real time use and not scalable for order of magnitude larger datasets. In these experiments, true SSD/TSSD is calculated to allow comparison with the scalable yet suboptimal dimensionality approximation. Since the distances after the pre-processing step were desired, this had to be done individually for each pre-processing mode described in Section 3.2.1.

4.1.2 Evaluation of the database search result

The ladybug camera used in this work took several images from each visited place as described in Section 3.1. Four images were used from each place when performing the database searches. However, the four images were treated completely independently by the algorithm. $k$ sorted nearest neighbours were thus returned to each individual images, i.e. in total $k \times 4$ neighbours were returned each time the database was called. These neighbours represented places which were visually similar to the location in the query image. Therefore, they could be interpreted as four sets of ranked pose hypotheses, or more correctly spatial proximity hypotheses. These hypotheses should be passed on to some localisation back-end before a final pose estimate could be obtained. This implies that the accuracy in a final pose estimate depends on the performance of the back-end.
However, this back end was out of scope for this work. The essential when evaluating the database algorithm in this work was thus to determine how good a final estimate could be at best and the following definition, illustrated in Figure 4.1, was made. If a true pose neighbour is said to be found among the $k$ nearest neighbours to a query image then it is true that it can be found among the $k$ nearest neighbours to at least one of the four individual images forming a query image strip.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure4_1.png}
\caption{A query generates several sets of ranked hypotheses, $k$ nearest neighbours, when performing a database search. The objective is to find a pose neighbour to the query. A lower rank of an identified pose neighbour is considered to be a better search result. If a hypothesis with rank two from one of the generated hypotheses turns out to be a correct match while the matches from the other hypotheses sets are worse, then it is stated that the true pose neighbour can be found among the 2 nearest neighbours to the query.}
\end{figure}

Consider the following example to illustrate the concept of a correct database match:

- A new Ladybug image forms a query. It is splitted into four individual images and the neighbours of each are requested from the database.

- The 1st nearest neighbour to image one of four happens to be the nearest ground truth pose neighbour.

- The other three images return incorrect or lower ranking pose neighbours by chance.
In this case, it is reported that the true pose neighbour can be found among the 1st nearest neighbours to the query.

Important to point out is that there will not exist any knowledge about the true pose to compare the hypotheses with when running the algorithm in a real system. Knowledge about which neighbour is actually the best spatial proximity hypothesis will therefore not be available. It is this task, to determine the most likely hypothesis, that is handed over to the localisation back-end. It could for example be done by testing the hypothesis against other data or by forming a topological connectivity network of some kind to indicate the relative layout of the images. The critical thing for the algorithm in this thesis is actually to ensure (or ensure probabilistically to some percentage) that the set of hypotheses will include the true pose neighbour if it exists in the database. For a given desired percentage guarantee (e.g. 90%), how many neighbours must be picked? The lower number of required neighbours/hypotheses and the higher percentage guarantee, the more efficient will the wider system be able to be. The performance of the database was therefore evaluated based on these two parameters.

**Calculation of probability of generating a correct pose hypothesis**

The probability of finding a true closest pose neighbour among some \( k \) nearest neighbours for different \( k \), or in other words the probability of finding a true pose hypothesis if only the \( k \) best ranked hypotheses are considered, was used to evaluate the database performance. High probability and a low number of neighbours indicate an efficient database according to previous reasons.

The desired probability was estimated from the sampled query sets described in Section 3.1.3. The query sets contains images from some of the places in the database. The number of nearest neighbours needed before a true pose neighbour could be found among them was calculated for each query image and in a second step could the probability of finding a correct hypotheses among the \( k \) nearest neighbours be computed.

A binary test was carried out for some different given numbers of \( k \) nearest neighbours. The question to be answered was if a correct pose hypothesis could be found among these \( k \) neighbours. The answers from each query were independent of the answers from all the others. A binary test like this is called a Bernoulli test and the probability \( f(n_{\text{hit}}(k), n_{\text{query}}, p) \) of getting \( n_{\text{hit}}(k) \) number of successes (yes answers) when the number of queries are \( n_{\text{query}} \) follows the binomial distribution

\[
f(n_{\text{hit}}(k), n_{\text{query}}, p(k)) = \binom{n_{\text{query}}}{n_{\text{hit}}(k)} p(k)^{n_{\text{hit}}(k)} (1 - p(k))^{n_{\text{query}} - n_{\text{hit}}(k)}.
\]

(4.1)

Here \( p(k) \) is the probability of getting a yes in a single Bernoulli trial when considering \( k \) nearest neighbours and this was the probability of interest in this work.
In other words, the probability to be computed was the probability that any of the $k$ nearest SSD neighbours corresponded to the true pose of a query image randomly selected from some underlying, unknown distribution. The intuitive probability estimate $\hat{p}$ in (4.2) can be derived from the binomial distribution, using for example the maximum likelihood method.

$$\hat{p}(k) = \frac{n_{hit}(k)}{n_{query}}$$  (4.2)

The formula in (4.3) for computing the two sided $(1 - \alpha)\%$ confidence interval for this probability estimate can be derived if the binomial distribution is approximated with a normal distribution. This approximation is justified by the central limit theorem.

$$\hat{p}(k) \pm z_{1-\alpha/2} \sqrt{\frac{\hat{p}(k)(1 - \hat{p}(k))}{n_{query}}}$$  (4.3)

where $z_{1-\alpha/2}$ is the $1 - \alpha/2$ percentile of the standard normal distribution\(^1\). If computing a 95% confidence level, then $z_{1-\alpha/2} \approx 1.96$. The confidence interval does only depend on the probability estimate $\hat{p}(k)$, which is in the range $[0, 1]$ for all $k$, and the number of query images. Two different sets were used in this work, the same direction set with $n_{query} = 552$ and the opposite direction set with $n_{query} = 539$. Figure 4.2 shows all possible widths on the confidence interval for these two different sizes of the query sets. One can see that the intervals are rather small, at maximum 0.045 for all possible $k$ which means that the probability estimates computed later on in this chapter are accurate. It also implies that, when comparing probability estimate graphs with each other as done later in Section 4.2, the confidence intervals can not intersect if the graphs are separated with more than $2 \times 0.045 = 0.09$ units.

\(^1\)A normal distribution with expected value 0 and variance 1.
Figure 4.2: The width of the 95% confidence interval for the probability estimate for the two different query image sets used in this work and for all possible probability estimates.

4.2 Pre-processing evaluation

The first thing to be determined was if the true SSD between two images is correlated to the pose from where the images were captured. The pre-processing method which highlights this correlation best and under most circumstances could also be determined from the same test. This was done by combining the true closest pose neighbours ground truth with the true closest SSDs ground truth. The probability of finding a true pose neighbour to a query image among its \( k \) true closest SSD neighbours in the database was computed. This was repeated for all seven pre-processing methods suggested in Section 3.2.1. It was also done for the case where no pre-processing except RGB to gray-scale conversion was used to ensure that the pre-processing did not have the opposite effect as desired.

The query and the database sets were captured under quite similar lighting conditions. Therefore, the test was repeated for two different noise types added to the query to simulate bigger differences. The two noises used are

1. The whole image was multiplied with a random number \( e \in \mathcal{U}[0.5, 1.5] \) sampled from a uniform distribution.

\[
I_{\text{RGB}}^{\text{noisy}}(x, C) = e I_{\text{RGB}}(x, C)
\] (4.4)
2. Each pixel $x$ in the image was multiplied with a random number $e(x) \in \mathcal{U}[0.5, 1.5]$ sampled from a uniform distribution.

$$I_{\text{RGB}}^\text{noisy}(x, C) = e(x)I_{\text{RGB}}(x, C)$$ (4.5)

After multiplication were the pixel values saturated to $[0, 255]$ and also rounded to the closest integer. This was to keep the same image representation as in the original images.

Multiplying a pixel with a number changes its intensity, i.e. it makes the pixel darker or lighter. Multiplying with a number smaller than 1 makes the pixel darker and multiplying with a number greater than 1 makes the pixel lighter. Examples of this can be seen in Figure 4.3 where a whole image has been multiplied with 0.5 and 1.5 respectively.

![Figure 4.3: The image in the center has been multiplied by 0.5 (left) and 1.5 (right) respectively to illustrate the effect of the added noise.](image)

The same direction query set was used when evaluating the suggested preprocessing methods since only the effects from environmental dynamic changes were to be considered and not any rotational invariance aspects. Hypotheses for all queries in this set were generated and used when calculating the probabilities in Figure 4.4-4.6. One can see that the SSD measure seems to be related to the pose quite well. This relation is very clear, even without any specific pre-processing, when no noise is added in Figure 4.4. In fact, with no extra noise the only two pre-processing methods which significantly increases the probability of finding a desired match are method 2, rgb normalization, and method 6, histogram equalization on the intensity image. However, all pre-processing methods except 3, hue, improves the result when noise of type 1 is added to the query images in Figure 4.5. Using only hue as image representation is not an option at all since it is significantly worse then the other pre-processing methods, both with and without noise. Method 6 gives the highest probability of finding a true pose neighbour when considering only a small number of closest SSD neighbours, both without and with noise of type 1 or 2, Figure 4.5 - 4.6. With this pre-processing the probability of finding a true pose neighbour as the single absolute nearest SSD neighbour is about 88% when no noise is added. Important to point out is also that the confidence interval for the probability graphs obtained with this method 6 can not intersect with any other of the graphs for small number of nearest neighbours.
according to the reasoning in Section 4.1.2. Method 6, histogram equalization on
the intensity image, was therefore considered to be the best choice and all other
pre-processing methods were discarded from further evaluations.

![Graph](image_url)

(a) The probability of finding the true pose neighbour in the
set of $k$ true nearest neighbours

![Graph](image_url)

(b) Detailed graph for $k \leq 15$

**Figure 4.4:** Probability (y axis) of finding a true closest pose neighbour among the $k$ true closest SSD neighbours (x axis). No noise is added to the images.

The average time it took to do an exhaustive SSD search in the full image space
4.2 Pre-processing evaluation

(a) The probability of finding the true pose neighbour in the set of $k$ true nearest neighbours

(b) Detailed graph for $k \leq 15$

**Figure 4.5:** Probability (y axis) of finding a true closest pose neighbour among the $k$ true closest SSD neighbours (x axis). Noise type 1 is added to all query images.
4 Experimental results

(a) The probability of finding the true pose neighbour in the set of $k$ true nearest neighbours

(b) Detailed graph for $k \leq 15$

Figure 4.6: Probability (y axis) of finding a true closest pose neighbour among the $k$ true closest SSD neighbours (x axis). Noise type 2 is added to all query images.
was also noted for further performance comparisons with the complete algorithm. This time was approximately 19.7 seconds per query image strip (a strip consists of four individual images) including the pre-processing time for the chosen histogram equalization method.

### 4.3 Evaluation of the approximate search

A dimension reduction means loss of information and a nearest neighbour search in the low dimensional space spanned by $n_{PC}$ principal components will therefore not generate the exact same result as a search in the full space. However, if choosing only the $k$ closest approximate neighbours to a query image from the database, the non approximate SSDs have to be computed only for a small subset of the database. This significantly speeds up the image retrieval process so that the database can be scaled to significantly larger sizes and so that it can be used in real time applications. The $n_{PC}$ and $k$ parameters will have influence over the probability of finding a true SSD neighbour in the database subset. It is desirable to make this probability as large as possible since we from Section 4.2 know that a true pose neighbour often can be found if only the true SSD neighbours are found. The approximate database search was therefore executed for the same and the opposite direction query sets merged together into one single set of query images. The result was then compared to the true SSD:s ground truth and the number of approximate neighbours $M$ needed to have a probability of 90% to find the first $N$ true closest neighbours were computed. Note that the approximate search was preceded by the pre-processing choosen in Section 4.2.

The test was done for some different number of principal components between 1 and 20. The upper limit of the number of principal components was set to 20 since the KD-tree implementation used had an efficiency dip for data with larger dimensionality.

The approximate search was also evaluated for two different sets of training images. Firstly was the database itself used as a training set. If all database images are already collected and no new images are to be added, for example as in offline applications, this will be the optimal choice of training images since it will make the PCA approximation as good as possible. However, PCA is a time expensive procedure which it would be impossible to redo each time a new image is added to the database without loosing real time performance of an online application. It is therefore also of interest to know if the dimension reduction transform can be learned from another training set disjunctive from the database. If so, the principal component analysis only has to be carried out once even if new images are inserted. The third set of images described in Section 3.1.1 was therefore also used as a second training set.
All the results can be seen in Figures 4.7-4.8. It can be seen that the gain of using more principal components in the dimension reduction transform decreases dramatically after 10 for both choices of training data sets. One can also see that not using the database as training image set is highly possible. For having a probability of 90% to find the one true closest SSD neighbour after search in the lower dimension one would need to consider the 203 approximate closest neighbours when using the database and 221 when using the disjunctive set. This would mean a reduction of the total database size with slightly more than 99% in both cases.

\[ \text{nr of true neighbours requested, } N \]
\[ \text{nr of approximative neighbours needed, } M \]

\[ \text{PC1} \]
\[ \text{PC3} \]
\[ \text{PC5} \]
\[ \text{PC10} \]
\[ \text{PC15} \]
\[ \text{PC20} \]

(a) Different choices of principal components used for dimension reduction.

(b) 15 principal components used for dimension reduction.

**Figure 4.7:** Number of approximate closest neighbours, \( M \), (y axis) needed for having a probability of 90% of finding all \( N \) true closest SSD neighbours (x axis). The database is used also for the PCA.
4.3 Evaluation of the approximate search

(a) Different choices of principal components used for dimension reduction.

(b) 15 principal components used for dimension reduction.

Figure 4.8: Number of approximate closest neighbours, $M$, (y axis) needed for having a probability of 90% of finding all $N$ true closest SSD neighbours (x axis). A set disjunctive from the database is used for the PCA.
An alternative way of tuning the $n_{pc}$ parameter is to look at the eigenvalues to the principal components obtained by PCA. The eigenvalues can, as mentioned in Section 2.2.1, be interpreted as how much of the data that can be explained by the corresponding principal component vector. Figure 4.9 shows these eigenvalues plotted in decreasing order. One can clearly see a breakpoint in the graphs where the information gained by adding more components decreases dramatically. This knee can be found around the 4th eigenvalue and this implies that only 4 eigenvalues are sufficient. However, it turned out that the computational time gained when considering a fewer number of nearest neighbours, possible when using the larger number of principal components, was greater than the time gained when decreasing the amount of principal components to 4. The choice of using 15 components seemed therefore to be most reasonable.

With these two above mentioned experiments in mind, the number of principal components was chosen to be 15 and the proportion of the database kept after the approximate search was set to 1%. Also, the disjunctive set was used in all further evaluation since the hopes were that the database could become useful in online applications.

![Figure 4.9: The eigenvalues corresponding to the principal components obtained with PCA plotted in decreasing order. The PCA has been performed on two different sets of data with similar results.](image.png)
4.4 Evaluation of the complete algorithm

The complete algorithm as suggested in Chapter 3 was run for both the same and the opposite direction query sets and for the design parameter values determined above in Section 4.2 and 4.3. The probability of finding a true pose neighbour to a query image among the k closest neighbours remaining after database image retrieval was computed and the results can be seen in Figure 4.10 - 4.11. Figure 4.10 shows that the result is quite satisfying for the same direction query set. The probability of finding a true pose neighbour among the closest neighbours after database image retrieval is about 87%. The difference is not big at all compared to the 88% obtained when doing a true SSD closest neighbour search instead of an approximate SSD search after image dimension reduction. Important to point out is also that just over 10 nearest neighbours are enough for a 95% chance of generating a correct hypothesis as shown in Figure 4.10.

![Figure 4.10](image.png)

**Figure 4.10:** Probability (y axis) of finding a true closest pose neighbour among the k closest SSD neighbours (x axis) after database image retrieval. The same direction query set is used.
However, Figure 4.11 shows that the method is very poor for recognising query places when the camera equipment has been rotated 180 degrees. It can be seen that the probability drops to below 30% when using SSD with no image aligning (i.e. no TSSD) even though as many as 300 nearest neighbours are considered.

![Graph](image)

**Figure 4.11:** Probability (y axis) of finding a true closest pose neighbour among the k closest SSD neighbours (x axis) after database image retrieval. The opposite direction query set is used.

The average computational time per query image for this test was noted as well and it had decreased to about 0.25 seconds including pre-processing compared to the 19.7 seconds for the exhaustive search.
4.4 Evaluation of the complete algorithm

4.4.1 TSSD

The issue with rotational invariance could possibly be solved by using the TSSD similarity measurement instead of the SSD similarity measurement (aligning images before computing the SSD) as described in Section 2.1.3. The probability of finding a true pose neighbour among the first $k$ true closest TSSD neighbours was computed and compared with the corresponding result for the true SSD neighbours to see if any improvements could be done. The result can be seen in Figure 4.12.

The true TSSD seems to have little correlation with the pose compared to the true SSD. It seems like the image transformation step in TSSD aligns the query image to the database images in such a way that also images which are not very similar in their original form, gets a small distance to each other after the transformation. This is in a way a good result since data aligning was to be avoided as much as possible in this work. If the TSSD had proven itself to be much more useful than the SSD similarity measurement then, it would only have proven that data aligning is rather essential after all. Also, an approximate search after the closest TSSD neighbours can not easily be done in the KD-tree and a completely different approach might have been necessary to explore instead if TSSD had outperformed SSD.

The database hit probability was also computed for the case where an approximate SSD search was done in the KD-tree to make sure that SSD is a better measure of similarity then TSSD. The difference to previous tests is that the approximate neighbours retrieved were now sorted by their TSSDs instead of their SSDs in the last step of the algorithm. The TSSDs might have been a better measure of similarity when a big portion of the database images, which easily can be classified as very different from the query image, had already been sorted out. Parameter values determined in Section 4.2 and 4.3 above were used for the approximate search.

The result from this test can be seen in Figure 4.13. The performance of the TSSD measure is still not as good as the performance of the SSD measure for the same direction query set. For the opposite direction query set the probability of getting a correct database hit was increased when using the TSSD and considering a quite large number of neighbours after image retrieval. However, for only a small number of neighbours, which was more interesting, was the performance of the TSSD and SSD measurements very similar. The final conclusion was therefore that the SSD measure was the best choice.
4 Experimental results

Figure 4.12: A comparison between the SSD ground truth and the TSSD ground truth. The graphs show the probability (y axis) of finding a true closest pose neighbour among the $k$ true closest SSD/TSSD neighbours (x axis).
4.4 Evaluation of the complete algorithm

Figure 4.13: A comparison between the database hit probability when sorting the approximate neighbours by their true SSDs and true TSSDs respectively. The graphs show the probability (y axis) of finding a true closest pose neighbour among the k first images obtained from the database.
4.4.2 True SSD compared to approximate SSD

The last step in the suggested algorithm is to compute the true SSD:s, computed in the full image space, between the query image and the k approximate closest neighbours, found after image reduction to a lower dimensional space. To try to speed things up even more then already done, a comparison between the result before and after this step was made. The question to be answered was how much the result is improved by sorting the approximate database neighbours by their true SSD:s. The answer can be seen in Figure 4.14. One can see that skipping this last sorting step decreases the probability of finding a true pose neighbour among the first k suggested hypotheses remaining after database look-up. However, the average computational time per query image was much shorter. The computational time for a look-up was now reduced from the 0.25 seconds per query image strip mentioned before to about 0.05 seconds. It could even be reduced to 0.03 seconds if only the first approximate closest neighbour was of interest.
4.4 Evaluation of the complete algorithm

Figure 4.14: Probability (y axis) of finding a true closest pose neighbour among the k closest SSD neighbours (x axis) after database image retrieval. Comparison between the result obtained after just finding the the k approximate closest neighbours in the KD-tree and the result obtained after also sorting these by their true SSD:s to the query image.
An image database for fast place recognition has been suggested. Images from a ladybug camera, which takes four images each sample, are stored in the database. The database is also trained using principal components analysis to learn a dimension reduction transform which can be used to transform images into a lower space. A query with four related ladybug images can then be fed to the database and a fast closest SSD neighbours search can be done in a KD-tree to obtain pose hypotheses. Each individual ladybug image generates $k$ hypotheses ordered according to their likeliness of being correct.

Some different suggestions of how to tune the database has been evaluated and the final proposition can be seen in Table 5.1. There are some different tuning options to choose between. The first is to find $k$ neighbours in the database KD-tree and then sort these by their true SSD:s to the query image. This tuning gives a higher probability of generating a correct hypothesis but the database look-up time is a bit longer. If the look-up time is critical then the last sorting step can be excluded with some loss of accuracy in the database image retrieval.

The database turned out to have a high probability of generating correct pose hypotheses if the ladybug camera was not rotated when capturing the query images compared to when capturing the database images. It also showed real time potential since the image retrieval times were small and the PCA did not need to be based on the database itself, but could be pre-computed from a set of visually similar images. The database look-up time together with the probability of generating a correct pose hypothesis can be seen in Table 5.2 where a comparison between three different tuning options are made.
### Table 5.1: Final design parameters suggestions

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-processing method</td>
<td>Histogram equalization on image intensity image</td>
</tr>
<tr>
<td>Number of principal components used for dimension reduction</td>
<td>15</td>
</tr>
<tr>
<td>KD-tree: Splitting criteria</td>
<td>Sliding midpoint rule</td>
</tr>
<tr>
<td>KD-tree: Bucket size</td>
<td>1</td>
</tr>
<tr>
<td>KD-tree: Exact or approximate neighbours search</td>
<td>Exact</td>
</tr>
<tr>
<td>Number of approximate closest neighbours to search for</td>
<td>1% of database size or 1 neighbour</td>
</tr>
<tr>
<td>SSD/TSSD/nothing in last sorting step</td>
<td>SSD or no sorting</td>
</tr>
</tbody>
</table>

### Table 5.2: Comparison of the image retrieval performance for three different tunings of the database. The look-up time includes time for pre-processing query images but not for pre-processing database images or building the KD-tree. Each hypotheses set returned by the database contains four individual hypotheses/images, one for each camera on the ladybug equipment.

<table>
<thead>
<tr>
<th>Tuning description</th>
<th>Time [s]</th>
<th>Prob of generating correct hypothesis among $k$ most likely hypotheses sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr of approx neighbours: 1% of database size Sort approx neighbours by: SSD</td>
<td>0.25</td>
<td>87%</td>
</tr>
<tr>
<td>Nr of approx neighbours: 1% of database size Sort approx neighbours by: No sorting</td>
<td>0.05</td>
<td>82%</td>
</tr>
<tr>
<td>Nr of approx neighbours: 1 Sort approx neighbours by: No sorting</td>
<td>0.03</td>
<td>82%</td>
</tr>
</tbody>
</table>

The look-up time includes time for pre-processing query images but not for pre-processing database images or building the KD-tree. Each hypotheses set returned by the database contains four individual hypotheses/images, one for each camera on the ladybug equipment.
The ANN library used for implementing the KD-tree comes with settings for approximate nearest neighbour search which could speed up the process even more. These options were not utilized in this work due to reasoning that properties of a non-approximate search should be explored first before trying to optimize things further. However, future improvements of the algorithm suggested in this work could perhaps involve explorations of these settings.

The biggest drawback with this algorithm is the lack of rotational invariance. It is possible that this issue can be overcome with a better camera placement on the ladybug sensing equipment then used in this work. (See Section 3.1 for a description of the ladybug camera). However, it is not always possible to change the camera setting arbitrarily and a good extension of the suggested algorithm could be to make its rotational invariance properties more independent of the camera placement. For this purpose could for example the Fourier-Mellin transform [Derrode and Ghorbel, 2001] be explored since its resulting spectrum is invariant to both image scaling, translations and rotations.

Another useful extension of this work is the development of the localisation base-end needed for interpretation of the pose hypotheses into an actual pose estimate. A particle filter [Arulampalam et al., 2002] could perhaps be useful for this. The particle filter has successfully been used in many applications to solve non-linear or non-gaussian tracking problems. Another option could be to explore the approach taken by Byun and Kuipers [1991] where a topologically map is used to determine known sequences of distinct places for a robot to travel between. Loop closures are then successfully detected by comparing the true sequence of places which the robot travels through with possible sequences derived from the map.
Derivation of Principal Components Analysis

Let $\mathbf{x} \in \mathbb{R}^n$ be a random vector for which to compute a set of basis vectors $\mathbf{p}$ in such a way that an approximation $\hat{\mathbf{x}}$ of $\mathbf{x}$ can be obtained according to (A.1) below.

$$
\hat{\mathbf{x}} = \sum_{i=1}^{m} y_i \mathbf{p}_i \quad m \leq n
$$

(A.1)

The number of new basis vectors, $m$, should be less then or equal to the dimension $n$ of $\mathbf{x}$. If $m = n$ then $\hat{\mathbf{x}}$ will in fact not be an approximation of $\mathbf{x}$, but will be exactly equal to $\mathbf{x}$. The basis vectors $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_m$ are the principal components and $y_1, y_2, ..., y_m$ are the new coordinates for the coordinate system spanned by $\mathbf{p}$. The coordinate $y_i$ can be formed by taking the inner products of $\mathbf{x}$ with corresponding basis vector $\mathbf{p}_i$ according to (A.2).

$$
y_i = \mathbf{x}^T \mathbf{p}_i \quad i = 1, 2, ..., m
$$

(A.2)

Assume that $k$ observations of the random vector $\mathbf{x}$ are available. Then a $k \times n$ observation matrix $\mathbf{X}$ can be formed where each row corresponds to an observation $\mathbf{x}_j^T$ and $j = 1, 2, ..., k$. Let then $\mathbf{P}_m$ be a $k \times n$ matrix where the columns corresponds to an orthogonal (uncorrelated) basis for $\mathbf{X}$. The two equations (A.1) and (A.2) above can then be expressed in matrix form according to

$$
\hat{\mathbf{X}} = \mathbf{Y}_m \mathbf{P}_m^T
$$

(A.3)

$$
\mathbf{Y}_m = \mathbf{X} \mathbf{P}_m.
$$

(A.4)

$\mathbf{Y}_m$ is a $k \times m$ matrix where the columns corresponds to the coordinates for each column vector (basis vector) in $\mathbf{P}_m$. 
If introducing the constraint that the column vectors of $P_m$ must have unit norm and recalling that the column vectors also are orthogonal to each other, one can easily see that (A.5) must hold. In this equation $I_m$ is the $m \times m$ identity matrix.

$$P_m^T P_m = I_m$$  \ (A.5)

The problem is now to determine the matrix $P_m$ in such a way that the error when approximating $x$ with $\hat{x}$ is as small as possible. If we let $p_{m+1}, \ldots, p_n$ be another set of orthonormal basis vectors such that $p_1, \ldots, p_n$ spans the whole $\mathbb{R}^n$, then the problem can be stated as minimizing the expression in (A.6).

$$\varepsilon = x - \hat{x} = \sum_{i=m+1}^n y_ip_i$$  \ (A.6)

An appropriate way of minimizing the residuals is to minimize the expected value for the mean square of the residuals

$$\xi = E[|\varepsilon|^2] = E[\varepsilon^T \varepsilon] = E \left[ \left( \sum_{i=m+1}^n y_ip_i \right)^2 \right]$$  \ (A.7)

The expectation operator is linear and $p_i$ are orthonormal vectors for all $i$. The expression describing $\xi$ can therefore be simplified according to (A.8).

$$\xi = \sum_{i=m+1}^n E[y_i^2]$$  \ (A.8)

This expression can be even further expanded by use of (A.2) (note that $y_j$ is a scalar which implies that $y_j = y_j^T$).

$$\xi = \sum_{i=m+1}^n E \left[ (p_i^T x)(x^T p_i) \right] = \sum_{i=m+1}^n p_i^T E[xx^T] p_i = \sum_{i=m+1}^n p_i^T R p_i$$  \ (A.9)

Here $R$ denotes the covariance matrix for $x$. The expression in (A.9) is a multi-variable expression and can be minimized by using the method of Lagrangian multipliers. This method yields the set of first order differential equations

$$0 = \frac{\partial}{\partial p_i} \xi = 2(R p_i - \lambda_i p_i) \quad , i = m + 1, \ldots, n.$$  \ (A.10)
This can be recognised as an eigenvalue problem. Thus, the solution consists in the eigenvalues and eigenvectors of the covariance matrix of $\mathbf{x}$. Now, since the $p_i$ are determined by the eigenvectors of $\mathbf{R}$ the minimization problem in (A.8) can be reduced to minimizing the sum of the eigenvalues $\lambda_i$ corresponding to the set of basis vectors not used in the approximation of $\mathbf{x}$.

$$\min_\xi \xi = \min_{\lambda_i} \sum_{i=m+1}^{n} \lambda_i$$  \hspace{1cm} (A.11)

To summarize, the best approximation, under the restriction that only $m$ out of in total $n$ basis vectors are to be used, is to use the $m$ eigenvectors to the data correlation matrix with the largest eigenvalues as illustrated in A.12 - A.13.

$$\left\{ \lambda_1, \lambda_2, \ldots, \lambda_n \right\} = \text{eigval}(\mathbf{R})$$
$$\left\{ p_1, p_2, \ldots, p_n \right\} = \text{eigvec}(\mathbf{R}), \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \hspace{1cm} (A.12)$$
$$\mathbf{P}_m = \begin{pmatrix} p_1 & p_2 & \cdots & p_m \end{pmatrix} \hspace{1cm} (A.13)$$
The Lucas-Kanade algorithm

The Lucas-Kanade (LK) algorithm is an algorithm for aligning an image $I(x)$ to a template image $T(x)$. The aim is to find warp parameters $p$ to the warp $\tilde{x} = W(x, p)$ so that $I$ can be warped into $T$ as accurate as possible. The algorithm can be derived from the optimization problem in (B.1). This problem is, by the LK algorithm, solved in an iterative process where, in each step, it is assumed that an approximate estimate of the warp parameters is known. The warp can then be updated with the solution of $\Delta p$ after every iteration.

$$
\min_{\Delta p} \sum_x [I(W(x, p + \Delta p)) - T(x)]^2 
$$  (B.1)

A derivation of the algorithm can be found in [Baker and Matthews, 2002] and it can be summarized as in Algorithm 3. In this algorithm the Hessian $H$ has to be re-evaluated at each iteration which implies great computational costs.
Algorithm 3 Lucas-Kanade

(∗ Algorithm for aligning images ∗)

Require: Template image $T(x)$, Image $I(x)$, Initial warp parameters guess $p_{\text{guess}}$, Warp $W(x, p)$

1: $p \leftarrow p_{\text{guess}}$
2: while alignment not good enough do
3:     Compute $I(W(x, p))$ by warping $I(x)$ with $W(x, p)$
4:     Compute the error image $T(x) - I(W(x, p))$
5:     Compute the gradient $\nabla I$ and warp it with $W(x, p)$
6:     Compute and evaluate the Jacobian, $\frac{\partial W}{\partial p}$, of the warp at point $(x, p)$
7:     Compute the steepest descendent images $\nabla I \frac{\partial W}{\partial p}$
8:     Compute the Hessian, $H = \sum_x [\nabla I \frac{\partial W}{\partial p}]^T [\nabla I \frac{\partial W}{\partial p}]$
9:     Compute $\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x, p))]$
10:    Compute $\Delta p = H^{-1} \sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x, p))]$
11:    Perform the warp update $p \leftarrow p + \Delta p$
12: end while
Derivation of the Inverse Compositional image alignment algorithm

The first step in the derivation of the Inverse Compositional algorithm is to perform a first order Taylor expansion of the objective function in (2.18). This expansion generates the following simplified problem in (C.1)

\[
\min_{\Delta p} \sum_x [T(W(x, 0)) + \nabla T \frac{\partial W}{\partial p} \Delta p - I(W(x, p))]^2
\]  

(C.1)

Here the Jacobian \( \frac{\partial W}{\partial p} \) is evaluated at \((x, 0)\).

\( W(x, 0) \) is the identity warp and this gives that

\[
T(W(x, 0)) = T(x)
\]  

(C.2)

Since an optimum is to be found a necessary condition is that the first derivative of the objective function with respect to \( \Delta p \) must be equal to zero. Derivation of the Taylor expanded objective function in (C.1) gives that (C.3) must hold

\[
0 = \sum_x \left[ \nabla T \frac{\partial W}{\partial p} \right]^T \left[ T(x) + \nabla T \frac{\partial W}{\partial p} \Delta p - I(W(x, p)) \right]
\]  

(C.3)

Solving this with respect to \( \Delta p \) yields the solution in (C.5).

\[
\sum_x \left[ \nabla T \frac{\partial W}{\partial p} \right]^T \left[ \nabla T \frac{\partial W}{\partial p} \right] \Delta p = \sum_x \left[ \nabla T \frac{\partial W}{\partial p} \right]^T [I(W(x, p)) - T(x)]
\]  

\[
\Leftrightarrow
\]  

(C.4)
Derivation of the Inverse Compositional image alignment algorithm

\[
\Delta p = \left( \sum_x \left[ \nabla^T \frac{\partial W}{\partial p} \right]^T \left[ \nabla^T \frac{\partial W}{\partial p} \right] \right)^{-1} \sum_x \left[ \nabla^T \frac{\partial W}{\partial p} \right]^T \left[ I(W(x, p)) - T(x) \right] \tag{C.5}
\]

Let the Hessian \( H \) be defined as in (C.6). The solution can then be written as in (C.7).

\[
H = \sum_x \left[ \nabla^T \frac{\partial W}{\partial p} \right]^T \left[ \nabla^T \frac{\partial W}{\partial p} \right] \tag{C.6}
\]

\[
\Delta p = H^{-1} \sum_x \left[ \nabla^T \frac{\partial W}{\partial p} \right]^T \left[ I(W(x, p)) - T(x) \right] \tag{C.7}
\]
Bibliography


Upphovsrätt
Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.
Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

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

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

Copyright
The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

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

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

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

© Hanna Nyqvist