A markerless augmented reality system using one-shot structured light

Bingyao Huang

Follow this and additional works at: https://rdw.rowan.edu/etd

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you - share your thoughts on our feedback form.

Recommended Citation
Huang, Bingyao, "A markerless augmented reality system using one-shot structured light" (2015). Theses and Dissertations. 4.
https://rdw.rowan.edu/etd/4

This Thesis is brought to you for free and open access by Rowan Digital Works. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Rowan Digital Works. For more information, please contact LibraryTheses@rowan.edu.
A MARKERLESS AUGMENTED REALITY SYSTEM USING ONE-SHOT STRUCTURED LIGHT

by

Bingyao Huang

A Thesis

Submitted to the
Department of Electrical & Computer Engineering
College of Engineering
In partial fulfillment of the requirement
For the degree of
Master of Science in Electrical & Computer Engineering
at
Rowan University
June 30, 2015

Thesis Chair: Ying Tang
Acknowledgments

I would like to express my sincere gratitude to my advisor Dr. Ying Tang for the continuous support of my Masters study and research, for her patience, motivation, enthusiasm, and immense knowledge. She taught me how to think critically, question thoughts and express ideas. Her guidance helped me in all the time of research and writing of this thesis. Besides the care in study and research, her generous support helped me pass through the hard times when I was trying to adapt to a new life in this country. I could not have imagined having a better advisor and mentor for my Masters study. I hope that one day I would become as good an advisor to my students as Dr. Tang has been to me.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Haibin Ling and Dr. Ravi Ramachandran for their encouragement, insightful comments, and practical advice.

My sincere thanks also goes to Mr. Karl Dyer for the generous help on the 3D-printed camera mount and slider, without his help I would have not finished the system test and data collection.

I am also grateful to my friend Christopher Franzwa, his support and care helped me overcome setbacks when I first came this country. I greatly value his friendship.

I would also like to thank my family for the support they provided me through my entire, without your love, encouragement and editing assistance, I would not have finished this thesis.
Abstract

Bingyao Huang

A MARKERLESS AUGMENTED REALITY SYSTEM USING ONE-SHOT STRUCTURED LIGHT
2014-2015
Ying Tang, Ph.D.
Master of Science in Electrical & Computer Engineering

Augmented reality (AR) is a technology that superimposes computer-generated 3D and/or 2D information on the user’s view of a surrounding environment in real-time, enhancing the user’s perception of the real world. Regardless of the field for which the application is applied, or its primary purpose in the scene, many AR pipelines share what might be thinned down to two specific goals, the first being range-finding the environment (whether this be in knowing a depth, precise 3D coordinates, or a camera pose estimation), and the second being registration and tracking of the 3D environment, such that an environment moving with respect to the camera can be followed. Both range-finding and tracking can be done using a black and white fiducial marker (i.e., marker-based AR) or some known parameters about the scene (i.e., markerless AR) in order to triangulate corresponding points. To meet users’ needs and demand, range-finding, registration and tracking must follow certain standards in terms of speed, flexibility, robustness and portability. In the past few decades, AR has been well studied and developed to be robust and fast enough for real-time applications. However, most of them are limited to certain environment or require a complicated offline training. With the advancement of mobile technology, users expect AR to be more flexible and portable that can be applied in any uncertain environment. Based on these remarks, this study focuses on markerless AR in mobile applications and proposes an AR system using one-shot structured light (SL). The markerless AR system is validated
in terms of its real time performance and ease of use in unknown scenes.
# Table of Contents

Abstract ................................................................. iv

List of Figures ......................................................... vii

List of Tables ........................................................... viii

Chapter 1: Introduction ................................................. 1

  Research Background and Objectives ............................... 1
  Research Contribution ................................................. 2
  Organization .......................................................... 3

Chapter 2: Literature Review ........................................... 4

  Augmented Reality ...................................................... 4
    Display devices ..................................................... 5
    Camera pose estimation ........................................... 7
    Virtual contents registration and rendering ...................... 8
    Marker-based AR ..................................................... 8
    Markerless-based AR ................................................. 8

  3D Object Reconstruction ............................................ 11
    System calibration .................................................. 12
    Acquiring 2D images from different views ......................... 12
    Finding correspondence ........................................... 12
    Registration ........................................................ 12
    Surface construction and optimization .......................... 12
    Stereo Vision ........................................................ 13
    Active range sensing ............................................... 14
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Table of Contents (Continued)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Structured Light</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Markerless Tracking and Pose Estimation</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Feature detection</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>Feature description</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>Feature matching</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Pose estimation</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Chapter 3: Fast 3D Reconstruction using One-Shot Spatial</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Structured Light</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Overview of the Approach</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3D Reconstruction Process</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Color-coded structured light</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Image processing for grid extraction</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Color labeling and correction</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Neighbor completion using hamming distance</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>Chapter 4: Markerless Tracking and Camera Pose Estimation</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>System Calibration</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Feature Based Tracking</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Camera Tracking and Pose Estimation Using Perspective-n-Points</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Chapter 5: Experiment and Results</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>3D Reconstruction</td>
<td>54</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>HMD displays</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Marker-based AR</td>
<td>9</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Block Matching</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Range data scanning</td>
<td>16</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Corner definition by Harris [44]</td>
<td>22</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Feature matching</td>
<td>24</td>
</tr>
<tr>
<td>Figure 7</td>
<td>RANSAC used to filter matching outliers</td>
<td>31</td>
</tr>
<tr>
<td>Figure 8</td>
<td>System architecture</td>
<td>33</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Color Grid Pattern</td>
<td>36</td>
</tr>
<tr>
<td>Figure 10</td>
<td>The proposed markerless AR system</td>
<td>42</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Converting from object to camera coordinate systems</td>
<td>44</td>
</tr>
<tr>
<td>Figure 12</td>
<td>System setup</td>
<td>53</td>
</tr>
<tr>
<td>Figure 13</td>
<td>SL reconstruction process</td>
<td>54</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Reconstructed point cloud and mesh</td>
<td>55</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Camera tracking and pose estimation using PnP</td>
<td>57</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Accuracy of the proposed markerless pose estimation</td>
<td>59</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Frame rate (fps) of the proposed method presented</td>
<td>60</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Overall performance</td>
<td>61</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. Camera and projector intrinsic parameters</td>
<td>52</td>
</tr>
<tr>
<td>Table 2. Camera-projector pair rotation matrix</td>
<td>52</td>
</tr>
<tr>
<td>Table 3. Camera-projector pair translation vector</td>
<td>54</td>
</tr>
<tr>
<td>Table 4. Accuracy of the markerless pose estimation</td>
<td>58</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Research Background and Objectives

We live in a three-dimensional spatial world that is perceived by our vision system, where the objects we see is tangible and real. With the development of technology, digital information is brought to human, however, this kind of intangible virtual information is less intuitive for human to understand due to the limitation of human vision system and brain. Interfaces and tools have been invented and developed in an attempt to bring us closer to the exponentially increasing digital information, one of which is Augmented Reality (AR) that brings the real and the virtual together (i.e., computer generated graphics/video). AR has many applications in our life. Regardless of the field for which the application is applied, or its primary purpose in the scene, many AR pipelines share what might be thinned down to two specific goals, the first being range-finding the environment (whether this be in knowing a depth, precise 3D coordinates, or a camera pose estimation), and the second being registration and tracking of the 3D environment, such that an environment moving with respect to the camera can be followed. To meet users’ need, the design of range-finding, registration and tracking must consider speed, flexibility, robustness and portability. In the past few decades, AR has been well studied and developed to be robust and fast enough for real-time applications. But its flexibility and portability face a big challenge as most of the AR development is either limited to a certain environment or requires complicated offline training. As mobile devices become popular, users expect a more flexible and portable AR that can be applied in any uncertain environment. Based on these remarks, this thesis fo-
cuses on AR in mobile applications. In particular, we propose a 3D reconstruction (range-finding) method using structured light (SL). We then apply this method to our proposed AR system that allows fast one-time range-finding and accurate tracking in real-time, and easy-of-use in arbitrary unknown scenes without any offline training and/or add-on sensors needed.

**Research Contribution**

The goal of this research is to provide proof-of-the-concept of a markerless AR system that can be performed in real-time (30Hz) on mobile devices. In particular, the system attempts to accomplish the range-finding component without using a priori or comparative data set of the scene, but rather a known projection of structured light (SL). By doing this, range-finding can be accomplished for any unknown environment, requiring only one instance of data regarding the scene. In addition, the proposed method can be easily embedded on camera-equipped (with a flash light) mobile devices. Our specific contributions are as follows.

This work is primarily a proof-of-concept that robust markerless AR can be performed in real-time (30 Hz) on mobile devices. Our main contributions are as follows:

- We propose a flexible and portable AR system that can be used in any unknown scenes with no offline training.
- We propose an SL-based range finding method to reconstruct exact 3D information of the real world
- We demonstrate that a mobile device with a camera and flash light through a film is suffice to apply our real-time markerless AR system
Organization

This thesis is organized as follows. Chapter 2 describes the related work and techniques in AR with a focus on markerless (mobile) AR. The general 3D reconstruction process of our proposed AR system is presented in Chapter 3, followed by the detail on tracking, camera pose estimation and graphics rendering in Chapter 4. Chapter 5 gives the testing and evaluation of the proposed system in comparison with the ground truth. Finally, the conclusions and future research directions are presented in Chapter 6.
Chapter 2

Literature Review

Augmented Reality

Augmented reality (AR) superimposes computer-generated 3-D graphics or 2-D information on the user’s view of a surrounding environment in real-time, enhancing the user’s perception of the real world. AR is both interactive and registered in 3D as well as combines real and virtual objects [17]. In the definition of Milgram’s Reality-Virtuality Continuum [71] defined by Paul Milgram and Fumio Kishino there is a continuum that spans between the real environment and the virtual environment comprise Augmented Reality and Augmented Virtuality (AV) in between, where AR is closer to the real world and AV is closer to a pure virtual environment.

Another definition of AR is given by Azuma et al. [17] that AR is not restricted to a particular type of display technologies such as head-mounted display (HMD), nor limited to the sense of sight. AR can potentially apply to all senses, augmenting smell, touch and hearing, etc. With this context, our thesis only focuses on computer vision-based AR that estimates camera pose (localization, orientation) using visual tracking and view geometry constraints without any add-on sensors other than a camera. 3D virtual objects are then rendered from the same viewpoint that the images of the real scene are being taken by the tracking camera. The term of AR in the rest of this thesis is referred as computer vision-based Augmented Reality.

The goal of an AR application could be to complement existing information in the scene, such as overlaying augmented text on an historic building [54, 114] or giving more
educational information in a museum [6, 18, 19] or a book [23, 55]. AR also finds its applications in image guided surgery [33, 43, 58], reverse engineering [32], navigation [47], advertising and commercial [2, 48]. Another purpose could be to simply draw attention to the virtual data such as an entertainment game [11, 96]. Among all the applications, there are three essential key factors to account for a reliable and flexible AR system: display devices on which virtual contents are shown to users; tracking and camera pose estimation mechanism such that an environment moving with respect to the camera can be followed; and virtual content registration and rendering for proper display of the augmented information on devices.

Display devices. The display technologies can be grouped into three categories: head-mounted display (HMD), projective and handheld.

HMDs are wearable display devices on the head or as part of a helmet that places the real scene and images of virtual environment over a user’s view of the world (Fig. 1). HMD can either be video-see-through or optical see-through with a monocular or binocular
display. Video see-through HMDs provide users with a video view of both real world and virtual graphics. This technology requires at least one digital camera that captures the real scene and passes it through the video display to users. Given that the real and virtual contents are both processed as digital images, there are no synchronization issues. The optical see-through employs an optical mirror to allow views of physical world to be passed through the lens and graphically overlay information to be reflected in the user’s eyes, e.g., Google Glass. Since the optical see-through physically display real scenes with digitally-displayed virtual contents, synchronization of both displays is a technique challenge.

Projective displays project virtual information directly onto a real object with either video-projectors, for example, ARBlocks [86], optical elements, holograms and radio frequency tags, etc., or other tracking technologies. One of the potential advantages of projective displays is that they can be used without a wearable display device. But the down side is that the necessary set-up makes it inconvenient for outdoor uses. Another shortcoming with this type of displays is that cameras and projectors are difficult to synchronize under different lighting conditions.

Handheld displays are typically mobile devices such as smart phones, tablets and watches. The devices are usually attached with a camera or paired with other image acquisition and processing devices to stream videos to the display. Handheld displays are highly suitable for mobile AR applications due to their portability, mobility, ubiquity and low cost, owing to the development of the mobile computing technology recently [24]. They use video-see-through techniques to superimpose virtual contents onto the real environment and employ built-in sensors, such as digital compasses, accelerometer, gyroscope, barometer and/or GPS units to help and enhance their six degree of freedom (DoF) tracking. With the devel-
opment of digital camera, high definition screen such as retina display [111], and powerful CPU, GPU and RAM the smart phones and tablets are capable of real-time AR applications with high definition displays. However, their small display is the limitation for interactive 3D user experience. A promising solution is to combine HMDs with handheld devices, so the HMDs serve as image acquisition and displays while handheld devices serve for computing and processing server.

**Camera pose estimation.** Precise geometric registration of virtual contents convinces observers to accept the virtual content as part of the real environment rather than separated or overlaid illusion of information. In most AR applications, this registration requires a correct and real-time estimation of the user’s viewpoint with respect to the coordinate system of the virtual contents [39]. Namely, the camera pose with respect to the real scene must be known at each frame to be rendered. The camera pose can be estimated using an appropriate 6-Dof tracking system, where a 3D rigid transformation of the camera localization in the scene is calculated, including 3D rotation (3-Dof) and 3D translation (3-Dof) in the world coordinate system. This process usually involves 3D reconstruction (or range-finding) and visual tracking. Given that the camera pose with respect to the real scene changes over time, camera pose estimation must be performed fast enough to support real-time tracking.

In sensor based AR, optical(laser, IR), accelerometer, gyrometer, GPS, magnetic, ultrasonic sound techniques are applied/combined to enable a reliable tracking mechanism. However, in computer vision based AR, the relative localization between the camera and real scene in computer vision-based AR changes in real-time. Thus, a camera pose estimation algorithm must be performed fast to meet real-time tracking requirement.
**Virtual contents registration and rendering.** This process consists of generating and superimposing virtual contents using the estimated camera pose in real-time. The virtual contents may be modeled offline, for example, characters in an AR game, or generated online, such as, the wearable outdoor navigation application [70].

Out of the aforementioned three key factors, our thesis primarily focuses on tracking and camera pose estimation in unknown circumstances. Range-finding and tracking can be done using a fiducial marker or some natural features or parameters about the scene in order to triangulate corresponding points. The former method is referred to as a marker-based AR system while the latter is known as a markerless system.

**Marker-based AR.** The use of fiducial markers, although important to provide a robust tracking mechanism, presents a couple of drawbacks. The major restriction is that markers often cannot be attached to objects that are supposed to be augmented. While this problem might be solved using more than two cameras (at least one for markers tracking and the other for image taking), this significantly reduces the overall tracking and registration quality. In addition, such an approach is not suitable for most mobile devices like mobile phones. For instance, the use of markers may pose sometimes unwanted objects on the outputting video frames, which can reduce the visual aesthetic or get in the way of a desired view, particularly in mobile devices where resolution and/or field of view is limited. Finally, applying markers at all target locations to be augmented can be time-consuming like a large scale scene or even impossible, for example, in outdoor intangible scenarios. Thus, in a mobile scenario markerless tracking approaches using natural features of the environment to be augmented for tracking are preferable.
**Markerless-based AR.** Existing markerless AR applications extract pre-stored spatial information from objects followed by mapping invariant feature points to calculate the pose [26, 65, 99]. Strong feature detectors are usually used to identify target points and the random sample consensus (RANSAC) [38] algorithm is then employed to validate the matching of feature points. Once the features are matched, a perspective transformation is used to estimate camera’s translation and rotation poses with respect to the object from frame to frame. Markerless pose trackers mainly rely on natural feature points (sometimes called keypoints or interest points) visible in the user’s environment, so its real-time camera pose estimation becomes challenging.

A perusal of current literature provides a number of prominent markerless AR techniques, which can be categorized into three types: structure from motion (SfM), Parallel Tracking and Mapping (PTAM) and model-based systems.

SfM tries to work on a completely unknown scene and uses motion analysis and some known camera parameters to calculate the camera pose [4, 31]. Using MonoSLAM [31] as
an example, it applies SfM to do simultaneous localization and mapping (SLAM), where
the probabilistic 3D map is initialized using a planar target containing four known features
and then dynamically updated by using Extended Kalman Filter (EKF).

PTAM system [56] uses a stereo pair of images of a planar scene for initialization,
and bundle adjustment is then applied to determine and update the 3D locations of interest
points.

Model-based methods are considered robust against noise and efficient that make use
of models of the tracked objects, such as CAD models or 2D templates of items with
distinguishable features [120] (e.g., lines, circles, cylinders and spheres) [27, 119]. Once a
connection is made between the 2D image and 3D world frame, it is possible to calculate
the camera pose by projecting the features coordinates in 3D to 2D image coordinates.
An optimal pose estimation is found by minimizing the distance between the projected
2D coordinates and the corresponding observed 2D features coordinates. [27] describes a
real-time 3D model-based tracking algorithm for a ”video see through” monocular vision
system. This model-based method introduces an tracking algorithm that uses implicit 3D
information to predict hidden movement of the object thus improving the robustness and
performance. [115] proposes an illumination insensitive model-based 3D object tracking
system, where a Kalman filter is used to recursively refine the position and orientation. [85]
applies a hybrid (edge-based and texture-based) tracking system for out-door augmented
reality, where the system operates at about 15-17 frames on a handheld device. Recently,
[106] shows deformable surfaces like paper sheets, t-shirts and mugs can also serve as
potential augmentable surfaces. However, there is still space to improve its robustness and
minimize the computational costs.
These methods accomplish tracking in parallel with range-finding. They either require an initialization stage for the 3D model of the target object or require additional hardware for image processing to achieve real-time AR. For instance, [27] acquires the 3D model of the target object in priori, making it impossible to work in uncertain environment; [4] uses a server for online image processing and 3D point cloud reconstruction and Wi-Fi communication with mobile device. In short, all of the methods need at least some amount of the spatial environment information to be known either a-priori, or in a sequence of video frames, to accomplish the actual range-finding.

Several attempts have been made to improve the current markerless AR techniques for mobile applications. However the performance is still not sufficient enough. A modified version of SURF is proposed in [22], but far from real-time on mobile devices. Wagner et al propose a feature detection and matching system for mobile AR [109], where a modified version of SIFT and outliers rejection algorithms speed up the computation to achieve approximately 26 FPS. An AR application has been adapted to iPhone [57], where a reduced version of the PTAM is adopted. However, the method has shown severely reduced accuracy and low execution speed. [103] proposes a markerless augmented reality system for mobile devices with only a monocular camera. To enable a correct 3D-2D match, the EPNP [61] technique is used inside a RANSAC loop and inlier tracking uses optical flow.

3D Object Reconstruction

A prerequisite to estimate camera pose is the knowledge of the 3D coordinates of a reference object. This can be done through 3D reconstruction (rang-finding), a process of capturing the real shape and appearance of the object. 3D reconstruction has wide applica-
tions in medical and industrial fields, and has also been largely used to generate models for virtual and augmented reality. The process usually involves five major steps:

**System calibration.** A 3D reconstruction system consists of at least one digital camera. To properly register the point cloud, the transformation between image frame, camera frame and world frame must be known, which is expressed as camera intrinsic and extrinsic parameters. The process of obtaining those camera intrinsic and extrinsic parameters is called calibration.

**Acquiring 2D images from different views.** To completely reconstruct the surface of the entire object, often 8-10 images of the object from different views are needed.

**Finding correspondence.** A common method to get range data is to compare the 2D images and find the correspondences between each feature point through feature matching under camera poses constraints.

**Registration.** Given the perspective transformations from each view to object, the 3D coordinates are uniquely determined by its 2D correspondences. This is often completed by triangulation. Then the range data is transformed and combined together to form a single point cloud in the 3D world coordinate system. The whole process is called registration.

**Surface construction and optimization.** Once the point cloud is created, a connected mesh can be produced by connecting the 3D points to form edges. There are many well-established algorithms to do so, such as, marching cube algorithm [67], Poisson Surface Reconstruction [53] or the one in [46] that uses the neighborhood relationship of existing points.

In the rest of this section, three main 3D reconstruction methods are reviewed: stereo
vision, time of flight and structured light

**Stereo Vision.** Stereo vision system consists of two or more calibrated cameras capturing digital images of objects from different views. The captured images share some common object information, whose sparse correspondences or dense correspondences are used to match the prominent parts, namely corners, edges, or patch of the object, for triangulation. The former one is called feature-based matching, which first extracts a set of potentially matchable image locations using either interest operators or edge detectors, and then searches for corresponding locations in other images using patch-based metrics. More recent work in this area has focused on the use of extracted, highly reliable features as seeds to grow additional matches [63]. Similar approaches have also been extended to wide baseline multi-view stereo problems and combined with 3D surface reconstruction [42, 64]. While sparse matching is still occasionally used, most stereo vision today focuses on dense correspondence, since it is required for many applications such as image-based rendering or modeling. Such dense correspondence, often called block matching, consists of the following four steps: matching cost/score computation for each block pair, cost minimization (score optimization), disparity assignment according to the minimal cost match and disparity refinement. Usually the matching cost is the sum-of-squared-differences (SSD) or correlation be- tween two blocks from two images (Fig.3). To get a better result, global optimization algorithms are studied, such as probabilistic (mean-field) diffusion [92], expectation maximization (EM) [10], graph cuts [12], or loopy belief propagation [35, 37, 104, 110]. Those algorithms view a digital image as a graph model and make explicit smoothness assumptions that the image is smooth between two adjacent pixels. Finally, the matching is solved as a global optimization problem.
**Active range sensing.** Stereo vision uses passive video cameras, so it is also called passive 3D reconstruction. When applied to a scene in poor light condition or with textureless objects, for example, a plaster bust, this method usually has difficulty to find and match features, even with the epipolar constraint. Active 3D reconstruction, on the other hand, tackles this problem well. It uses light source to add an artificial pattern or texture on objects, thus greatly improving the precision of feature matching. This active illumination technique has been successfully applied to 3D laser scanning [34] and structured light scanning [104] with high reliability.

One of the active methods is called time-of-flight that measures the time a light signal flies between the light sensor and objects of interest. The theory behind the method is simple. When the illumination source is switched on for very short time and sends out a single light pulse or pulses, the resulting light pulse reaches the scene and is reflected by the objects in the field of view. The sensor then captures the reflected light. Depending upon the distance between illumination source and the object, there is a delay, $T$, for the camera to capture the reflected light, which is a function of the distance, $D$, and the speed
of light, $c$:

$$T = 2 \times D/c$$

Thus, the distance is only related to the delay and a constant $c$

$$D = Tc/2$$

where $T$ is the delay and $D$ is the distance from the illumination source to the object.

A good example of this system is LIDAR (LIght Detection And Ranging) [7], similar to RADAR(RAdio Detection And Ranging), a remote sensing method that uses a spatially narrow laser beam to measure ranges. To account for sensor noise, [28] describes an integrated method that combines a 3D super-resolution with a probabilistic scan alignment approach.

Velten et al. [107] proposes a time-of-flight 3D reconstruction method that use diffusely reflected light, the authors claim it achieves sub-millimeter depth precision and centimeter lateral precision over 40 cm $\times$ 40 cm $\times$ 40 cm of hidden space.

Another popular method uses a laser or light stripe sensor to sweep a plane of light across the scene or objects while observing it from an offset viewpoint [29]. A stripe is observed as an intersection between the light plane and object surface. As the stripe sweeps across the object, it deforms its shape according to the object’s surface shape. It is then a simple matter of using optical triangulation to estimate the 3D locations of all the points seen in a particular stripe. (Fig.4)

**Structured Light.** Structured light 3D reconstruction method is known for its robustness against noise and correspondence unambiguity. Instead of using a single video
camera, structured light allows an active illumination sensor (e.g. projector) to project encoded stripe patterns onto objects that create synthetic texture. Structured light follows the same routine as stereo systems if considering the active illumination sensor (e.g. projector) as an inverse camera. In terms of the correspondence problem in stereo vision based reconstruction, the synthetic texture helps match textureless surfaces, thus producing a better correspondence of features captured by two cameras.

Projecting known encoded patterns makes the correspondence between pixels unambiguous and is robust against occlusions [94]. This technique has been used to produce large numbers of highly accurate registered multi-image stereo pairs and depth maps for the purpose of evaluating stereo correspondence algorithms and learning depth map priors and parameters [98].

3D reconstruction using structured light has been well studied in the past few decades due to its wide applications in reverse engineering [82], augmented reality [100], medical
imaging [78] and archaeological finds [69]. In terms of codification strategy structured light can be classified into four general types: discrete spatial multiplexing, time-multiplexing, continuous frequency multiplexing and continuous spatial multiplexing [89]. When considering the projections, structured light can be sequential (multiple-shot) or single-shot. More detail can be referred to [41].

Most of the early work focuses on a temporal approach that requires multiple patterns to be projected consecutively onto stationary objects. Obviously, such requirement makes temporal approaches unsuitable for mobile and real-time applications.

Recently, researchers have devoted lots of efforts in speeding up the data acquisition process by designing techniques that need only a handful of input images or even a single one, so-called "one-shot" 3D image acquisition. For example, Zhang et al. [117] propose a multi-pass dynamic programming algorithm to solve the multiple hypothesis code mating problem and successfully apply to both one-shot and spacetime methods. Ali et al. [104] model a spatial structure light system using a probabilistic graphical formulation with epipolar, coplanarity and topological constraints. They then solve the correspondence problem by finding a maximum posteriori using loopy belief propagation. Kawasaki et al. [52] use a bicolor grid and local connectivity information to achieve dense shape reconstruction. Given that the proposed technique does not involve encoding positional information into multiple pixels or color space, it provides good results even when discontinuities and/or occlusions are present. Similarly, Chen et al. [21] present a specially-coded vision system where a principle of uniquely color-encoded pattern is proposed to ensure the reconstruction efficiency using local neighborhood information. To improve color detection, Fechteler and Eisert [36] propose a color classification method where the color classifi-
cation is made adaptive to the characteristics of the captured image, so distortion due to environment illumination, color cross-talk, and reflectance is well compensated.

In spite of the aforementioned development, there is still space to further improve one-shot methods, particularly in terms of speed. For example, the method reported in [104], takes 10 iterations to converge, which costs about 3 minutes to recover thousand intersections. Similarly, the approach in [36] takes a minute to reconstruct an object with 126544 triangles and 63644 vertices.

Markerless Tracking and Pose Estimation

Given tracking and pose estimation are vital to our project, this section provides an overview of the current development in those areas and explain in detail the key steps involved for creating an appropriate computer vision based markerless tracking system. Unlike most marker-based tracking methods, markerless tracking is designed for uncontrolled and unknown environment, where the tracking process works without adjusting the object or the environment to be tracked, such as placing fiducial landmarks or references. The natural features of the scene such as textures, blobs, corners, edges, lines and circles are used to uniquely identify the objects. In general this markerless tracking process involves four main mechanisms, feature detection, description, matching and pose estimation. The first three steps find strong features of the object and produce description/feature vectors of the target object. The target is then tracked over the frames by matching descriptors/feature vectors. Provided the tracking information and camera initial pose in a 3D coordinate system, the camera’s new pose with respect to the object is determined. To account for a robust pose estimation, several conditions must hold:
• Robustness: the tracking must be robust against noise and able to recover from failure

• Performance: the tracking must be fast to ensure AR performed in real-time.

• Set-up time: the initialization time must be as short as possible for a practical AR application.

To address the problems above, many markerless tracking and pose estimation methods are proposed, all of which can be categorized into two classes: recursive tracking [103] and tracking by detection [5]. Recursive techniques start the tracking process using an initial guess or a rough estimation, and then refine or update it over time. They are called recursive because the next estimation relies on the previous one, for example, feature matching is performed on two consecutive frames. In contrast, tracking by detection techniques can do frame by frame computation independently from previous estimation. In this case, a priori information about the environment or the objects to be tracked is needed, for example, a computer generated object 3D model. The feature matching is only between the current frame and the initial frame. The advantage of recursive tracking is smoothness and robustness against sudden changes, but it may result in an error accumulation over the entire tracking process. While tracking by detection is free from error accumulation [112], it is much more computationally expensive given that a classifier [3, 50] or object 3D model training [27] is needed.

**Feature detection.** To allow for an accurate pose determination, feature detection must meet several requirements [39]:

**Fast computation time.** It must be possible to calculate a rather large number of features points and associated descriptors in real-time in order to allow for a pose estimation at an acceptable frame rate.
Robust against lighting, viewing angles and noise. The set of feature points to be selected and their calculated descriptors shall not vary significantly under different lighting, noise and viewing angles. Some feature detection algorithms are highly sensitive to the change of lighting and image taking devices noise, since they use image intensity as features. In most AR applications, the users’ position and orientation are usually not fixed. Thus, any feature points that are restricted to only one distance or viewing angle will be useless for AR. All effects are quite common, particularly in outdoor environment.

Scale invariant feature choices. The scale of the captured object changes as the user moves around the scene especially in outdoor environments due to perspective projection. Thus, scale invariance must retain in the entire AR application, namely feature points visible from one distance will not disappear when getting closer or farther, allowing for a continuous tracking event.

Visual markerless pose trackers mainly rely on natural features in the user’s environment. Those features include blob-like image regions that has higher or lower intensities [30, 66, 72], edges [16], corners that are intersections of edges with stronger features [14, 44, 68, 73, 87, 95, 101], lines and circles [116]. Research has proved that corner detectors are more efficient than blob or edge detectors, which allow for a more precise position determination.

Majority of feature detection algorithms work by computing a corner response function $C$ across the image. Pixels which exceed a intensity-related threshold are then classified as corners [87].

Moravec [73] computes the sum-of-squared-differences (SSD) between a patch around a candidate corner and patches shifted from the candidate a small distance in a number of
directions. \( C \) is then the smallest SSD to be obtained, ensuring that extracted corners are those locations which change maximally under translations.

Harris [44] builds a Hessian matrix of an approximated second derivative of the SSD as:

\[
H = \begin{bmatrix}
I_{xx} & I_{xy} \\
I_{xy} & I_{yy}
\end{bmatrix}
\]  

(2.1)

Harris then defines the corner response to be

\[
C = |H| - k(\text{trace}(H))^2
\]  

(2.2)

where eigenvalues are \((\lambda_1, \lambda_2)\) and

\[
|H| = \lambda_1 \lambda_2
\]  

(2.3)

\[
\text{trace}(H) = \lambda_1 + \lambda_2
\]  

(2.4)

A window with a score \( C \) greater than a threshold is considered a "corner".

Following Harris’s framework, Brown and Szeliski [14] define \( C \) as harmonic mean of the eigenvalues \((\lambda_1, \lambda_2)\) of \( H \):

\[
C = \frac{|H|}{\text{trace}(H)} = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}
\]  

(2.5)

In a similar fashion, Shi and Tomasi [95] assume affine image deformation and conclude that it is better to use the smallest eigenvalue of \( H \) as the corner strength function:
$C = \min(\lambda_1, \lambda_2)$  \hfill (2.6)

Lowe [68] builds an image pyramid with different scales and resolutions. The scale invariance is obtained by convolving the image pyramid with a Difference of Gaussians (DoG) kernel, retaining locations which are optima in scale as well as in space.

**Feature description.** Once feature point are detected and extracted, how can computer tell which feature point in this image matches that in another image. Feature vectors or descriptors are designed to describe the attributes of a feature point by taking structural, neighborhood and region information into consideration. A feature descriptor must contains the necessary information to distinguish two different features. For a markerless AR system, feature descriptors should be scale and orientation invariant, not only allowing for fast calculation but also be able to describe the concerning interest point as unique as
possible in order to reduce ambiguity and mismatches.

Several different feature descriptors have been proposed in the last decade, such as SIFT [68], SURF [8] an improved version of SIFT, binary descriptors BRIEF [15], ORB [88], BRISK [62], and FREAK [1].

The feature vector of SURF is almost identical to that of SIFT. SIFT describes the features using the local gradient values and orientations of pixels around the keypoint. To create a keypoint descriptor, the gradient magnitude and orientation at each image sample point in a region around the keypoint location is firstly calculated. These samples are then counted into orientation histograms summarizing the contents in subregions of $4 \times 4$. Then for each subregion, the length of each arrow is assigned the sum of the gradient magnitude in/near that direction. Then a SIFT descriptor is a 128-element vector consisting of sum of the gradient magnitude. The high dimensionality makes it difficult to use SIFT descriptor in real time, while SURF improves on SIFT by using a box filter approximation to the Gaussian derivative kernel. This convolution is sped up further using integral images to reduce the time spent on this step [91]. As described in [8] an orientation is first assigned to the keypoint when constructing a SURF descriptor. Then a square region is constructed around the keypoint and rotated according to the orientation. Similar to SIFT, the region is split up regularly into smaller $4 \times 4$ square subregions. For each subregion, we compute a few simple features at $5 \times 5$ regularly spaced sample points. The horizontal and vertical Haar wavelet responses $dx$ and $dy$ are computed and summed up in each subregion and form a first set of entries to the feature vector. The absolute values of the responses $dx$ and $dy$ are also calculated, and together with the sum of vector to form a four-dimensional descriptor. And for all $4 \times 4$ subregions, it results in a vector of length 64.
BRISK is a 512 bit binary descriptor that computes the weighted Gaussian average over a selected pattern of points near the keypoint. FREAK (Fast Retina Keypoint) is an improved version of BRISK which is also a binary descriptor. This method is inspired by human retinal pattern in the eye, thus it samples the points using weighted Gaussians at a region around the keypoint. Comparisons among these feature descriptors in term of rotation, scaling, change of viewpoint, brightness shift, Gaussian blur are pretested in [1].

**Feature matching.** Once a set of robust features have been detected and described, they will be matched by comparing the distances between descriptors over the images (Fig.6).

Feature matching is one of the most critical tasks in the entire tracking pipeline. A large number of features may better describe the points however, the high number of potential feature correspondences will result in a large computational effort, slowing down the tracking process significantly. Additionally, wrong feature pairs (so called outliers) would result
in wrong pose estimation.

Feature matching by image patches (also called template/block matching) is the easiest way to find valid feature correspondences between images. The image patch matching first selects a patch or a constant windows size (e.g. 8×8 pixels) to extract features, for example, intensity and then compares this patch with all patches in other images. The goal is to find the best matching patch with the closest distances using the summed square distance function (SSD) or the sum of absolute distance function (SAD). Although image patches are very error-prone with respect to changing light conditions or viewing angles, they are not scale or orientation invariant. Thus, image patches can only be used for recursive tracking while error accumulation is the drawback if no error checking mechanism is used.

To produce a reliable tracking result, scale and orientation invariant features such as SIFT, SURF, FAST, BRISK and FREAK are usually used. Those features can be matched by finding the two descriptors with e.g. smallest summed square distance or smallest summed Hamming distance (SHD) if the descriptor is binary, for example BRISK and FREAK. Further, the SSD or SHD should lie below a specified threshold to ensure a reliable matching result. The threshold has to be defined carefully to avoid rejecting valid feature correspondences on one hand, and accepting too many wrong feature pairs on the other hand. Each type of feature detector needs its individual threshold to achieve an adequate ratio between accepted and rejected feature correspondences.

If no information can be used to reduce the number of potential feature candidates, e.g. from previous tracking iterations, an initial mapping can be extracted by a brute-force search. However, brute-force is computational intensive with $O(nm)$ for $n$ image features and $m$ map features thus it should be applied as rare as possible. Fast Library
for Approximate Nearest Neighbors (FLANN) usually performs classification faster than brute-force even with a large database of images. This method as detailed in [75] basically performs a parallel searching of randomized hierarchical trees to achieve an approximate matching of binary features. As discussed in [75], the hierarchical clustering tree gives significant speedup over linear search over a binary and float descriptors.

To speed up the searching process for the best matching descriptors, some research uses a kd-tree [9] or applies an feature culling mechanism [39]. For example the method can uniquely categorize SURF or SIFT features into two different kinds of blob-like features [39]: dark features with brighter environment or bright features with darker environment. Thus, the matching process can be speedup if two SURF features belong to different categories without calculating vector distances.

**Pose estimation.** Camera pose is about the camera’s localization $(X, Y, Z)$ and orientation $(\theta_x, \theta_y, \theta_z)$ in the world coordinate system, where $(\theta_x, \theta_y, \theta_z)$ are rotation around $X, Y, Z$ axis respectively. Given a set of objects’ 3D points $O^0 = (P_1, P_2, P_3...P_i)$ in the world coordinate system, a pose estimation means to find an object’s 3D pose $[R|t]$ from a set of 2D point projections $o^j = (p_1, p_2, p_3...p_i)$ at the $j^{th}$ frame. Note that the projection relationship means the correspondences $C = \{O^j \iff o^j\}$ that are matches of an object’s 3D points $O^j$ detected on the camera frame $o^j$. With a sufficient number of correspondences the camera pose can be determined uniquely by:

\[
\begin{align*}
\mathbf{o}^j &= M \mathbf{O}^j, \quad (\mathbf{O}^j = \mathbf{O}^0) \quad (2.7) \\
M &= A[R^j|t^j] \quad (2.8)
\end{align*}
\]
where $A$ is camera intrinsic matrix, $R^j$ and $t^j$ are rotation matrix and translation vector that brings the camera to its current pose at the $j^{th}$ frame. The definitions of $R$ and $t$ are then given in Eqs. (2.9), and (2.13).

$$R = R_Z R_Y R_X = \begin{bmatrix}
  r_{11} & r_{12} & r_{13} \\
  r_{21} & r_{22} & r_{23} \\
  r_{31} & r_{32} & r_{33}
\end{bmatrix}$$  \hspace{1cm} (2.9)

$$R_X = \begin{bmatrix}
  1 & 0 & 0 \\
  0 & \cos(\theta_x) & -\sin(\theta_x) \\
  0 & \sin(\theta_x) & \cos(\theta_x)
\end{bmatrix}$$  \hspace{1cm} (2.10)

$$R_Y = \begin{bmatrix}
  \cos(\theta_y) & 0 & \sin(\theta_y) \\
  0 & 1 & 0 \\
  -\sin(\theta_y) & 0 & \cos(\theta_y)
\end{bmatrix}$$  \hspace{1cm} (2.11)

$$R_Z = \begin{bmatrix}
  \cos(\theta_z) & -\sin(\theta_z) & 0 \\
  \sin(\theta_z) & \cos(\theta_z) & 0 \\
  0 & 0 & 1
\end{bmatrix}$$  \hspace{1cm} (2.12)

$$t = \begin{bmatrix}
  t_x \\
  t_y \\
  t_z
\end{bmatrix}$$  \hspace{1cm} (2.13)

The most general version of the pose estimation problem requires to estimate the 6-DoF of the extrinsic parameters $[R|t]$ and five intrinsic parameters in $A$: focal length, principal point, aspect ratio and skew. It could be established with a minimum of 6 correspondences,
using Direct Linear Transform (DLT) algorithm [97]. There are, though, several simplifications to the problem that turn into an extensive list of different algorithms to improve the accuracy of the DLT.

In practical AR system, several different markerless pose estimation algorithms are implemented [40, 61, 84, 97], a comparison among them is given by [77]:

**Direct Linear Transformation (DLT).** These algorithms intend to directly estimate the camera pose \([R|t]\) ignoring certain restrictions on the solution space.

**Perspective n-Point (PnP).** The most common simplification of DLT is Perspective-n-Point (PnP) problem [79, 98]: given a set of correspondences \(C = \{O^j \leftrightarrow o^j\}\) between 3D points and their 2D projections, we seek to retrieve the pose \((R \text{ and } t)\) of the camera w.r.t. the world and the focal length \(F\). At least \(n\) points are needed, usually camera intrinsics \(A\) is a prior knowledge.

**A priori information estimators.** Before pose estimation, some information regarding the camera pose is often available. The algorithms belonging to this class use the priori information as initials to provide more reliable and faster results.

Regardless of restrictions or assumptions, almost all algorithms share a similar routine of direct linear transform (DLT), which recovers a \(3 \times 4\) camera matrix \(M\), recall (2.7):

\[
M = \begin{bmatrix}
M_{11} & M_{12} & M_{13} & M_{14} \\
M_{21} & M_{22} & M_{23} & M_{24} \\
M_{31} & M_{32} & M_{33} & M_{34}
\end{bmatrix} = A[R|t] = \begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
r_{11} & r_{12} & r_{13} & t_1 \\
r_{21} & r_{22} & r_{23} & t_2 \\
r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\] (2.14)
From the perspective projection equation (4.2):

\[
\begin{align*}
  u_i &= \frac{M_{11}X_i + M_{12}Y_i + M_{13}Z_i + M_{14}}{M_{31}X_i + M_{32}Y_i + M_{33}Z_i + M_{34}} \quad (2.15) \\
  v_i &= \frac{M_{21}X_i + M_{22}Y_i + M_{23}Z_i + M_{14}}{M_{31}X_i + M_{32}Y_i + M_{33}Z_i + M_{34}} \quad (2.16)
\end{align*}
\]

where \( \mathbf{p}_i = (u_i, v_i) \) are the projected 2D points of the object’s points \( \mathbf{P}_i = (X_i, Y_i, Z_i) \).

This system of equations can be solved in a linear fashion for the unknowns in the camera matrix \( M \) by multiplying the denominator on both sides of the equation. The resulting algorithm is called the direct linear transform (DLT) and is commonly attributed to [97]. In order to compute the 12 (or 11) unknowns in \( M \), at least six correspondences between 3D and 2D locations must be known.

More accurate results for the entries in \( M \) can be obtained by directly minimizing the set of Equations (2.16), using non-linear least squares with a small number of iterations.

Once the entries in \( M \) have been recovered, it is possible to recover both the intrinsic calibration matrix \( A \) and the rigid transformation \((R, t)\) using RQ factorization.

In the case where the camera is already calibrated, i.e., the matrix \( A \) is known, we can perform pose estimation using at least 3 points, this method is known as P3P [40], but the results are noisy. The so-called P3P provides at most four different poses due to geometric ambiguities. Thus, necessary further feature correspondences are used to determine the unique pose. Several approaches have been proposed to solve the P4P, P5P or even PnP. The PnP problem can also be solved by the iterative method [102] or the non-iterative efficient algorithm EPnP [61] with the efficiency of \( O(n) \) (where \( n \geq 4 \)). The key implementation of the algorithm with such linear complexity is, to represent the \( \mathbf{P}_i \) as a weighted sum of
control points \((\omega_1...\omega_m)\) where \(m \leq 4\), and to perform all further computation only on these points. Although the more feature correspondences the more accurate the extracted pose is, a set of detected feature correspondences usually contain invalid feature pairs due to mismatching leading to inaccurate or even invalid pose calculations. Thus, invalid features correspondences must be exposed explicitly. The well-known RANSAC [38] algorithm can be used to determine the subset of valid correspondences from a large set of feature pairs with outliers. RANSAC randomly chooses few feature pairs to create a rough pose using the algorithm such as P3P, and tests this pose geometrically for all of the remaining pairs. Several iterations are then applied using different permutations of feature pairs, resulting in pose where most feature correspondences could be confirmed (Fig. 7). However, RANSAC does not optimize the final pose for all valid feature correspondences. As [77] explains, the quality of RANSAC reduces with the increasing ratio of outliers. When the outlier ratio reaches 60\%, it is not possible to estimate a robust camera pose within a reasonable time frame. Therefore, the pose should be refined to reduce the overall error of all applied feature correspondences. One refinement method is to optimize the pose with respect to the total projection error. Commonly, several iterations of nonlinear least squares algorithms e.g. the Levenberg-Marquardt (LM) algorithm [83] are sufficient to converge to the final pose. Further, different feature correspondences may be weighted e.g. according to their expected position accuracy to increase the pose quality. For example, an M-estimator [83] can be applied automatically neglecting the impact of outliers or inaccurate feature positions.
Figure 7. RANSAC used to filter matching outliers. The colored correspondences are computed using the SIFT feature detector providing many good matches along with a small number of outliers. Left: All colored correspondences are used, including the outliers. Right: RANSAC is used in order to find the set of inliers (green correspondences). Figure obtained from OpenCV documentation [13].
Chapter 3

Fast 3D Reconstruction Using One-Shot Spatial Structured Light

Introduction

This chapter presents a fast 3D reconstruction method based on one-shot spatially multiplexed structured light. In particular, a De Bruijn sequence [105] is used for encoding a color grid. To improve the resolution and field of view (FOV) under larger dynamic scenes, we extend our prior work [100] to use 8 colors instead of 6 [81]. After adding two more colors, the resolution is increased from 841 points to 4356 points. But this also brings up a problem for color detection. To compensate that, we propose a Hamming distance based method to improve the robustness. With only a single shot of a color grid, the depth information of a target object is extracted in real-time. Given our focus on mobile applications with dynamic scenes, some detail, such as sub-pixel accuracy, is no longer our concern. Instead, with the aim of achieving a balance of efficiency and accuracy, we strive to improve the speed of the proposed method while a decrease in accuracy can be tolerated by the system. In our experiment, the entire process from projecting a pattern to triangulating for mesh reconstruction takes less than 1 second.

The rest of the chapter is organized as follow: Section  overviews our proposed approach, followed by its detail in Section .

Overview of the Approach

The proposed structured-light method consists of a color camera and a projector, both of which are calibrated with respect to a world coordinate system and have their intrinsic and extrinsic parameters known. When the projector projects a color-coded light pattern onto
a 3D object, the camera captures its image immediately. The 3D information of the object is then obtained by analyzing the deformation of the observed light pattern with respect to the projected one and identifying their correspondence.

Matching pairs of light points and their images present several challenges. First, with the aim to project only one light pattern for real-time applications, there is a tradeoff between reliability and accuracy to select proper color stripes in terms of the number of colors, length of codeword, and dimensions. In our method, a two-dimensional 8-color De Bruijn spacial grid is used to offer a confident solution as explained in Section III.A. The more colors to use, the higher density and resolution. But the more colors, the better chance of color misdetection. To overcome this drawback, our method uses a two-level color label correction method. The grid pattern preserves spatial neighborhood information for all captured grid points. For instance, if two points are connected vertically/horizontally on the captured image, their correspondences must follow certain topological constraints (e.g., the same vertical/horizontal color). Those constraints are then used to correct color detection errors as detailed in Section III.B. When dealing with more complex scenes that involve shadows, occlusion, and discontinuities, observed geometric constraints can be misleading [104]. Thus, a De Bruijn-based Hamming distance minimization method is established for further correction.
3D Reconstruction Process

As illustrated in Fig.8, the reconstruction process starts with projecting, capturing, and image processing in order to retrieve the grid overlain on the object. The recovered stripe substrate forms a 2D intersection network to solve the correspondence problem. Without loss of generality, the structured light codification is first presented in Section 2, followed by a brief explanation of the image processing to obtain a 2D colored intersection network in Section 3. The detailed methods to remedy color detection errors and incomplete neighborhood information due to occlusion are then described in Section 4 and 5.

**Color-coded structured light.** Our projected grid pattern is composed of vertical and horizontal colored slits with a De Bruijn sequence. A De Bruijin sequence of order $n$ over an alphabet of $k$ color symbols is a cyclic sequence of length $k^n$ with a so-called window property that each subsequence of length $n$ appears exactly once [105]. When applying such a sequence to both vertical and horizontal stripes, it forms a $m \times m$ grid, where $m = k^n + 2$ and a unique $k$-color horizontal sequence overlaid atop a unique $k$-color vertical sequence only occurs once in the grid. Let $C$ be a set of color primitives, $C = \{cl|cl = 1, 2, 3, ..., 8\}$, where the numbers represent different colors. Thus, each crosspoint in the grid matrix, $v_i$, is an intersection of a vertical stripe and a horizontal stripe with its unique coordinate $(x_i, y_i) \in \Omega$, where $\Omega$ is the camera’s image plane and colors $(c^h_i, c^v_i)$, where $c^h_i, c^v_i \in C$, representing the horizontal and vertical colors of $v_i$, respectively. Therefore, the color-coded grid network can be interpreted as a undirected graph

$$G = (V, E, P, \xi, \vartheta, \chi, \tau), \quad \text{where}$$
\( V = \{ v_1, v_2, \ldots, v_{m \times m} \} \) represents grid intersections.

- \( e_{ij} \in E \) represents a link between two nodes \( i \) and \( j \) in the grid.

- \( P \) is a vector of 2D coordinates of \( v_j \) in the camera’s image plane, where \( p_j = (x_j, y_j); \ (x_j, y_j) \in \Omega. \ell(P) \) denotes the length of \( P \).

- \( \xi: V \to C \) is the color labels associated with a node where \( \xi(v_i) = (c^h_i, c^v_i); \xi^h(v_i) = c^h_i, \xi^v(v_i) = c^v_i \).

- \( \vartheta: V \to \{\vartheta(v_1), \vartheta(v_2), \ldots, \vartheta(v_{m \times m})\} \), where
  \[ \vartheta(v_i) = \bigcup_j N_{bj}(v_i); \ j \in \{north, south, west, east\}; N_{bj}(v_i) \in V \] represents the neighbors of \( v_i \). When the \( j^{th} \) neighbor of \( v_i \) does not exist, \( N_{bj}(v_i) \) is assigned a value of NULL.

- \( \chi: \vartheta \to \{0, 1\} \) is a decision function with a binary output.
  \[ \forall j \in \{north, south, west, east\} \]
  \[ \exists N_{bj}(v_i) \in \vartheta(v_i) = NULL \Rightarrow \chi(\vartheta(v_i)) = 0, \]
  \[ \text{else} \quad \chi(\vartheta(v_i)) = 1. \]

- \( \tau: V \to P \) is a function for getting the 2D coordinates of \( v_i \), where \( \tau(v_j) = p_j \)

In our implementation, \( C = 1, 2, \ldots, 8 \), representing Red, Yellow, Lime, Green, Cyan, Blue, Purple and Magenta. The colors used for the horizontal stripes are Red, Lime, Cyan and Purple, while Yellow, Green, Blue and Magenta are used for vertical stripes. They are evenly distributed in HSV space for better color detection. Fig. 9 shows the resulting grid. The total of 8 colors are significant in improving the density and resolution in comparison
to other earlier works [113], [20].

**Image processing for grid extraction.** There are several image processing methods used in our implementation to retrieve the stripe substrate material overlain on the real world scene. First, an adaptive thresholding with a Gaussian kernel is used to correctly segment the projected and deformed grid. The idea is to check if the RGB value of every pixel is greater than a threshold which is the Gaussian average of a local intensity. Given that eight colors are used for vertical and horizontal stripes in our implementation, such a threshold is not fixed but rather adapts to the local intensity of a $9 \times 9$ neighborhood of the pixel. The Hilditch thinning algorithm [45] is then applied until the skeleton of the grid is obtained. Based on the skeleton of the grid, intersections can then be located. The idea is to count the number of total rises, (i.e. changes in the pixel value from 0 to 1) and the total neighbors with a value of one or high within a $3 \times 3$ window centered at a pixel position of interest. The position is considered as an intersection of the grid as long as the total rises and the number of neighbors with a value of one are not less than 3 or the total rises is greater than 1 and the number of neighbors with a value of one is great than 4.
Due to the complexity of the scene (e.g., shadows, distortion, depth discontinuity, and occlusion, etc.), there exist spurious connections and holes, especially at the border of the background and the object. To this end, the watershed algorithm [108] is first used for the segmentation of the target object from the background, resulting in two grid networks. The traversal algorithm used in our prior work [100] is then applied to the networks individually with the purpose of determining the northern \((Nb_{north}(v_i))\), southern \((Nb_{south}(v_i))\), eastern \((Nb_{east}(v_i))\) and western \((Nb_{west}(v_i))\) neighbors of each intersection \(v_i\). The algorithm moves a \(3 \times 1\) (for the north-south traversal) or \(1 \times 3\) (for the west-east traversal) window along the direction it traverses to the most recently chosen path pixel, until either an intersection is found in this manner, or no intersection is found in an allotted maximum distance from the original observed intersection. An intersection \(v_i\) with four neighbors determined is considered a perfect intersection which will be used for triangulation. Due to the aforementioned challenges, not all the intersections have complete neighborhood information. To avoid of losing those intersections when decoding the codeword for triangulation, further process is needed as elaborated in Section .

**Color labeling and correction.** Each intersection in our color-coded grid is represented by its unique neighborhood information, particularly the colors of its neighbors. To determine the color of each intersection, the captured color grid is blurred with a \(3 \times 3\) Gaussian filter with a mask of skeleton grid obtained in Section III.B. The purified color grid is then converted from RGB to HSV for color labeling of each pixel, where 8-pair thresholds corresponding to the Hue ranges of the 8 colors are applied. To compensate for the color crosstalk, two morphological openings (i.e., \(5 \times 1\) and \(1 \times 5\) structuring elements are used to separate horizontal and vertical stripes.
To determine the color labels of each intersection (i.e., $c^h_i, c^v_i$), a special vote majority rule is applied to a $9 \times 9$ neighborhood window of the intersection, where a $360^\circ$ angular search is used to populate the histograms of 8 color labels. Each histogram of a particular color label has 18 bins, each of which corresponds to a non-overlapping $20^\circ$ range. As stated in Section III.A, the color labels 1, 3, 5, and 7 are used for the horizontal stripes, while 2, 4, 6, 8 for the vertical stripes. Let $\rho_k$ counts the number of observations that fall into each of the $\kappa$ bins in a histogram and $\rho^c_{\kappa}$ is a type of $\rho_{\kappa}$ in the histogram of the color label $cl$. The color labels for the intersection can then be determined as follows:

$$c^h_i = g(\max_{cl=1,3,5,7}^{\kappa} (\rho^c_{\kappa}))$$ \hspace{1cm} (3.1)

$$c^v_i = g(\max_{cl=2,4,6,8}^{\kappa} (\rho^c_{\kappa}))$$ \hspace{1cm} (3.2)

where $g(x) : x \rightarrow cl, cl \in C$ maps $x$ to its corresponding color label.

Although the special vote majority rule substantially minimizes the impact of ill color detection on color labeling for an intersection, there are still errors. In view of those errors, a topological constraint is then used for further correction. If intersections are linked horizontally in the camera image, their correspondences must be collinear. There is more than that. The intersections should have the same color labels as they are on the same horizontally stripe. The same idea should apply to the intersections on the same vertically stripe. Thus when plotting the histogram of the color labels of the intersections on the same row or column, further correction is performed, where the row/column histogram has 4 bins,
each corresponding to one of the four colors used for horizontal/vertical stripes.

\[(c^h_i) = g(\max(\rho^h_\kappa)), \kappa = 1, 2, 3, 4 \] (3.3)

\[(c^v_i) = g(\max(\rho^v_\kappa)), \kappa = 1, 2, 3, 4 \] (3.4)

where \(\rho^h_\kappa\) and \(\rho^v_\kappa\) are a type of \(\rho_\kappa\) in the histogram of color labels for horizontal and vertical stripes, respectively.

**Neighbor completion using hamming distance.** Each intersection \(v_i\) together with its four neighbors forms two unique De Bruijn subsequences, one for the horizontal string (i.e., \((\xi^h(Nb_{west}(v_i)), c^h_i, \xi^h(Nb_{east}(v_i)))\)) and the other for the vertical string (i.e., \((\xi^v(Nb_{north}(v_i)), c^v_i, \xi^v(Nb_{south}(v_i)))\)). One difficult in finding the correct correspondences is that not all intersections have complete neighborhood information even after the traversal elaborated in Section III.C. The codewords resulted from those so-called “imperfect” intersections are literally broken, making it impossible to map them to the projected De Bruijn color grid. To this end, our method proposes a Hamming Distance-based amendment procedure. The procedure first defines a function \(f(v_i)\) that returns color labels of itself and its neighbors’ for the intersection \(v_i\),

\[f(v_i) = [(r^v_w, r^h_w); (c^v_i, c^h_i); (r^v_n, r^h_n); (r^v_e, r^h_e); (r^v_s, r^h_s)]\] (3.5)

where \((r^v_w, r^h_w); (r^v_n, r^h_n); (r^v_e, r^h_e); (r^v_s, r^h_s)\) represent western, northern, eastern, and southern neighbors’ horizontal and vertical color labels of \(v_i\). If \(v_i\) has complete neighborhood information, the value of \(r\) can be retrieved from \(\xi()\) in the grid graph; otherwise, the neighbors
are virtual with the colors to be determined through the procedure. The algorithm then
traverses the retrieved intersections using a $3 \times 4$ or $4 \times 3$ moving window, depending on
the task of either labeling vertical or horizontal color of the missing neighbors. For each
intersection with missing neighbors, the procedure assigns the color labels of its missing
neighbors to 0. For instance, if the northern neighbor of $v_i$ is missing, $r^v_{w} = r^h_{w} = 0$. Doing
so makes the originally broken codewords complete while invalid. Such an amended but
faulty word should have the minimal Hamming distance to its corresponding correct one.
According to this idea, the procedure then creates virtual neighbors of each ”imperfect”
intersection with correct color labels. Compared to real intersections, virtual intersections
only have color labels without coordinates. The detailed amendment procedure is given in
Algorithm 1 shown in Appendix.A. For clarity, the algorithm only includes the procedures
of creating virtual neighbors and assigning their vertical color labels. The same procedures
except for the moving window size should be applied to determine their horizontal colors.

In Algorithm 1 (Appendix.A), the inputs are

- **$G$**: original retrieved intersection grid.
- **$h^v = \{s_0, s_1, \ldots, s_{m \times m-1}\}$**: a matrix set where each element $s_j$ is a $3 \times 4$ matrix
  with each row element forming a vertical De Bruijn subsequence $(a_{0j}, a_{1j}, a_{2j}, a_{3j})$
of length 4,
  \[
  s_j = \begin{bmatrix}
  a_{0j} & a_{1j} & a_{2j} & a_{3j} \\
  a_{0j} & a_{1j} & a_{2j} & a_{3j} \\
  a_{0j} & a_{1j} & a_{2j} & a_{3j}
  \end{bmatrix}.
  \]
- **$H$**: the function for calculating per-element Hamming distance between two matri-
The outputs are:

- $r_s^v$, $r_w^v$, $r_e^v$ and $r_n^v$.

- $G'$: amended intersection grid with virtual neighbors.

The two matrices $\pi^v$ and $\pi^h$ are of size $3 \times 4$ and $4 \times 3$ for horizontal and vertical separately.
Chapter 4

Markerless Tracking and Camera Pose Estimation

Introduction

The proposed markerless AR system consists of two main stages: range finding using SL (Chapter 3) and markerless tracking by matching SURF descriptors and camera pose estimation using PnP. The system architecture is shown in Fig. 10. The initialization, image processing and reconstruction(range finding) are described in detail in Fig. 8. In this chapter, the tracking and pose estimation process (the loop in Fig. 10) are discussed.

![Diagram of the proposed markerless AR system](image)

*Figure 10. The proposed markerless AR system*

System Calibration

A binocular 3D reconstruction system consists of an optical camera and a projector, where the projector is used for pattern projection and the camera for image acquisition. The precision and accuracy of reconstruction and tracking is highly dependent on the intrinsic and extrinsic parameters of the camera and projector. Thus a proper system calibration is a prerequisite for correct computation of any scaled or metric results from their images.
The intrinsic camera parameters consist of horizontal and vertical focal lengths \( f_x, f_y \), the principal point \((c_x, c_y)\) of the camera and the skew. In our system the skew is assumed to be zero. \( f_x \) and \( f_y \) can be calculated from camera/projector focal length \( F \), Field Of View(FOV) and the principal point is the image center. In order to compensate distortion of the camera, the radial and tangential distortion coefficients up to the second order need to be computed. In order to calibrate these, a sufficient amount of 2D-3D correspondences has to be created, which is typically done using a known target such as a chessboard. For the calibration of a low resolution camera, we use the calibration method proposed by Zhang [118] through OpenCV [13] library. In this method, the size of the chessboard and a single cell is defined by user. Several shots of the chessboard are then captured and its inner corners are detected. The reprojection error between the detected corners and the chessboard corners is reduced by iteratively updating the intrinsic and extrinsic parameters of the camera.

Extrinsic parameters are translation vector \( t \) and rotation matrix \( R \) that bring the camera to the projector in the 3D world coordinate system.

For each image the camera takes on a particular object, we can describe the pose of the object relative to the camera coordinate system in terms of a rotation and a translation; see Fig. 11.

A pinhole camera model describes a perspective transformation from 3D to 2D, where a scene view is formed by projecting 3D points into the image plane using Eq. 4.1.

\[
sp' = A[R|t]P' \tag{4.1}
\]
Figure 11. Converting from object to camera coordinate systems: the point \( P \) on the object is seen as point \( p \) on the image plane; the point \( p \) is related to point \( P \) by applying a rotation matrix \( R \) and a translation vector \( t \) to \( P \) [49]

or

\[
\begin{bmatrix}
    u \\
    v \\
    1
\end{bmatrix} =
\begin{bmatrix}
    f_x & 0 & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    r_{11} & r_{12} & r_{13} & t_1 \\
    r_{21} & r_{22} & r_{23} & t_2 \\
    r_{31} & r_{32} & r_{33} & t_3
\end{bmatrix}
\begin{bmatrix}
    X \\
    Y \\
    Z \\
    1
\end{bmatrix}
\]

(4.2)

where:

- \((X, Y, Z)\) are the coordinates of a 3D point \( P \) in the world coordinate space
- \((u, v)\) are the coordinates of the projection point \( p \) in pixels
- \(A\) is a camera matrix, or a matrix of intrinsic parameters
- \((c_x, c_y)\) is a principal point that is usually at the image center
• $f_x, f_y$ are the focal lengths expressed in pixel units.

The intrinsic parameters are independent on the scene viewed. Once calibrated, it can be re-used as long as the focal length is fixed. The joint rotation-translation matrix $[R|t]$ is called a matrix of extrinsic parameters. It describes the camera motion around a static scene, or vice versa, rigid motion of an object in front of a still camera. The rigid transformation above is can be written as (when $z \neq 0$):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$

$$y' = y/z$$

$$u = f_x * x' + c_x$$

$$v = f_y * y' + c_y$$

Real lenses usually have some distortion, mostly radial distortion and slight tangential distortion. So, the above model is extended as:
\[
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} = R \begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} + t
\]

\[x' = x/z\]
\[y' = y/z\]

\[x'' = x' + k_2 x'^3 + k_3 x'^5 + k_6 r^2 + 2p_1 x'y' + p_2 (r^2 + 2x'^2)\]
\[y'' = y' + k_2 y'^3 + k_3 y'^5 + k_6 r^2 + p_1 (r^2 + 2y'^2) + 2p_2 x'y'\]

where \(r^2 = x'^2 + y'^2\)

\[u = f_x * x'' + c_x\]
\[v = f_y * y'' + c_y\]

Where \(k_1, k_2, k_3, k_4, k_5,\) and \(k_6\) are radial distortion coefficients. \(p_1\) and \(p_2\) are tangential distortion coefficients. The distortion coefficients do not depend on the scene viewed. Thus, they also belong to the intrinsic camera parameters. For the distortion we take into account the radial and tangential factors [13]. For the radial factor we use the following formula:

\[x_{corrected} = x (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)\] (4.3)
\[y_{corrected} = y (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)\] (4.4)

So for an pixel point at \((x, y)\) coordinates in the original image, its position on the corrected image should be \((x_{corrected}, y_{corrected})\). Tangential distortion occurs when the camera lenses are not perfectly parallel to the imaging plane. It is corrected using the
formulas below:

\[ x_{\text{corrected}} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \] \hspace{1cm} (4.5)

\[ y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \] \hspace{1cm} (4.6)

So we have five distortion parameters presented as one row matrix with 5 columns:

\[ \text{Distortion coefficients} = (k_1 \ k_2 \ p_1 \ p_2 \ k_3) \] \hspace{1cm} (4.7)

The projector’s intrinsic and camera-projector pair extrinsic parameters can be calibrated using the aforementioned method by treating the projector as a reverse camera, we use the software provided by [74] for system calibration.

**Feature Based Tracking**

As reviewed in Section , in a practical AR application, markerless tracking must meet the following requirement:

- **Robustness:** the tracking must be robust against noise and be able to recover from failure.

- **Performance:** the tracking must be fast enough to ensure AR perform in real-time.

- **Set-up time:** the initialization time must be as short as possible for a practical AR application.

In our markerless system, SURF detector and descriptor are used due to its robustness against weak light condition, invariant to scale/orientation and fast computation. All of
these features ensure a real-time tracking.

SURF detector is recognized as a more efficient substitution for SIFT. It has a Hessian-based detector and a distribution-based descriptor generator.

The SURF detector is based on the Hessian matrix, defined as

\[
\mathcal{H}(x, \sigma) = \begin{bmatrix}
L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\
L_{xy}(x, \sigma) & L_{yy}(x, \sigma)
\end{bmatrix}
\] (4.8)

\(L\) is the convolution of the Gaussian second order derivative with the image \(I\) in point \(x\). In the implementation, they use a simpler box filter instead of Gaussian filter in [68]. An image pyramid is built and convolved with the box filter to obtain scale-invariance properties.

To describe a SURF keypoint, an orientation is first assigned to the keypoint. For each direction a square region is constructed around the keypoint SURF descriptor is extracted. The region is split up regularly into smaller \(4 \times 4\) square subregions. The Haar wavelet responses in horizontal \(dx\) and vertical \(dy\) are calculated and summed up over each subregion and form a first set of entries to the feature vector. The absolute values of the responses \(|dx|\) and \(|dy|\) are also calculated, and together with the sum of vector to form a four-dimensional descriptor. And for all \(4 \times 4\) subregions, it results in a vector of the length of 64.

In our system, the camera captures an image of the scene at the initialization stage and computes the SURF descriptors of each single detected keypoint. Note that the 3D coordinates of these keypoints are already calculated in the previous 3D reconstruction stage as described in Chapter.3. The 3D coordinates of the keypoints are assigned to the 3D coordinates of their nearest color grid intersections. In the end, these keypoints and
their descriptors, 2D locations are stored as a reference.

As the camera/object moves around the scene, SURF descriptors are extracted from each incoming frame. Once SURF generates its float point descriptor, a Brute Force k-nearest neighbors (k-NN) based matching algorithm is applied to find the 2 nearest matches between the descriptors. The distance metric is $L^2$ norm. To get rid of the mismatched keypoints pairs and guarantee strong correspondences, a threshold of distance ratio is applied, only when the ratio between the best match distance and the second to the best match is below a specific threshold e.g. 0.7, the best match is determined as a ”correct” one.

Then, the detected 2D/3D feature correspondences are taken to determine the exact camera 6-DoF pose by applying several Perspective-n-Point (PnP) iterations, followed by a pose refinement step with a non-linear least-square algorithm. If enough feature correspondences have been detected and validated by the RANSAC iterations, the number of used features $n$ will be decreased, therefore speeding the pose estimation process and the 6-DoF camera pose is computed using the $n$ features.

**Camera Tracking and Pose Estimation Using Perspective-n-Points**

In our case, to allow for an accurate pose estimation the camera intrinsics are first calibrated using the method in Section . With the known camera intrinsics, iterative PnP [83] that is based on Levenberg-Marquardt optimization is used for pose estimation. This function finds such a pose that minimizes reprojection error, which is the SSD between the observed 2D projections on camera frames and the projected object points using formula (4.1). Although iterative PnP is slower than P3P [40] or EPnP [61], it returns the most robust estimation. The pose estimation procedure is as follows:
1. A set of object’s points are selected and their 3D world coordinates are reconstructed using the SL range-finding method described in Chapter 3: \((P_1, P_2, P_3, ... P_i)\), \((i \geq 6)\). Their initial 2D coordinates in the camera image plane are \((p^0_1, p^0_2, p^0_3, ... p^0_i)\) (Fig.15a).

2. The SURF descriptor \((d^0_1, d^0_2, d^0_3, ... d^0_i)\) of the initial 2D keypoints \((p^0_1, p^0_2, p^0_3, ... p^0_i)\) is extracted in the initial frame (frame 0).

3. As the camera/object moves in the scene, a set of keypoints \((p^j_1, p^j_2, p^j_3, ... p^j_k)\) in the current incoming frame (\(j^{th}\) frame) are detected and described by SURF, so do their descriptors \((d^j_1, d^j_2, d^j_3, ... d^j_k)\).

4. A Brute Force 2-NN (k-NN \(k = 2\)) based matching algorithm (Algorithm 1) is applied to find the correspondences between the initial 2D keypoints’ descriptors \((d^0_1, d^0_2, d^0_3, ... d^0_i)\) and SURF keypoints’ descriptors \((d^j_1, d^j_2, d^j_3, ... d^j_k)\) in current frame (\(j^{th}\) frame) using \(L_2\) norm. In our practical application, a stable and fast matching result is obtained using the parameters \(\tau = 3\), and \(i, k > 6\) in Algorithm 1.

5. An iterative PnP algorithm is used to solve the rotation matrix \(R\) and translation vector \(t\) using the correspondences between 3D coordinates \((P_1, P_2, P_3, ... P_i)\), \((i \geq 6)\) and new 2D coordinates \((p^j_1, p^j_2, p^j_3, ... p^j_k)\).

6. Repeat steps 3-5 to update the camera pose as camera move around the scene (Figs. 15).
Algorithm 1 Brute Force k-NN based matching algorithm

1: Create a matching score matrix $\Delta$
2: for $m \leftarrow 1$ to $k$ do
3:   for $n \leftarrow 1$ to $i$ do
4:      $\Delta[m,n] = \delta_{mn} = L_2(d_m^0, d_n^1)$
5:   end for
6: end for
7: for $m \leftarrow 1$ to $k$ do
8:   $q_{mn} = \min(\delta_{m1}, \delta_{m2}, \delta_{m3}, \ldots, \delta_{mi})$
   $q_{ml}' = \min((\delta_{m1}, \delta_{m2}, \delta_{m3}, \ldots, \delta_{mi}) \setminus q_{mn})$
9:   if $\frac{q_{mn}}{q_{ml}'} > \tau$ then
10:      A strong match pair of discriminators $(d_m^0, d_n^1)$ is defined by \{m, n\}
11: end if
12: end for
13: The created matching score matrix $\Delta$:
14: $\Delta = \begin{bmatrix} 
\delta_{11} & \delta_{12} & \delta_{13} & \ldots & \delta_{1i} \\
\delta_{21} & \delta_{22} & \delta_{23} & \ldots & \delta_{2i} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\delta_{k1} & \delta_{k2} & \delta_{k3} & \ldots & \delta_{ki} 
\end{bmatrix}$

51
Chapter 5

Experiment and Results

Our implementation setup is composed of (a) a desktop with an Intel quad-core (3.40 GHz) i7-2600k CPU, 8G DDR3 1600 memory, and a 64-bit Windows 7 operating system; (b) a Point Grey Flea3 color camera running at a resolution of $640 \times 480$; and (c) an Optima 66HD DLP projector set to the resolution of $1280 \times 800$ as shown in Fig. 12. When the projector projects a color-coded light pattern onto a 3D object, the camera located at 1-meter away from the object captures the image. Both camera and projector are calibrated with their intrinsic and extrinsic parameters in Tables. 1, 2 and 3. To validate the accuracy and speed of our proposed method, a plaster bust with a complex surface is used in our experiments. The proposed markerless AR system then performs at 30 frames per second (fps), which is considered real-time in our context.

Table 1

*Camera and projector intrinsic parameters*

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>$f_x$</th>
<th>$f_y$</th>
<th>$c_x$</th>
<th>$c_y$</th>
<th>$k_1$</th>
<th>$k_2$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$k_3$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera</td>
<td>1796.10</td>
<td>1792.37</td>
<td>259.97</td>
<td>288.02</td>
<td>-0.23</td>
<td>-5.46</td>
<td>0.00</td>
<td>0.00</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td>Projector</td>
<td>1404.45</td>
<td>1348.78</td>
<td>477.97</td>
<td>276.15</td>
<td>-0.48</td>
<td>2.57</td>
<td>-0.03</td>
<td>-0.01</td>
<td>0.00</td>
<td>0.09</td>
</tr>
</tbody>
</table>

Table 2

*Camera-projector pair rotation matrix*

<table>
<thead>
<tr>
<th>Rotation matrix (R)</th>
<th>$r_{11}$</th>
<th>$r_{12}$</th>
<th>$r_{13}$</th>
<th>$r_{21}$</th>
<th>$r_{22}$</th>
<th>$r_{23}$</th>
<th>$r_{31}$</th>
<th>$r_{32}$</th>
<th>$r_{33}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cam-proj pair</td>
<td>0.95</td>
<td>0.06</td>
<td>-0.32</td>
<td>0.03</td>
<td>0.99</td>
<td>0.11</td>
<td>0.33</td>
<td>-0.09</td>
<td>0.94</td>
</tr>
</tbody>
</table>
Figure 12. System setup: a projector on the bottom-left, camera on the right and a plaster bust in the scene.
Table 3

Camera-projector pair translation vector

<table>
<thead>
<tr>
<th>Translation vector (t)</th>
<th>$t_x$</th>
<th>$t_y$</th>
<th>$t_z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cam-proj pair</td>
<td>377.29</td>
<td>-291.63</td>
<td>243.83</td>
</tr>
</tbody>
</table>

3D Reconstruction

![Figure 13](image1)

*Figure 13.* SL reconstruction process: (a) A plaster bust in our experiment. (b) Color grid pattern projected onto the plaster bust. (c) Extracted color grid from (b)

Figs. 13 gives the object (Figs. 13a) and the captured intersection network with $62 \times 60$ color stripes (Figs. 14b). Due to the highly deformed surface, especially at the hair, eyes, and nose areas, there are several holes as depicted in Fig. 13c. Most of them are due to occlusion distortion. Note that with the help of the watershed algorithm, the impact of sharp depth discontinuity between the background and object is minimal. The reconstructed point cloud and mesh are shown in Fig. 14. In the experiment, the total number of intersections in the projected De Bruijn grid ($66 \times 66$) is 4356. However, four horizontal and six vertical stripes are out of the camera’s FOV, resulting in the size of the captured grid $62 \times 60$ with 3720 intersections. Due to occlusion distortion, there are 393 intersections missing,
especially in the areas of hair, eyes and boarders of the bust and background. With this
ground truth, our method is able to recover 3327 intersections with 267 labeled wrong and
110 containing incomplete neighborhood information. This corresponds to 88.6% correct
assignment.

![Reconstructed point cloud and mesh: (a) and (b)](image)

*Figure 14. Reconstructed point cloud and mesh: (a) and (b)*

With the focus on local optimization, such as the vote majority rule for color detection
and correction, and De Bruijn-based Hamming distance to improve neighborhood infor-
mation of intersections, our method achieves the best speed in comparison to the existing
one-short methods [104] (the running time for 1000 grid intersections was about 3 min-
utes), [21] (the running time for 1110 grid intersections was about 313 ms). In unoptimized
C++ code, the running time for 3327 grid intersections was about 1 second.

**Markerless Tracking and Pose Estimation**

In our AR system the world origin is at the camera optical center. As shown in Fig. 11 the
direction pointing from the camera optical center to the camera face is defined as positive
Z. Similarly, positive Y is defined as the direction from the camera optical center to the
camera bottom and positive X the direction from the camera optical center to the camera
right. The clockwise rotation around X, Y, Z axes (i.e., rz) is termed as pitch, yaw, and roll, respectively. In a similar fashion, the translations along X, Y, and Z axes are defined as tx, ty, tz respectively.

A tripod with a camera ball mount is used to enable camera rotation and translation. For the ground truth, an iPhone 6 is mounted on the top of the camera. Thus, while the iPhone rotates with the camera, its built-in sensor (i.e., InvenSense MP67B) can precisely measure the rotation. On the other hand, a tape ruler is used to physically measure the translation. Eventually, by moving the tripod around the scene, the performance of the developed system is tested in terms of camera rotation and translation. In our system setup (Chapter 5), we test the propose markerless AR system in a scene of 100 inches × 60 inches × 60 inches, while the bust (or a part of the bust) stays in the camera’s FOV.

In the first set of experiments, the rotation and translation are computed and measured between the camera initial pose and the current frame. As exemplified in Fig. 15, the first set of SURF keypoints (red dots) are extracted in the initial frame (Fig. 15a), where an initial pose is estimated using the 3D-2D correspondences and visualized by the X, Y, and Z axes. The X axis (red) points to the right, Y axis (green) down, and Z axis (blue) the inside of the image. While the camera moves in the scene, a new set of SURF keypoints are extracted for new pose estimation as shown in Figs. 15b, 15c and 15d. Note that the number of keypoints to be extracted varies each frame. Even when occlusion occurs as shown in Fig. 15d, our AR system is still robust enough to correctly compute the pose.

In the second set of experiments, the camera’s motion data (i.e., rotation and translation) is obtained and compared with the ground truth at 27 different displacements of the camera. Due to sensor noise, we acquire 10 groups of the motion data at each displace-
Figure 15. Camera tracking and pose estimation using PnP. The camera moves around the scene and its poses are estimated in real-time. The rotation and translation along X(red), Y(green), Z(blue) axis are rx, ry, rz and tx, ty, tz. (a) SURF keypoints (red dots) are extracted at initial pose. (b) Camera moves close to bust. (c) Camera moves away from bust. (d) A correct estimation with partial occlusion.

ment, including the pose estimation from our AR system (i.e., Cam(rx), Cam(ry), Cam(rz), Cam(tx), Cam(ty), Cam(tz)) and the physical measurement (True(rx), True(ry), True(rz), True(tx), True(ty), True(tz)). The mean of the 10 data points are plotted and their corresponding errors in comparison to the ground truth are presented in Fig. 16. Figs. 16a and 16c are for the rotation (in degrees), and Figs. 16b and 16d the translation (in inches). As can be seen in Figs. 16a and 16b, the plots are well superimposed. With the tracked object being 60 inches from the camera, the average absolute error of our system is 1.98 inches in
translation and 1.07 degrees in rotation (with a standard deviation of 1.91 inches and 2.42 degrees). More detail can be found in Table 4. It can be noted that, the maximum error is 8.89 inches in translation and 7.25 degrees in rotation due to the following reasons:

- A rough calibration of the camera and projector pair causes 3D reconstruction and tracking error.

- The physical measurement using tape ruler and iPhone 6 gyroscope has a low precision of 0.5 inch in translation and 1 degree in rotation.

- To allow for a real-time AR application, we set the camera shutter to be 15 ms and gain to be 20 db, which results in noise increase and sometimes a flash effect that lightness and contrast change suddenly on the incoming frame.

Table 4

Accuracy of the markerless pose estimation: Mean Error (inches and degrees), Standard Deviation, and Maximum Error

<table>
<thead>
<tr>
<th></th>
<th>error(rx)</th>
<th>error(ry)</th>
<th>error(rz)</th>
<th>error(tx)</th>
<th>error(ty)</th>
<th>error(tz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>-0.03</td>
<td>-1.22</td>
<td>1.95</td>
<td>3.69</td>
<td>-0.90</td>
<td>1.35</td>
</tr>
<tr>
<td>std</td>
<td>2.88</td>
<td>2.45</td>
<td>1.95</td>
<td>2.17</td>
<td>1.61</td>
<td>1.95</td>
</tr>
<tr>
<td>max</td>
<td>6.98</td>
<td>1.97</td>
<td>7.25</td>
<td>8.89</td>
<td>1.17</td>
<td>8.65</td>
</tr>
</tbody>
</table>

Processing Speed Analysis

One of the development goals is to achieve real-time AR performance that supports users’ interactivity. When moving and rotating the camera along X, Y and Z axises respectively our AR system consistently performs at an average of 30 fps as shown in Fig. 17. Note that
Figure 16. Accuracy of the proposed markerless pose estimation: (a) and (c) camera translation measured using the tape ruler and our AR system. (b) and (d) corresponding errors (inches, degrees).

The measure of FPS is done when moving the camera in the scene described in Section 5.2. The FPS is calculated by 1/processing time, where the process time that includes tracking and pose estimation for a single frame.

To understand how each process plays in the overall system speed for future improvement, we look into the time complexity of each process carefully. First, we measure the time spent on the three processes (i.e., initialization, image processing, and reconstruction) that only occur once and only once in the AR pipeline. The initialization process includes all the stages that create and assign global variables, such as De Bruijn color grid pattern, camera-projector calibration, and the first frame. All the necessary procedures
that find intersections correspondences belong to image processing, for example, adaptive thresholding to extract color grid, Gaussian blur to remove image noise, Hue thresholding, morphological operations, and skeletonization, etc. The reconstruction involves intersection neighbors traversal, intersections color labeling, Hamming distance-based color label correction, intersection 2D coordinates computation by decoding De Bruijn sequence, and intersections triangulation. As can be seen in Fig. 18, the initialization costs 3641 ms, image processing 693 ms and reconstruction 502 ms. The skeletonization (366 ms), morphological operations (201 ms) and thresholding (126 ms) are relatively expensive processes.

Second, we evaluate the tracking and pose estimation in a similar fashion. As shown in Fig. 18b, the total running time for this process is 34 ms (29.4 fps). Although iterative PnP and brute-force matching algorithms are used, the proportion of matching (5 ms) and pose estimation (6 ms) in the entire iterative tracking and pose estimation process are smaller than SURF feature detection and description (25 ms). According to Fig. 17, the three procedures are executed at the average of 30 Hz. For each tracked frame, there are
approximate 10 SURF feature points to be tracked. As also shown in Fig. 17, there are some drops in the FPS curve due to the increase of detected SURF feature points. For instance, at the frame 193, the system detects 15 feature points. The extra time spent on feature description, matching and iterative PnP pose estimation of those points results in a reduced fps of 10.5.
Chapter 6

Discussion and Conclusion

This thesis proposes an markerless augmented reality system that uses one-shot spatially multiplexed structured light. In our AR system, the 3D points are obtained by SL reconstruction, 2D points are obtained by SURF and pose is simultaneously estimated using 3D-2D correspondences. The application of SL successfully overcomes the deficiency of existing marker and markerless AR methods that either utilize fiducial markers, prior information about the real world scene, or environment-environment comparison, thereby making the proposed methodology more suitable for AR in an unknown and uncertain environment. The fast one-shot real-time 3D reconstruction (range-finding) method uses De Bruijn encoded color pattern. Unlike previous one-short approaches, our formulation focuses on local optimization, where a Hamming distance minimization method is used for finding missing topological information and a special vote majority rule is applied for color detection and correction. The proposed range-finding method is approved to be efficient with a desirable accuracy achieved, even when applied to a very complicated scene.

Note that with the focus on mobile applications, such as mobile interactive augmented reality [100], we emphasize more on speed over accuracy. Thus we only use about 10 in our application to sufficient estimate for camera pose as a larger number of 3D points may significantly slow down the process of tracking and pose estimation. The proposed feature points based tracking and pose estimation method is tested in an indoor application, the average absolute error is 1.98 inches in translation and 1.07 degrees in rotation, which is considered reliable in our scenario. The entire system was tested on a desktop and was
shown to operate at speeds of approximately 30 Hz with tracking and pose estimation in each frame.

**Discussion and Future work**

The evolution of the proposed algorithm is in its infancy and has many areas of potential improvement, particularly in the following aspects:

- This work is only a proof-of-concept that our AR system can be implemented on mobile devices. One of our next step is to transplant the code to mobile devices for verification.

- Our AR system is based on SL and visual tracking, so color and feature points detection is sensitive to the light condition. Therefore, an adaptive color and feature detection algorithm with automatic outliers rejection mechanism is worth of future investigation to enhance the robustness of 3D reconstruction and tracking.

- Another future step that is work-in-progress is to propose a camera-projector pair extrinsic parameters calibration using SL. This method uses correspondences between projected color grid pattern and camera frame to estimate an essential matrix using RANSAC. The rotation matrix and translation vector are then obtained by decomposing the essential matrix. This calibration method simplifies the process described in Section , since no extra software is needed.

- To improve the accuracy of the reconstruction method, the study of global optimization algorithms, such Markov Random Field (Section ) is another future direction that find an optimal solution for the correspondence problem in Section , where topological and epipolar constraints will be considered as score functions.
• The performance of the proposed pose estimation method is highly dependent on feature detection, description and matching algorithm as illustrated in Fig. 18b. Although in our implementation, the SURF detector, descriptor and brute-force 2-NN matching algorithm are sufficient to achieve a 30 Hz AR, this performance will decrease when applied to mobile devices, due to hardware limitation. The future work is to find a balance between computational cost and reliability of the tracking method. We will test other faster detectors, descriptors and matching algorithms, such as FAST detector, BRISK, FREAK descriptor and Hamming distance based matcher (if use binary descriptor) or FLANN’s hierarchical-clustering approach [75]. Another routine is to consider feature matching as a classification problem. Instead of creating feature descriptors and matching algorithms, feature correspondences can also be determined by classifiers [80] specifying the characteristics of feature points [39,60].

• As hardware of mobile devices evolves, acceleration methods are becoming popular to mobile applications. For example, modern mobile devices are equipped with multi-core CPU and GPU. The acceleration methods on desktop can also be applied to mobile devices, such as parallelization through CPU and GPU acceleration.

• The tracking and pose estimation method in our AR system requires the object of interest to stay in the camera’s FOV though the entire process. However, this is not always true, particularly for outdoor applications where occlusions and view angle loss happens. Feature map that is used to store detected features, such as the one in PTAM [56] and MonoSLAM [31] can be created and combined with optical flow algorithms [59] to address this issue. As the system works around the scene, the feature map dynamically updates itself. Even when occlusions occur, the lost feature
can be easily retrieved from a different angle in the map.

• Finally, we plan to compare our AR system with other markerless AR applications, such as SfM-based [76, 90], SLAM-based [31], model-based [4, 27, 65] and hybrid [59, 103] techniques, from speed, flexibility, robustness and portability perspectives.
References

Proceedings of the IEEE Computer Society Conference on Computer Vision and 


Multiple Instance Learning. IEEE Transactions on Pattern Analysis and Machine 

augmented reality system for context-aware architectural, engineering, construction 
and facility management (AEC/FM) applications. Visualization in Engineering, 
1:3, 2013.

for augmented reality applications. Journal of Real-Time Image Processing, 


Computer vision and image . . ., 2008.


I. Zaragoza, and M. Alcaniz. Treating cockroach phobia using a serious game on a 
mobile phone and augmented reality exposure: A single case study. Computers in 

graph cuts. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 


Appendix A

Hamming-Distance-Based Amendment

1: \textbf{for} \ i = 0 \rightarrow \ell(P) - 1 \ \textbf{do}

2: \quad \textbf{for} \ d = 0 \rightarrow 2 \ \textbf{do}

3: \quad \textit{Initialize}

4: \quad \textbf{for} \ q = 0 \rightarrow 3 \ \textbf{do}

5: \quad \pi_{dq} = 0

6: \quad \textbf{end for}

7: \quad \textbf{end for}

8: \quad \lambda = i

9: \quad \textit{Assign value to each element of} \ \Pi^v

10: \quad \textbf{while} \ \chi(\vartheta(v_\lambda)) = 0 \ \text{and} \ \text{Nb}_{east}(v_\lambda) \neq \text{NULL} \ \textbf{do}

11: \quad \pi_{11}^v = \xi^v(v_\lambda)

12: \quad \textbf{if} \ \text{Nb}_{west}(v_\lambda) \neq -1 \ \textbf{then}

13: \quad \pi_{10}^v = \xi^v(\text{Nb}_{west}(v_\lambda))

14: \quad \textbf{end if}

15: \quad \textbf{if} \ \text{Nb}_{north}(v_\lambda) \neq \text{NULL} \ \textbf{then}

16: \quad \pi_{01}^v = \xi^v(\text{Nb}_{north}(v_\lambda))

17: \quad \textbf{end if}

18: \quad \textbf{if} \ \text{Nb}_{east}(v_\lambda) \neq \text{NULL} \ \textbf{then}

19: \quad \pi_{12}^v = \xi^v(\text{Nb}_{east}(v_\lambda))

20: \quad \textbf{end if}
21: if $\text{Nb}_{\text{south}}(v_\lambda) \neq \text{NULL}$ then
22: \[
\pi_{21}^v = \xi^v(\text{Nb}_{\text{south}}(v_\lambda))
\]
23: end if
24: if $\text{Nb}_{\text{north}}(\text{Nb}_{\text{west}}(v_\lambda)) \neq \text{NULL}$ then
25: \[
\pi_{00}^v = \xi^v(\text{Nb}_{\text{north}}(\text{Nb}_{\text{west}}(v_\lambda)))
\]
26: else if $\text{Nb}_{\text{west}}(\text{Nb}_{\text{north}}(v_\lambda)) \neq \text{NULL}$ then
27: \[
\pi_{00}^v = \xi^v(\text{Nb}_{\text{west}}(\text{Nb}_{\text{north}}(v_\lambda)))
\]
28: end if
29: if $\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(v_\lambda)) \neq \text{NULL}$ then
30: \[
\pi_{02}^v = \xi^v(\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(v_\lambda)))
\]
31: else if $\text{Nb}_{\text{east}}(\text{Nb}_{\text{north}}(v_\lambda)) \neq \text{NULL}$ then
32: \[
\pi_{02}^v = \xi^v(\text{Nb}_{\text{east}}(\text{Nb}_{\text{north}}(v_\lambda)))
\]
33: end if
34: if $\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(v_\lambda)) \neq \text{NULL}$ then
35: \[
\pi_{22}^v = \xi^v(\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(v_\lambda)))
\]
36: else if $\text{Nb}_{\text{east}}(\text{Nb}_{\text{south}}(v_\lambda)) \neq \text{NULL}$ then
37: \[
\pi_{22}^v = \xi^v(\text{Nb}_{\text{east}}(\text{Nb}_{\text{south}}(v_\lambda)))
\]
38: end if
39: if $\text{Nb}_{\text{south}}(\text{Nb}_{\text{west}}(v_\lambda)) \neq \text{NULL}$ then
40: \[
\pi_{20}^v = \xi^v(\text{Nb}_{\text{south}}(\text{Nb}_{\text{west}}(v_\lambda)))
\]
41: else if $\text{Nb}_{\text{west}}(\text{Nb}_{\text{south}}(v_\lambda)) \neq \text{NULL}$ then
42: \[
\pi_{20}^v = \xi^v(\text{Nb}_{\text{west}}(\text{Nb}_{\text{south}}(v_\lambda)))
\]
43: end if
if $\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda)) \neq \text{NULL}$ then

\[
\pi_{13}^v = \xi^v(\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda)))
\]

end if

if $\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda))) \neq \text{NULL}$ then

\[
\pi_{03}^v = \xi^v(\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda))))
\]

else if $\text{Nb}_{\text{east}}(\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(v_\lambda))) \neq \text{NULL}$ then

\[
\pi_{03}^v = \xi^v(\text{Nb}_{\text{east}}(\text{Nb}_{\text{north}}(\text{Nb}_{\text{east}}(v_\lambda))))
\]

end if

if $\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda))) \neq \text{NULL}$ then

\[
\pi_{23}^v = \xi^v(\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(\text{Nb}_{\text{east}}(v_\lambda))))
\]

else if $\text{Nb}_{\text{east}}(\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(v_\lambda))) \neq \text{NULL}$ then

\[
\pi_{23}^v = \xi^v(\text{Nb}_{\text{east}}(\text{Nb}_{\text{south}}(\text{Nb}_{\text{east}}(v_\lambda))))
\]

end if

\[
\Pi^v = \begin{bmatrix}
\pi_{00} & \pi_{01} & \pi_{02} & \pi_{03} \\
\pi_{10} & \pi_{11} & \pi_{12} & \pi_{13} \\
\pi_{20} & \pi_{21} & \pi_{22} & \pi_{23}
\end{bmatrix}
\]

\[
s_j^* = \arg \min_{s_j} (H(\Pi^v, s_j))
\]

$v_\lambda = \text{Nb}_{\text{east}}(v_\lambda)$

Create virtual neighbors with color labels if needed

if $\text{Nb}_{\text{south}}(v_\lambda) = \text{NULL}$ then

\[
r_{n}^v = a_{1j}^*
\]

end if

if $\text{Nb}_{\text{south}}(v_\lambda) = \text{NULL}$ then
65: \[ r_s^v = a_{i,j}^s \]

66: \textbf{end if}

67: \textbf{if} \( Nb_{south}(v_\lambda) = \text{NULL} \) \textbf{then}

68: \[ r_w^v = a_{0,j}^s \]

69: \textbf{end if}

70: \textbf{if} \( Nb_{south}(v_\lambda) = \text{NULL} \) \textbf{then}

71: \[ r_e^v = a_{2,j}^s \]

72: \textbf{end if}

73: \textbf{end while}

74: \( v_\lambda = Nb_{east}(v_\lambda) \)

75: \textbf{end for}