

The University of Birmingham

**A USER-ORIENTED MARKERLESS
AUGMENTED REALITY FRAMEWORK
BASED ON 3D RECONSTRUCTION AND
LOOP CLOSURE DETECTION**

by

YUQING GAO

A thesis submitted to the University of Birmingham for the degree of
DOCTOR OF PHILOSOPHY (Ph.D.)

School of Engineering
Department of Electronic, Electrical & Systems Engineering
University of Birmingham
November 2016

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

ABSTRACT

Visual augmented reality aims to blend computer-generated graphics with real-world scenes to enhance the end users' perception and interaction with a given environment. For those AR applications which require registering the augmentations in the exact locations within the real scene, the system must track the user's camera motion and estimate the spatial relationship between the user and the environment. Vision-based tracking is one of the more effective tracking methods, and uses relatively low-cost and easy-access cameras as input devices and, furthermore, exploits several computer vision (CV) techniques to solve the problem. It can typically be divided into marker-based and markerless methods. The marker-based AR applications have, in the past, proved sufficiently robust and accurate, and the marker-based tracking methods have been widely supported by almost every AR software development kit developed and marketed to date. However, they always require the introduction of artificial markers into the workplace, which may be undesirable in some cases (e.g. outdoors), due to deterioration over time as a result of exposure to weather effects, or due to requirements not to tamper with objects and sites of historic or religious significance. In contrast, markerless tracking methods attempt to make use of the natural features extracted from the original environment as their reference. Several CV-based methods, such as Structure from Motion (SfM) or visual SLAM, have already been applied for the process of unsupervised markerless template training, and many research projects have applied markerless tracking methods for solving AR issues within their chosen application area. However, a general development framework supporting the higher-level application designers or developers, supporting the customisation of AR applications for different environments and different purposes, is rare.

The present research proposes a conceptual markerless AR framework system, the process for which is divided into two stages – an offline database training session for the designers, and an online AR tracking and display session for the final users. In the offline session, two types of 3D reconstruction application, RGBD-SLAM and SfM are integrated into the development framework for building the reference template of a target environment. The performance and applicable conditions of these two methods are presented in the present thesis, and the application developers can choose which method to apply for their developmental demands. A general developmental user interface is provided to the developer for interaction, including a simple GUI tool for augmentation configuration. The present proposal also applies a Bag of Words strategy to enable a rapid “loop-closure detection” in the online session, for efficiently querying the application user-view from the trained database to locate the user pose. The rendering and display process of augmentation is currently implemented within an OpenGL window, which is one result of the research that is worthy of future detailed investigation and development.

ACKNOWLEDGEMENTS

I would like to express my sincere, respectful gratitude to my supervisor, Professor Robert J. Stone, for his guidance, supervision, constant encouragement and great help throughout these years. Thank you for supporting me with your unlimited patience, positive attitude and the warmest heart, which helped me to overcome the hardest times during my PhD study.

I would like to express my appreciation to Dr. Mike Spann, for his precious suggestions and academic supports. I would also extend my gratitude to Dr. Jingjing Xiao, for helpful discussion and collaboration. Her endlessly intellectual enthusiasm impressed me deeply. My special thanks to Dr. Cheng Qian for his tremendous assistance throughout my PhD study and for being such a sincere friend. I would like to thank Professor Peter Gardner, Mary Winkles for their kind help and support for my PhD programme. Many thanks to Vishant Shingari, Chris Bibb, Mohammadhossein Moghimi, Laura Nice, and all the colleagues in the Human Interface Technologies (HIT) team for their help and being great friends. I would like to express my sincere gratitude to Deborah Cracknell and Mark Du'chesne from The National Marine Aquarium, for sharing the valuable data and information with me. I also want to thank James Robert, whose writing has inspired me so much.

The greatest thanks go to my beloved family. I cannot express my gratitude to my mother and father, who always give me unconditional support and love. I could not have achieved anything without your support. Finally, I would also like to thank all of my friends who supported thus far. Thank you!

TABLE OF CONTENTS

ABSTRACT	1
TABLE OF CONTENTS	3
LIST OF ILLUSTRATIONS	7
LIST OF TABLES	12
LIST OF ABBREVIATIONS	14
Chapter 1 Introduction	18
1.1. Scientific and Novel Contributions.....	27
1.2. Thesis outline	29
Chapter 2 Literature Review and Background.....	32
2.1. Augmented Reality	33
2.1.1. Visual AR technologies	35
2.1.2. AR development frameworks.....	52
2.1.3. AR system evaluation.....	54
2.2. Computer vision methods in AR	57
2.2.1. Camera and camera calibration	60
2.2.2. Visual features	65
2.2.3. CV-based localisation and mapping.....	82
2.2.4. Image retrieval for loop closing	97
2.3. Hardware, software supports and datasets for evaluation.....	101

2.3.1.	Hardware	101
2.3.2.	Software.....	110
2.3.3.	CV datasets for evaluation.....	112
2.4.	Problems and Challenges.....	121
Chapter 3	Preparatory Studies.....	124
3.1.	AR application development and requirement audience survey	124
3.1.1.	Questionnaire analysis	125
3.1.2.	Conclusion	139
3.2.	Geometric transformations	140
3.2.1.	Reference frames and coordinate systems.....	141
3.2.2.	3D-to-3D rigid transformations	145
3.2.3.	3D-to-2D camera projections	148
Chapter 4	3D Reconstruction for Template Training.....	151
4.1.	RGBD-SLAM.....	154
4.1.1.	Graph-based SLAM.....	158
4.1.2.	System implementation	161
4.1.3.	Output data	173
4.2.	Structure from motion.....	174
4.2.1.	Incremental SfM	175
4.2.2.	VisualSfM.....	185
4.3.	Reconstruction evaluations	189

4.3.1.	A small study on visual features.....	192
4.3.2.	The evaluation of 3D sparse reconstruction based on camera pose estimation	201
4.4.	Conclusion	219
Chapter 5	Vision-based Template Training and User Tracking	221
5.1.	Bag of words and vocabulary	226
5.1.1.	Visual word clustering.....	227
5.1.2.	BoW signatures	232
5.2.	Image retrieval and loop closing.....	234
5.2.1.	Chow-Liu tree.....	235
5.2.2.	Location representation and likelihood	238
5.3.	Implementation	241
5.4.	Results and evaluations.....	245
5.5.	Conclusion	257
Chapter 6	Conceptual User Interface and Use Case Scenario	258
6.1.	Conceptual user interface.....	258
6.1.1.	Development framework	259
6.1.2.	AR registration and display	264
6.2.	Use case scenario	268
6.2.1.	Background and requirements	269
6.2.2.	Usability evaluation plans	272

6.3. Conclusion	273
Chapter 7 Conclusions and Future Work	275
7.1. Conclusions.....	275
7.2. Future research and development	279
PUBLICATION	281
REFERENCES	282
APPENDICES	293
A. AR application development & requirement audience survey	293
B. Software and development supports	302
C. Conversions between rotation matrices and quaternions.....	309
D. Distance metrics.....	311
E. Skew-symmetric matrix representation of cross product.....	312
F. RGBD-SLAM output files.....	313
G. VisualSfM file formats	315

LIST OF ILLUSTRATIONS

Figure 1-1: The screenshot of Virtual Wembury.....	18
Figure 1-2: Virtual Wembury Event.....	19
Figure 1-3: Simplified representation of an RV Continuum.	19
Figure 1-4: Real hands in virtual world.....	20
Figure 1-5: Virtual rabbit on user’s hand with <i>Handy AR</i>	20
Figure 1-6: The system diagram of the proposed user-oriented markerless AR framework.....	26
Figure 1-7: The flow chart of the content of each chapter.	29
Figure 2-1: Contemporary smartphone hardware sensors.....	38
Figure 2-2: An AR campus navigation app on iPad developed using the Wikitude SDK.	40
Figure 2-3: An AR shipwreck (Maria) app on iPad developed using the Metaio SDK.	41
Figure 2-4: A sample program of ARToolkit with a black and white square fiducial marker.....	44
Figure 2-5: The sample markers provided by ARToolkit and ARTag.....	45
Figure 2-6: An <i>AR Spitfire</i> app on iPad developed using the Vuforia SDK: the stone image (shown left) is target for tracking.	48
Figure 2-7: The relationships between lower-level developers – end users of an AR system.	52
Figure 2-8: The relationships between lower-level developers – higher-level developers – final users of an AR system.....	54
Figure 2-9: The factors affecting usability testing methods.....	56
Figure 2-10: The perspective projection procedure of a pinhole camera model where	

upper case X,Y,Z denote camera coordinates and lower case x, y denote image coordinates.....	61
Figure 2-11: Black-white chessboard pattern with size of 9x6 provided by OpenCV library.	63
Figure 2-12: the unregistered depth image (upper) and the registered depth image (lower) with their corresponding colour image captured by Kinect.	65
Figure 2-13: n point segment test corner detection in an image patch.	71
Figure 2-14: Lowe (2004)'s Pyramid Scheme.	73
Figure 2-15: Maxima and minima of the DOG images are detected by comparing a pixel (marked with X) to its 26 neighbours (marked with circles).....	74
Figure 2-16: the value of the box filter of any rectangle can be computed using integral image representation.....	76
Figure 2-17: The generation of a SIFT keypoint descriptor.....	78
Figure 2-18: The position of P should be chosen by minimising the sum of squared errors between the measured p_i and re-projection \hat{p}_i	89
Figure 2-19: Some affordable input camera devices that can be used for vision-based AR.....	102
Figure 2-20: Diffuse rendering of the integrated LIDAR 3-D triangle meshes for the datasets fountain-P11 (upper) and Herz-Jesu-P8 (lower).....	115
Figure 2-21: An example of Precision-Recall curves to multi-class.	120
Figure 3-1: The profession of the respondents.	125
Figure 3-2: The respondents' understanding of AR.	126
Figure 3-3: The categories of the AR applications used declared by the respondents.	127
Figure 3-4: The intentions of the respondents to apply AR in their own works.	128

Figure 3-5: The areas that the respondents would like to develop AR for.....	129
Figure 3-6: The AR development experience of the respondents.	130
Figure 3-7: The importance of the factors that affect the respondents' evaluation of an AR SDK.....	133
Figure 3-8: The difficulty of the unfavourable situations that impede the respondents on their AR development.....	134
Figure 3-9: The AR use case requirements of the respondents.	138
Figure 3-10: A reference coordinate system based on a rectangle fiducial marker.	143
Figure 3-11: The transformations between world coordinate system, camera coordinate system and image pixel coordinate system.....	144
Figure 3-12: Space point P has four different coordinates with respect to the world reference frame and three different camera reference frames.	147
Figure 3-13: The process of mapping a 3D point from world coordinates to pixel coordinates.....	148
Figure 3-14: An image with fisheye lens distortion before distortion correction (left) and after correction (right).	150
Figure 4-1: The flowchart of graph-based RGBD-SLAM.	155
Figure 4-2: An ideal (without noise and error) procedure of visual SLAM.....	156
Figure 4-3: A pose-graph representation of a SLAM process.....	159
Figure 4-4: The GUI of RGBDSLAM v2.	161
Figure 4-5: The sequence diagram of RGBDSLAM v2.....	162
Figure 4-6: The strategy of RGBDSLAM v2 for loop closure detection by selecting three types of candidates from older Nodes.	168
Figure 4-7: The coordinates of same space points are changed due to the different	

reference frames.	169
Figure 4-8: The class graph between the custom class Node of RGBDSLAM v2 and the class of Vertex and Edge used for graph-based optimisation within g2o framework..	172
Figure 4-9: The flowchart of monocular SfM-based sparse 3D reconstruction.	175
Figure 4-10: Epipolar geometry constraints between two pinhole camera models.....	178
Figure 4-11: The sparse point cloud reconstruction of a car generated by VisualSfM (upper) and the dense point cloud reconstruction based on the VisualSfM result (lower).	188
Figure 4-12: Homography correspondences between two images.....	197
Figure 4-13: The trajectory comparison diagrams for the results of [freiburg1_desk].	214
Figure 4-14: The trajectory comparison diagrams for the results of [freiburg1_floor].	215
Figure 4-15: The trajectory comparison diagrams for the results of [freiburg1_360]..	216
Figure 4-16: The trajectory comparison diagrams for the results of [freiburg3_long_office_household].....	217
Figure 4-17: The trajectory comparison diagrams for the results of [fountain-P11] and [castle-P11].....	218
Figure 5-1: The work flow of the proposed online session to query input images from the trained database.	222
Figure 5-2: an example of the Hamming distance between two 8 binary bitsets. The hamming distance equals to the number of total differences between the two bitsets.	229
Figure 5-3: K-means clustering (upper) and a 3-level depth hierarchical k-means tree (lower).	233
Figure 5-4: The whole process of offline template training and how the online session makes use of the data contained within the trained database for user camera pose	

estimation.	242
Figure 5-5: first row: 3 (out of 25) frames in test data; second row: the corresponding frames of the first row in query data which were taken at same place, same time but 3 weeks later.	246
Figure 5-6: The trajectory comparison diagrams for the online tracking results of [freiburg1_desk].	253
Figure 5-7: The trajectory comparison diagrams for the online tracking results of [freiburg1_desk2].	254
Figure 5-8: The trajectory comparison diagrams for the online tracking results of [freiburg3_long_office_household].	255
Figure 6-1: The input-process-output diagram of a 3D reconstruction-based AR development framework, equates to the offline session mentioned previously.	260
Figure 6-2: The simple GUI tool contained within the AR development framework..	262
Figure 6-3: The OpenCV augmented views of part of the RGB frames contained within the dataset [freiburg3_long_office_household].	265
Figure 6-4: The OpenGL augmented views of part of the RGB frames contained within the dataset [freiburg3_long_office_household].	267
Figure 6-5: The National Marine Aquarium.	269
Figure 6-6: The Eddystone Reef Tank and its function zone.	270
Figure 6-7: Two-tier architecture of the Eddystone Reef Tank area.	270
Figure 6-8: The upstairs passage and physical marine animal models hung from the ceiling.	271

LIST OF TABLES

Table 2-1: The requirements of the proposed AR frameworks	59
Table 3-1: The satisfaction degree of the respondents on their applications.	130
Table 3-2: The satisfaction degree of the respondents on their applications.	131
Table 3-3: The importance and difficulty scores on the factors that affect the respondents AR development experience.....	136
Table 3-4: The importance scores on the factors that affect the respondents evaluating a SDK.	139
Table 4-1: The public datasets used in the present research.....	190
Table 4-2: A performance test on different feature detector – extractors (bracketed are %RSD).....	194
Table 4-3: A performance test on different matching methods for different types of feature (bracketed are %RSD).....	198
Table 4-4: The performance test on ORB and SiftGPU which has limited the number of features to detect (bracketed are %RSD).....	200
Table 4-5: The performance test of RGBDSLAM v2 on different RGBD sequences.	205
Table 4-6: The performance test of VisualSfM on RGB data contained within the RGBD dataset (bracketed are total processing time without extra time for exporting & importing files).	209
Table 4-7: The performance test of VisualSfM on outdoor RGB sequences.....	211
Table 5-1: The BoW cluster algorithm for SIFT-like descriptors suggested by Cummins & Newman (2011).....	230
Table 5-2: The processing time and retrieval precision of frame-to-frame local feature matching method by using SiftGPU and ORB features (bracketed are %RSD).	247

Table 5-3: The processing time and retrieval precision of FAB-MAP with different feature descriptors and vocabularies.....	249
Table 5-4: The test results of the proposed online tracking stage on two office-like environments, [freiburg1_desk] and [freiburg3_long_office_household], which have been restored in the offline session as presented in Section 4.3.2 (bracketed are %RSD).	251

LIST OF ABBREVIATIONS

2D	2-dimensional.
3D	3-dimensional.
API	Application Programming Interface.
AR	Augmented Reality.
AV	Augmented Virtuality.
Avg.	Average.
BA	Bundle Adjustment.
BoW	Bag of Words.
BRIEF	Binary Robust Independent Elementary Features.
CAD	Computer-Aided Design.
CPU	Central Processing Unit.
CV	Computer Vision.
DLT	Direct Linear Transformation.
DoF	Degree of Freedom.
DoG	Difference of Gaussians.
EKF	Extended Kalman Filter.
E-PnP	Efficient Perspective-n-Point.
FAB-MAP	Fast Appearance-Based Mapping.
FAST	Features from Accelerated Segment Test.
FAST-ER	Features from Accelerated Segment Test - Enhanced Repeatability.
FLANN	Fast Library for Approximate Nearest Neighbours.
fps	frames per second.
GIS	Geographical Information System.

GLOH	Gradient Location and Orientation Histogram.
GPS	Global Positioning System.
GPU	Graphics Processing Unit.
HCI	Human-Computer Interaction.
HMD	Head Mounted Display.
IMU	Inertial Measurement Unit.
INS	Inertial Navigation System.
iOS	iPhone Operating System.
IR	Infra-Red.
ISMAR	International Symposium on Mixed and Augmented Reality.
kd-tree	<i>k</i> -dimensional tree.
KL	Kullback–Leibler.
LHS	Left-Handed Coordinate System.
LIDAR	LIght Detection And Ranging.
LM	Levenberg-Marquardt.
LoG	Laplacian of Gaussian.
LS	Least-Squares.
MLESAC	Maximum Likelihood Estimation SAmples and Consensus.
MR	Mixed Reality.
NMA	National Marine Aquarium (Plymouth)
NN	Nearest Neighbour.
No.	Number.
NVM	N-View Match.
OBJ	Object.

OpenCV	Open Source Computer Vision.
OpenGL	Open Graphics Library.
OpenNI	Open Natural Interaction.
ORB	Oriented Fast and Rotated BRIEF.
P3P	Perspective-3-Point.
PC	Personal Computer.
PCA	Principal Components Analysis.
PCD	Point Cloud Data.
PCL	Point Cloud Library.
PMVS2	Patch-based Multi View Stereo Software.
PnP	Perspective-n-Point.
PROSAC	PROgressive SAmples Consensus.
RANSAC	RANdom SAmples Consensus.
RGB	Red, Green and Blue.
RGBD	Red, Green, Blue and Depth.
RHS	Right-Handed Coordinate System.
RMSE	Root Mean Squared Error.
ROS	Robot Operating System.
RSD	Relative Standard Deviation.
RTAB-Map	Real-Time Appearance-Based Mapping.
RV	Reality-Virtuality.
SBA	Sparse Bundle Adjustment.
SDK	Software Development Kit.
SfM	Structure from Motion.

SIFT	Scale-invariant feature transform.
SLAM	Simultaneous Localisation And Mapping.
sSBA	sparse Sparse Bundle Adjustment.
SSD	Sum of Squared Differences.
SUSAN	Smallest Univalve Segment Assimilating Nucleus.
SVD	Singular Value Decomposition.
tf-idf	term frequency–inverted document frequency.
UI	User Interface.
USAN	Univalve Segment Assimilating Nucleus.
USB	Universal Serial Bus.
VR	Virtual Reality.
WiPS	Wi-Fi Positioning System.
XML	Extensible Markup Language.

Chapter 1 Introduction

Augmented Reality (AR) is a cutting-edge technology that has been in existence for several years, developing in parallel with its so-called “immersive” counterpart, *Virtual Reality (VR)*. As described in Milgram *et al.* (1994), a common view of a VR environment is one in which the participant or observer is totally immersed into a completely synthetic world, which may exceed the bounds of physical reality and in which the physical laws no longer hold. For example, Cheng (2015) developed a virtual reality reconstruction – *Virtual Wembury* – which allows users to take a ‘walk’ along a virtual south-west coastal path in Devon from different places in the real world (e.g. in a hospital) by using VR headsets and hand controllers, as shown in Figure 1-1 and Figure 1-2.



Figure 1-1: The screenshot of Virtual Wembury.¹

¹ **Virtual Wembury 2015:** <https://www.youtube.com/watch?v=tyH-4IGrPnE>



Figure 1-2: Virtual Wembury Event.

(Cheng, 2015)

The virtual environment and the real environment can be viewed as lying at opposite ends of a continuum – a *Reality-Virtuality (RV) continuum* –, as illustrated in Figure 1-3 below.

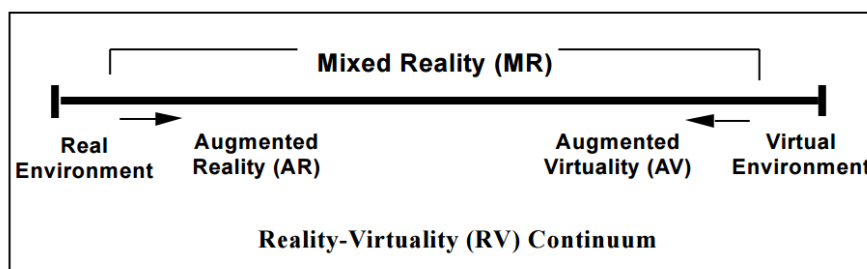


Figure 1-3: Simplified representation of an RV Continuum.

(Milgram *et al.*, 1994)

The area between the extremities of the RV continuum is considered as a generic *Mixed*

Reality (MR) environment, in which both the real world and the virtual word objects are mixed together. Both *Augmented Reality (AR)* and *Augmented Virtuality (AV)* are considered as subcategories of MR. An AV environment is principally virtual but augmented through the use of real objects, such as showing real hands in a virtual environment (Figure 1-4). As opposed to AV, an AR environment is principally real with added virtual enhancement, such as placing a virtual rabbit on top of the user's hand (Figure 1-5).

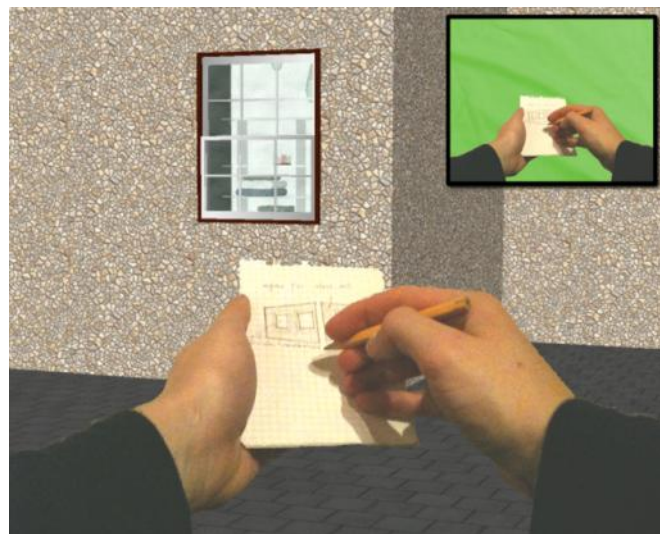


Figure 1-4: Real hands in virtual world.

(Bruder *et al.*)



Figure 1-5: Virtual rabbit on user's hand with *Handy AR*.

(Lee & Hollerer, 2007)

The goal of AR, generally speaking, is to enhance the end users' perception and interaction with a given real environment by superimposing computer-generated ("virtual") information upon it (Carmigniani & Furht, 2011). Visual "augmentations", a slight variation on the theme of AR, aim to blend virtual visual information (e.g. 3D interactive graphics) with views of real world and to display the synthesised results to the end users. Visual AR has been widely used in several application areas which can be traced back, as in a survey reported by Azuma (1997), to many research projects and studies. For education and training, AR has been adopted into both academic and corporate settings (Lee, 2012). Núñez *et al.* (2008) have introduced an AR system for teaching Inorganic Chemistry at the university-level by setting up a collaborative environment that supports several groups of students interacting with material and compound structures. Their experience shows that the students enjoy the system and their motivation for learning Inorganic Chemistry is improved. *Construct3D* (Kaufmann, 2004) is another AR system designed for higher education, which provides a natural setting for face-to-face collaboration of teachers and students to learn mathematics and geometry. The author stated that the main advantage of using AR is that complex spatial problems and spatial relationships may be comprehended better and faster than with traditional methods by working directly in 3D space. According to Sherstyuk *et al.* (2011), AR and MR technologies significantly expand mannequin functionality in medicine and medical education (e.g. Bichlmeier *et al.* (2008), Kondo *et al.* (2004)) due to the high cost of human error. A variety of AR techniques have been also integrated into image-guided surgery system, such as a 3D AR navigation system using autostereoscopic images developed by Liao *et al.* (2010). Their system spatially projects 3-D images onto the surgical area which is viewed via a half-silvered mirror, increasing

the surgical instrument placement accuracy and reducing the procedure time as a result of intuitive 3-D viewing. In cultural heritage sites, AR can be used to inform travellers of historical stories on-site and even present the original look of buildings or landscapes that no longer exist by constructing virtual 3D models and laying them over the ruins (e.g. Guo *et al.* (2009), Kakuta *et al.* (2008), Papagiannakis *et al.* (2002)). According to Livingston *et al.* (2011), many AR projects have been proposed to meet various military needs and challenges, such as the *SuperCockpit* (Furness, 1969) system, where flight and target data were superimposed onto the pilot's visual field to assist localisation, and the *Battlefield Augmented Reality System* (Livingston *et al.*, 2002), which dealt with similar functions for the dismounted warfighter in both operation and training. Huang *et al.* (2011) also points out that some characteristics of AR, such as a better sense of immersion, maintenance of real world understanding, appealing experiences for users and an absence of the need to build complicated virtual surroundings, meet many requirements of the exhibition and entertainment industry.

A visual AR system generally includes one or more processes by which the host computing system has to “learn” about features and structures in the real world or “workspace” from a form of “reference template”, which can be one or more objects or *fiducial* markers² located within the workspace, or even the whole workspace scene, so that augmented objects can be inserted accurately and meaningfully. This learned “knowledge” is then utilised for running the client application of the end users who may wish to perceive and interact with the AR view, by estimating the location and

² **Fiducial marker:** an object placed in the field of view of an imaging system which appears in the image produced, for use as a point of reference or a measure.

orientation of the end users with respect to the reference template, and, thus, registering and rendering the augmented content appropriately. Several research projects and applications have, in the past, focused on different stages of the AR process by exploiting various types of sensors, different training, tracking and image registration methods and techniques, and a range of display technologies (wearables, Smartphones, tablets, PC “kiosks”, etc.) (Zhou *et al.*, 2008). For an AR application that requires an accurate (as opposed to approximate) registration of virtual objects, vision-based methods are preferred, for use with or without other techniques, such as fiducial markers placed in the environment, or hybrid integration with the GPS location of the end user. Vision-based techniques, whereby the available visual information in a scene can be collected by an optical sensor (such as a camera), can be used to identify, reference and recognise the spatial relationship between the cameras or the AR users and the environment.

Two major vision-based methods have typically been considered in the literature. The first relates to artificial marker-based methods, which require the introduction of fiducial markers or 3D models (with abundant visual features) into the environment, the spatial geometry of each of which are known. The second is more of a natural feature-based method, which requires the learning and training of visual feature information from the original target environment. The latter method is less intrusive and more flexible than the former since “they function anywhere without the need for special labelling or supplemental reference points” (Johnson *et al.*, 2010), and can be applied in both indoor and outdoor environments. However, the database creation stage (a.k.a. reference template/map training) within natural feature-based AR is more complex, since the geometric structure of a workspace may be complicated which is hard for

people to restore it with 3D modelling applications precisely and manually. Many *computer vision (CV)*-based methods, such as *Structure from Motion (SfM)* (Ullman, 1979) or *visual Simultaneous Localisation and Mapping (visual SLAM)* (Riisgaard & Blas, 2003), have already been applied for unsupervised learning and training of the reference template, and for tracking the end users. It can be found that most of the recent researchers who have attempted to apply natural feature-based methods for solving AR issues within their chosen application area (such as the examples described above) have always concentrated on undertaking the database training stage themselves (stated as “one off prototypes” in Dünser & Billinghurst (2011)). Alternatively, some applications have allowed the end users to create the reference map and inset virtual information for a more general purpose, then, to view the augmentation results in the same online session (e.g. Klein & Murray (2007)). However, this kind of system has assumed that the users who want to create the augmentation and the users who want to view the augmentation are the same people. This does not apply to all practical applications, many of which will have two kinds of end user – (a) the **higher-level application designers or developers**, who may not have a professional background (in such techniques or process as CV, SLAM, etc.) as the lower-level AR researchers, but want to apply AR technology to their own developments, and (b) the **final users** who will use the developed application (or product).

The present research concentrates on developing an approach focused on the user-oriented, markerless visual AR framework. Unlike previous research studies, this proposed work is not presented as an AR solution with specific aim but instead sets out to provide a **general development framework** for higher-level developers to create

their own applications for a **specific place**. The motivation is that some AR projects, such as heritage sites or exhibitions (e.g. Papagiannakis *et al.* (2002)'s *Ancient Pompeii*), expect users to visit the sites physically and experience an AR environment specifically designed for the place. The proposed development framework allows people to build such a customised AR application by providing the visual information of the targeted places to augment, and their end users will see the AR view through the AR applications only when they visit the place. With regards to user-oriented, **both** the AR application developer and the individual who finally views or experiences the end result of the AR application are considered as end users. Hence the proposed AR process is divided into two stages – an **offline database training session** for the application developers, and an **online AR tracking and display session** for the final application users. The whole system diagram of the proposed framework is presented in Figure 1-6.

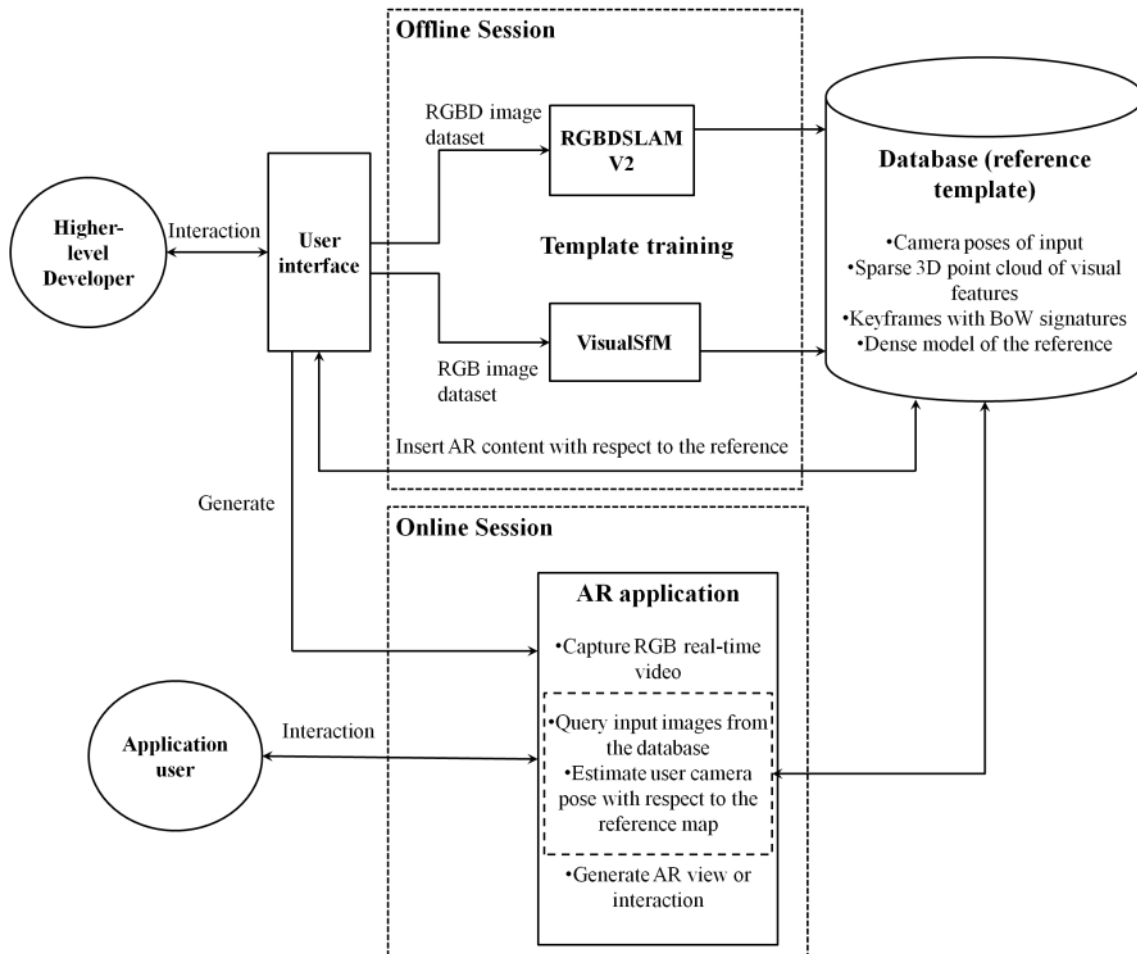


Figure 1-6: The system diagram of the proposed user-oriented markerless AR framework.

With consideration given to delivery technologies that will be within the financial reach of typical AR application/product users (including Smartphones, tablets, etc.), the proposed system is designed to be low-cost and, in the main, purely vision-based. The offline session will process RGB or RGBD images which are easily collected by most users for the recovery of the spatial structure of the target environment and for the creation of a reference map for the online session. Two major 3D reconstruction methods known as *VisualSfM* (Wu, 2011; 2013) and *RGBD-SLAM* (Endres *et al.*, 2014) are applied and tested, and their respective assets and drawbacks are compared and analysed. Moreover, in order to deal with the often-overlooked role of application

developers, the proposed framework offers a conceptual development framework for such developers. This allows them to interact with the offline process by selecting input data sources for template training, and further to define the relative location of augmented information from the system-constructed map, inserting this information into the database with a simple GUI tool. The online session requires RGB image sequences as an input to locate the end user's location in relation to the reference map, and then projects and renders the defined augmented information onto them. To find the landmarks efficiently, and to retrieve the keyframes with the highest probabilities in order to match the initial query frames from the database, the present proposal also applies *Fast Appearance-Based Mapping (FAB-MAP)* (Cummins & Newman, 2008) with training a visual vocabulary of *Bag of Words (BoW)* (Sivic & Zisserman, 2003) to describe image features of each reference keyframe. Several visual features and their respective BoW-generating are tested and evaluated on both accuracy and efficiency in a “*loop-closure detection*” task in the online session. The estimated pose of each input image is then used for displaying the AR view to the end users. The results are analysed statistically and visually using public datasets of different environments.

1.1. Scientific and Novel Contributions

The major contributions presented in this thesis are studying and comparatively evaluating several technologies for the different stages of the proposed markerless AR framework described above, which can be summarised as follows:

- 1) Study of the user types and requirements through an online AR application development and requirement audience survey (Chapter 3).

- 2) Application of low-cost vision-based 3D reconstruction / mapping technologies for an offline, marker-less reference template training, which allows both RGB and RGBD image datasets as the input source. The evaluation and comparison of two existing 3D reconstruction applications, – RGBD-SLAM v2 (Endres *et al.*, 2014) for RGBD input and VisualSfM (Wu, 2011; 2013) for RGB input – with several public datasets. Different visual feature detection and description methods are used for checking their performance, such as SiftGPU (Wu, 2007) and ORB (Rublee *et al.*, 2011). The accuracy of 3D reconstruction results were assessed by comparing estimated camera poses with the ground truth of the real camera trajectories (Chapter 4).
- 3) Application of the BoW and the FAB-MAP (Cummins & Newman, 2008) approaches to enable a rapid loop closure detection process for retrieving those online input RGB images without prior knowledge from a trained database. Two visual vocabulary generating methods are used for both SIFT-like and ORB visual descriptors. The retrieval results are used for user tracking and the performance is tested and compared with a public dataset in terms of accuracy and processing rate (Chapter 5).
- 4) Development of a conceptual “black-box” style markerless AR development framework exploiting the above technologies and the provision of a general user interface to the AR developer for interaction. A simple GUI tool for augmentation configuration was also provided. A plan presenting a future user evaluation on the development framework with a real use case is also presented (Chapter 6).

1.2. Thesis outline

The logical sequence of the chapters in the thesis is briefly summarised in Figure 1-7.

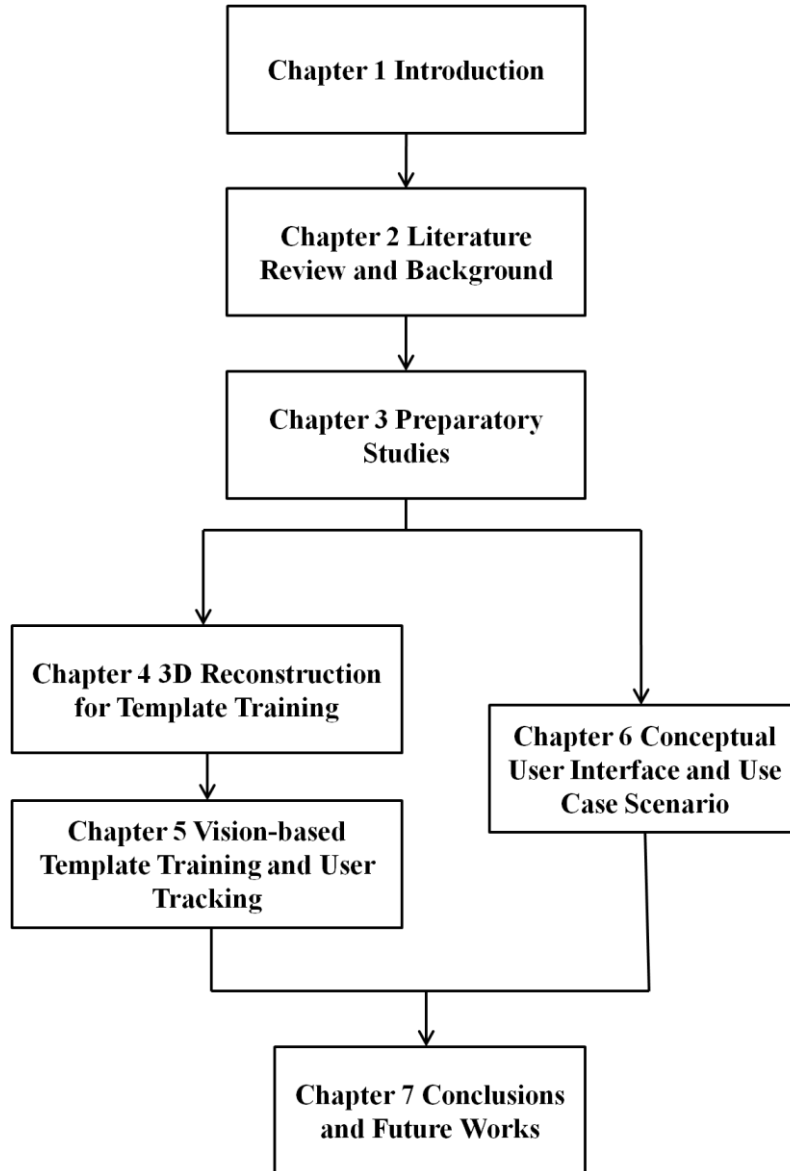


Figure 1-7: The flow chart of the content of each chapter.

Chapter 2 presents a literature review and background of AR, especially visual AR, addressing the related computer vision methods and technologies. The existing problems are analysed and the challenges are indicated at the end.

Chapter 3 firstly reports a questionnaire analysis of the AR application development and requirement based on an online survey. Then the definitions of different reference frames used in the present thesis are presented with the frequently used geometric transformations in vision-based AR, including 3D-to-3D rigid transformations between different spaces, and 3D-to-2D camera projections from space to image plane.

Chapter 4 presents the main process, 3D reconstruction of the offline session in the proposed AR framework. The principles and implementations of two reconstruction methods, RGBD-SLAM for RGBD data input and SfM for RGB data input, are introduced in detail. Two existing applications RGBD-SLAM v2 and VisualSfM for each method are applied and evaluated with several public datasets of different environments. The effect of making use of the different visual feature detection and description methods (e.g. SiftGPU and ORB) is also presented and discussed.

Chapter 5 presents details on how to make use of the 3D reconstruction results for training the reference template, and how to perform a rapid loop closure detection to retrieve the online input RGB image from the trained database for further user tracking. The visual word clustering and BoW encoding approaches for both SIFT-like and ORB visual descriptors are introduced. The FAB-MAP approach, which scores the matching likelihood between two images due to their BoW signatures, is also presented. The performance of online user tracking is tested on a set of public dataset and is evaluated statistically and visually at the end.

Chapter 6 presents the conceptual design of user interface for proposed AR

development framework with a simple augmentation configuration GUI tool, and a basic AR browser for basic augmentation registration and display. The further user evaluation of the development framework on a real use case is planned at the end.

Chapter 7 draws conclusions by giving a summary of the proposed work addressing some of the pros and cons of the techniques and processes, and suggests future work to improve the framework developed in the present thesis.

Chapter 2 Literature Review and Background

This chapter presents a comprehensive review of the previous research and the component technologies of AR covered in the present thesis. It starts with the introduction of AR, and the process of visual AR with several different tracking methods (sensor-based and vision-based) are mainly reviewed in Section 2.1.1. The AR application and development framework are analysed from the perspective of two different end users (developers and application final users) in Section 2.1.2. The evaluation methods of AR system then are discussed in Section 2.1.3.

In Section 2.2, the computer vision technologies used in AR are discussed. The topic starts from the cameras which are used to acquire visual information about the workspace and their calibrations (Section 2.2.1). The several kinds of visual features which can be extracted from the image data to represent an image are reviewed in Section 2.2.2. This section includes feature detection, representation and how to make use of these processes to compare images for the purpose of camera pose estimation within an AR system.

The visual features than are used in the processes for solving the problem of localisation and mapping in Section 2.2.3, which can be used for AR system to restore a markerless reference template from the original environment. This section includes two 3D reconstruction approaches: structure from motion and visual simultaneous localisation and mapping.

The process of online user tracking requires finding the input query image quickly from

the reconstructed reference model stored in a trained database, known as a loop closure detection problem. The image retrieval techniques used to solve this issue are discussed in Section 2.2.4.

Subsequently, a survey of hardware devices and software packages that are available for system implementations of vision-based AR systems is presented in Section 2.3.1 and 2.3.2 respectively. Then the available datasets and benchmarks for testing and evaluating the performance of each technique inside the proposed system (*i.e.* 3D reconstruction or mapping, image retrieval) are discussed in Section 2.3.3.

This chapter finally concludes with an overview of existing problems and challenges of an AR system based on the discussion in the previous sections, and suggests a user-oriented markerless AR framework to deal with the them (Section 2.4).

2.1. Augmented Reality

The term “augmented reality (AR)” is believed to have been coined by former Boeing researcher Tom Caudell in 1990, although the first AR prototypes appeared earlier than the coining of the term in the late 1960s and 1970s (Johnson *et al.*, 2010). AR is generally considered as a derivative of Virtual Reality (VR). Immersive VR allows the participant to explore and interact in real time in a variety of simulated scenarios of varying levels of detail or fidelity. As described by Stone (1996), VR permits intuitive interaction with real-time three dimensional databases. A VR system may support multi-sensory experiences which can include smell, and touch (‘haptics’), but mostly

hearing and sight. The artificial virtual environment can be delivered to the participant through a computer monitor, a head-mounted display, large-screen wall display or some form of stereo-surface projection. As Wilson (1999) argued, different types of technologies, including AR, can provide very different experiences, which one to be selected is very dependent on the needs of particular tasks (such as the capabilities required of the end user, the task environment and, of course, the finance available for hardware and software technologies).

In contrast to the common aim of VR systems that attempt to “immerse” their end users into an “enclosed” computerised virtual environment, AR aims to enhance the user’s perception and interaction with the real world by superimposing virtual information (*i.e.* sound, video, graphics or any other virtual data generated by computer) onto or within a physical real-world environment (Carmigniani & Furht, 2011). AR applications cover a wide range of multidisciplinary areas including medical, manufacturing, visualisation, path planning, entertainment and military training (Azuma, 1997). A brief introduction of several applications has already been given in Chapter 1. Azuma (1997) also describes AR systems as possessing three characteristics: 1) they combine real and virtual elements; 2) they are interactive in real time; 3) they present fused images that are registered in the real world. Although audio information is involved in some research studies (e.g. narrative-based audio-only games presented in Roden *et al.* (2007)), the augmentations are mostly associated with vision – thus referring to the 3rd characteristic mentioned above. The visual augmentation is the goal to be achieved in many cases since blending the virtual information with the real world in a visually

coherent manner can provide the participant with stronger visual perception³. The problems and technologies associated with visual augmented reality are the main foci of the present thesis.

The relative technologies of building a comprehensive AR system and the usability evaluation methods for AR interfaces are described and discussed in the following sections. The detail of each method and program used in the present research is presented in the remaining chapters.

2.1.1. Visual AR technologies

According to Krevelen & Poelman (2010), visual AR systems must perform a range of typical tasks including sensing (capturing), tracking (measuring), registration (rendering) and interaction (API) to support their applications. From the perspective of the final users of the AR applications, these tasks can be interpreted as an *online* loop process which firstly acquires information from the environment (a.k.a ‘workspace’) around the users (especially the scene the users are looking at), estimates the location and motion of the users (particularly their head motion or point of view), generates augmented views by superimposing pre-defined virtual objects upon the images of the real world, and finally displays them to the users, in some cases supporting additional human-computer interaction which depends on the specific aim of each AR application. In addition to the actual running of the AR application, a system training stage is generally

³**Visual Coherence in Augmented Reality:**

<http://ael.gatech.edu/lab/research/technology/visual-coherence/>

inevitable before (*offline*) or during the process described above. The training stage registers the virtual objects to their actual real-world locations, often defined by the application designers or developers. It requires the system to identify and learn the geometric structure of a “reference”, such as a *fiducial marker*, or a specific object or real-world features, so that the location of the virtual objects to insert can be defined with respect to that reference. When the system training stage has been completed, the actual running of the application will display the augmentation to the final users as the reference is being recognised from the workspace.

Several techniques have been applied to deal with the above tasks, as evidence in the literature. Zhou *et al.* (2008) reviewed a ten-year development of the AR “community”, presented at the ISMAR (International Symposium on Mixed and Augmented Reality) conference. They concluded that tracking techniques are one of the most popular and challenging concerns for most AR researchers. The tracking process enables the detection and measurement of changes in the viewer’s position and direction, and these can then be properly reflected in the rendered graphics during registration process. There are two representative types of tracking methods, known as *sensor-based* and *vision-based*, which have been applied to various AR systems and applications, in many instances based on the hardware used and the tracking accuracy required.

Zhou *et al.* (2008) describes sensor-based tracking techniques exploiting magnetic, acoustic, inertial, optical and/or mechanical sensors. Each of these techniques feature different advantages and drawbacks, and these have been reviewed in Rolland *et al.* (2001). The literature describing purely sensor-based tracking is mostly focused on the

approaches by using *Global Positioning System (GPS)* or *Global Positioning System / Inertial Navigation System (GPS/INS)* integration sensors, which can directly provide geographical position and orientation of the sensors' carrier (e.g. a robot or a human end-user) without additional computation. Examples include Roberts *et al.* (2002), who describe a technique which determines the user position with respect to the reference frame of the geographical database by using integrated GPS/INS sensors, and allows users to “look” into the ground to “see” underground features. Reitmayr & Schmalstieg (2004) describe a system for collaborative navigation and information browsing tasks in an urban environment. Schall *et al.* (2009) present a system that uses Kalman filtering for fusion of GPS with barometric heights, and also for an inertial measurement unit with gyroscopes, magnetometers and accelerometers to estimate the orientation, velocities, accelerations and sensor biases. Santana-Fernández *et al.* (2010) describe a *GIS (Geographic Information System)*-based guidance system that allows the driver of a tractor to see the real agricultural plot through eye monitor glasses with the treated zones in a different colour. GPS/INS sensors have become lightweight in recent years and are, today, embedded in many popular smart-phones and tablet computers (e.g. iPhone, iPad and Samsung series, as shown in Figure 2-1), which makes them easier to be obtained and used for research applications.



Figure 2-1: Contemporary smartphone hardware sensors⁴.

The literatures focusing on mobile GPS/INS-based design and applications includes that by Chang & Qing (2010), who present the system design of a Multi-Object Oriented AR system for a location-based adaptive mobile learning environment, and a study of the scenario. Choi *et al.* (2011) who describe a system that allows users to create geospatial tags for certain sites and propose a way to organise and group such geospatial tags and how to efficiently search and find the tag that users be interested in. Lee *et al.* (2012) introduce an application entitled *CityViewAR* that provides geographical information about destroyed buildings and historical sites that have been affected by the earthquakes with the geo-located information provided in 2D map views, AR visualisation of 3D models of buildings *on-site*, immersive panorama photographs, and list views. Most of the commercial mobile AR Software Development Kits (SDKs) support GPS/INS based

⁴ **Indoor navigation using smartphones:**

<https://connect.innovateuk.org/documents/3347783/3709547/Indoor+Navigation+using+smarthphones.pdf/ae5a631f-55d3-4ffe-9a13-0ec23cd09287>

tracking. Figure 2-2 and Figure 2-3 are two custom AR applications for the iPad developed with the *Wikitude* SDK and *Metaio* SDK respectively. Figure 2-2 shows a GIS-based AR campus navigation application developed by the author using the Wikitude Software Development Kit. The campus buildings are all tagged and shown on a virtual “radar”; the building information will be displayed in the text box on the bottom of the screen by pressing corresponding floating button. Figure 2-3 shows another conceptual GIS-based AR application for heritage. The geo-tag bubble is floating on the direction where the real-world target object (in this case a distant shipwreck located in the West of England!) is located; the text information and the 3D target model will be displayed by pressing the bubble. Note that the GPS signal is usually limited and even blocked when in an indoor environment. Thus in the case of indoor situations, as shown in Figure 2-2 (lower) and Figure 2-3, a *Wi-Fi Positioning System (WiPS)* was used instead for indoor localisation. The basic idea is that wireless access points are deployed along with Wi-Fi, which enable wireless devices to locate the approximate position (Bahl & Padmanabhan (2000) cited in Boonsriwai & Apavatjirut (2013)).

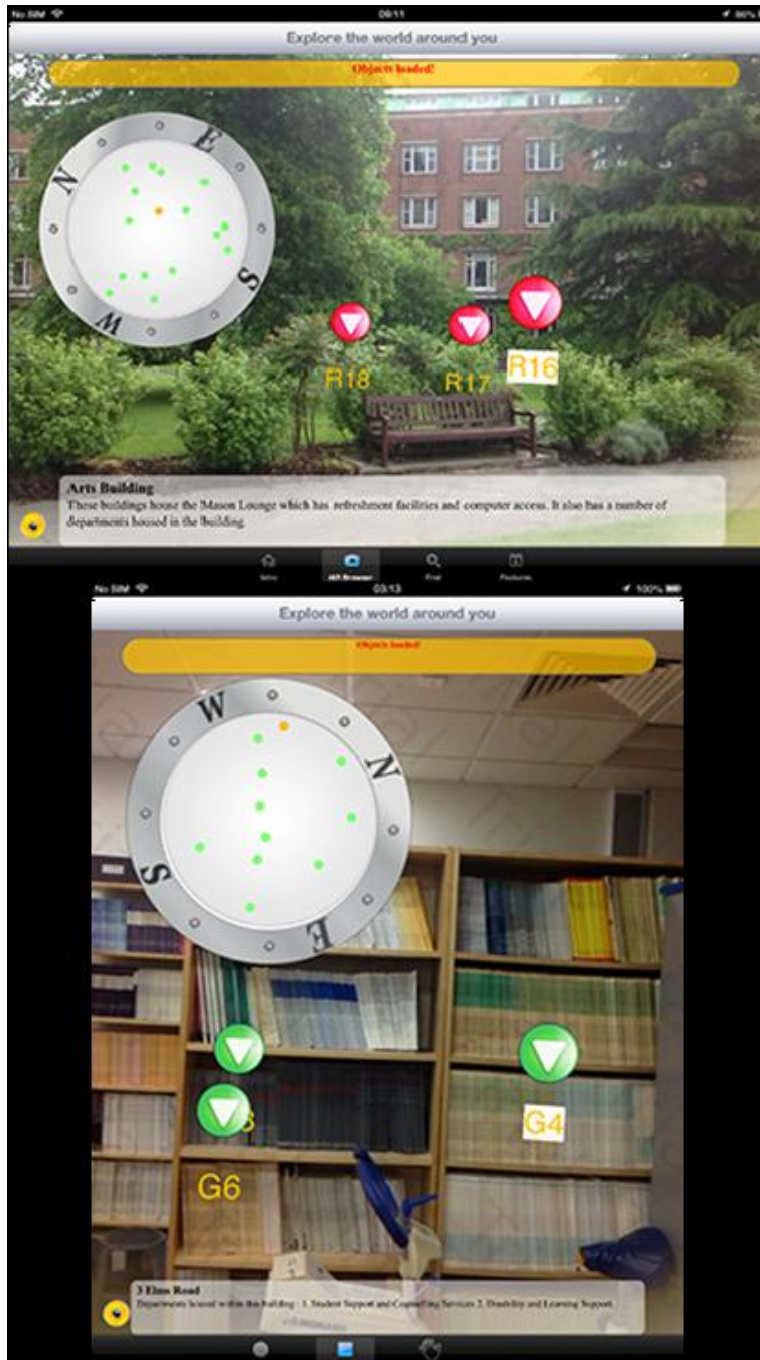


Figure 2-2: An AR campus navigation app on iPad developed using the Wikitude SDK.

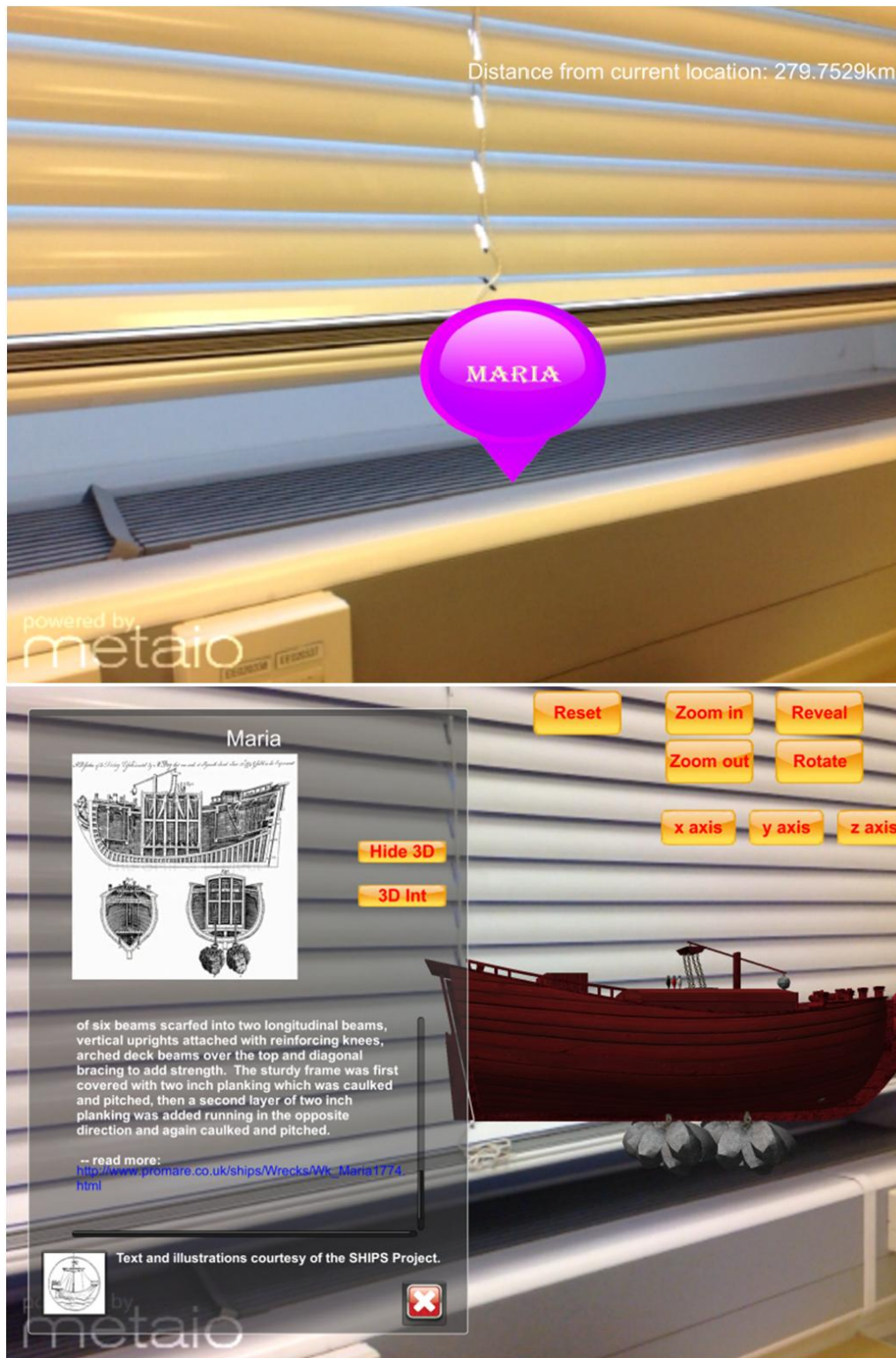


Figure 2-3: An AR shipwreck (Maria) app on iPad developed using the Metaio SDK.

Due to the fact that the visual AR experience is embodied in the augmented views, no

matter which tracking method is used, the system will make use of the tracking results (*i.e.* the measurement of user motions) to align the virtual information to the view of the real world and display them to the users properly, known as *registration* process. Non-vision-sensor-measured motions are usually uncoupled to the visual information acquired by the system. As reported in Van Krevelen & Poelman (2007), plain GPS was accurate to about 10-15 meters; with the *Wide Area Augmentation System* technology the precision might be increased to 3-4 meters. *Differential* GPS could yield a 1-3 meter accuracy but required a local base station to send a differential error-correction signal to the roaming unit. The precision could be enhanced further to centimetre-level by applying a *Real Time Kinematic* technique based on carrier-phase positioning, in which the phase of the received carrier with respect to the phase of a carrier generated by an oscillator in the GPS receiver (Langley, 1998). However Fukuda *et al.* (2014) indicated that the use of GPS-based AR registration could be problematic due to the expensive hardware required for highly precise and accurate registration which may not always be available for the users. As for the relatively affordable location and orientation sensors in smart-phones, Blum *et al.* (2012) tested several smart-phones at the time and found that the GPS sensors exhibited errors with means of 10-30m; the mean errors of compass sensors were around 10-30°. The standard deviations around these means could be relatively high, mainly depending on the surrounding environment (e.g. the surrounding building). A more recent positioning accuracy study of smart-phones released by location data company Place IQ demonstrated a very similar result (cited in Buczkowski (2016)), in which the average variance within a city is roughly 30 metres and the actual range of results was actually very wide (from 1m to 204m). The accuracy comes down to interplay of several factors, including signal source (e.g. GPS or Wi-Fi),

environment and personal use. In conclusion, if the AR applications do not require a very precise registration, such as only viewing a floating planar picture or information bubble on a general location, it will be appropriate to acquire an approximate positional relation between the user and the real world by using relatively low-cost GPS/WPS integration sensors. Otherwise more expensive hardware for location and orientation is required.

In order to achieve a more realistic augmented view of perspective 3D models which will align to the background environment tightly, accurately and reliably, it becomes important to focus more on the use of computer vision-based technologies. In fact, vision-based tracking techniques have been one of the most active areas of research in the AR domain according to Zhou *et al.* (2008) and (Bostanci *et al.*, 2013). Vision-based tracking methods generally determine the relative motions of the users with respect to the workspace by finding trained fiducial markers or 3D models through the input image data. Hence, vision-based tracking can be categorised in terms of both *marker-based* tracking and *marker-less-based* tracking (Herling & Broll, 2011). In early research studies, many investigators focused on using planar markers with high-contrast pattern, such as those made available for *ARToolkit* (Kato & Billinghurst, 1999) and *ARTag* (Fiala, 2005; Cawood & Fiala, 2008).

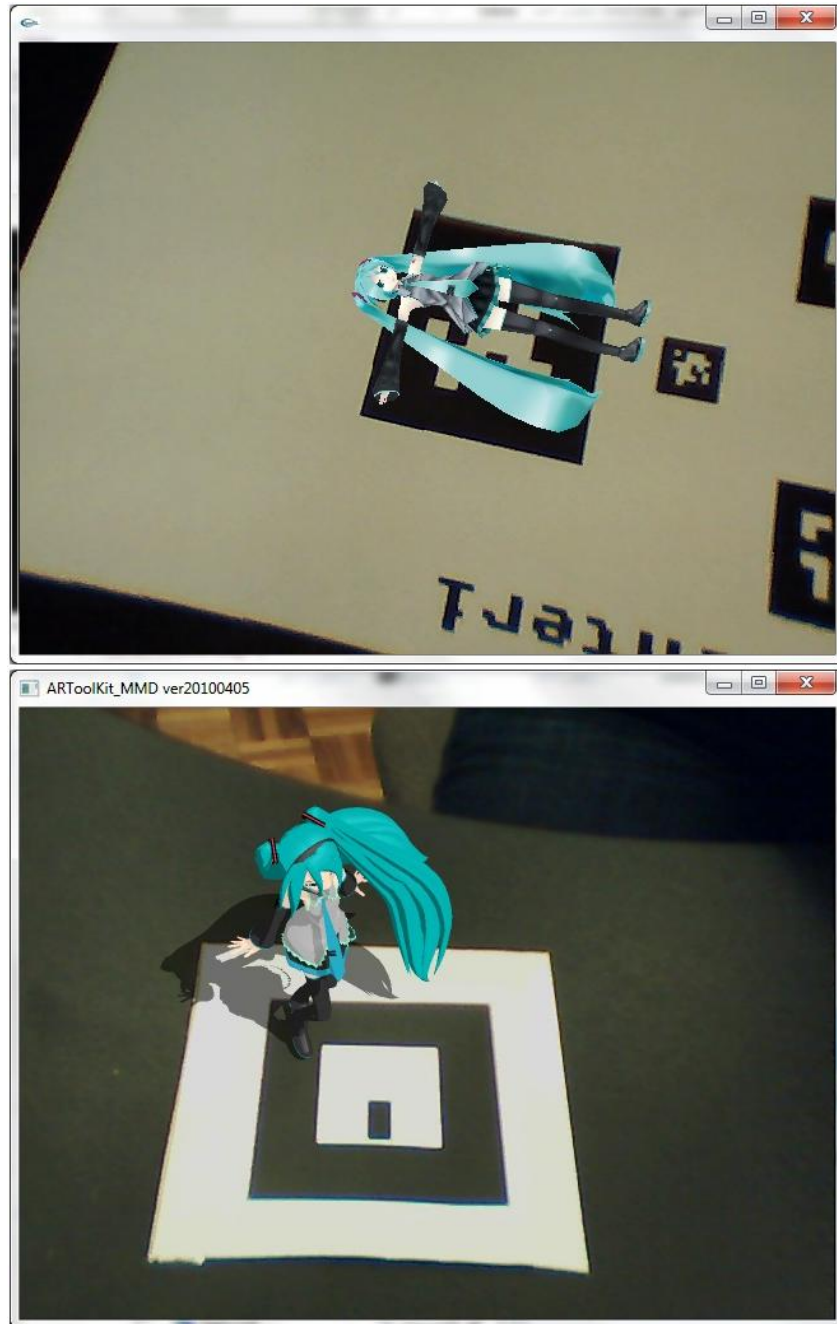


Figure 2-4: A sample program of ARToolkit with a black and white square fiducial marker.

The simplest form of planar marker can be a black-and-white square fiducial marker, such as the one shown in Figure 2-4. The reason for using black and white markers is because they deliver a higher contrast target in different luminance conditions, making them easier to be detected. The relative motion between the marker plane and the user

camera can be calculated by finding 3D-to-2D point correspondences from the 3D world to the 2D image, a process which will be described in Sections 3.2.3. DeMenthon & Davis (1992) stated that four known correspondences are sufficient to calculate the relative pose between the marker and the camera. The four corner points of a rectangular (or square) marker just meet the request, which are relatively robust and can be estimated by finding intersections of edge lines. The workflow of the black-and-white square marker detection routine is given in Baggio (2012), which can be summed up as follows. Firstly, the input image should be converted to greyscale and a binarisation operation is performed to transform each pixel to black or white – thus it will be possible to locate marker candidates by finding closed contour shapes with four vertices from the binary image. Then these candidates need to be unwrapped to square form by using perspective transformation – thus the pattern inside square can be verified by either *code-decoding* (e.g. ARTag) or *template matching* (e.g. ARToolkit) (Sun et al., 2011). The sample marker designed for these two methods are given in Figure 2-5.

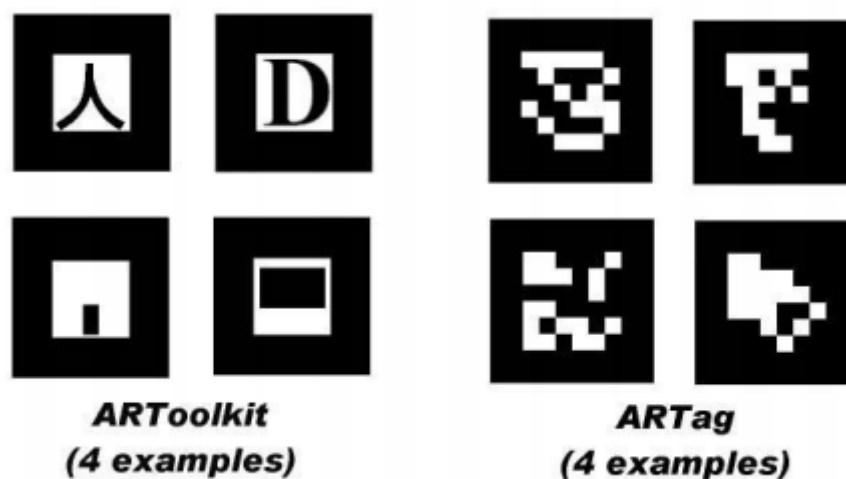


Figure 2-5: The sample markers provided by ARToolkit and ARTag.

(Fiala, 2005)

The traditional template matching method provided by ARToolkit involves defining a sampling grid inside the pattern, which is sampled to provide a 256 (or 1024) element feature vector. Then this feature vector is compared by correlation to a template marker library to identify the potential markers. However, Fiala (2005) commented that the use of correlation caused high levels of false marker detections where there are none, and high inter-marker confusion rates. The uniqueness of each marker also degenerates when user trains and adds more templates into the library. Meanwhile, the processing time of matching rises as each candidate must be correlated with all templates in the library. Moreover, the method stores twelve prototypes for each template marker to address the four possible rotations and three different light conditions, which causes more time to be consumed. On the other hand, Fiala (2005)'s ARTag improved the performance of ARToolkit by applying a code-decoding technique for marker identification and verification. As shown in Figure 2-5 (right), the interior of each square border is divided into 6x6 square grids, and each grid can be only black or white to carry one bit of digital data. Thus a marker can be translated into a 36-bit word which is easier and more robust to verify, and no graphic template training is required to construct the library.

Baggio (2012) indicates that the traditional fiducial marker-based tracking methods usually benefit from their relatively cheap detection algorithm and robust performance against lighting changes. However these methods are also restricted in applications due to the following issues: 1) the basic form of markers must be black and white with square borders, which cannot provide aesthetic value; 2) the size of supported markers is limited (e.g. 2^{36} in ARTag case); and 3) the marker cannot be tracked if partially

overlapped. Therefore, an advanced version of the planar marker utilises free-style patterns with naturally occurring visual features has appeared. Here, the visual features, including distinct points, lines, edges or textures (see Section 2.2.2 for detail) are going to be detected, extracted, and used as the reference for tracking. Natural feature-based markers literally can have any form and texture (with sufficient visual features mentioned above). They can be colourful images which are more pleasant to look at (certainly more so than the very visually intrusive black and white square form patterns, and especially when viewed in real-world scenes of nature), and they can also convey specific meaning to the users via the images themselves. Since the visual features within a natural feature-based markers are treated as discrete units rather than a whole pattern, the marker with partial occlusion can still be used for pose estimation if there are sufficient features matched between the query and template, as shown in Figure 2-6 – even when the stone marker for tracking in the lower-right was folded in from one corner, a valid tracking result was delivered with the AR *Spitfire*. The computational cost largely depends on what kind of feature detection, extraction and matching algorithms are used. According to Amin & Govilkar (2015), there are various AR SDKs have already supported natural feature-based AR tracking on mobile devices. Figure 2-6 just shows a natural feature-based AR application on an iPad developed using the Vuforia SDK. Many recent media advertisement campaigns prefer to use this kind of AR approach for their products, such as the *Starbucks' Cup Magic* application and Disney's AR billboard (Russell, 2012).

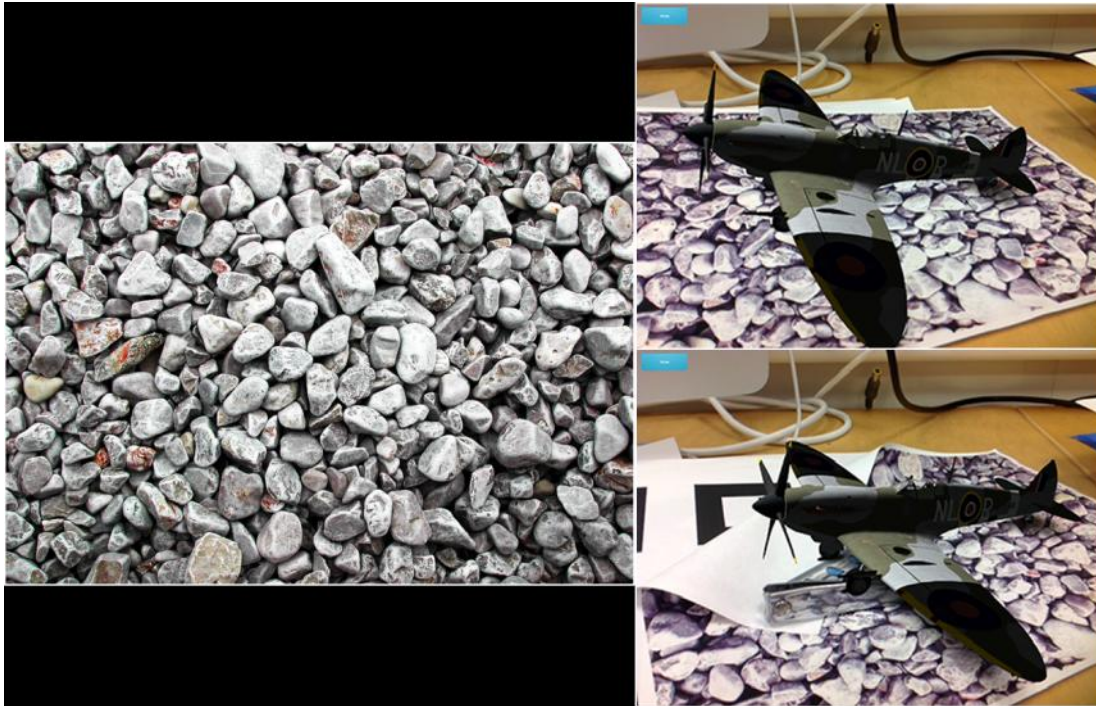


Figure 2-6: An AR *Spitfire* app on iPad developed using the Vuforia SDK: the stone image (shown left) is target for tracking.

By tracking trained planar markers in the environment, AR applications retrieve the position of the user and provide reasonable robust, rich and convenient AR experiences to, for example, exhibition and museum visitors (e.g. White *et al.*, 2007; Kolstee & van Eck, 2011; Jevremovic & Petrovski, 2012; Mor *et al.*, 2012) – all at a relatively low computational cost. However, individual planar markers do not support wide-angle ($>180^\circ$) rotation of the tracking. In other words, if the users move to the back of the pattern plane, the tracking will fail and no augmentation will be performed. To overcome this issue, some researchers have exploited 3D model markers, including Rabbi & Ullah (2014) who present an application of a cubic marker with six planar fiducial markers on its faces, which has shown to deliver a robust form of tracking during occlusion or from different angles of view.

The marker-based methods generally require introducing man-made reference models (2D images or 3D objects) into the workspace for tracking during the AR process. The models are normally artificial and designed to be easily recognisable and distinguishable from the background environment. However these features restrict marker-based system in their use in outdoor field settings since some artificial markers (e.g. physical, printed products) are fragile in such environments and can cause aesthetic and conservation issues in both urban and rural areas. For this reason, many researchers have focused on vision-based tracking, thus shifting more toward a markerless approach to AR – or on *natural feature-based* markerless tracking as opposed to the *artificial marker-based* methods described above. These kinds of AR systems attempt to learn the 3D geometric structure of a scene, such as a crafted Computer-Aided Design (CAD) model (e.g. Comport *et al.*, 2003; Pressigout & Marchand, 2006), or a more complex commercial textured 3D model (e.g. Reitmayr & Drummond, 2006) of some objects from the original environment with distinguishable features. Similarly to the 2D image markers, the systems will track the visual features of the given models to estimate the positional relations between the users and the environment, but in this case, setting up the reference coordinate system to describe the spatial position of each detected feature can become a problem. If the given models were based on an artificial object, such as a cuboid furniture or a building which has an explicit and angular polygon structure and were relatively easy to create by hand (*i.e.* using 3D modelling software such as *Google Sketchup*, *Autodesk 3dsMax* or *Blender*), then the spatial information of visual features can be identified through the model construction. However, a regular polygonal structure is not available in all situations, especially in the natural environments, which may be devoid of any man-made objects. Most of the structures in the real world are too

complex to manually create reference models for them with a high geometric accuracy. To identify the spatial position of the features will be very difficult.

In order to deal with the reference model training issue for markerless tracking, unsupervised (automatic) 3D reconstruction – techniques to restore the shape and appearance of real objects by computer – is introduced to AR systems for creating reference models from the original environment. The 3D scanning technologies can be divided into two types: *contact* and *non-contact* (Curless, 1999). Contact solutions require probing the object through physical touch which may modify or damage object during the scanning. In contrast non-contact solutions (X-ray, synthetic aperture radar, photogrammetry, laser scanning) are preferred in recent reports from the 3D reconstruction community. The non-contact solutions can be further divided into *active methods* and *passive methods* (Remondino & El-Hakim, 2006). The passive methods typically refer to image-based modelling, where the vision sensor, such as *monocular* or *stereoscopic* camera, is used to provide a set of digital images or video that requires further processing to derive the 3D object coordinates (Hassner & Basri, 2006; Wu, 2013), and is characteristic of portable and low-cost. The methods to recover 3D information from 2D images are described in Sections 2.2.3 and 4.2. In contrast, the active methods directly provide range data containing the 3D coordinates necessary for the model generation. This is based on using quite costly active sensors, but can provide a highly detailed and accurate representation of most shapes. The range data can be acquired from structured light (Geng, 2011), time-of-flight lasers (Levoy *et al.*, 2000; Cui *et al.*, 2010) or even ultrasound (Jordan *et al.*, 2009), which actively scan the object. Thus the process is also known as *3D scanning*. Since most of the active sensors are

costly, heavy and quite complicated to use or modify, the combinations of image- and range-based sensors have appeared, such as Microsoft's *Kinect* and the *Asus Xtion* (a.k.a. *RGBD cameras*) which are portable and directly provide both colour images and dense depth information at the same time, at a relatively low price point. The RGBD-based modelling is presented in Henry *et al.* (2010) and Endres *et al.* (2014). By taking account of the above discussion, the single vision-based (monocular camera) and RGBD-based (*Kinect*) methods are used in the present research for constructing the reference spatial structure models of the target environment in which to perform AR. Both methods of the reconstruction are presented in Chapter 4, with the utilisations of the reconstruction results for reference template (*i.e.* the reference model or map to track) and database training in Chapter 5.

Some AR research projects and applications also developed hybrid tracking methods which mainly combined computer vision and other sensing technologies together to complement each other. Reitmayr & Drummond (2006) present a good example of a hybrid method which is based on several techniques: GPS sensing for initialising an approximate range, edge-based tracking for accurate localisation, gyroscope measurements for dealing with fast motions, gravity and magnetic field measurements for eliminating drift, and previous frame storage to act as a reference with online frame selection to re-initialise automatically after dynamic occlusions or failures. Although the accuracy of AR user tracking was improved, the hybrid tracking methods required specialised hardware and sensors and fuzzy integration of multiple sensors, which also requires a significant computing resource to deal with them.

2.1.2. AR development frameworks

Krevelen & Poelman (2010) explain that prototyping *frameworks* of AR systems is a process which supports the supplying easy integration of AR devices and the quick creation of *user interfaces (UIs)* which are developed independently from their final applications. From this perspective, one can hypothesise that there are actually two types of end user who will be engaged in the development of an AR system. One type accounts for the “user” mentioned thus far who will be the ultimate users of the developed. The other type is the “developer”, who has clear application design objectives and desires to implement them by using AR system frameworks. Many research projects reported in the past have specific application foci when attempting to solve practical AR problems, such as those AR systems introduced in Chapter 1. The developers of these examples usually design and develop the applications directed towards the final users, as shown in Figure 2-7, and take no account of designing a general user interface of the frameworks for the higher-level developers (since they are lower-level developers themselves).

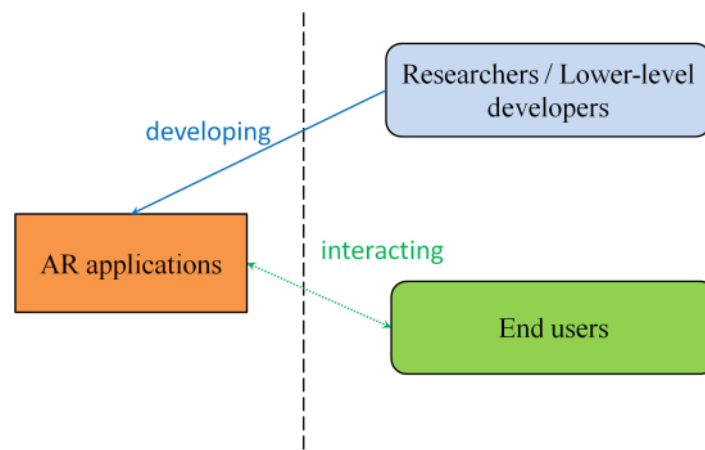


Figure 2-7: The relationships between lower-level developers – end users of an AR system.

Nevertheless, to bring AR technology out of the lab for more people, developing the

“user experience” of a system should not only concentrate on the final users but also those potentially higher-level developers who may not have a professional background (like the computer vision knowledge reviewed in Section 2.2) but wish to apply AR technology to their own developments (as shown in Figure 2-8). This is also known as end-user development (Lieberman *et al.*, 2006), and is supported by several AR SDKs such as above-mentioned Wikitude (Figure 2-2) , Metaio (Figure 2-3), ARToolkit (Figure 2-4), Vuforia (Figure 2-6), and so on. Different AR SDKs support development of AR applications on different platforms (e.g. *iOS*, *Android* for mobile device or *Windows*, *Linux* for PC) and provide several approaches (*i.e.* sensor-based, marker-based and 3D model-based) for the developer’s specific requirements. The developer typically interacts with the SDK framework to specify the reference to be identified or tracked, and then inserts the desired virtual information to be displayed with respect to the chosen reference in the real world. The reference can be the quantised geographical position and orientation for geo-based user tracking, the 2D image pattern or barcode for planar marker-based user tracking, or the CAD data for 3D model-based user tracking. Most of the existing vision-based development frameworks allow developers to customise the 2D markers. Some of them (e.g. Vuforia) even support developers to scan 3D objects from the real world as the reference, but require relatively harsh lighting and clear conditions of the environment for scanning⁵. Metaio also supports an easy, instant 3D mapping and tracking online for small workspace. A comparative study of AR SDKs are presented by Amin & Govilkar (2015).

⁵ **Vuforia Object Scanner:** <https://library.vuforia.com/articles/Training/Vuforia-Object-Scanner-Users-Guide>

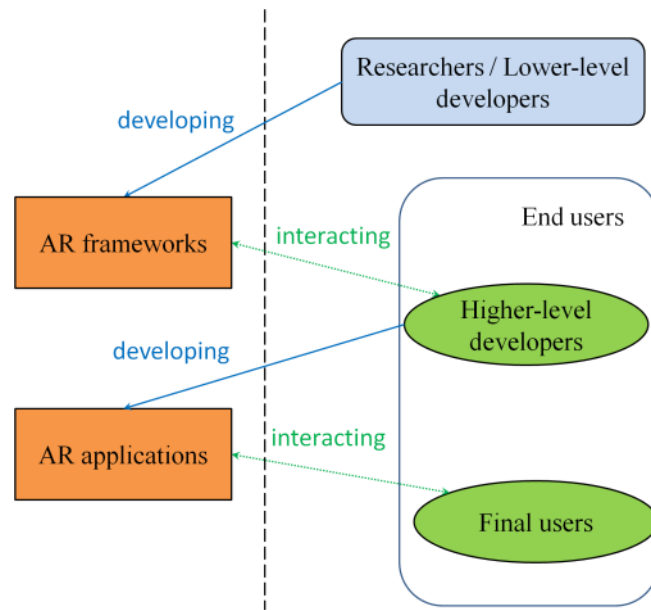


Figure 2-8: The relationships between lower-level developers – higher-level developers – final users of an AR system.

2.1.3. AR system evaluation

As described above, the implementation of an AR application or system involves various internal techniques and several existing research studies have focused on one or two such techniques in particular, such as the important tracking issue mentioned in Section 2.1.1, in which the accuracy is sometimes considered as a “criterion” for measuring the robustness of an AR system (e.g. Reitmayr & Drummond (2006); Carozza *et al.* (2014)). AR tracking is actually a 3D pose estimation problem and can be evaluated by comparing to the ground truth map data. Similar evaluation metrics for 3D mapping are discussed in Section 2.3.3 below. However, as well as the technique adopted, the experience of AR system end users should not be ignored. As a *Human-Computer Interaction (HCI)* system, one of the basic questions that needs to be solved is exactly **who** are the users and what are their needs (Fjeld, 2003). Dünser & Billingham (2011) also point out that, in order to bring the technology out of the

research labs and into people's everyday lives, strong user-centred system designs and processes are required, and the AR systems need to be evaluated with actual users. Nevertheless, they also indicate that there are only a few suitable formal methods for evaluating AR interfaces in comparison to other technologies, since the researchers prefer to design and create one-off prototypes for their specific aim instead of a general framework for an AR system. In spite of this, Kostaras & Xenos (2009) summarised that most of the individual studies on the usability of specific AR applications were based on existing methods of *Usability Evaluation*, which can be divided into the following categories:

- 1) *Inquiry methods (user report)*: specifically *Questionnaires* and *Interviews*, which can gather subjective data that relate to the opinions and preferences of users on different factors, but are not, on their own, sufficient for obtaining secure conclusions.
- 2) *Inspection methods*: the usability of the user interface is inspected by an evaluator (expert) instead of the actual end users, which can be less time consuming than the other categories (Nielsen, 1994b). The typical inspection methods include *heuristic evaluation* (Nielsen & Molich, 1990) in which the evaluators examine the user interface according to 10 standard usability principles given in Nielsen (1994a) and *cognitive walkthrough*, which focuses on how well a novice user is able to accomplish pre-defined tasks with the interface without prior training (Rieman *et al.*, 1995).
- 3) *Testing methods*: real users are involved in these methods to measure the extent to which the interface meets its intended purpose and satisfies the target user population. The factors affecting testing methods are presented in Figure 2-9.

The typical testing methods include *Think Aloud Protocol* where the users will vocalise their thoughts, feelings, and opinions while interacting with the interface (Minati *et al.*, 2009); *Co-Discovery Method*, where the users will attempt to perform tasks together while being observed (Minati *et al.*, 2009); and *Laboratory Evaluation* which is performed in a controlled environment and which mimics a real-life scenario(Alshamari & Mayhew, 2008).

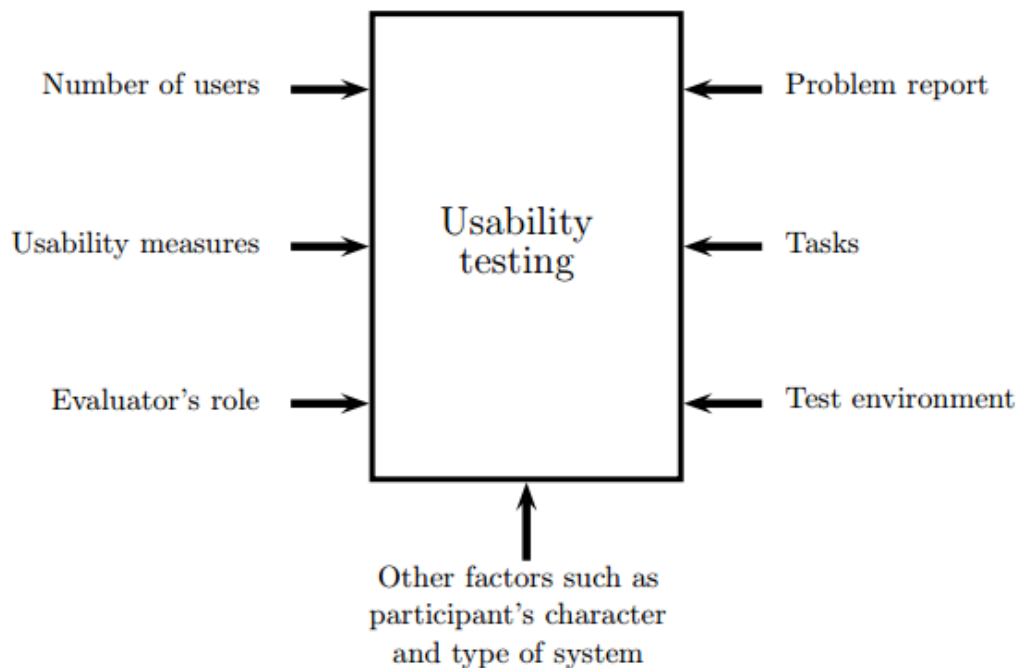


Figure 2-9: The factors affecting usability testing methods.

(Alshamari & Mayhew, 2008)

Martinez & Bandyopadhyay (2014) chose four evaluation methods – cognitive walkthrough, heuristic evaluation, laboratory observation and questionnaire administration – to evaluate a project called “Augment – 3D Augmented Reality”, to

determine which methods would be more suitable in case of AR interfaces based on comparing the evaluation results obtained. However, they discovered that a combination of methods delivers the best overall results, as not all design problems were found by individual methods alone. They suggest that design guidelines for AR evaluation based on their results should involve:

- 1) Combining one usability inspection method with one usability testing method to obtain a reliable outcome.
- 2) Using more than one expert in the inspection methods.
- 3) Ensuring, in the case where a questionnaire-based method is chosen, that the number of users who complete it should be large enough and contain a variety of participants, including AR experts.

2.2. Computer vision methods in AR

As mentioned earlier, there are various technologies involved in visual AR system development, particularly with regard to four major processes: sensing, tracking, registration and interaction. Most of these technologies are inextricably linked with *Computer Vision (CV)*. Vision is one of the most important senses that allow humans to perceive and understand the world surrounding them. The eyes will sense light from the environment and the brain will interpret the meaning of the stimuli arriving at the retina of the eye. The original idea of computer vision technologies was focused on attempts to duplicate the effect of human vision. The optical sensors (*i.e.* cameras) are used to obtain the visual information which will be depicted in the form of two-dimensional (2D) images, which can then be perceived and understood electronically by computer

(Sonka *et al.*, 2014). CV-related research covers a wide range of applications, including industrial inspection and control processes (Labudzki & Legutko, 2011), robot guidance and autonomous navigation (Courtney *et al.*, 1984; Mataboni, 1992; DeSouza & Kak, 2002), detection and recognition of the human face (Hsu *et al.*, 2002; Yang *et al.*, 2002), gestures (Wu & Huang, 1999; Patsadu *et al.*, 2012) or other objects (Ullman, 1996; Torralba *et al.*, 2003), image indexing and retrieval (Idris & Panchanathan, 1997; Sivic & Zisserman, 2003), three-dimensional (3D) reconstruction (Hassner & Basri, 2006; Ma *et al.*, 2012), and so on. Most of these applications, including vision-based AR systems, achieve the major task of understanding the **workspace** through the input data and identification of the **camera state**.

Generally, the image processing techniques used in visual AR systems include camera calibration, image acquisition and representation, template training and matching, spatial information recovery and augmentation registration. Specifically, the proposed framework aims to help higher-level developers to create and register perspective AR content (e.g. virtual 3D models) accurately based on a **specific** place. The requirements based on a real use case scenario of an indoor aquarium (the **UK's National Marine Aquarium**⁶, NMA, located in Plymouth) described in Section 6.2 with the proposed AR framework are briefly listed in Table 2-1.

⁶ **The National Marine Aquarium:** <http://www.national-aquarium.co.uk>

Table 2-1: The requirements of the proposed AR frameworks

Development framework for higher-level developers		Use case scenario of the National Marine Aquarium (developers)
Input device	Low-cost visual sensor or depth camera	
Target environment	Both indoor and outdoor	Indoors
Based on specific place	Yes	Indoors function zone in front of the <i>Eddystone Reef Tank</i> ⁷
Request on accurate AR registration	Yes	Registering 3D marine animal models and animations to the specified location inside the function zone seamlessly
Tracking type	Markerless-based	Marker-based or markerless-based
Request on real-time processing	No	
Developed application for final end users		Use case scenario of the National Marine Aquarium (visitors)
Operating platform	Computer or mobile devices	Mobile devices (smartphones or tablet computer)
Input device	Integrated RGB camera of the platform device or webcam	Integrated camera of the mobile device
Output device	Monitors of the platform device	Display screen of the mobile device
Request on visiting specific place	Yes	The function zone of the Eddystone Reef Tank
Request on real-time processing	Yes	

⁷ **Eddystone Reef Tank:** One of the NMA’s largest tank features, based on the ecosystems found near the Eddystone Lighthouse Rocks, 19km south of Plymouth – see:

<http://www.360imagery.co.uk/virtualtour/leisure/nma/>

In order to select the appropriate technologies to meet the requirements above, a number of typical techniques and algorithms are reviewed and discussed in the following subsections, and the design decisions are given at the end of this section.

2.2.1. Camera and camera calibration

A camera is a form of optical equipment which captures the reflected light from the environment to achieve similar functions to those of the human eye. The sensed image data can take the form of individual photographs or image sequences constituting videos. In computer vision, most single-lens camera devices can be simplified into a monocular pinhole camera model (see the dashed box in Figure 2-10) in image processing (Hartley & Zisserman, 2003). In contrast to the monocular camera, a stereoscopic camera has two or more separate lenses to simulate the binocular vision of human and to capture 3D images. However, technologically a stereoscopic camera can also be depicted by a set of monocular camera models where each of its lenses is replaced by an individual pinhole camera.

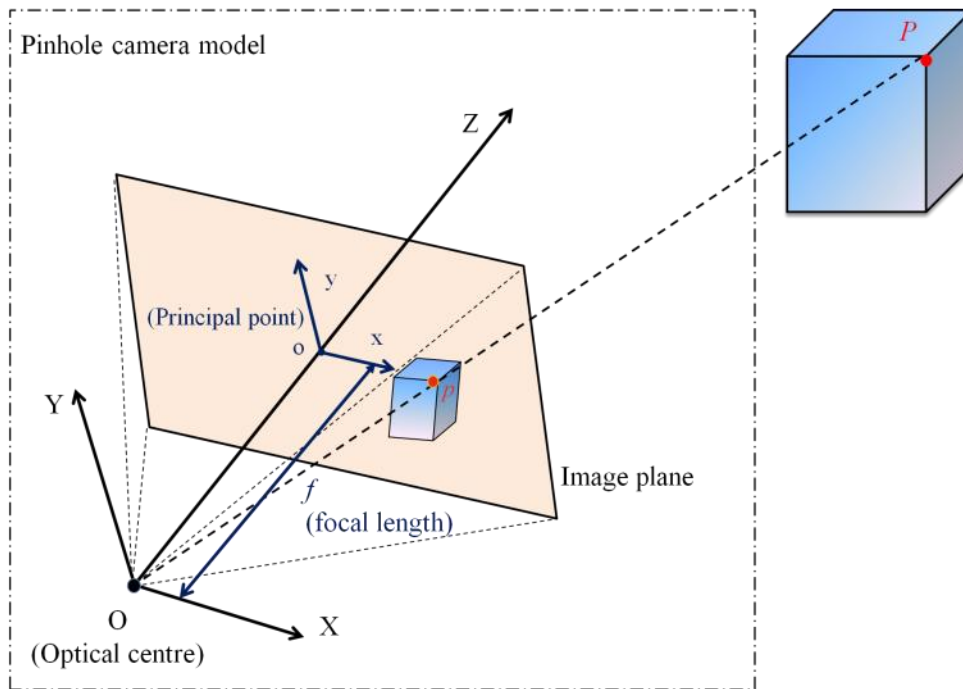


Figure 2-10: The perspective projection procedure of a pinhole camera model where upper case X,Y,Z denote camera coordinates and lower case x, y denote image coordinates.

The ideal pinhole camera model mainly consists of an *optical centre* (a.k.a. projection centre) and an image plane, which defines a 3D reference frame to express the spatial relationships between the camera and the objects around it. This local reference frame is called *camera reference frame*. As can be seen in Figure 2-10, the coordinate system has its origin at the optical centre, the X-Y plane parallel to the image plane, and the Z-axis along the optical axis perpendicular to the image plane. The location of the image plane can be described with the shortest length to the optical centre – known as *focal length*, and the intersection point where the optical axis joining to the image plane, referred as image centre or *principal point*. Figure 2-10 also presents the projection procedure from a 3D object (with respect to the camera coordinate system) to the 2D camera image plane: by looking at the 3D point P , the reflected light from P going through the image plane and arriving at to the optical centre, and the intersection p on

the image plane is the projected image point of the P .

2.2.1.1 Camera calibration

In practice, real camera devices perform perspective projection to map a 3D scene to 2D images, which is controlled by *intrinsic camera parameters*. As the name indicates, they are intrinsic properties of the camera devices. The pictures taken using the same camera share the same intrinsic parameters. The intrinsic camera parameters include the focal length and the principal point mentioned above, and additionally the lens distortion which is caused by lens imperfections or intentionally introduced by a fisheye lens for creating a wide panoramic or hemispherical image (Horenstein, 2005). Camera calibration refers to the process of finding these parameters, which is important to the visual AR applications in the quest to achieve the best user experience, since the AR process will insert the virtual objects to the scene of the input images and project the augmentations on the user display screen correctly with these intrinsic camera parameters (Baggio, 2012).

The intrinsic camera parameters can be provided by the manufacturer or computed through a known target for calibration purpose (e.g. a chessboard pattern plane shown in Figure 2-11 (Heikkila & Silvn, 1997; Zhang, 2000)). There are several implementations for processing camera calibration, such as the camera calibration toolbox in *Matlab* (Bouguet, 2004) and the *camera_calibration* sample code provided by *OpenCV* library (Bradski, 2000). The main idea of camera calibration is to take several images from different viewports of a set of annotated 3D points to determine their projected points on the images. Specifically if *OpenCV* is used for calibration, then

the 3D points will be extracted from each inner corners of the black-white square within the chessboard pattern. A pattern reference frame is defined. Since the pattern is flat, the Z axis of the reference frame is assumed to be perpendicular to the pattern plane and all points on the pattern are located at $Z = 0$. The X and Y axes are assumed to be aligned with the grid of pattern thus the 3D positions of corner points can be identified by giving the actual size of the square.

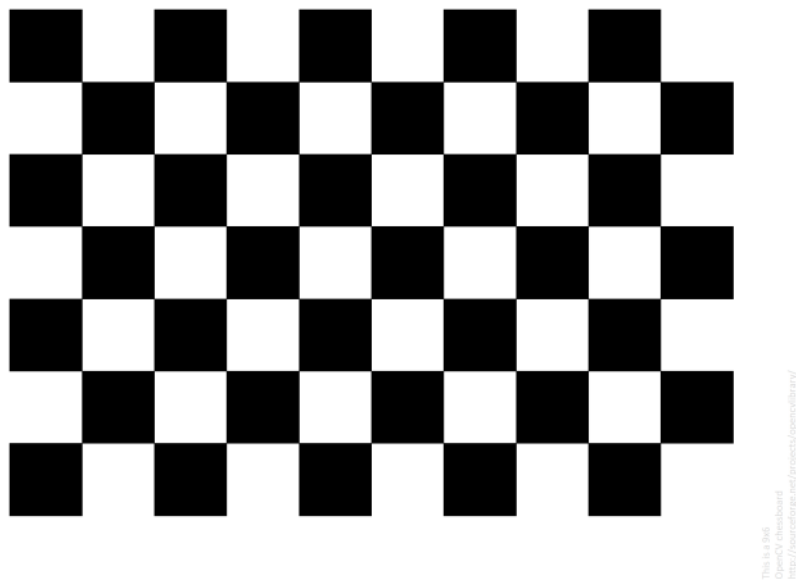


Figure 2-11: Black-white chessboard pattern with size of 9x6 provided by OpenCV library.

The basic principle of camera calibration involves taking known 3D points, measuring the 2D image points and finding the intrinsic camera parameters from those correspondences. The mathematical details are described in Section 3.2.3.

2.2.1.2 Kinect sensor calibration

In the present research, the Microsoft Kinect 1.0 is specifically used to obtain RGBD input data (unless otherwise noted, all references to the "Kinect" in this section concern the Microsoft Kinect 1.0 product). The Kinect combines a monocular colour camera, an

Infra-Red (IR) camera and an IR speckle projector to provide traditional colour images and depth information for each pixel at a certain frame rate (*i.e.* 30 fps for Kinect). The sensor data are read and stored as colour and depth images by utilising an Open Source software framework *OpenNI*. The calibration of the colour camera is quite similar to the approaches described in Section 2.2.1.1. The depth information is determined by using the IR camera and speckle projector as pseudo-stereo pair, and it can be calibrated by detecting a chessboard in the IR image too, which has described in Burrus (2012) and Reimann (2015). In fact it is not necessary to calibrate a Kinect by hand since the *OpenNI* camera driver provides default intrinsic camera models with reasonably accurate focal lengths. The lens distortion is ignored due to the low-distortion lenses used by Kinect. In addition to the intrinsic parameters calibration mentioned so far, it should be noted that the colour camera and the depth camera are generally working concurrently, but the acquired videos may slightly out of sync. The asynchronous colour-depth frame pairs can be dropped by checking the difference between their timestamps (measured in microseconds by OpenNI). It should also be noted that there is a space displacement between the lenses of the colour and depth cameras, thus the imaging ranges of the cameras are a bit different. An example is given in the upper of Figure 2-12. This can be solved by mapping depth pixels with the corresponding colour pixels through a registration process which is supported by some devices, like Kinect, and the calculations can be performed in hardware and accessed through the OpenNI API. The registration result is shown in the lower of Figure 2-12. Alternatively a custom calibration of RGBD camera can also be performed for achieving rigorous results. The entire set of calibration methods are presented in Herrera *et al.* (2011) and Zhang & Zhang (2014).



Figure 2-12: the unregistered depth image (upper) and the registered depth image (lower) with their corresponding colour image captured by Kinect.

2.2.2. Visual features

In computer vision, *visual features* refer to the elementary characteristics of the content in images, which include colour, texture, shape, and so on. As stated in Dong (2013), most image processing systems, such as those underpinning image annotation or retrieval systems, use visual features to match and recognise objects from images, and the performance of such systems are heavily dependent on the features used for image representation. These can be categorised into *global features* and *local features*. Through the description of Lisin *et al.* (2005), the global features describe an image as a whole and have the ability to generalise an entire object with a single vector, while the local features represent the image patches computed at multiple points in the image.

Generally the global features are not very robust for image matching tasks since partial changes, such as occlusion and clutter, occur in the images and can affect the global description of an image. Instead, the local features extracted from multiple interest points of an image will describe the image by segmenting it into several individual patterns with no interference with each other. These patterns are very useful in determining the similar parts from different images. The following subsections will focus on introducing different local features and their corresponding detection, representation and matching methods.

2.2.2.1 Feature detection

Detecting visual features is commonly regarded as a low-level image processing operation and is performed as the first processing stage in many vision-based systems, especially for image retrieval, classification and object recognition. Since the features are numerical representations of images, they can be used for comparing different images and calculating the similarity. The standard of comparison depends upon the method used for feature representation. A very basic form of feature representation is the *colour histogram* (Swain & Ballard, 1991). The pixels on images can be represented by different amounts on some colour axes (e.g. red, green and blue in RGB space). Each colour axis is viewed as a channel, and each channel has a related intensity value. If the images are monochromatic, then a pixel can be represented by a single intensity value, known as *greyscale*. The colour histogram represents the distribution of colour in an image by accumulating the number of pixels at each different intensity value found in that image. For example, an 8-bit RGB image allocates three bits for red ($2^3=8$ possible intensity values), three bits for green ($2^3=8$ possible intensity values), two bits for blue

($2^2=4$ possible intensity values), so there are totally 256 ($8*8*4$) discrete possible colour for each pixel on the image. Stricker & Orengo (1995) mention that, although using classical colour histogram results in the retrieval of the images whose colour compositions are similar to the given query image, due to the lack of texture and geometric information contained in images it is difficult to identify a specific object from images. For this reason, the representation methods of texture and shape features have been considered, including *edges*, *corners* and *blob*.

Edges are defined as sets of pixels in the image that border two homogeneous regions of different intensities (Gonzalez & Woods, 2008). Thus, edge points usually have a strong gradient (vector denoting the maximum rate of change of image intensity) magnitude, which can exhibit various degrees of discontinuities in image intensity dependent on the shape of edges in images. The commonly used edge detection algorithms include Canny (1986), Roberts (1963), Sobel & Feldman (1968) and Prewitt (1970). These are first-order derivative methods that rely on the computation of image gradients and searching for local directional maxima of the gradient magnitude. Second-order derivatives methods, such as Marr & Hildreth (1980), are also used for feature detection by searching for *zero-crossing* (the point where the sign of function changes) in the second derivative instead of the peaks of the first derivative, which produce thinner edges (fewer edge points). The edge detection methods mentioned above have been classified and evaluated in Sharifi *et al.* (2002). Edge detection aims to determine the location, orientation and strength of each edge point such that a two-dimensional image pattern can be represented by a set of one-dimensional curves. Thus edge detection is always performed as an early stage in image analysis and pattern recognition. However, for

researches into processes such as those involving robot navigation, 3D reconstruction and the vision-based tracking task in AR systems, there is a desire to obtain an understanding of surrounding environment or self-locating the camera and its carrier (e.g. a vehicle, a robot or a human) through a sequence of images. For this reason, the relationships between the images need to be identified first, and this requires extracting robust (*i.e.* invariant to various changes in environment), distinctive, and repeatable local features from each image, which will then be used for image matching. In these cases, edge features are not a good choice for tracking. Firstly the homogeneous contour of edges is locally ambiguous, which means that the physical motions tangential to edge are indistinguishable and cannot be measured. This is known as *aperture problem* (Hildreth & Ullman, 1982). Furthermore, although most edge filters are brightness invariant, they are typically variant to other changes, such as colour (Koschan & Abidi, 2005) and scale (Lindeberg, 1998a).

Corners (a.k.a. interest points) are a more suitable choice for tracking. They are defined as the intersection of two edges, or points, for which there are two dominant and different edge directions in a local neighbourhood of the point (Muda *et al.*, 2014). Thus, corner points will have a strong gradient in two distinct directions. Harris & Pike (1988) concentrated on the extraction and tracking of corners in their work, since corner features are discrete, reliable and meaningful. However, the lack of connectivity of these feature points becomes a limitation for obtaining higher-level descriptions of surfaces and objects. Hence the authors decided to introduce a combined corner and edge detector, known as *Harris Detector* (Harris & Stephens, 1988): given an image $I(x, y)$, assume a “window” $w(x, y)$ located at position (x, y) is being shifted by (u, v) .

The difference between the original and the shifted window can be expressed as:

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2 \quad (2.1)$$

If (x, y) is a corner point, it means a significant change when the window is being moved in any direction. Hence the locations with large $E(u, v)$ need to be found out.

Since $I(x + u, y + v)$ can be approximated by a Taylor expansion $I(x + u, y + v) \approx I(x, y) + uI_x(x, y) + vI_y(x, y)$ where I_x and I_y are the partial derivatives of I , $E(u, v)$ can be approximated in matrix form as:

$$E(u, v) \cong [u \quad v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad (2.2)$$

where M is a 2x2 matrix computed from image derivatives, known as *Harris matrix*.

$$M = \sum_{x, y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad (2.3)$$

The corner response is measured by eigenvalues λ_1 and λ_2 of M . Therefore the following inferences can be made based on the magnitudes of the eigenvalues:

- 1) If both λ_1 and λ_2 are near to zero, it means a “flat” region.
- 2) If one of λ_1 and λ_2 is near to zero, another has large positive value, it means an edge point.
- 3) If both λ_1 and λ_2 have large positive values, it means a corner point.

Since the eigenvalue decomposition of the matrix M is computationally expensive, Harris & Stephens (1988) suggest that the corner response can be defined with the trace and determinant of M as following:

$$R = \det(M) - \kappa(\text{trace}(M))^2$$

$$\det(M) = \lambda_1 \lambda_2 \quad (2.4)$$

$$\text{trace}(M) = \lambda_1 + \lambda_2$$

Therefore the windows that have a response value R greater than a certain value are corners.

Another widely used corner detection algorithm is the *Smallest Univalue Segment Assimilating Nucleus (SUSAN)* (Smith & Brady, 1997), which is realized by a circular mask with a nucleus. According to Chen *et al.* (2009), the Harris Detector is superior to the SUSAN detector in terms of stability, anti-noise ability, and complexity. But as for timing results, Rosten & Drummond (2006) reported that the SUSAN was more efficient than the Harris. In addition, Rosten & Drummond (2006) propose a much more efficient corner detection algorithm – *FAST (Features from Accelerated Segment Test)*, which is based on a relaxed version of the SUSAN corner criterion, namely an accelerated segment test which uses a circle of 16 pixels (as shown in Figure 2-13) to classify whether a candidate point is actually a corner.

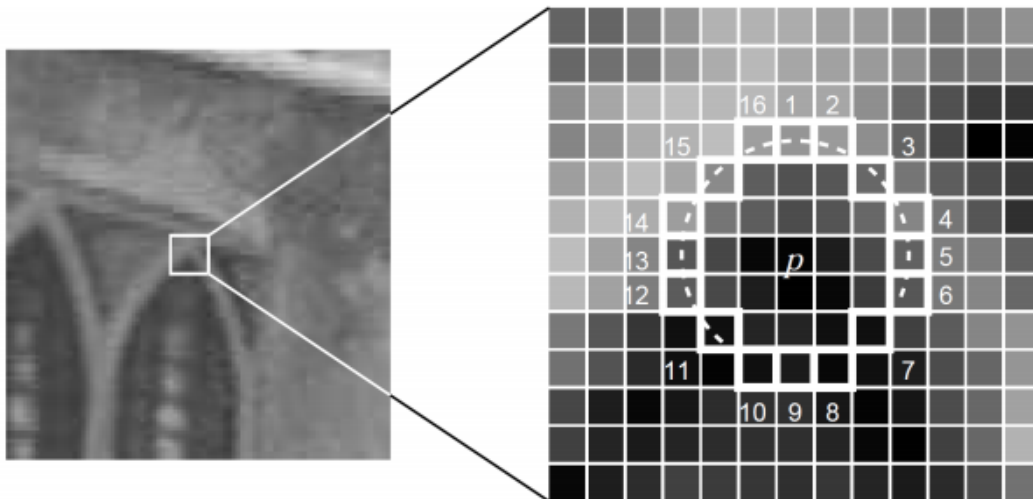


Figure 2-13: n point segment test corner detection in an image patch.

(Rosten & Drummond, 2006)

If there are n contiguous pixels having similar brightness (all brighter than a given threshold or all darker than another threshold) to the nucleus within the circle, where the suggested value of n is between 9 and 12, then the pixel under the nucleus is considered to be a corner. By applying machine learning techniques, the computation time and resources of FAST can be more optimal. An improvement detector FAST-ER (Enhanced Repeatability) is proposed in Rosten *et al.* (2010), where they train a decision tree with an optimised structure suitable for points with high repeatability to compare the pixels and decide whether the point of nucleus is a corner or not.

Corner features are basically invariant to translation, rotation and brightness, and relatively efficient (especially the FAST detector is very fast), but they are still not invariant to scale and are sensitive to high-level noise. To solve the scaling problem, blob features are taken into account. Relative to a corner feature which is considered as an individual interest point, a blob feature refers to an interest region, in which some

properties are constant or vary within a prescribed range of values (*i.e.* all the points in a blob can be considered in some sense to be similar to each other). Based on this fact, a blob feature can be treated as a *keypoint* to track, which is located at the centre of the region. One major technique for blob detection is based on local extrema (Pandey *et al.*, 2014), such as *Laplacian of Gaussian (LoG)* operator which usually results in strong positive responses for dark blobs (on a light background) and strong negative responses for bright blobs (on a dark background) (Lindeberg, 1998b). Lowe (1999) applied an LoG filter within different scale spaces to detect scale-invariant feature candidates in their well-known *scale-invariant feature transform (SIFT)* algorithm, which includes a feature detector and descriptor. The SIFT detector searches and identifies stable features across multiple scales from the convolution of a Gaussian kernel (at different scales) of the input image. Specifically, given an input image $I(x, y)$, the scale space representation of the image at a certain scale σ defined by $L(x, y, \sigma)$ is obtained by convolving the image by a variable scale Gaussian kernel $G(x, y, \sigma)$ where

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (2.5)$$

and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2.6)$$

Since the computation of LoG operators is time consuming, Lowe (2004) improved the computational performance of SIFT by an efficient approximated filter – *Difference of Gaussian (DoG)* – which can be computed from the difference of two nearby scales separated by a constant multiplicative factor k :

$$\begin{aligned}
 D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\
 &= L(x, y, k\sigma) - L(x, y, \sigma)
 \end{aligned}
 \tag{2.7}$$

Thus an *image pyramid* scheme is applied to produce the DoG, in which the scale space is separated into several *octaves*. In each octave the original image is repeatedly convolved with Gaussians by different scale parameter σ to produce a set of scale space images and the adjacent smoothed images are subtracted to produce the DoG, as shown in Figure 2-14.

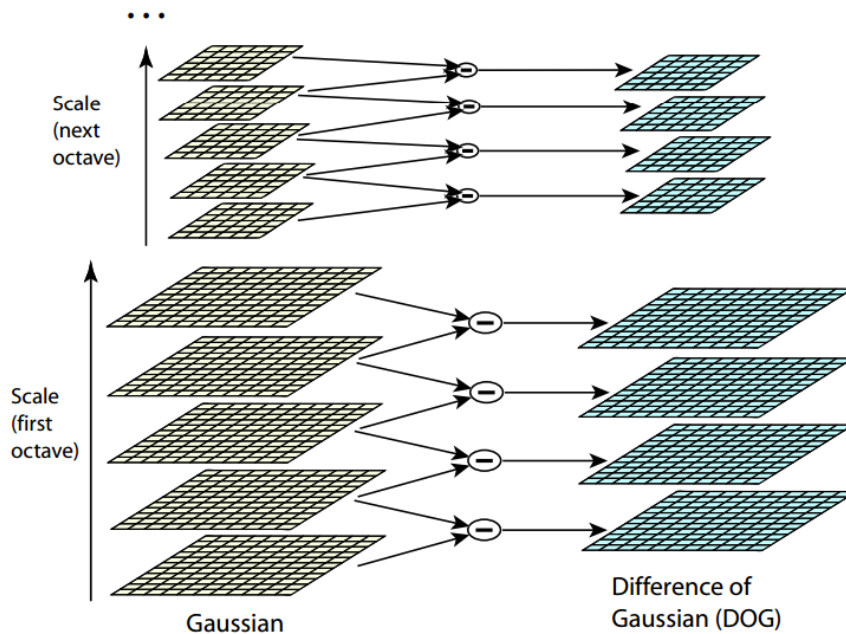


Figure 2-14: Lowe (2004)'s Pyramid Scheme.

After that the smoothed images are down-sampled by a factor of 2 to start the next octave. The potential keypoints are located by detecting maxima and minima of DoG in scale space. Each point is compared to its neighbours in the current image and two adjacent images in the scale above and below, as shown in Figure 2-15.

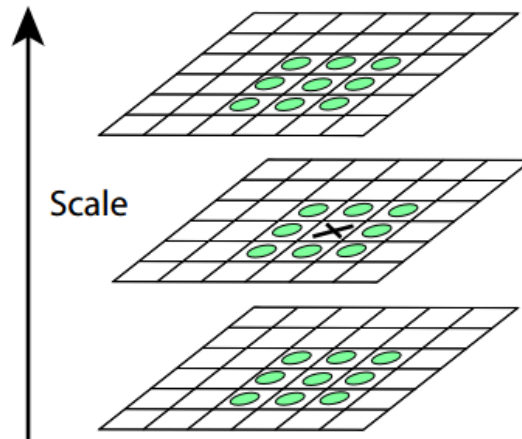


Figure 2-15: Maxima and minima of the DOG images are detected by comparing a pixel (marked with X) to its 26 neighbours (marked with circles).

(Lowe, 2004)

More accurate location of extrema is obtained by Taylor series expansion of scale space. After that the intensity at this extreme is tested and the points with bad contrast are rejected. Since the DoG also has a higher response for edges, the points with strong edge responses need to be removed for the reason discussed above. A strong edge response in the DoG will have a large principal curvature across the edge but a small one in the perpendicular direction. Just like the Harris corner response described in (2.4), the trace and determinant of a *Hessian matrix*, – whose eigenvalues are proportional to the principal curvatures of DoG, are used here for filtering. Assume the quantity $\frac{(r+1)^2}{r}$ is at a minimum when the two eigenvalues are equal and it increase with r .

Whether the ratio of principal curvatures is below some threshold r , or not, is depending on:

$$\frac{\text{trace}(H)^2}{\det(H)} < \frac{(r+1)^2}{r} \quad (2.8)$$

Thus the points that have a ratio between the principal curvatures greater than r are rejected.

Although the SIFT detector has overcome the scale-variant issue and applied the DoG filter for accelerating, it is still far more time-consuming than the corner detectors reviewed above (Chen *et al.*, 2009; Miksik & Mikolajczyk, 2012). A faster and more improved approach based on SIFT is *Speeded-Up Robust Features* (SURF) (Bay *et al.*, 2008). The SURF detector applies a *box filter* for calculating image convolution to approximate LoG in which each pixel after the box filter has a value equal to the average value of all pixel values in a given rectangle in the original image. The box filter can be computed efficiently with the help of *integral images*. Each pixel of an integral image is the sum of all the pixels that were above it and on its left in the original image. For example, the value of the box filter of the rectangle D with integral image value on each corner (denoted by $\mathbf{ii}(\text{corner})$ below) shown in Figure 2-16 can be computed as $\mathbf{ii}(4) + \mathbf{ii}(1) - \mathbf{ii}(2) - \mathbf{ii}(3)$.

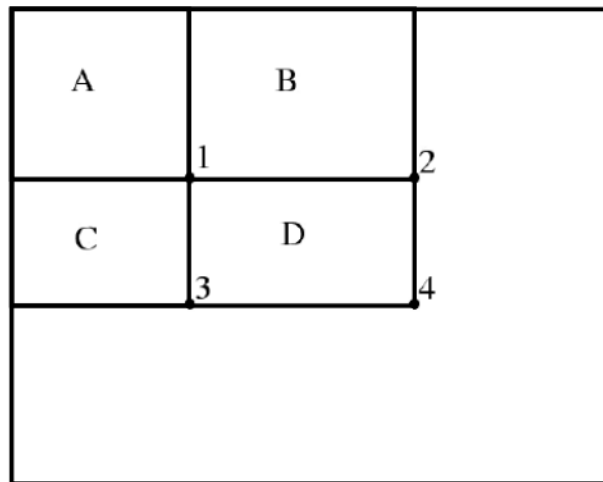


Figure 2-16: the value of the box filter of any rectangle can be computed using integral image representation.

The use of integral images allows the up-scaling of the filter at a fixed computational cost, and SURF does not down-sample the size of images to build image pyramid which saves a considerable amount of time. SURF detection of blobs relies on the Hessian matrix. The determinant of the Hessian matrix is used as a measure of local change around the point and points are chosen where this determinant is maximal. The SURF algorithm also provides descriptor method to represent the keypoints, which will be discussed in Section 2.2.2.2. In addition to translation, rotation and scale invariance of the feature detectors discussed above, Mikolajczyk & Schmid (2004) indicate the affine-invariant challenge against changes in viewport and propose a Harris- / Hessian-Laplace detector where the Harris measure of the determinant of the Hessian matrix is used to select the location of interest points, and the characteristic scale of a local structure is indicated by Laplacian. A comparison of affine region detectors is given in Mikolajczyk *et al.* (2005).

2.2.2.2 Feature representation and matching

Once an interest point with robust local feature is detected in images, a distinctive signature can be computed using the pixel information around the interest point to represent it for subsequent processing (e.g. feature matching). This process is called descriptor extraction. The simplest descriptor could be the raw pixel values in a small patch around the interest point, also called a template. But consider that the local appearance of a feature patch will usually vary from image to image throughout a tracking task. Therefore, more complex descriptions are needed for allowing an invariant match to orientation, scale and even affine changes within the feature patches while still preserving discriminability between non-corresponding patches (Szeliski, 2010).

Distribution-based descriptors are one of the techniques that can be used for describing local image regions, using histograms to represent different characteristic of feature appearance. A simple example is the colour or intensity histogram described above. Park *et al.* (2000) and Won *et al.* (2002) propose a local edge histogram descriptor using the global and semi-local edge histograms generated directly from the local histogram bins and this can be used to measure the similarity between images. One of the most common histogram-based descriptors can be used for corner and region features described so far is a SIFT-like descriptor (Lowe, 2004), which is based on the gradient distribution in the detected regions and is represented by a 3D histogram of magnitude, locations and orientations of the gradient. In this case, the scale of the keypoint is used to choose the Gaussian smoothed image $L(x, y)$ with proper scale. Then the gradient magnitude $\mu(x, y)$ and orientation $\theta(x, y)$ are computed using pixel differences:

$$\mu(x, y) = \sqrt{(\mathbf{L}(x+1, y) - \mathbf{L}(x-1, y))^2 + (\mathbf{L}(x, y+1) - \mathbf{L}(x, y-1))^2} \quad (2.9)$$

$$\theta(x, y) = \tan^{-1}((\mathbf{L}(x, y+1) - \mathbf{L}(x, y-1)) / (\mathbf{L}(x+1, y) - \mathbf{L}(x-1, y)))$$

The gradient histogram has 36 bins covering 360 degrees of orientation and is weighted by a gradient magnitude and Gaussian window. The highest peak in the histogram is taken and any peak above 80% is also considered to calculate the orientation. Thus the orientation is assigned to each keypoint to achieve rotational invariance. The feature descriptor is created from a 4x4 gradient window containing a histogram of 4x4 samples per window in 8 directions, which finally results in a 128-D SIFT descriptor vector, as depicted in Figure 2-17.

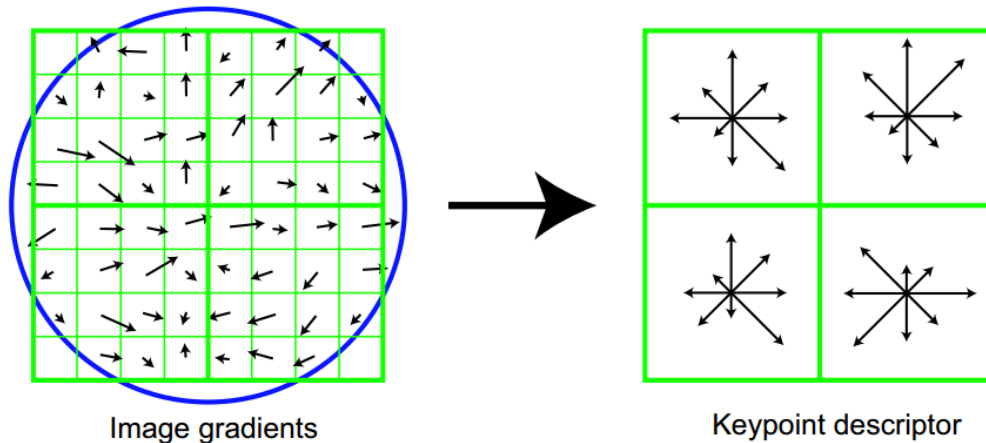


Figure 2-17: The generation of a SIFT keypoint descriptor.

(Lowe, 2004)

Based on the SIFT descriptor, Ke & Sukthankar (2004) developed a PCA-SIFT descriptor which uses principal components analysis (PCA) to normalise the gradient patch instead of histograms. Similarly, Mikolajczyk & Schmid (2005) propose another variant of SIFT descriptor – a gradient location and orientation histogram (GLOH) descriptor – which changes the location grid and uses PCA to reduce the size. SURF

(Bay *et al.*, 2008) also provides a histogram-based descriptor in which the main orientation is obtained from the largest sum value of the *Haar wavelet* response around the keypoint. The neighbourhood is split into 4x4 sub-regions and each sub-region has a 4-D descriptor vector about the sum of values of the responses and the sum of the absolute values of the responses in both x and y orientation. Hence the final SURF descriptor is a 64-D vector which has less dimensions and is faster than the SIFT descriptor. A comparison and evaluation of SIFT, PCA-SIFT and SURF is given in Juan & Gwun (2009) who also point out that choosing the most appropriate method mainly depends on the application. Although the comprehensive performance of SIFT seems better, it is still hard to consider its use in a real-time system due its high time-consuming qualities. A more recent development – *SiftGPU*, implements a fast SIFT method to ease this issue by using the GPU's calculation power. (Wu, 2007).

Binary descriptors are an alternative way to represent features. *Binary Robust Independent Elementary Features* (BRIF) (Calonder *et al.*, 2010) –, like SIFT and SURF, include a detector and a descriptor. This method creates a bit vector from the test responses of pixel intensity on smoothed image patches, and calculates an n -dimensional bit-string which uniquely defines a set of binary tests (n could be of different lengths, e.g. 128, 256, and 512). BRIF descriptor is efficient to compute but does not provide rotation invariance. Rublee *et al.* (2011) propose a new feature, ORB (*Oriented Fast and Rotated BRIF*), by using the BRIF descriptor with the FAST detector, and they enhance both techniques by addition of a fast and accurate orientation component to FAST, making the BRIF descriptor rotation invariant. The corner orientation is measured by the *intensity centroid* (Rosin (1999) cited in Rublee *et al.*

(2011)), which defines the moments (a certain particular weighted average) of a patch as:

$$M_{o_{pq}} = \sum_{x,y} x^p y^q I(x,y) \quad (2.10)$$

for $p, q = 0, 1, 2 \dots$ the centroid is then determined with these moments:

$$\text{Centroid} = \left(\frac{M_{o_{10}}}{M_{o_{00}}}, \frac{M_{o_{01}}}{M_{o_{00}}} \right) \quad (2.11)$$

The orientation of the path is then given by:

$$\theta = \text{atan2}(M_{o_{01}}, M_{o_{10}}) \quad (2.12)$$

where atan2 is the quadrant-aware version of arctan .

Similarly, *Binary Robust Invariant Scalable Keypoints* (BIRSK) (Leutenegger *et al.*, 2011) also uses a binary descriptor dependent on an extended FAST detector, which searches for maxima in a 3D scale-space. *Fast Retina Keypoint* (FREAK) (Alahi *et al.*, 2012) is a novel keypoint descriptor inspired by the human retina. The binary string of the descriptor is computed by efficiently comparing image intensities over a retinal sampling pattern. The applied methods are based on both BRISK (for orientation) and ORB (for sampling pairs).

The local feature descriptors described above can be used to match images and to understand the spatial changes around the matched features. To match simple template descriptors, an error metric such as the sum of squared differences, cross-correlation or normalised cross-correlation can be used to directly compare the intensities in small patches around feature points (Szeliski, 2010). Then the result is checked against a threshold to determine if a match is found. Since a histogram can be represented by a multi-dimensional vector and the similarity of two histograms can be measured by

calculating the Euclidean distance (see Appendix D for detail) between the vectors, to match histogram-based descriptors, the best matching results from a set of descriptors to another set can be found by searching the *nearest neighbour* (known as NN search). To reject wrong matches, a *ratio test* can be performed following a *k*-NN matching by only accepting the first NN if the distance ratio between the first and second NN is less enough – because correct matches need to have the closest neighbour significantly closer than the closest incorrect match to achieve reliable matching (Lowe, 2004). Another mismatch removal methods involve checking *geometric consistency*, for which the detected feature points in one image can be mapped to their point correspondences in another image through a geometric (2D-to-2D) transformation, such as *epipolar geometry*, *homography* and *affine* (Lowe, 2004). Whether comparing template or histogram-based descriptors, the time consumed in calculating the similarity and searching the matches between the vectors is closely dependent on the number of vector dimensions, and the computational cost of the descriptors with high dimensions (e.g. 128 for SIFT) will be higher. A popular method for improving this issue is applying *Fast Library for Approximate Nearest Neighbours* (FLANN) (Muja & Lowe, 2009), which indexes the feature descriptors by using either randomised – *kd-trees* or a *hierarchical k-means tree* to reduce the search time. On the other hand, the binary nature of the binary descriptor allows use of the Hamming distance for matching, which can be implemented efficiently using a bitwise XOR operation followed by a bit count on the result. For large binary descriptor datasets, FLANN is not suitable and conventional “*brute-force*” search is used instead. However, alternative approximation methods for matching binary features faster have recently been presented in Muja & Lowe (2012) and Yan *et al.* (2015).

As previously mentioned, for applications such as robot navigation, 3D reconstruction and vision-based AR, the aims of feature matching are to recover the space structure of the environment and to estimate the camera motion between the images. The related approaches are described in Section 2.2.3.

2.2.3. CV-based localisation and mapping

2D images are considered as the projections of the 3D real world, which can be used for learning the spatial structures of the captured scenes or objects and locating their camera sensor with respect to them. This process is usually called *localisation* and *mapping*. Mapping is a process that attempts to recover the 3D structure of the environment and then construct a *map* to describe the relative position and orientation of each object with respect to a chosen reference frame. The process of localisation tracks the motions of the camera and recovers the camera model of each image related to the constructed map. Note that the phrase ‘camera’ below will not refer to a real camera device, but to the pinhole camera model associated with a specific image (see Figure 2-10). Therefore:

- 1) for a consecutive image sequence, a spatial transformation (including translation and rotation) from any image to another is called *camera motion*;
- 2) for an individual image, the absolute 3D position (*i.e.* translation) and orientation (*i.e.* rotation) of its camera model with respect to the reference frame are called *camera pose*.

These are described as *extrinsic camera parameters*. In computer vision, the localisation

and mapping problem can be solved by employing *Structure from Motion* (SfM) techniques. Or, alternatively, in the robotics community for example, a similar problem is commonly referred to as *Simultaneous Localisation and Mapping* (SLAM). The methods that make use of vision sensors to solve the SLAM problems are known as visual SLAM. Both SfM and SLAM techniques are described and discussed below.

2.2.3.1 Structure from motion

Bolles *et al.* (1987) state that 2D images have an inherent ambiguity: an image only captures two out of three viewing dimensions, and hence an infinity of 3D scenes can give rise to the same 2D image. This means that any individual 2D image cannot supply enough information to enable the reconstruction of its source 3D scene. However, if there are multiple images of the same scene, but these are separated by a distance, including the stereo pairs from the stereoscopic camera or the image sequences acquired by the moving a monocular camera, then the lost depth information may be recovered. The method for recovering structure (*i.e.* depth information) by observing the motion of objects in the image plane is called structure from motion (Ullman, 1979). Since SfM generally takes multiple 2D images as its input, it can be regarded as a monocular vision-based process – as opposed to a multi-view or stereoscopic process. Both *computer stereo vision* (Marr & Poggio, 1976) and SfM technologies aim to extract 3D information or further reconstruct a 3D representation from a scene or an object through digital images. The images used may be acquired simultaneously, as in the case of using a stereo rig, or acquired sequentially by moving a camera relative to the scene (Hartley & Zisserman, 2003). The discrimination between the calibrated stereo rigs and SfM has been interpreted in Baggio (2012): a calibrated rig can be a multi-view camera or a set

of two or more monocular cameras, where the relative motions between the lens/cameras are already known (*i.e.* “calibrated”). Meanwhile the SfM uses a set of images whose camera motions are totally unknown and need to be solved through the process. The SfM algorithms based on multiple images are generally categorised into two methods: *batch* methods and *incremental/online* methods (Kim & Chung, 2003). Batch methods determine the scene structure and the camera poses by using all image measurements simultaneously, and this produces the most optimal estimates. The advantage is that the reconstruction errors can be distributed meaningfully across all measurements. Several batch SfM methods have been summarised in Hartley & Zisserman (2003). Some of these are known as factorisation or factorisation-like methods, which are linear and based on direct SVD (*Singular Value Decomposition*) factorisation of the image point measurements. One of the limitations is that they require every 3D point to be visible in every view hence they are not applicable to sparse modelling problems. More recently Crandall *et al.* (2013) describe factorisation methods to obtain a good initialisation and then apply *Bundle Adjustment* (introduced below) as a final nonlinear refinement step to obtain accurate camera parameters and scene structure. Since the batch-based methods process all images at once, they cannot be used for applications which have strict real-time requirements, such as robot navigation system which needs to plan an action based on immediate information being acquired immediately. Instead, the incremental SfM algorithms will solve the problem progressively with a recursive form. The images are acquired one by one as the process is running, and the program will perform real-time updates to the constructed map after each image is processed. Owing to the fact that the approaches of these two categories share the core techniques of SfM, the remainder of this section will mainly interpreting

the processes of incremental SfM methods.

If the images take the form of rigid scenes or objects, to obtain camera motions and recover the geometric structure, SfM implementations typically attempt firstly to detect robust features from the target scenes. The features need to be observed across multiple consecutive images from the input sequences, then the matched feature correspondences between the consecutive images – which give a set of images feature trajectories over time – will be determined (Dellaert *et al.*, 2000; Corke *et al.*, 2007). The local features reviewed in Section 2.2.2 are usually selected for this aim since they are suited for the matching task. The recovery of the relative camera motions through feature matches between the images and the acquisition of spatial positions of these image points with respect to the world reference frame are introduced in Hartley & Zisserman (2003), Nistér *et al.* (2006) and Ma *et al.* (2012). Generally, for the incremental SfM methods, two images with abundant feature correspondences will be selected as an initial *two-view*. The camera motion between these two images will be recovered first by using methods based on epipolar constraints, or, more specifically, *essential matrix* – an epipolar constraint between each of the correspondences within two calibrated views (*i.e.* the intrinsic camera parameters of both cameras are known) – is decomposed to give the desired camera motion (Longuet-Higgins, 1987). Longuet-Higgins (1987) indicated that an essential matrix could be estimated from eight or more point correspondences by solving a linear equation. Since the essential matrix is rank-deficient, it turns out that seven correspondences are enough to estimate this matrix whilst enforcing the rank 2 constraint in the matrix (Hartley & Zisserman, 2003). Nistér (2004) proposed a more efficient 5-point algorithm that requires finding the roots of a

tenth degree polynomial. Consider that not all correspondences being used here are correct matches; the raw result needs to be checked by *Random Sample Consensus* (RANSAC) (Fischler & Bolles, 1981).

RANSAC can fit a model to data which containing a significant percentage of gross errors. The performance evaluations of several RANSAC implementations are presented in Choi *et al.* (1997). In the case of the essential matrix estimation, RANSAC will randomly sample a subset of correspondences with a certain number k (eight, seven or five for each algorithm described above) to calculate an initial estimation. Then the remaining correspondences will be tested by the initial estimation. The correspondences which satisfy the estimation will be called as *inliers*, otherwise *outliers*. The process of data sampling and fitting will be repeated for several trials, and the sample set with the largest number of inliers will be kept. The ultimate estimation will be recalculated by using the inliers to find an optimal result. To increase the probability of finding a true set of liners from the random sampling, a sufficient number of trials must be undertaken. The required number of trials will grow quickly with the number of sample correspondences k , and the estimations using fewer correspondences are less sensitive to outliers. Therefore the minimum k is preferred (Szeliski, 2010). The well-known improved variants of RANSAC include MLESAC (*Maximum Likelihood Estimation Sample and Consensus*, (Torr & Zisserman, 2000)), PROSAC (*Progressive Sample Consensus*, (Chum & Matas, 2005)), and *Preemptive* RANSAC (Nist ́r, 2005).

Once the essential matrix is determined, the relative camera translation and rotation between the two cameras can be recovered by using SVD, which will be described in

Section 4.2.1.1. Returning to the selected initial two-view, one of these two camera reference frames is generally selected as the world reference coordinate system in order to describe camera poses and the location of spatial points. Then the camera poses of the initial two-view are obtained from their relative motion and are used to triangulate their inlier correspondences into 3D points, known as *triangulation* (Hartley & Sturm, 1997). In brief, the triangulation problem is to find the intersection of the two lines in space. As depicted in Figure 2-10, if a spatial point is visible in an image, there will be a ray in space which connects the point, its projection on the image plane and the camera optical centre. The ray is defined by the camera model, whose optical centre and image plane are defined by the intrinsic camera parameters which have been gained by performing camera calibration (see Section 2.2.1.1). The positional relation between the camera model and the world reference frame is given by the extrinsic camera parameters, namely the camera pose, which have been calculated through the previous methods. Thus, if a point is observed by two cameras, there will be two rays starting from the respective optical centre, through the corresponding image point and finally meeting in the location of the spatial point, forming a “spatial triangle”. With the known pair of rays for each correspondence between two images, the intersected point can be computed. One of the most common methods is linear triangulation (Hartley *et al.*, 1992) which is described in detail in Section 4.2.1.2. The main idea is that from the two views to triangulate, a total of four linear equations can be obtained for solving the 3D position, P , of each correspondence. However, the back-projected rays of a correspondence cannot be guaranteed to intersect due to measurement noise which causes the equations to not be satisfied precisely. For this reason, the four linear equations are converted into the form $AP = \mathbf{0}$ for a suitable 4x4 matrix A and the aim

then becomes to find a nonzero solution for P to minimise $\|AP\|$ (subject to the condition $\|P\|=1$). Hartley & Sturm (1997) discuss two methods – the Linear-Eigen method which uses the SVD or Jacobi’s method for finding eigenvalues of symmetric matrices to solve the problem, and the Linear-LS method which finds a least-squares solution to this problem by using a *pseudo-inverses* method or by using the SVD (the numerical analysis methods used here can be found in Atkinson (2008)). Neither of these two methods are projective invariant and only the Linear-LS method is affine invariant. Hartley & Sturm (1997) describe another kind of method which considers that the correct 3D positions could be chosen by minimising the sum of the squared errors between the measured image positions and re-projected positions (as shown in Figure 2-18). The solution can be found by iteratively adjusting two weight values of the equations until the process converges, in other words, the best weight $1/w_0$ and $1/w_1$ are found, where the w_i is the scale factor of each image and depends on the value of P . This method can be applied to either the Linear-Eigen or the Linear-LS to refine the solution. The disadvantage of this method is that it sometimes fails to converge in unstable circumstances, thus Hartley & Sturm (1997) use their non-iterative optimal Polynomial method as a backup which requires a sufficiently good initialisation.

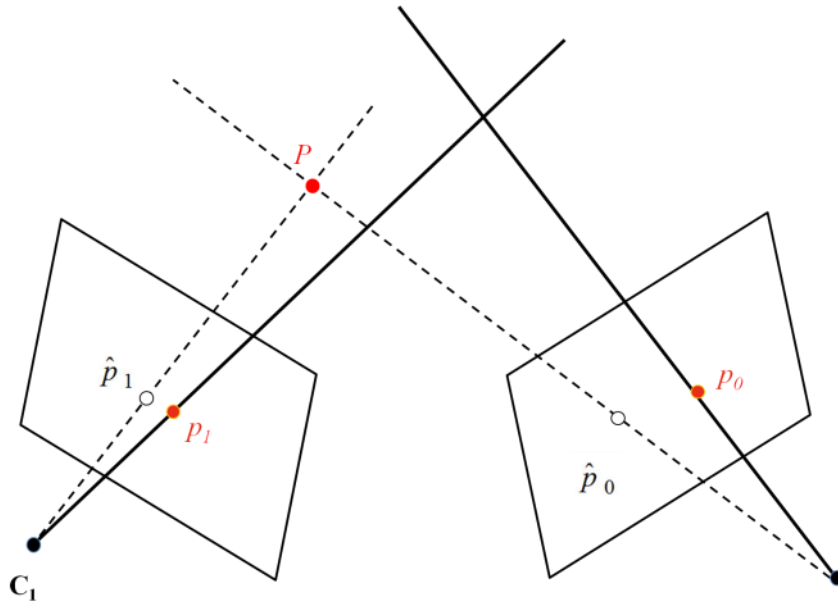


Figure 2-18: The position of P should be chosen by minimising the sum of squared errors between the measured p_i and re-projection \hat{p}_i .

After recovering the first two camera poses and the 3D positions of their inlier correspondences, the camera poses of other image are gradually recovered with respect to the world reference frame by finding 3D-to-2D correspondences between the recovered 3D points and their correspondence in the images. The inlier correspondences between the new processed image and each previous one will be triangulated recursively to recover more 3D points to the world reference frame, which is known as incremental reconstruction. The problem of recovering camera pose by using n sets of 3D-to-2D correspondences is called the *Perspective- n -Point* (PnP) problem (Szeliski, 2010). There are several algorithms for solving this, but all of them require that the chosen point correspondences should not be coplanar. The minimal amount of correspondences, n , needed for solving this problem is 3 (*i.e.* P3P). Fischler & Bolles (1981) noticed that four possible solutions can be found through a P3P equation system. Thus a fourth 3D-to-2D correspondence is required to remove ambiguity and find the

best solution among the four (Gao *et al.*, 2003). Consider the impact of noisy data (*i.e.* outliers), – more correspondences are expected to be involved for solving PnP to make the solution robust. Thus, more preferable methods use a 6-point *Direct Linear Transformation* (DLT) algorithm (Hartley & Zisserman, 2003) to guess an initial pose, subsequently optimising it with the iterative technique which directly minimises the squared re-projection error for the 2D points as a function of the unknown pose parameters by using non-linear least squares (Szeliski, 2010). However, the computational complexity of these methods is relatively high – so much so that it is unlikely to achieve an acceptable real-time performance. Alternatively, there is a faster, non-iterative solution, known as *Efficient PnP* (E-PnP), which can be used for the general cases of $n \geq 3$. The computational complexity grows linearly with n , but is somewhat less accurate than that for the iterative solutions (Lu *et al.*, 2000). In addition, the RANSAC methods mentioned above are also commonly used with a PnP method to filter out the outliers. Apart from being used in SfM, PnP is also a core technique used in markerless AR tracking which makes use of the correspondences between the trained 3D template points and the matched 2D image points to estimate user pose in respect of the real world.

As mentioned in the triangulation stage, by considering the noise effect, the best 3D position solutions of the correspondences between the images are selected based on re-projection error minimisation. In that case, only recovered 3D points are taken account of. However, the process of incremental SfM is recursive. The camera pose estimation is dependent on both observed 2D image features and their 3D correspondences, which have already been recovered by triangulating the correspondences from previous images

with their estimated camera pose. Therefore any error instances will be accumulated during the process. Bundle Adjustment (BA) (Triggs *et al.*, 1999) is commonly used in the SfM reconstruction refinement for producing jointly optimal 3D point coordinates and camera poses. As the name indicates, all of the structure and camera parameters are adjusted ‘in one bundle’ simultaneously by minimizing some cost function that quantifies the model fitting error. BA is always used as the last step of every feature-based 3D reconstruction algorithm, in effect to “polish” a rough reconstruction, as well as for use as an intermediate process in incremental SfM reconstructions to avoid an accumulation of errors (Byröd & Åström, 2010). It is classically expressed as a non-linear least squares problem where the cost function is assumed to be quadratic in the feature re-projection errors, and Lourakis & Argyros (2005) state that *Levenberg-Marquardt (LM)* (Levenberg, 1944; Marquardt, 1963) has become a very popular approach to solve this issue. This is because LM is relative easy to implement and it makes use of an effective damping strategy for quick convergence from a wide range of initial guesses. The LM algorithms involve the normal equations which iteratively linearise the function to be minimised in the neighbourhood of the current estimate, but for a *dense* linear system with a large number of unknown parameters, the computational costs could be very demanding. However, the normal equations matrix used in SfM has a *sparse* block structure owing to the lack of interaction among parameters for different 3D points and cameras (because the projection of a certain point on a certain image does not depend on any other 3D points and camera). Based on this, Lourakis & Argyros (2009) develops a BA package with a tailored, sparse variant of the LM algorithm, called *Sparse Bundle Adjustment (SBA)*, which takes advantage of the zeroes pattern in the normal equations of avoiding storing and operating on zero-

elements, thus gaining tremendous computational benefits. Konolige & Garage (2010) implements an efficient version of SBA, called *sparse SBA (sSBA)*, in which the secondary structure (*i.e.* relations among cameras) is also sparse. By making use of multi-core CPU as well as multi-core GPUs, Wu *et al.* (2011) propose a new inexact Newton type bundle adjustment algorithm that exploit hardware parallelism for efficiently solving large scale 3D scene reconstruction problems. This BA algorithm is then used in an incremental SfM-based 3D reconstruction application, called *VisualSfM* (Wu, 2011; 2013), with their previous work SiftGPU mentioned in Section 2.2.2.2 as well.

Although there is no additional requirement for SfM input data except 2D images, there is a well-known inherent ambiguity problem of monocular vision – an absolute scale of recovered results cannot be determined without appropriate additional sensors. Actually the absolute scale factor is not so important for many applications, but if it is needed (e.g. robot autonomous navigation or multi-area SLAM), then it can be solved by manual calibration or in cooperation with other sensors. For example, Kim & Chung (2003) proposed an omni-directional stereo vision sensor where SFM and stereo functioned in cooperation to remove the disadvantage of each algorithm.

2.2.3.2 Visual simultaneous localisation and mapping

Simultaneous localisation and mapping (SLAM) is a fundamental computational problem in the robotics community where it is necessary to describe a scenario in which a robot is required to move through a previously unknown, mostly static environment, incrementally constructing and updating a map of that environment whilst

simultaneously localising itself with respect to that map. Developments in this field have been applied to both indoor and outdoor robot navigation (Temeltas & Kayak, 2008), unmanned aerial/underwater vehicles (e.g. Bryson & Sukkarieh, 2008; Ferreira *et al.*, 2012), and planetary rovers (e.g. Ingrand *et al.*, 2007).

SLAM is more of a concept than a specific algorithm, and, thus, the two main tasks involved – localisation and mapping – can be implemented using different methods. A SLAM system should solve the following problems: landmark extraction, data association, state estimation, state update and landmark update (Riisgaard & Blas, 2003). There is no fixed method for each step, and “*landmarks*” and their extraction methods are depending upon the type of sensor in use, such as laser, sonar, vision. These are very similar to the 3D reconstruction methods referred to in Section 2.1. However, no matter what type of sensor is being used, the SLAM process can be described in a general form: if robot is moving around, the sensor mounted on robot will acquire data from the environment at a certain sampling rate. The landmarks are distinctive and easily recognisable features in the environment, which can be sensed by the sensor as *observations*. By re-observing landmarks, the robot will associate the newly acquired data with the old data to further achieve an *odometry measurement* between the different states (*i.e.* the change of the robot’s positions over time). The landmarks which have not previously been seen will also be updated to the map and be used for re-observation in next move of the robot. This process is quite similar to the SfM methods described in Section 2.2.3.1. Actually, incremental SfM can be considered as an approach of visual SLAM – a sub-branch of SLAM which mainly makes use of vision (e.g. a portable and low-cost camera as a sensor to acquire image data). So-called

landmarks refer to stable image features and data association is performed by feature matching and 3D position recovery of found correspondences. The camera pose of images can be estimated due to 3D-to-2D correspondences and then more feature points can be recovered in space.

Besides monocular images used in SfM, visual SLAM also uses a *stereovision* approach to solve the SLAM problem (Lemaire *et al.*, 2007). Stereovision uses sequences of stereo frames captured by multi-view stereoscopic cameras or calibrated stereo rigs. The stereovision-based SLAM is presented in Mallet *et al.* (2000), Sola *et al.* (2008) and (Schleicher *et al.*, 2009). Alternatively, the recent emergence of RGBD cameras (such as Microsoft's Kinect) can directly provide colour images and associated dense depth information simultaneously. There are several literatures of RGBD-based SLAM (Henry *et al.*, 2010; Endres *et al.*, 2012; Sturm *et al.*, 2012; Endres *et al.*, 2014), but these have mostly been confined to projects in an indoor environment. This is due to the operational limitations of the IR projector and sensor when deployed in outdoor environments, as listed in Abbas & Muhammad (2012):

- 1) Limited field of view preventing an agile operation.
- 2) Short range, not providing the scale for typical outdoor applications.
- 3) Infrared saturation in direct sunlight.

Despite these, Abbas & Muhammad (2012) attempt to push the limits on the Kinect's capabilities to obtain a minimally acceptable performance on an outdoor mine detection application by taking advantage of the slow nature of mine detection tasks and by designing the robot to operate in indirect sunlight when lighting does not saturate

Kinect's IR sensor.

In practice, during a state estimation process, errors can be caused by both estimation models and measurement tools. Performing a full bundle adjustment to all undetermined variables after all state estimation processes have been finished, can definitely sort out this issue. However, the refinement in “the last step” for real-time applications is not so important because most of their operations require that accurate estimations should be updated instantly for each time step. Not only that, as the number of cameras and landmark observations increases, a full bundle adjustment can be quite time-consuming, hence it is unlikely to be used for intermediate refinement. Therefore, *sequential optimal* methods, which are suitable for real-time processes and will approximate the global BA to fit within fixed computational bounds, are preferable. There are two approaches that have been used in visual SLAM and have proved to be successful: *filtering* methods and *keyframe-based* methods (Strasdat *et al.*, 2012). Filtering will marginalise out the past poses and retain the observations to summarise the information gained over time with a probability distribution. One of the most common filtering methods is *extended Kalman filter* (EKF) (Haykin, 2001) which uses Gaussian probability distributions. The detailed information of EKF and many other Bayesian filtering techniques (e.g. particle filters) are reviewed in Chen (2003). Keyframe-based methods will retain the optimisation approach of a global BA, but, computationally, will select only a small number of past frames to process. Strasdat *et al.* (2010) compared filtering versus keyframe BA for monocular SLAM in terms of accuracy and computational costs. They concluded that, in order to increase the accuracy, it is usually more profitable to increase the number of observations than the number of frames to

process. Thus keyframe BA is more efficient than filtering to achieve an identical level of accuracy.

The SLAM problem can also be intuitively represented by a *graph* whose nodes correspond to camera poses or landmarks, and in which an edge between two nodes encodes a sensor measurement if a certain landmark can be observed from a certain pose. The *graph-based SLAM* problem was first proposed by Lu & Milios (1997). The optimal map can be computed by finding a configuration of the nodes that is maximally consistent with the measurements (Grisetti *et al.*, 2010). Keyframe-based methods apply a sparse version of this graph whose nodes represent poses of selected keyframes and are connected by pose-to-pose constraints obtained from odometry measurements. Generally, for sequential processes, each new camera pose can be estimated by matching the current image data to the last node, but the measurement errors in such recursive system accumulate fast when robot has been travelling for a long time. Therefore, the constraints between current pose and much older keyframe nodes are expected to be constructed, known as *loop closure detection*. Loop closures can be found by using the topological structure of the constructed graph (Engel *et al.*, 2014), or by using image retrieval approaches such as visual bag of words described in Section 2.2.4. Once the new pose node and its links connected to old nodes are added to the graph, the graph can be optimised using a generic graph optimisation framework (e.g. Kümmerle *et al.*, 2011).

2.2.4. Image retrieval for loop closing

Image retrieval techniques aim to search and retrieve images from a large database of digital images (Del Bimbo, 1999). Alternatively, this can be interpreted as an appearance-based information retrieval process. In a visual SLAM system, image retrieval techniques are one of the effective ways to recognise already-mapped areas, known as **loop closure detection** (Angeli *et al.*, 2008). Specifically, some input images are saved at regular intervals during the map construction to represent visual information of different locations. Detecting a loop closure to a visited place can be used for the data association problem. This would be easy for most SLAM systems, owing on the continuity of their input, which means each new image will have a high probability is being able to close loops with its immediate predecessors. Moreover, detecting large loop closures to those much older pose nodes (*i.e.* non-adjacent nodes in a sequence) can establish more constraints for the current pose node to make graph-based optimisation more robust. In addition, loop closing can also help to re-localise a robot when lost. This issue is similar to AR user tracking when starting from a random place: has this place been mapped in the trained database? If so where is this place relative to the constructed map? For batch-based SfM, the image collection is usually unordered. The simplest way to determine whether a pair of images matches is by performing a full comparison, namely a brute-force/exhaustive test. Despite the fact that the same linear search method can also be used for querying an image from a database, the cost is proportional to the number of all candidates to compare and becomes unacceptable expensive when the database becomes large. To reduce the times of comparing, the images in the dataset should be organised and indexed by their own feature descriptions(Rui et al., 1999). The main idea here is to use global features – a

single descriptor to represent an entire image – and so a numeric similarity between query and each object in the database can be computed and ranked (Jansen & Rieh, 2010). These global descriptors can be derived from edge and colour histograms (Starner *et al.*, 1998; Lamon *et al.*, 2001), sets of texture features (Torralba *et al.*, 2003), or edge map and gradient orientation histograms (Kosecka *et al.*, 2003). However, such global features are not very robust to portion changes between images, such as lighting, perspective change and occlusion. Therefore use of the score schemes based on local features is preferable, as described in Košecká *et al.* (2005) who characterises each location by a set of representative images, and then finds matched robust local features (*i.e.* SIFT) between the query and each model view. The likelihood of the matching is computed by employing a voting approach to assess the amount of similarity. Another representation methodology is *bag of words (BoW)* (Sivic & Zisserman, 2003), which treats each image as a set of visual words. Visual word is a data structure that carries the feature information amongst the training data in an image retrieval system (Baeza-Yates & Ribeiro-Neto, 1999). Specifically, each visual word corresponds to a descriptor vector of an invariant local feature, taken from a *visual vocabulary* which is built from a training database. Therefore an image can be represented by a binary vector whose length is equivalent to the size of the vocabulary and each element indicates the presence or absence of a corresponding word (Cummins & Newman, 2007). The ‘distance’ between such two vectors can be used to assess the similarity between their respective images, and then the retrieval results can be ranked by a NN search (Newman *et al.*, 2006; Wang *et al.*, 2006). Based on BoW scheme, Cummins & Newman (2007) propose a probabilistic framework which performs location-matching between places that have been visited within the world, as well as providing a measure of the

probability of being at a new, previously unvisited location. They explicitly account for *perceptual aliasing* in the environment – identical but indistinctive observations receive a low probability of having come from the same place. An Open Source appearance-based navigation system *FAB-MAP (Fast Appearance-Based Mapping)* (Cummins & Newman, 2008) is based on this work, and its advanced version is proposed in Cummins & Newman (2011) which is capable of dealing with a very large scale place recognition.

Contrary to the offline training needed in the methods described so far, Angeli *et al.* (2008) presents an incremental real-time method with BoW to detect loop-closures in a Bayesian filtering scheme, computing the probability that the current image comes from an already-perceived scene. The system is able to work indoors and outdoors without prior information on the environment type, but unlike FAB-MAP, this system cannot perform evaluation of the distinctiveness of visual words and may be affected by perceptual aliasing.

The proposed AR framework in the present thesis for higher-level developers aims to implement a markerless AR application based on a specific place specified by the developers. As shown in Table 2-1, due to the request of markerless tracking and accurate registration, the development system needs to learn the appearance and spatial structure of the target environment to augment. The developers can only provide the environment information (e.g. the visual information of the Eddystone Reef Tank of the **National Marine Aquarium**) obtained by a low-cost sensor – *i.e.* RGBD data for indoor environment or RGB data for both indoor and outdoor – thus the environment

learning procedure is actually a vision-based 3D reconstruction / mapping problem which can be solved by SfM or visual SLAM systems reviewed above. Of relevance here, two applications which have been reported with reasonable high accuracy on 3D reconstruction (see Section 2.3.3) – VisualSfM (Wu, 2011; 2013) taking RGB input and RGBD-SLAM v2 (Endres *et al.*, 2014) taking RGBD input – are utilised in the proposed development framework for training the reference map of the targeted place. The detail and performance of these two methods is described further in Chapter 4. One should notice that these two methods are not the only ones that can be used in the proposed framework; any open source vision-based 3D mapping approaches with high accuracy can be used as an alternative selection.

With regard to the AR application, the procedure can be described as follows: when application end users (e.g. visitors to the National Marine Aquarium) capture some visual information via their input devices, the application needs to determine whether the input is taken at the targeted place which has been registered with AR content by developers before (e.g. superimposing animated 3D marine animal models onto the specific locations within the Eddystone Reef Tank). This is basically a loop closure detection problem inside SLAM, but requesting high processing rate for meet the real-time requirement as well as high accuracy for accurate AR registration. Thus an efficient image retrieval technology is expected in this stage to locate the data with the closest appearance to the input from the trained reference map. The FAB-MAP (Cummins & Newman, 2008; 2011) with BoW approach – which has been reported with robust performance on handling loop closing problem inside large area and real-time SLAM problem – is selected in the proposed framework for serving the application

online tracking. The detail and performance is described in Chapter 5.

2.3. Hardware, software supports and datasets for evaluation

There are several hardware devices and software available to support the system implementations of vision-based AR discussed above, including the camera used for inputting image data, the GPU used for assisting with the general purpose calculation, and the software and methods supporting 3D reconstruction for AR template training. After implementation, the performance of system needs to be tested and assessed. According to the discussion in Section 2.1.3, the system can be evaluated from both a subjective (qualitative data-based) and objective (quantitative data-based). Several datasets and evaluation benchmarks are created and developed in the past for the two major non-user based CV technologies – 3D reconstruction/mapping and image retrieval – used in the proposed framework. The following subsections will describe and evaluate some of the hardware, software and CV datasets used for supporting and evaluating a vision-based AR system.

2.3.1. Hardware

As mentioned in Section 2.1.1, a visual AR application performs sensing, tracking, registration and interaction during system time. This can be described as *input-process-output* model and each stage of the model is associated with particular hardware support. This section focuses on those **non-exclusive** hardware devices used in AR which means they should be relatively easy to access by most people (including researchers,

developers and the general public).

Input / camera devices



Figure 2-19: Some affordable input camera devices that can be used for vision-based AR.

The major input sensor used for vision-based AR is camera, whose task is capturing the source data for both 3D template training offline and AR user tracking online. Particular for 3D reconstruct procedure, a performance comparison of several cameras for 3D reconstruction is presented in Thoeni *et al.* (2014). This input camera can be a common monocular camera, a stereoscopic camera or an RGBD camera, as shown in Figure 2-19.

The monocular camera is ideal for AR applications since it has already been integrated into several general-purpose devices, such as the personal computers, laptops or smartphones, thus the owners of these devices can make use of the camera directly. At the time of writing, the frame rates and capture resolutions of laptop webcams are around 30 fps at 640 x 480 or even higher (sometimes higher resolutions are accompanied by a lower frame rate), and smartphones such as iPhone5 provide a resolution of 3264 x 2448 for single capture and 1080p (note that "p" here is short for "progressive scanning", although 1080p usually refers to 1920x1080 pixels) for 30 fps video recording⁸. In addition to the computer-integrated camera, recent add-on webcams, such as lightweight and portable *Logitech Webcam* series (e.g. the Logitech HD Pro Webcam C920 in Figure 2-19 ①), can be connected to a computer or a laptop easily by using a standard USB or a firewire cable which often allows for faster data transference than USB, with better video quality. An independent camera device, like the popular light action camera *GoPro Hero* series (e.g. the GoPro Hero4 Black in Figure 2-19 ②), can also be used for providing high quality image data for 3D reconstruction offline.

Stereo webcams have also appeared recently, such as the *Minoru 3D Webcam* (Figure 2-19 ③), which supports a maximum resolution of 800 x 600 and a maximum frame rate of 30 fps with two discrete webcams integrated together. Stereo images are generated in anaglyph form (*i.e.* 3D video)⁹. Another recently appeared lightweight stereo camera with relatively cheap price is *StereoLabs ZED* (Figure 2-19 ④),

⁸ **iPhone 5 - Technical Specifications:** https://support.apple.com/kb/SP655?locale=en_GB

⁹ **Say Hi to the Minoru 3D Webcam!** : <http://www.minoru3d.com/>

supporting a maximum resolution of 2208 x 1242 and a maximum frame rate of 15 fps. This device allows a long range measurement (1.5 to 20m) for both indoors or outdoors. However, the depth map computation is performed using GPU and the actual depth calculation speed is heavily depends on the graphics card used.

An RGBD camera is an alternative to obtain the associated depth information of the RGB images. Typical and affordable RGBD cameras include the Asus Xtion Pro Live (Figure 2-19: ⑤), the Microsoft Kinect 1.0 (Figure 2-19: ⑥) and its successor Microsoft Kinect 2.0 (Figure 2-19: ⑦). The Xtion and the Kinect 1.0 are structured light-based sensors, performing a triangulation process between an infrared camera and an infrared projector embedded in the devices. The projector produces a pattern of dots which are projected onto the entire field of view, and then the camera internal process receives the reflected pattern and computes the corresponding depth for each image pixel. The internal structures of these two devices are basically similar and both provide 30fps 3D video. However, they have their own advantages and disadvantages. Kinect 1.0 has a bigger size and is of a greater weight than the Xtion; it also requires an external power supply which makes it inconvenient for moving. The depth range of the Kinect is between 0.8m to 4.0m against the range of 0.8m to 3.5m for the Xtion. They both support a higher RGB resolution at 1280 x 1024, although the depth resolution is still only 640 x 480. On the other hand, the Kinect 1.0 features a motor that can be controlled remotely by a particular application, whilst the Xtion only allows manual positioning. The Xtion only supports a USB 2.0 interface which is a considerable disadvantage for users with more recent laptops or PCs which may only support USB 3.0.

On the other hand, the Kinect 2.0 – an advanced time-of-flight-based Kinect sensor that indirectly measures the time it takes for IR light pulses from a projector to a surface, and back to the sensor – provides a somewhat lower depth resolution (512 x 424), but higher RGB resolution (1920 x 1080) and wider depth range (0.5m to 4.5m). A comparison between two generations of the Kinect is conducted in Pagliari & Pinto (2015). They state that the structured light approach is not always robust enough to provide a high level of completeness of the framed scene, meaning that, regardless of the actual resolution of the IR sensor, the actual depth readings may only cover a small percentage of this resolution, The tiny areas that fall between dots is considered as ambiguities which cannot achieve a depth estimate. On the other hand, the time-of-flight sensor spills each pixel in two accumulators and a clock regulates which one of the pixel side is the one currently active. The Kinect 2.0 relies on measuring the differences between two accumulators to obtain depth information, each one containing a portion of the returning IR light. In order to eliminate the depth ambiguity issue of the Kinect 1.0, the Kinect 2.0 acquires images at multiple frequencies (*i.e.* 120MHz, 80MHz and 16MHz). Longer wavelengths allow for measuring longer distances with low resolution, while shorter wavelengths give higher resolution. The precision can then be improved by using the two measurements together. Furthermore, as mentioned in Section 2.2.3.2, Kinect 1.0 cannot be used under infrared light sources, but the Kinect 2.0 can remedy this issue with its built-in ambient light rejection, where each pixel individually detects when that pixel is over-saturated with incoming ambient light (Lau, 2013). Pagliari & Pinto (2015) also comment that the Kinect 2.0 can even work properly outdoors when the scene has low ambient IR light, but it is still difficult for scenes that exist under

direct sunlight.

Computer processor and GPU

The main tasks of an AR system, such as template training, tracking and augmentation registration mentioned above, are all processed within a computer, which can be a desktop or laptop PC (e.g. Simões *et al.*, 2013; Kurihara & Sagawa, 2014), a handheld tablet computer (e.g. Zoellner *et al.*, 2009; Haugstvedt & Krogstie, 2012) or even a high-powered smartphone (e.g. Klein & Murray, 2009; Lee *et al.*, 2012). The performance of the system depends upon the processing capacity of the device and the computational costs of the implemented methods. In general, the computing power of mobile device is much limited due to typical market demands for smaller sizes and lower cost, but most of them can deal with the tasks which do not require expensive computational overheads to achieve (such as marker-based tracking described in Section 2.1.1). However, the more complex processes such as SLAM and 3D reconstruction described in Section 2.2.3, or for investigations underpinning experimental research still require implementation using a computer with powerful processor (also referred to as *CPU*). One of the crucial but hardest requirements of AR applications is achieving real-time (or near real-time) registration and image display/update. In order to accelerate the computing process, some computers employ a multi-core processor which was originally provided for manipulating computer graphics and image processing (*i.e.* a *graphics processing unit* (GPU)) together with a CPU to solve large blocks of data in parallel. GPUs are more efficient than general-purpose CPUs since its massively parallel architecture, consisting of thousands of smaller, more efficient cores, has been designed for handling multiple tasks simultaneously. One use for GPU-accelerated

computing in AR technologies is VisualSfM (Wu, 2011) with SiftGPU (Wu, 2007) and Multicore Bundle Adjustment (Wu *et al.*, 2011), which will be described in Section 4.2.2. The graphics cards of most computers are produced by Intel, ATI/AMD or Nvidia. Most modern versions of GPU support to be accessed with the graphics APIs, such as *Direct3D* for Windows platform and *OpenGL* for cross-platform. They allow user to enable GPU for general purpose processing but they also require skills in graphics programming. Nvidia has created another parallel computing platform and API called CUDA¹⁰, which is designed to work with programming languages such as C and C++ and does not require skills in graphics programming (in contrast to Direct3D and OpenGL). Actually, The ZED stereo camera mentioned above requires a CUDA-capable computer with a Nvidia graphics card for full capability.

Output / display devices

The output of a visual AR application refers to the augmented visual information to display, whereas the output device usually refers to the display monitor/screen of a stationary computer or mobile device. However, due to the aims of AR applications, the display devices can be a little different, such as the Magic Mirror system or Cisco's AR commercial application cited in – Carmigniani & Furht (2011). Here the output screen is used as a fake mirror, allowing users to try on virtual clothes before buying them. Another example is the driver guidance system of Santana-Fernández *et al.* (2010), where the output device takes the form of eye monitor glasses. This example belongs to

¹⁰ **CUDA – Parallel Programming and Computing Platform:**

http://www.nvidia.com/object/cuda_home_new.html

the prevalent class in VR and AR of *head mounted display* (HMD), designed to take advantage of the movements of the user's head (in contrast to handheld displays or computer monitor which are not, in the main, wearable) and, thus, enhance the reality of the augmentation. The latest generations of HMD include *Oculus Rift* and *HTC's Vive*, but technically they are VR-centric devices which require additional front-facing input camera and setups for generating AR view. The HMDs designed for AR includes *Immy Mark I* featuring Natural Eye Optics with 60 degree field of view and 1024 x 768 display resolutions, 3 forward facing cameras and integrated *Inertial Measurement Unit (IMU)*¹¹; *Meta 2 AR headset* supporting 90 degree field of view and 2560 x 1440 display resolution, 720p front-facing camera and sensor array for hand interactions and positional tracking¹²; and the *Microsoft's HoloLens* — a high-definition stereoscopic 3D optical smartglasses. In fact, the HoloLens is not only a display device, but mostly a platform, as discussed in the next subsection.

AR platforms

An AR platform can be understood as a device specifically designed for supporting AR implementations from both aspects of hardware and software. The hardware design is generally very comprehensive to support whole input-process-output work cycle within AR applications. The built-in software may produce some useful indirect data – such as determining the user pose or mapping the environment – automatically through the hardware by applying one or more technologies, such as those mentioned in Section 2.2.

¹¹ **Immy Inc.:** <http://www.immyinc.com/>

¹² **Meta Company:** <https://www.metavision.com/>

The data can be assessed by users using the provided SDKs and APIs, allowing for further AR application development. Specifically, the Microsoft's Next Generation Devices Team (Holmdahl, 2015) declared that their MR smartglasses HoloLens features an IMU for tracking head motion, several sensors (which includes a depth and a photographic video camera) for understanding surrounding environment, spatial sound and microphone for audio augmentation and communication, and a custom-made Microsoft Holographic Processing Unit along with the traditional CPU and GPU for computing and processing. This device was announced as a part of Microsoft's Windows Mixed Reality project featuring an AR operating environment in which any universal application can run. Other AR platforms with similar concept include *Intel RealSense* and *Google Tango*. The Intel RealSense technology consists of series of 3D cameras (R200 and SR300, mainly structured light-based) together with multi-platform support SDKs and APIs for "achieving depth perception, 3D imaging, interior mapping, and feature tracking"¹³. Actually the RealSense technology is not so much dedicated to improve the human/computer interaction for supporting a user-based system, commented in Shilov (2016), but has quickly evolved into a more general CV technology that will eventually be used for robots and drones. In contrast, although Google Tango uses approximately similar hardware to Intel RealSense (except Tango devices use fisheye motion cameras and different computing devices), it is a human-centred design aiming to give computers a human-like perception of space and motion. Google Tango chiefly targets mobile devices, focus on determining their position and orientation in real-time. The major types of functionality include motion-tracking, area

¹³**Intel® RealSense™ Technology:**

<http://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>

learning and depth perception, which allow application developers to create several user experiences including AR.

Although these new (at the time of writing) technologies promise the potentials for future AR development, there is no literature (again at the time of writing) that provides an evaluation or even evaluation metrics for these AR platforms.

2.3.2. Software

There are many AR SDKs that allow higher-level developers (shown in Figure 2-8) to create AR applications directly without the knowledge of the inner “functioning” of AR techniques. These AR frameworks are available as free or Open Source software toolkits, as well as those products targeting commercial organisations and include various key functions, such as tracking methods and virtual image rendering techniques. A recent comparative study of several popular AR SDKs is given in Amin & Govilkar (2015). It can be found that the system based on 2D artificial marker-based tracking have been widely supported by each SDK, thanks to the maturity of this particular approach and the reasonably impressive results it has demonstrated over the past few years (also discussed in Section 2.1.1). The application developer can design and customise the 2D pattern as a reference to track. Other supporting tracking methods include GPS/WPS -based, human face-based and 3D object-based. Most of the 3D models supported for use of tracking are a general 3D object such as cuboid and cylinder, or a calibrated model provided with the SDKs. Vuforia has also provided a 3D object scanner for users to create their own 3D models from the real world, but this

requires relatively harsh lighting and clear conditions of the environment. On the other hand, natural scene-based tracking is rarely supported, although note that Metaio has provided a function of instantly 3D mapping and tracking of small workspace.

The present thesis aimed to use 3D reconstruction technology to support system learning features and the creation of maps for each specific AR target environment. Except for using the selected VisualSfM for handling RGB-input and RGBDSLAM v2 for handling RGBD-input, there are several potential alternatives for the 3D reconstruction task of the proposed markerless development framework. For multi-view 3D reconstruction (*i.e.* RGB-input), Kersten & Lindstaedt (2012) investigated the following software for archaeological 3D reconstruction: 1) open-source software packages – VisualSfM and *Bundler* (Snavely *et al.*, 2008) (estimating camera poses and generating sparse point clouds) along with *PMVS2 (Patch based Multi View Stereo Software)* (Furukawa & Ponce, 2010) (generating dense point clouds through the output results of Bundler or VisualSfM); 2) free web service – *Autodesk 123D Catch*¹⁴; and 3) low-cost commercial software – *Agisoft PhotoScan*¹⁵. According to their various test results, they recommended VisualSfM as best solution, due to the balance between efficiency and geometric quality. Schöning & Heidemann (2015) propose a benchmark which ranked four most common multi-view 3D reconstruction software solutions – VisualSfM, Autodesk 123D Catch, Agisoft PhotoScan and *ARC 3D*¹⁶ – by comparing their produced 3D models qualitatively and quantitatively. VisualSfM ranked first again

¹⁴ **Autodesk 123D**: <http://www.123dapp.com/>

¹⁵ **Agisoft PhotoScan**: <http://www.agisoft.com/>

¹⁶ **ARC 3D**: <http://www.arc3d.be/>

through their comprehensive consideration of quality, academic licensing and runtime.

For RGBD sensor-based 3D reconstruction and mapping, the alternatives of RGBDSLAM include *KinectFusion* (Newcombe *et al.*, 2011) and *RTAB-Map (Real-Time Appearance-Based Mapping)* (Labbe & Michaud, 2014). Zhu *et al.* (2016) report that KinectFusion has a higher accuracy and real-time performance than RGBDSLAM – RGBDSLAM usually runs at 2 FPS while KinectFusion runs at 15 FPS. However they also point out that the biggest disadvantage of KinectFusion is that it can only build a limited size of map, since this method is memory-consuming and it uses a GPU memory. Consider that the 3D reconstruction process proposed in this thesis does not require a real-time performance, RGBDSLAM could be acceptable and its performance evaluation against VisualSfM is presented in Section 4.3.

2.3.3. CV datasets for evaluation

The proposed vision-based AR system consists of several CV-based technologies, as set out in Section 2.2. There are plenty of image databases for various CV research problems, and each of these problems requires particular evaluation metrics for assessing the performance of applied algorithms. Some of these databases are categorised and archived online, available for public use (e.g. CV Datasets on the web¹⁷ and CVonline: Image Databases¹⁸). In this section, the datasets for evaluating 3D reconstruction/mapping and loop closure detection methods are reviewed.

¹⁷ **CV Datasets on the web** : <http://www.cvpapers.com/datasets.html>

¹⁸ **CVonline: Image Databases**: <http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm>

3D reconstruction / mapping

Two kinds of 3D reconstruction or localisation and mapping method are available in this proposed work for learning a specific target environment: one is SfM, taking a set of RGB images as input; another is RGBD-based SLAM, taking RGBD data as input. 3D reconstruction and 3D localisation/mapping are not the same task. Visual SLAM applications – as the name implies – focus on learning the environment from the obtained visual information and locating the sensor with respect to the map they have built. Meanwhile, SfM-based applications put more focus on 3D reconstruction, which estimates 3D geometric information from the images for creation of virtual 3D model – either a meshed model or a set of point cloud. Thus the accuracy of data produced by CV-based methods for 3D reconstruction is generally evaluated by comparing the created models against the **ground truth**. Schöning & Heidemann (2015) state that, the ground truth data in most benchmarks or evaluations on multi-image 3D reconstruction is acquired by traditional terrestrial 3D laser scanners and light detection and ranging (LIDAR) systems, such as Zoller+ Fröhlich’s IMAGER 5003 laser scanner in Strecha *et al.* (2008), Zoller + Fröhlich’s IMAGER 5006h and IMAGER 5010 terrestrial laser scanners in Kersten & Lindstaedt (2012) and ATOS Compact Scan 2M 3D scanner in Mousavi *et al.* (2015). Further, Schöning & Heidemann (2015)’s benchmark require two criteria: 1) including real scene photographs as well as photographs taken in a controlled indoor environment; 2) the availability of a ground truth. They examined several multi-view datasets and finally chose the datasets *fountain-P11* and *Herz-JesuP8* (with integrated LIDAR 3D triangle meshes as ground truth, as shown in Figure 2-20) (Strecha *et al.*, 2008) as a real scene, and the dataset *Oxford Dinosaur* (Visual

Geometry Group, 2004) for a controlled indoor environment. Schöning & Heidemann (2015) then make use of an iterative closest point algorithm (Besl & McKay, 1992), aligning and registering the model with the ground truth. The minimal distance between every point of registered ground truth model to any triangular face of the reconstructed mesh is computed. The **mean value** and the **standard deviation** of all these distances are used for accuracy comparison between different reconstruction methods, and the computation **time** is also considered.

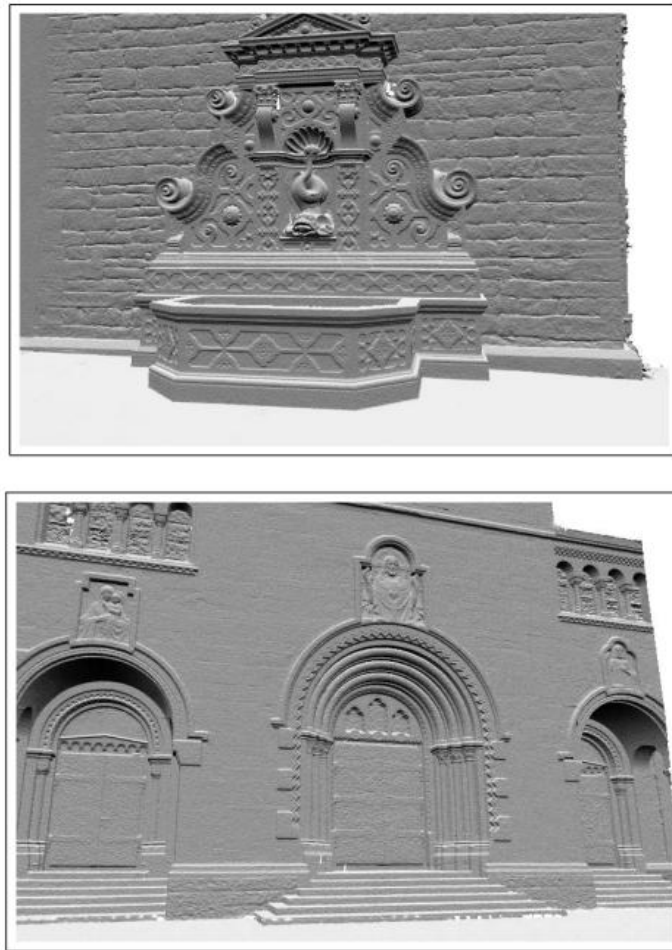


Figure 2-20: Diffuse rendering of the integrated LIDAR 3-D triangle meshes for the datasets fountain-P11 (upper) and Herz-Jesu-P8 (lower).

(Strecha *et al.*, 2008)

However, a good reconstructed meshed model is not necessary in the present thesis. The basic task of markerless AR tracking is closer to a SLAM problem, in which the accurate pose of user viewport is in demand and what need to be “reconstructed” is a reference map of the target environment which consists of both geometric information and recognisable visual features, *i.e.* the point cloud of keypoints. In this case, a meshed model of ground truth cannot match the requirement. In fact, the visual information can only be extracted by CV methods and it is hard to obtain so-called “real values” by other types of sensors as ground truth. However, as introduced in Section 2.2.3, both

SfM and SLAM methods contain the processes of camera pose estimation and map creation with 3D point clouds (which further becomes a dense model in SfM-based applications), and the resultant accuracy of these two processes are highly dependent on each other. Therefore the evaluation criterion designed for SLAM system which usually uses associated camera pose of each image as ground truth is considered instead, and Strecha *et al.* (2008)'s dataset also provide the ground truth of camera pose along with the model.

Since RGBD data can also be used as input in this proposal, Sturm *et al.* (2012)'s benchmark for the evaluation of RGBD SLAM systems is one of the options. 39 RGBD image sequences of an office environment and an industrial hall are provided, which are recorded from a Microsoft Kinect with highly accurate and time-synchronised ground truth camera poses from a motion capture system. The authors declared that this dataset is the first RGBD dataset suitable for the evaluation of visual SLAM systems and propose two evaluation metrics: 1) evaluate the end-to-end performance of the whole system by comparing its output (map or trajectory) with the ground truth; 2) compare the estimated camera motion against the true trajectory. The accuracy is then measured with **relative pose error** and **absolute trajectory error**. Assume $\mathbf{P}_1, \dots, \mathbf{P}_n \in \text{SE}(3)$ is a sequence of poses from the estimation and $\mathbf{Q}_1, \dots, \mathbf{Q}_n \in \text{SE}(3)$ is the sequence from the ground truth. The relative pose error at time step i is defined as

$$\mathbf{RPE}_i := (\mathbf{Q}_i^{-1} \mathbf{Q}_{i+\Delta})^{-1} (\mathbf{P}_i^{-1} \mathbf{P}_{i+\Delta}) \quad (2.13)$$

where Δ is fixed time interval. The absolute trajectory error at time step i is defined as

$$\mathbf{ATE}_i := \mathbf{Q}_i^{-1} \mathbf{S} \mathbf{P}_i^{-1} \quad (2.14)$$

where the rigid-body transformation \mathbf{S} corresponds to the least-squares solution that maps the estimated trajectory $\mathbf{P}_{1:n}$ onto the ground truth trajectory $\mathbf{Q}_{1:n}$. The errors over all time indices then are evaluated by computing *root mean squared error (RMSE)*, which gives less influence to outliers than computing mean error.

An alternative benchmark for RGBD SLAM is given in Handa *et al.* (2014). Their dataset is collected from two different environments: the living room and the office room. Just like Sturm *et al.* (2012), all RGBD image sequences are associated with ground truth trajectory, but moreover the sequences from the living room scene have camera pose information associated with a 3D polygonal model. Thus, these sequences can be used to benchmark both camera trajectory estimation and 3D reconstruction. One of the latest surveys of RGBD datasets – Cai *et al.* (2017) – compared Sturm *et al.* (2012)’s benchmark dataset with Handa *et al.* (2014)’s dataset, commenting that the latter “is more challenging and realistic since it covers large areas of office space and the camera motions are not restricted”.

Loop closure detection

Another CV-based key technique applied in the present work is visual loop closure detection. From a visual perspective, finding a loop closure can be expressed as if there is sufficient similarity between the current image and a map image (Liu & Zhang, 2013). In general the datasets collected for studying algorithm performance on navigation and mapping (*i.e.* SLAM) can also be used on loop closure detection, such as some sequences inside Sturm *et al.* (2012) RGBD dataset (e.g. [freiburg1_room]). More specifically, several loop closing-targeted researches made use of the SLAM datasets

for evaluating their methods which are described below. Cummins & Newman (2008) have tested their FAB-MAP system on their New College and City Centre image collections¹⁹ which are composed of images of outdoor urban environment collected by mobile robot, the corresponding coordinates of each image derived from interpolated GPS, the ground truth “mask” used for measuring the image-to-image correspondence matrix generated by the loop closing algorithm, aerial photo for visualising results and camera calibration information. Another dataset²⁰ for loop closing problem are provided in Angeli *et al.* (2008) as supplemental material, which includes an indoor image sequence with strong perceptual aliasing and a long outdoor image sequence. The dataset also contains the ground truth image-to-image correspondence matrix and camera calibration information. Glover *et al.* (2010) introduced an appearance-based SLAM for multiple times of day and they collected their dataset²¹ from a selection of streets in the suburb of St. Lucia with corresponding GPS data for experiment. The visual data were collected by traversing a route at five different times during the day to capture the difference in appearance between early morning and late afternoon. The route was traversed again, another five times, two weeks later for a total of ten datasets.

As mentioned in Section 2.2.4, visual loop closure detection can be considered as an

¹⁹ **FAB-MAP – dataset:**

http://www.robots.ox.ac.uk/~mobile/IJRR_2008_Dataset/data.html

²⁰ **Cognitive Robotics at ENSTA:: Loop Closure Detection – dataset:**

<http://cogrob.ensta-paristech.fr/loopclosure.html>

²¹ **St Lucia Multiple Times of Day dataset – dataset:**

<https://wiki.qut.edu.au/display/cyphy/St+Lucia+Multiple+Times+of+Day>

image retrieval problem. Therefore the evaluation metrics designed for information retrieval system are usually used for assessing the algorithm for loop closing. One of the most commonly used metrics is *precision-recall curve*, which has been presented in several loop closing-related works for performance evaluation, such as Cummins & Newman (2008) and (Liu & Zhang, 2013). *Precision* is a measure of result relevancy, while *recall* is a measure of how many truly relevant results are returned (Pedregosa *et al.*, 2011). They can be defined with true positive rate and positive predictive value respectively, as shown below:

$$\begin{aligned}
 precision &= \frac{tp}{tp + fp} \\
 recall &= \frac{tp}{tp + fn}
 \end{aligned}
 \tag{2.15}$$

Where *tp* refers to *true positives*, *fp* to *false positives* and *fn* to *false negatives*, defined as follows:

	Prediction: positive	Prediction: negative
Truth: positive	true positive (<i>tp</i>)	false negative (<i>fn</i>)
Truth: negative	false positive (<i>fp</i>)	true negative (<i>tn</i>)

Precision and recall are typically inversely related, A system with high recall but low precision returns many results of predication, but most of the results are incorrect when compared to the truth; a system with high precision but low recall returns very few results, but most of them are correct when compared to the truth. Hence the precision-recall curves can be used to find an appropriate trade-off between precision and recall, assisting on selecting algorithms for different requirements (e.g. high precision at the

lower recall or high recall at the lower precision). An example of the precision-recall curves are given in Pedregosa *et al.* (2011) , as shown in Figure 2-21.

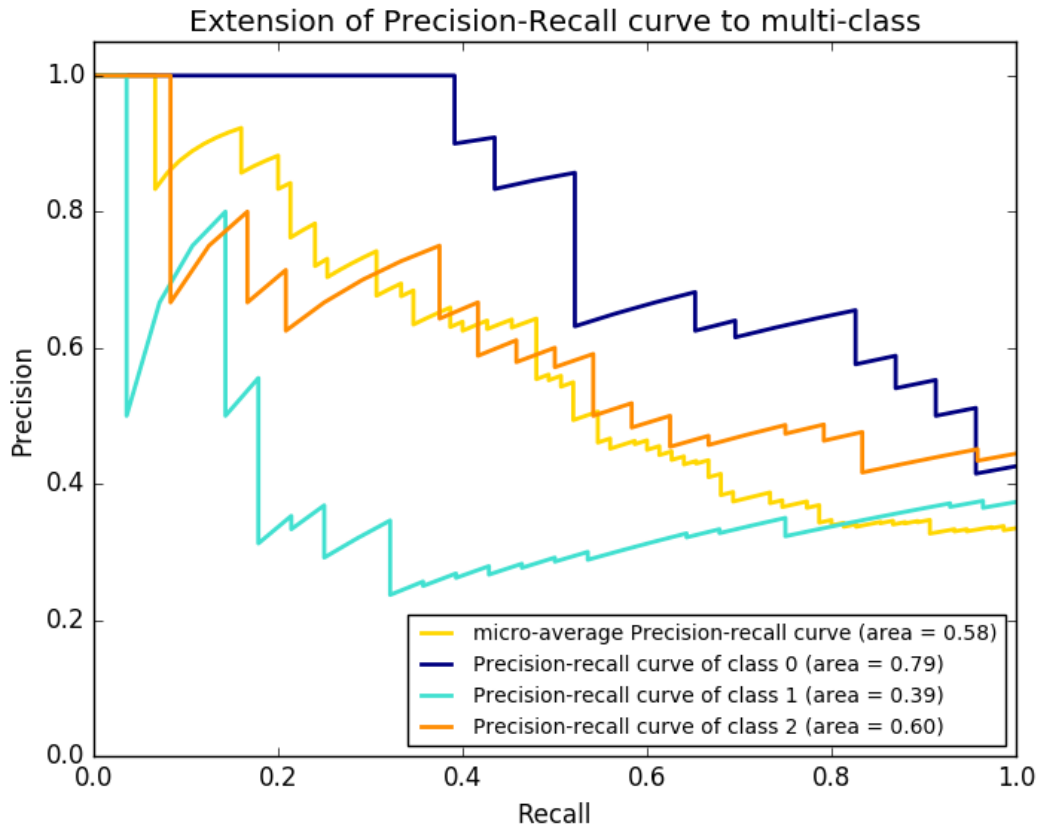


Figure 2-21: An example of Precision-Recall curves to multi-class.

Pedregosa *et al.* (2011)

In loop closure detection problem, the recall means the capacity of a system to detect a loop closure correctly when revisiting a mapped place while the precision means the proportion of the system detected loop closures are real loop closures. Whether a loop closure occurs or not is generally judged by the similarity between the images, which can form an *image-to-image correspondence matrix*. Entry (i,j) of this correspondence matrix will set to 1 if image i and image j were determined to be taken at the same place, or 0 otherwise. Thus the ground truth correspondence matrix provided in Angeli *et al.* (2008) and Cummins & Newman (2008)' datasets can be used to inspect the

performance of the algorithms.

2.4. Problems and Challenges

As discussed above, the choice of method to be adopted for tracking in an AR application is always one of the most challenging, yet essential problems to be overcome. Compared to exclusive sensor-based approaches, vision-based tracking methods seem to represent a more straightforward choice for the development and implementation of AR systems, since they will sense visual information instantly, and, as they do not require further data association between the sensors, the computed result can be directly used in registration process. Moreover, the camera images and videos are relatively low-cost and easy to obtain by people, including non-technical users. Vision-based AR systems attempt to recognise the pre-defined reference templates from the environment via the appearance of objects or other features, and further identify the position and orientation of the vision sensor (camera pose) with respect to these reference templates. This process is known as user tracking. An ideal AR user tracking system should result in accurate and efficient camera pose estimations, for enabling a refined registration of visual augmentation in real-time.

Vision-based AR tracking methods can be divided into several classes. As reviewed in Section 2.1.1, the results of artificial marker-based tracking, especially those of planar forms, are considered reasonably stable and robust. Marker-based tracking has been widely supported by many AR SDKs. In general, application developers are allowed to customise the appearance of a 2D planar marker, but for 3D markers, it is preferable to

use a general 3D polyhedron (e.g. cuboid) instead an arbitrary geometric structure. This is because the spatial positions of the visual features on the surfaces of a general 3D polyhedron are relatively easy to calculate due to its explicit geometric structure. Thus, it is possible to find the 3D-to-2D correspondences between the reference markers and the AR input images to perform camera pose estimations. However, the marker-based methods always require the introduction of artificial markers into the workspace. They are feasible for most flat media and some man-made situations, but are still unwanted in many other cases for aesthetic and conservation reasons, especially in preserved natural or historic environments. Even if they were accepted in such environment, it would be hard to maintain these markers in natural, outdoor settings. To take advantage of vision-based AR tracking and overcome the limitations of marker-based methods, the natural features of the original environment are extracted and used for tracking as markerless/natural feature-based tracking. In contrast to the marker-based methods which create and introduce additional artificial markers into the workspace deliberately, markerless methods attempt to make use of the visual information and the geometric structure of the original environment to create a reference map, and further estimate the user camera pose via this map. Markerless methods are relatively less intrusive and more flexible. However, one of the major difficulties is that the simple, general geometric structure described above rarely appears in a natural environment – whether it is a man-made environment or a completely rural setting. It is not appropriate to allow developers to create the reference model themselves, since to manually model a precise real-world scene with complex structures is quite difficult and highly time-consuming. For this reason, the results of the unsupervised 3D reconstructions implemented with CV-based technologies, such as visual SLAM and SfM, are considered to be used as the

reference for a markerless tracking process. The accuracy of the 3D reconstruction results to a large extent determines the performance of the AR registration.

On the other hand, and from an application development perspective, most of the recent research project reported in the literature concentrated on undertaking the template training stage themselves, or, alternatively, allowed the end users to create the reference map and then, in the same session, to view the augmentation results. However, this does not apply to all practical AR applications as discussed in Section 2.1.2. The higher-level designers and developers, who do not have a professional background (such as CV, SLAM, etc.) may intend to apply AR technology to develop their own applications with specific aims, – just like those who have made use of the existing marker-based AR SDKs for their development. This requires a markerless (e.g. 3D reconstruction-based) AR framework. The framework should provide an acceptable style of user interface to the higher-level developers, allowing them to restore the desired scene for tracking and to configure the augmented content for display. The framework should also provide an AR application interface to the application users, allowing them to interact with the application built by the developers.

Based on the problems described above, a user-oriented markerless AR framework has been suggested and the present research represents an attempt to implement it.

Chapter 3 Preparatory Studies

This chapter will present some preparatory studies of the proposed AR framework, which are divided into two parts: 1) the analysis of online survey results to identify the potential AR user types and their requirements (Section 3.1); and 2) the basic principle of geometric transformations frequently used in vision-based AR system (Section 3.2).

3.1. AR application development and requirement audience survey

Most of the AR-related projects and research studies, as mentioned in Section 2.1.2, were designed to be directed to the AR application end users. It is very common to perform questionnaires on application users in order to obtain their subjective opinions of the system. However, the motivation of this proposed framework represents a major focus on the potentially higher-level developers who may not be an AR “expert” but wish to apply AR technologies to their own developments. In order to identify who the target audiences of an AR system development framework are, and what are their requirements, a questionnaire was designed and displayed online to collect volunteer opinions of AR application development and requirement from different professions. A total of **80** responses were obtained and the analysis based on the results is performed in the following sections. A copy of the questionnaire is attached as Appendix A.

3.1.1. Questionnaire analysis

Profession distribution and understanding of AR

The respondents first specified their occupations. The answers can be roughly divided into Academic Profession, including researchers, research assistants, school instructors and students; High-Tech Industry Profession, including technical directors, advisors, computer-related developers and engineers; Business and Management, including production and marketing managers, founders, and managing directors; and “Other”, such as pewterers, composers and unemployed. The percentage of each of these categories is shown in Figure 3-1.

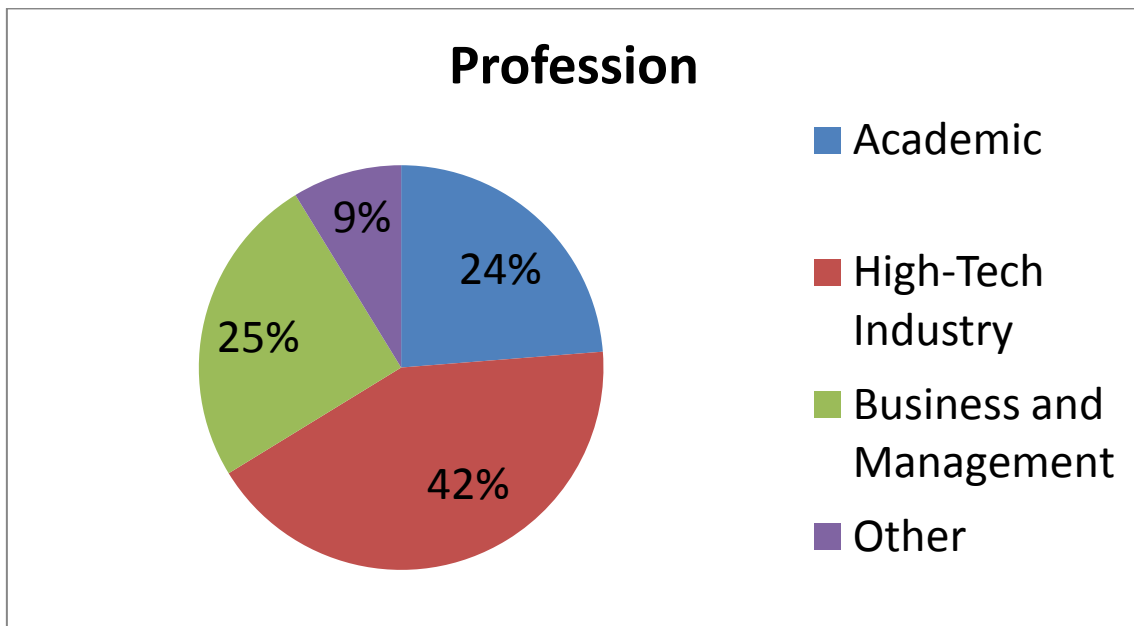
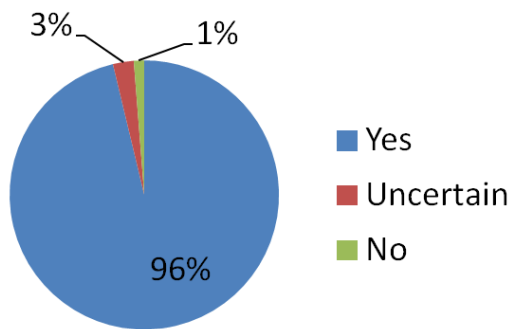


Figure 3-1: The profession of the respondents.

It can be seen that nearly half of these respondents pursue high-tech industry professions. Specifically, 8% of all the respondents declared that they are engaging in AR-related work.

Basic understanding of AR

**Do you know what
Augmented Reality (AR)
is?**



**Have you ever used any
AR applications?**

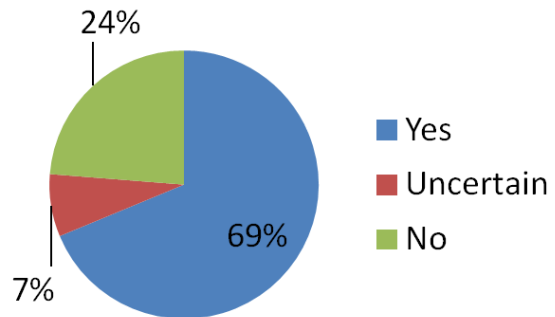


Figure 3-2: The respondents' understanding of AR.

For questions “Do you know what AR is?” and “Have you ever used any AR applications?”, the overwhelming majority of the respondents thought they knew what AR is (before a brief definition of AR was given after their answer to this question). After reading the definition, 69% respondents believed they had used AR applications before. Most of the AR applications used were AR games, especially the location-based markerless mobile game Nintendo’s *Pokémon GO*. Other applications can be divided into association tools, including navigation tools, collaboration tools and applications for the *Internet of Things* (e.g. *ThingWorx Studio*²²); education and training applications, including applications for school teaching, 3D flashcards for children, and various

²² **ThingWorx Studio:** <https://www.thingworx.com/platforms/thingworx-studio/>

training simulators (e.g. medical, defence, driving and flight); and “Other”, which mainly refers to various AR demos provided by some AR SDKs mentioned in Section 2.1.2. The statistics are shown in Figure 3-3.

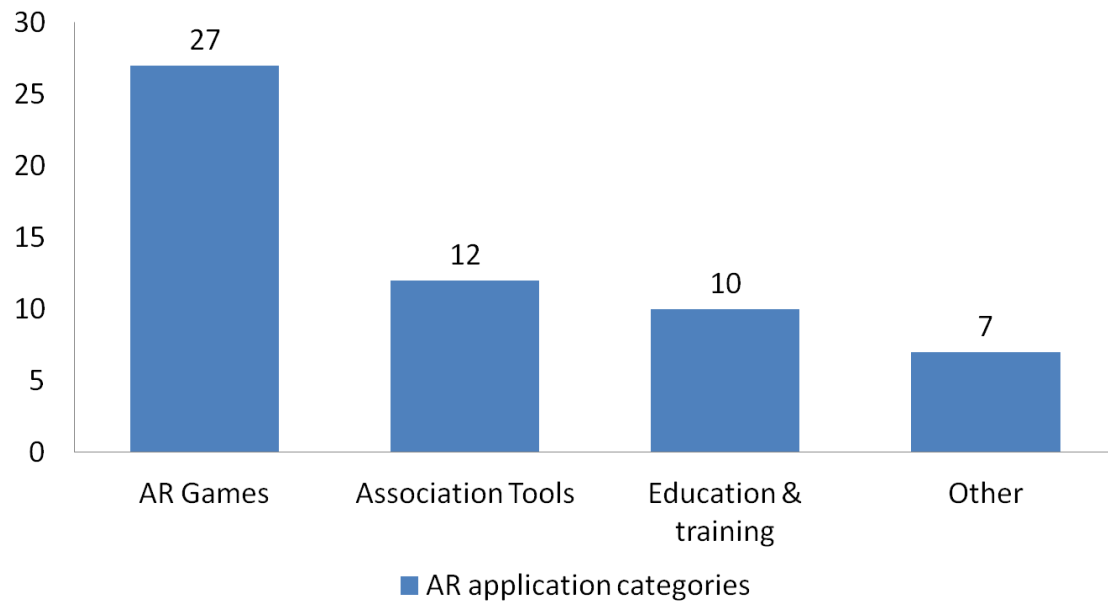


Figure 3-3: The categories of the AR applications used declared by the respondents.

AR application development intentions

Have you ever had a thought to apply AR in your own work or project?

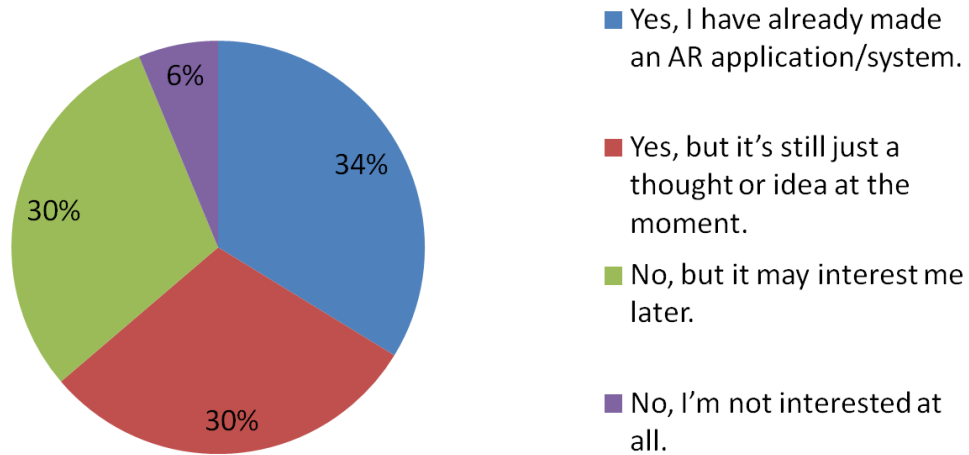


Figure 3-4: The intentions of the respondents to apply AR in their own works.

Figure 3-4 shows the results to the question asking whether the respondents had ever thought of applying AR in their own work or project. 64% of respondents declared having such a thought and more than half of them had already built an AR application or system before. 36% respondents never had such a thought before but most of them still expressed an interest in applying AR. Only 6% (five people) said they are not interested at all – two were working in the High-Tech Industry Profession, two in Business and Management, and the other one belonged to the “Other” category. The people who are not interested at all were not required to answer the following questions in the survey, but were invited to answer the final optional question: “If you have any other thoughts about AR, please write it below”. However, none of them gave an opinion. The remaining 75 respondents were asked which area they would like to develop AR for. The available options involved: Education or Training, Research, Tour guide for exhibitions or tourist attractions, Retail or Advertisement, and Entertainment.

The respondents were allowed to select more than one option and add other areas if they wished. The results are shown in Figure 3-5.

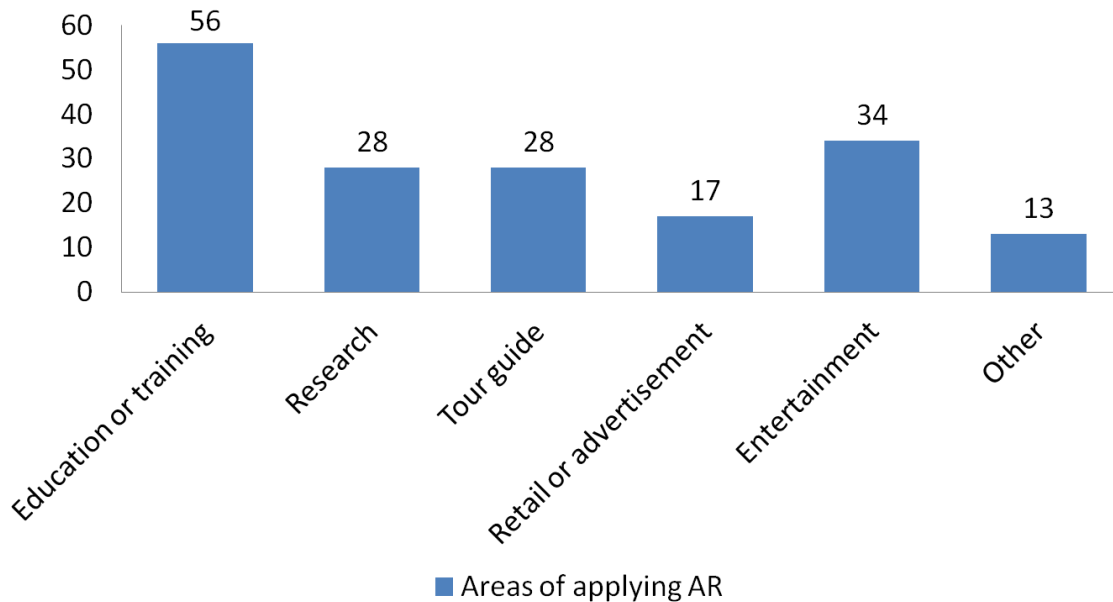


Figure 3-5: The areas that the respondents would like to develop AR for.

Most of the respondents indicated they would like to develop AR for Education or Training applications. The Entertainment category came second and there were also quite a number of people expressed a desire to develop AR for Research, Tour Guide, Retail or Advertisement. In the “Other” areas, there were four respondents who responded they would like to apply AR for defence, situational awareness (also a military/defence area of interest), three for medical, and one for each of manufacturing, data visualisation, telecommunications, collaboration, on-set cinematic, forensic security and interior design.

AR application development experiences

75 respondents who were interested in AR then progressed to the question of “Have you

ever developed an AR Application/System?” The answers are shown in Figure 3-6.

Have you ever developed an AR Application/System?

- No, but I have asked other people to develop one for use by me
- No.
- Yes, I have developed an application for my own use.
- Yes, I have developed an AR application for a third party.

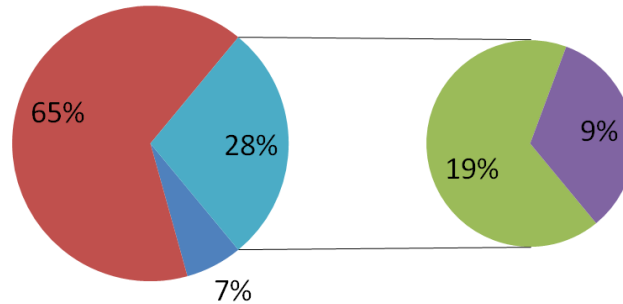


Figure 3-6: The AR development experience of the respondents.

There were 7% (four people) respondents who had asked other people to develop AR applications for them. Two of their applications were conducted in the military domain, and the other two focused on education and exhibition guide respectively. Then they were asked to rate the degree of satisfaction about their applications with 5 for Very Satisfied and 1 for Very Dissatisfied, as shown in Table 3-1.

Table 3-1: The satisfaction degree of the respondents on their applications.

Scores	1	2	3	4	5
Number of respondents			1	3	
Mean score (Variance)	3.75 (0.25)				

The reasons for satisfaction include “Immersive and engaging” and “new technology (was used)”. The reasons for dissatisfaction include “not quite what the client needed” and the quality of AR registration or display was “not very ideal”.

For 28% (21 people) of the respondents who had experience with AR application development, they described their applications as following: PhD research projects (three people), children-orientated Android application, system pod monitoring for mass transit, military, augmented marketing, mine site geological data assessment, emergency service situation management, showcasing the unique selling points of a product (two people), exhibition guide (two people), live home staging, medical and education (two people), AR wearable devices (two people), 3D reconstruction, fire control simulation, and machinery maintenance. Their satisfaction degrees on the developed applications are shown in Table 3-2.

Table 3-2: The satisfaction degree of the respondents on their applications.

Scores	1	2	3	4	5
Number of respondents		1	8	8	4
Mean score (Variance)	3.71 (0.71)				

The major reasons for satisfaction were cited as follows:

- AR just meets the requirement of the application
- The development tools were intuitive
- Good teamwork experience
- Technology breakthrough

On the other hand, the major reasons for dissatisfaction were cited as follows:

- The poor performance of the application caused by technological difficulties (such as unreliable tracking, hardware limitation)

- The development tools did not support the requirement
- The documentation of SDK was “messy”
- It was hard to craft a proper user interface for the application

Furthermore, these 21 respondents - with experience on AR application development - were asked to rate the importance of the factors influencing them in their evaluation of an AR SDK, as presented in Figure 3-7, They were also asked to select and rate the difficulty of the unfavourable situations they had encountered during the AR development, as presented in Figure 3-8.

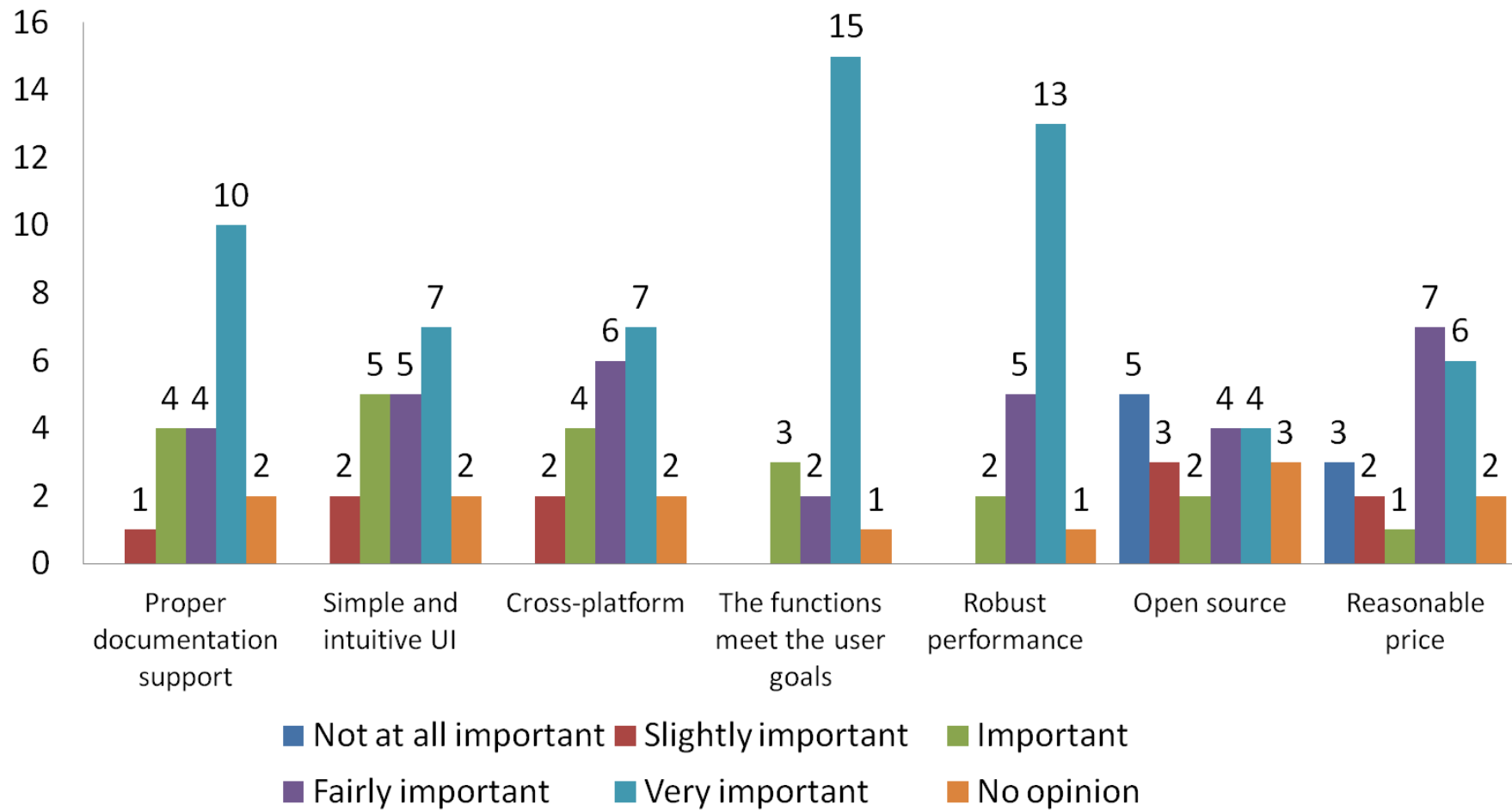


Figure 3-7: The importance of the factors that affect the respondents' evaluation of an AR SDK.

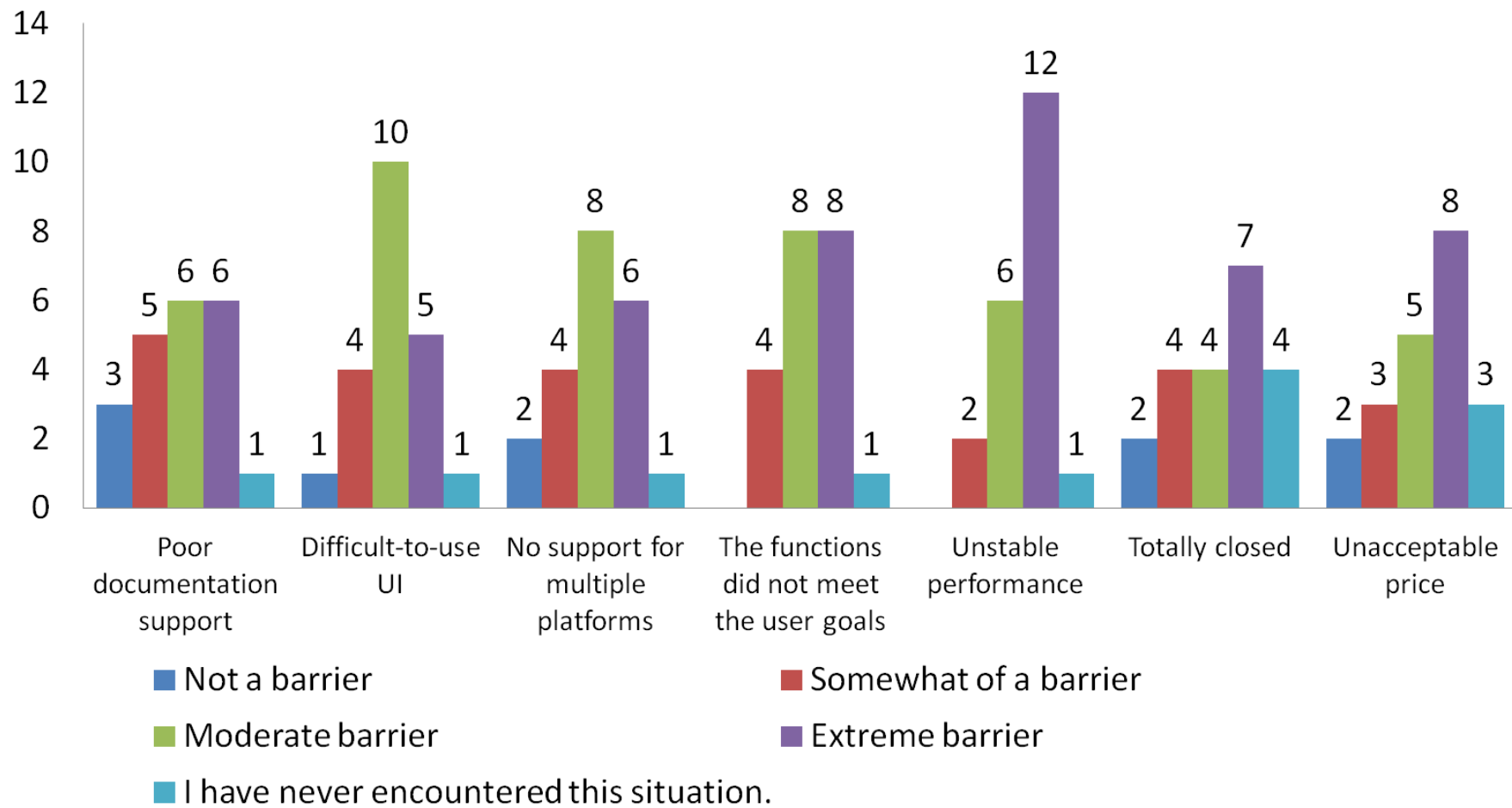


Figure 3-8: The difficulty of the unfavourable situations that impede the respondents on their AR development.

Some respondents also commented that --

- the SDK should be easy to integrate into routine site activities;
- the SDK should have an open and generic data interface which does not rely on a specific tracking technology or render/graphics engine;
- the price of SDK changes due to new release version, and the lack of support for the latest game engine release and device OS.

For each factor listed in Figure 3-7 and Figure 3-8, the mean values (and variances) of the scores on importance (5 for Very important and 1 for Not at all important) and difficulty (4 for Extreme barrier and 1 for Not a barrier) are calculated respectively and presented in Table 3-3. Note that **Importance scores I** does not consider the respondents who selected “no opinion”, while **Importance scores II** treats “no opinion” as score 0. Similarly, **Difficulty scores I** does not consider the respondents who selected “I have never encountered this situation.”, while **Difficulty scores II** treats them as score 0.

Table 3-3: The importance and difficulty scores on the factors that affect the respondents AR development experience.

Mean score (Variance)	Importance scores I	Importance scores II	Difficulty scores I	Difficulty scores II
Documentation support	4.21 (0.95)	3.81 (2.46)	2.75 (1.14)	2.62 (1.45)
User interface	3.89 (1.10)	3.5 (2.36)	2.95 (0.68)	2.81 (1.06)
Multiple platforms support	3.94 (1.05)	3.57 (2.36)	2.90 (0.94)	2.76 (1.29)
Suitable functions	4.60 (0.57)	4.38 (1.58)	3.15 (0.55)	3.00 (1.00)
Performance stability	4.55 (0.47)	4.33 (1.43)	3.5 (0.47)	3.33 (1.03)
Open source support	2.94 (2.52)	2.52 (3.26)	2.94 (1.18)	2.38 (2.35)
Price	3.58 (2.15)	3.24 (3.09)	3.06 (1.11)	2.62 (2.15)

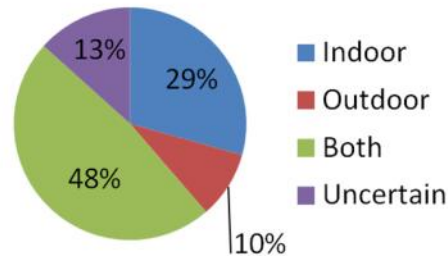
It can be seen that the respondents considered the extent to which an AR SDK could support the functions of meeting user goals as the most important factor, followed closely by performance stability. Open Source support was less important than other factors. The respondents encountered more difficulties with application performance and fewer difficulties with SDK documentation and Open Source support than other factors.

Use case requirements

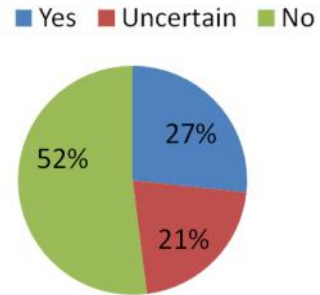
In this section, 75 respondents were asked to think about their needs and requirements in their specific case when applying AR. The questions and the responses are given in Figure 3-9. Most of the respondents required that their application could be used in both

indoor and outdoor environments. Indoor-centred came second, followed by outdoor-centred. 27% respondents would like to design their application for a specific place while 52% respondents would not. 21% respondents were unsure about this question. With regard to the tracking method, most of the respondents had no idea about it, followed by 29% respondents who required markerless tracking based on natural features of the original environment. There were also quite a number of respondents who decided to use marker-based tracking, followed closely by geo-based markerless tracking and hybrid tracking. One respondent selected “Other” and expected to use “flexible user anchor” for tracking. 42% respondents required accurate AR registration in their application, while 19% did not. However there were also 39% of respondents who were uncertain. Overall, there were a considerable number of respondents who were still not very clear with their use case requirement at the time the questionnaire was completed.

The general environment of your AR application is

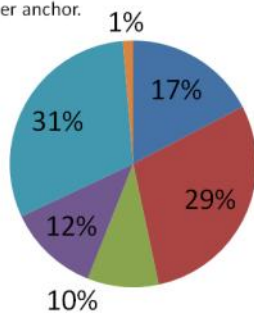


Is your application designed for a specific location?



The tracking method used in your application is

- Marker-based Tracking, based on artificial image markers or models.
- Markerless-based Tracking, based on natural features of the original environment.
- Markerless-based Tracking, based on geographic information.
- Hybrid Tracking.
- Uncertain.
- Other: Flexible user anchor.



Does your application require an accurate AR registration?

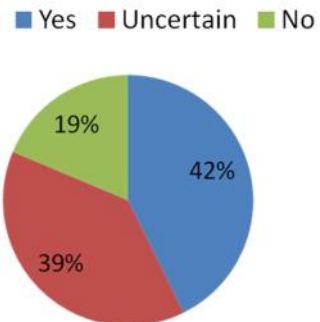


Figure 3-9: The AR use case requirements of the respondents.

Supplemental references

The final part of the questionnaire asked respondents to give their optional opinions on the importance of the factors that appear in Figure 3-7 relating to **general SDK** evaluation. The importance scores similar to Table 3-3 are given below as a comparative reference for AR SDK.

Table 3-4: The importance scores on the factors that affect the respondents evaluating a SDK.

Mean score (Variance)	Number of the responds	Importance scores I	Importance scores II
Documentation support	73	3.41 (0.65)	3.27 (1.09)
User interface	73	3.86 (1.54)	3.70 (2.07)
Multiple platform support	72	3.62 (1.89)	3.47 (2.34)
Suitable functions	73	4.22 (1.35)	3.99 (2.21)
Performance stability	73	<u>4.36</u> (1.02)	<u>4.18</u> (1.73)
Open source support	74	3.21 (2.23)	2.95 (2.82)
Price	73	3.77 (1.72)	3.62 (2.22)

Slightly different from Table 3-3, respondents considered performance stability as the most important factor when evaluating an SDK, although the importance of providing suitable functions to meet the end user goals scored well too. Generally speaking, Open Source support was still rated as less important than other factors.

3.1.2. Conclusion

80 volunteers from different professions answered an online questionnaire about their subjective opinions on AR application development and requirements. Most of them were shown to possess a basic understanding of AR and AR applications/systems. For 75 respondents who showed their interest in applying AR in their own works or projects, they were asked about their development intentions. The first two areas they expressed a

desire to apply AR were Education or Training and Entertainment. The respondents who had been involved in AR projects directly or indirectly described their applications, and showed a reasonable degree of contentment with their experience. 21 respondents who had experience of AR application development were asked to rate the importance and difficulty of several factors that might affect their AR development and evaluation of AR SDKs. They considered that an AR SDK's support for the functions of meeting user goals and robust performance as the most important factors, whilst they also encountered more difficulty on these two factors during actual development examples. With regard to the use case requirements, most of the respondents expected that their particular type of AR system could be used in both indoor and outdoor environments, not for a specific location, making use of markerless tracking based on natural features of the target environment, and requiring accurate AR registration. However, it should also be noticed that there were a quite number of the respondents who were not very clear when expressing their specific use case requirements

3.2. Geometric transformations

Geometric transformations are very frequently used operations in the vision-based AR process: in the sensing stage, the camera device will map the observed 3D objects to 2D image planes by using perspective projection transformation. The displacement of the camera between each two images is expressed by a 3D *rigid transformation* (also called *isometry* or *Euclidean transformation* since it forms the basis of Euclidean geometry (Galarza & Seade, 2007)). This camera motion consists of a rotation and a translation. In order to represent the camera location and the recovered spatial position of the image

features in a unified reference frame, a transformation with rotation and translation components between each camera reference frame and a pre-defined world reference frame needs to be established and applied. This can be obtained from its corresponding camera pose. The transformation within camera pose plays a particularly important role in both the offline training and online running stages. For offline 3D reconstruction described in Chapter 4, each input image is associated with a camera model. The camera pose of such a model is used to back-project 2D information from an image to the 3D space, restoring the spatial structure of the scene captured in the image. For online vision-based user tracking described in Chapter 5, the camera poses are used to register pre-defined virtual information into the world coordinate system. Then the augmentation can be rendered over the real-world view and subsequently projected onto the display screen. The specific methods used for pose estimation in the offline and online sessions are stated and discussed in Chapter 4 and Chapter 5 respectively.

In the present chapter, the basic knowledge of geometric transformations used throughout the thesis is introduced, including the definition of several reference frames (Section 3.2.1), the generic mathematical models of 3D rigid transformations (Section 3.2.2) and camera projections (Section 3.2.3).

3.2.1. Reference frames and coordinate systems

In order to describe the positions and orientations of objects in the 3D real world, it is important to define a *world reference frame* at the very beginning, in order to unify coordinate representation. One of the most common coordinate systems used in

computer graphics and image processing is the *Cartesian Coordinate System*, a.k.a. a rectangular coordinate system. The axes in the coordinate system are defined with two or three orthogonal vectors for 2D surface or 3D space respectively, and the unit lengths on each axis are equal (Heare & Baker, 1998). Unless stated, all coordinate systems mentioned below refer to the Cartesian Coordinate Systems. The world reference frame used in an AR system can be customised, which means it could be a geographic coordinate system (a common choice is *latitude, longitude and altitude system* (Crossley, 2015)) or an arbitrary 3D Cartesian coordinate system, depending on the specific aim of the different applications. For example, GPS / INS-based AR applications tend to use geographic coordinate systems as the reference frame, since GPS sensors can directly obtain their rough coordinates in latitude, longitude and altitude. In contrast, as mentioned in Section 2.1.1, developers of planar marker-based AR applications tend to select a reference frame with one of its axes perpendicular to the marker plane, as shown in Figure 3-10, in which the red-coloured axis is perpendicular to the marker plane (it became a red point at the centre of the right marker). In that way, the spatial coordinates of the points on the marker will have the same value on that axis, and other two-dimensional values are relatively easy to define by their image coordinates.

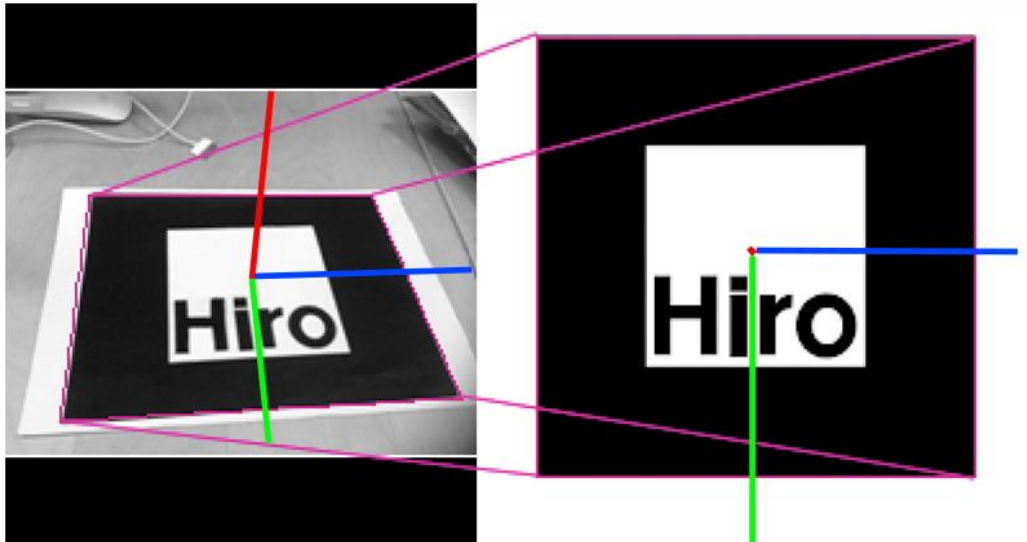


Figure 3-10: A reference coordinate system based on a rectangle fiducial marker.

In the present research, a world reference frame is needed to define the camera pose of each image and the global geometric structure. Again, as emphasised previously, the ‘camera’ refers to the pinhole camera model. Each camera model is associated with an individual image and can be defined with extrinsic and intrinsic camera parameters. The camera reference frame has been previously depicted in Figure 2-10, which is also a 3D Cartesian coordinate system that uses the optical centre as its origin and the optic axis as the Z-axis. The locations in images can be specified by using various coordinate systems and the pixel coordinate system is one of the most common choices, where the image is treated as a grid and each discrete unit represents a *pixel*. Since there are several different reference frames that exist in the workspace, an arbitrary 3D point P may have different coordinates in respect of different frame respectively. As depicted in Figure 3-11, the space point P can be described in either world coordinates P_W or camera coordinates P_C .

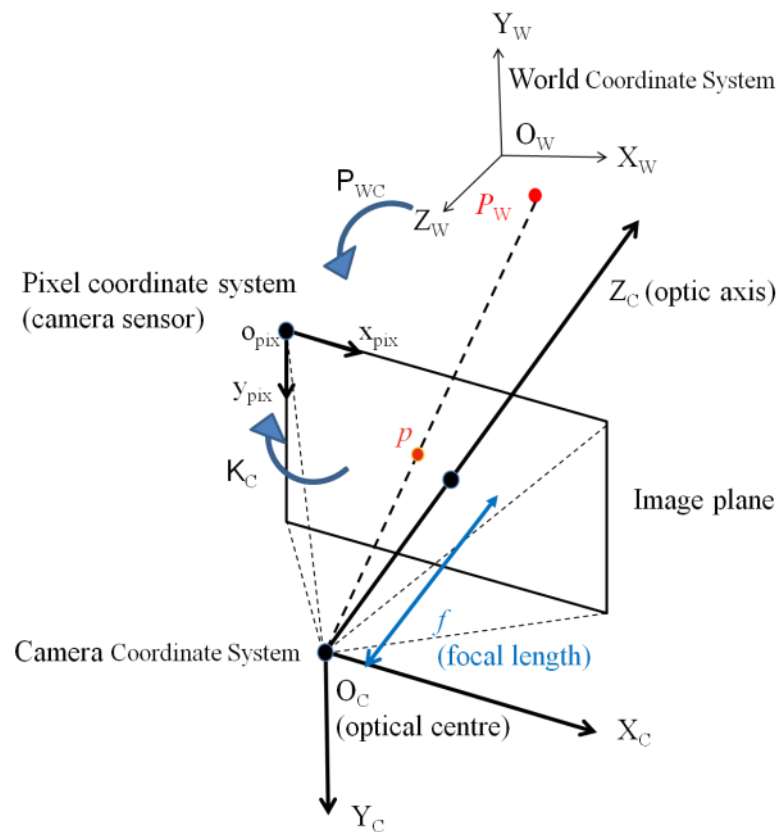


Figure 3-11: The transformations between world coordinate system, camera coordinate system and image pixel coordinate system.

Similarly if certain points in space are visible in multiple images, they may have different pixel coordinates in each image and their spatial positions in respect of each camera reference frame may also be different. The different coordinate representations of the same point can be treated as 3D-to-3D correspondences among the different cameras. These can be used to calculate the rigid displacement/motions between the camera models, and can further determine the camera pose of each individual image with respect to the world reference frame. Therefore, any given coordinates in the world coordinate system can be transformed to the camera coordinate systems with known camera poses. These are 3D-to-3D rigid transformations as described in Section 3.2.2 below. If the intrinsic camera parameters are known, then the projection transformation

can be performed between the camera coordinates and the image pixel coordinates, which are 3D-to-2D projections described in Section 3.2.3.

3.2.2. 3D-to-3D rigid transformations

3D rigid transformations can be applied to both objects and coordinate systems. For the former, the transformation is moving and rotating an object, like a camera, from one pose to another with respect to the fixed world reference frames. For the latter, the object is fixed, but for different reference frames it may have a different position and orientation, which can be calculated by using the transformation relations between the reference frames. In these cases, so-called 3D rigid transformations include rotations with a format of 3x3 matrices and translations with a format of 3D vectors. When such a transformation is applied to an object, the rotation matrix will change the orientation of the object's local frame and the translation vector will move the object with respect to its local frame. Rotations and translations can be combined into a 3x4 transformation

matrix $[R_{3 \times 3} \quad t_{3 \times 1}]$ or a 4x4 *homogeneous transformation matrix*
$$\begin{bmatrix} R_{3 \times 3} & t_{3 \times 1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

which represents a rotation followed by a translation.

The original 3D Cartesian coordinates (X, Y, Z) of space points, then, are represented by a 4D vector in *homogenous coordinates* by adding a fourth coordinate of 1 at the end: $(X, Y, Z, 1)$. This enables the use of a product operator for matrices to evaluate a sequence of translations and rotations (Chen, 2004). Therefore the arbitrary 3D point P in Figure 3-11 can be denoted by $P_W = (X_W \ Y_W \ Z_W \ 1)^T$ in the world coordinate

system and is being observed by a camera C . The alternative camera coordinates of this point are denoted by $P_C = (X_C \ Y_C \ Z_C \ 1)^T$ and can be obtained by left-multiplying a 4x4 homogeneous transformation matrix T_{WC} to world coordinates P_W :

$$P_C = T_{WC}P_W$$

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} \quad (3.1)$$

where R is the 3x3 rotation matrix and t is the 3D translation vector contained in the transformation T_{WC} between the world space and the camera space, which also implies the position and orientation of the camera with respect to the world reference frame (*i.e.* camera pose). The camera pose of each individual image can be calculated indirectly by chain-multiplying a series of relative transformations matrices together. An example is depicted in Figure 3-12:

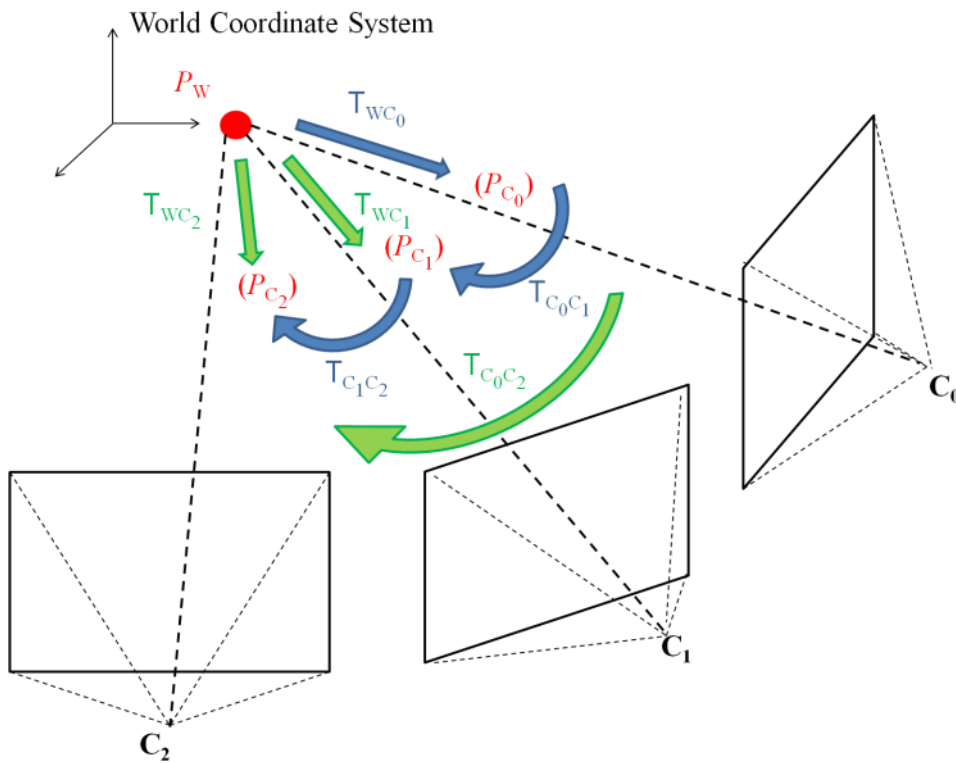


Figure 3-12: Space point P has four different coordinates with respect to the world reference frame and three different camera reference frames.

Suppose cameras C_0 , C_1 , and C_2 are observing a space point P which can be represented with world coordinates P_W or three different camera coordinates P_{C_0} , P_{C_1} and P_{C_2} .

Assume the transformation matrices from World to C_0 , C_0 to C_1 , and C_1 to C_2 are known as T_{WC_0} , $T_{C_0C_1}$ and $T_{C_1C_2}$, and then the transformations matrices from World to C_1 , C_0 to C_2 , and World to C_2 can be calculated by using the matrix product. It can be seen that the matrix multiplication allows transformations to be concatenated.

$$T_{WC_1} = T_{C_0C_1} T_{WC_0} \quad (3.2)$$

$$T_{C_0C_2} = T_{C_1C_2} T_{C_0C_1} \quad (3.3)$$

$$\begin{aligned}
T_{WC_2} &= T_{C_1C_2} T_{C_0C_1} T_{WC_0} \\
&= T_{C_1C_2} T_{WC_1} \\
&= T_{C_0C_2} T_{WC_0}
\end{aligned}
\tag{3.4}$$

3.2.3. 3D-to-2D camera projections

2D images are planar representations of the real world that can normally be obtained from a pinhole camera projection model. A 3D coordinate P can be mapped to a 2D pixel p in an image plane by using (3.5).

$$p = K T P \tag{3.5}$$

where K denotes a 3x3 (or 3x4 in homogeneous coordinates by adding a zero vector column in the right-most, as shown in (3.6)) *projection matrix* (a.k.a. camera intrinsic matrix) and T denotes a *view matrix* (a.k.a. camera extrinsic matrix) which is identical to the 3x4 or 4x4 homogeneous transformation matrix mentioned in Section 3.2.2. Specifically, two necessary transformations are performed here by multiplying these matrices, as shown in Figure 3-13.

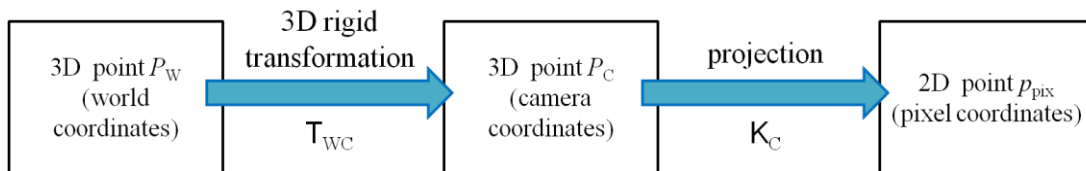


Figure 3-13: The process of mapping a 3D point from world coordinates to pixel coordinates.

Firstly the transformation matrix T_{WC} brings the world coordinates P_W from the world to the camera reference frame as camera coordinates P_C shown in (3.1). Then the

projection matrix K , which contains camera's intrinsic parameters determined by the actual camera device, projects the P_C to the image plane in pixel coordinates, as shown in (3.6).

$$w \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \quad (3.6)$$

where w is scaling factor (and it can be seen that $w = Z_C$), γ represents the skew coefficient between the x and the y axis, often be 0. An ideal pinhole camera model is affected by the focal length f , the shortest distance from the optical centre to the image plane, and the principle point (c_x, c_y) which is the point where the optic axis intersects the image plane. Since the aspect ratio of the pixel patterns may be uneven, focal length f is scaled by additional scale factors m_x and m_y in the x and the y direction respectively as $f_x = m_x \times f$ and $f_y = m_y \times f$. But in most cases the camera pixels are square and thus $f_x = f_y = f$.

Moreover, as mentioned in Section 2.2.1, real camera devices may produce lens distortion which should be taken into account. For distorted projection, the output pixel point can be corrected by applying Brown's distortion model (Brown, 1966) and five intrinsic distortion coefficients (comprising three radial factors d_1, d_2, d_3 and two tangential factors d_4, d_5) of the camera device, as shown below:

$$\begin{aligned} r_d &= \sqrt{(x_u - c_x)^2 + (y_u - c_y)^2} \\ x_c &= x_u(1 + d_1 r_d^2 + d_2 r_d^4 + d_3 r_d^6) + [2d_4 x_u y_u + d_5(r_d^2 + 2x_u^2)] \\ y_c &= y_u(1 + d_1 r_d^2 + d_2 r_d^4 + d_3 r_d^6) + [2d_5 x_u y_u + d_4(r_d^2 + 2y_u^2)] \end{aligned} \quad (3.7)$$

where (x_u, y_u) represents the undistorted pixel point directly calculated from (3.6). Its eventual position on the corrected output image should be (x_c, y_c) . *Vice versa*, the undistorted image can be calculated from a distorted image also (Figure 3-14).



Figure 3-14: An image with fisheye lens distortion before distortion correction (left) and after correction (right).

Chapter 4 3D Reconstruction for Template

Training

The AR framework proposed in the present research takes the form of a two-stage development, or “delivery” for two kinds of end-users: An offline training stage within a development framework is proposed for developers who wish to develop a particular AR application. An online AR running stage is proposed for final application users who wish to experience or exploit an AR experience. This chapter focuses on a 3D map reconstruction procedure inside the development framework.

According to the use case scenarios presented in the preceding chapters, the “particular AR application” created with this development framework is expected to be dedicated to a very **specific** place. Most of the existing visual SLAM-based AR applications do not focus on a specific workspace. They can learn visual information from a random place and start user tracking immediately for displaying AR content at runtime. However, this design is unnecessary in the present proposed framework because the users who will add augmentation and the user who will see augmentation are assumed to be different people. The concept here is quite like those location-based AR applications mentioned in Section 2.1.2, – especially those heritage site-related setups, in which the augmented content should be tightly coupled on the site location. However, those “one-off” application designs were starting at the bottom by lower-level developers who were engaged in AR-related research. They may seem elusive for people who expect to make use of their methods but have no knowledge of the inner techniques of AR. Hence **the proposed development framework attempts to build an**

easier interface between those higher-level developers and the markerless AR techniques. The higher-level developers who desire to add AR content relevant to a specific place with distinguishable visual features, such as the real use case relating to the National Marine Aquarium described earlier and in more detail in Section 6.2, only need to collect required image data (RGB or alternatively RGBD images or video-streams) from the target environment, and then allow the proposed offline system to learn and create the reference map (*i.e.* template). The developers then can set up customised virtual information (AR content) with respect to the template through another graphical user interface (GUI), as described in Section 6.1. Therefore, when the application end users visit this "specific place", the coupled AR content will be delivered to them. Since there is no special real-time requirement during the development – in fact, the accuracy of the template to the real world is deemed to be a more essential feature than processing time -the template training will be performed offline in this proposal, and will be referred to as **Offline Session**.

The template stores the visual and geometric information extracted from the user-specified workspace, which will be used for user tracking task during runtime of the AR application. The natural feature-based methods are chosen in the present proposal since the vision-based approaches basically only use RGB images as an input source. These are of relative low-cost and can be easily accessed by general public. Moreover, compared with the marker-based methods which require introduce artificial markers into the environment, the natural feature-based methods are less intrusive and more flexible. Without introducing an additional object, the training process has to learn visual features from target environment as a template and identify their spatial

information qualities (*i.e.* the 3D location) to construct a reference map, enabling the application developers to set up the augmented information with respect to this virtual map. **The present research proposes a semi-automatic template training strategy based on vision-based 3D reconstruction/mapping techniques.** Instead of 3D mesh surface models, the sparse 3D point clouds will be reconstructed from the input image data as a reference map. These cloud points are provided by feature correspondences of the matched images and are represented by the same descriptors which make them distinguishable landmarks for identifying and tracking.

Two effective vision-based 3D reconstruction methods are discussed in the remainder of this Chapter – known as RGBD-SLAM (Section 4.1) and SfM (Section 4.2). The main difference between these two methods is the requirement of the **input data** and the **applied environment**. RGBD-SLAM requires a continuous ordered RGBD dataset as input and the recent low-cost RGBD camera (*i.e.* Kinect 1.0) only works reliably indoors; SfM uses a general RGB image dataset as input, which is unnecessarily ordered but the *baseline* (the distance between the camera’s centres of projection) between images should be big enough. It can also deal with both an indoor and outdoor environment. Two representative applications of these two methods respectively, **RGBDSLAM v2** (Endres *et al.*, 2014) and **VisualSfM** (Wu, 2011; 2013) are applied in this proposal. Their performance of 3D reconstruction with different configurations (e.g. the visual features used) is tested and evaluated with public datasets in Section 4.3. The output will then be further used for AR application template training. The details on how to use them for user tracking will be introduced and discussed in Chapter 5.

4.1. RGBD-SLAM

RGBD-SLAM is one of the two approaches used in the present research for 3D reconstruction in the offline session. This kind of SLAM is based on a sequence of continuous RGB images or video stream with synchronously associated depth information (a.k.a. RGBD data). The RGBD-SLAM process can be described as moving a handheld / robot mounted RGBD sensor (e.g. Kinect) around an unknown workspace, creating the map, and locating the sensor by analysing the acquired RGBD data from the sensor at a certain time step. The sampling rate of most RGBD cameras can actually reach 30 frames per second, but the practical time step will depend upon the speed of the camera motion and the processing rate of the SLAM system, which should ensure an overlap between adjacent frames. A frame of RGBD data consists of a colour image captured by a common pinhole camera and a depth image in which each pixel contains a depth value and is associated with a pixel in the colour image. With these two associated images and intrinsic camera parameters, the visual and geometric information contained in a RGBD frame can be recovered to 3D space with respect to its local camera reference frame. The RGBD-SLAM process will determine the camera motions between frames, thus the local reconstruction of each RGBD frame can be brought to a unified world reference frame to develop a global reconstruction. The basic flow of a graph-based RGBD-SLAM is shown in Figure 4-1, and the detailed process is given below.

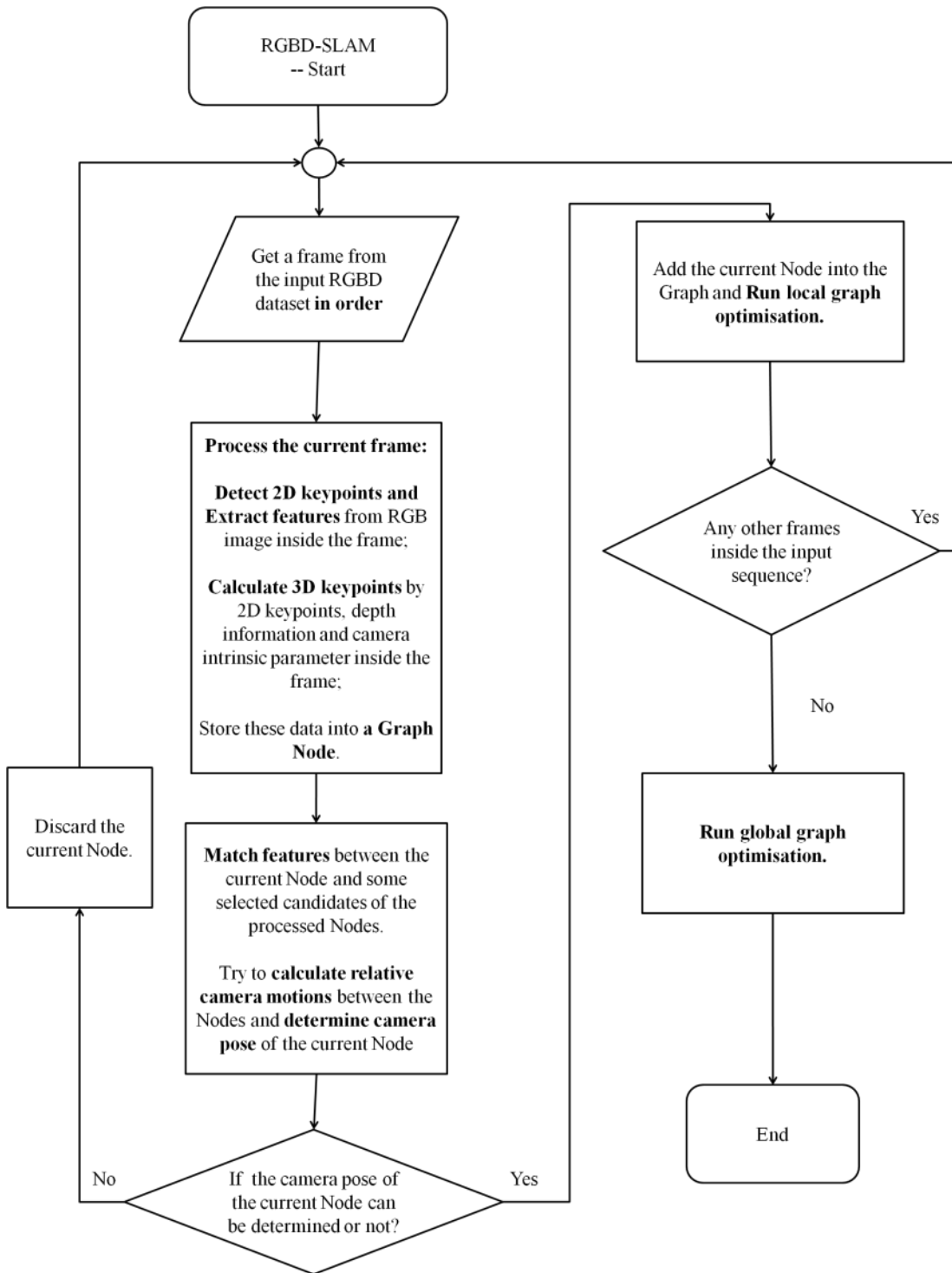


Figure 4-1: The flowchart of graph-based RGBD-SLAM.

Basically RGBD-SLAM belongs to the visual SLAM process referred in Section 2.2.3.2 which uses the visual features within the colour images as landmarks. The matchable

visual features of processed images are firstly recovered to 3D space (this is generally triangulated from the correspondences of first two input images if this is a monocular SLAM, but for RGBD data the depth of each image pixel is already known). If there are enough **landmarks** can be observed in the new input image, the system will make use of 3D-to-3D correspondences to estimate the new camera pose with respect to the old ones, and then the 3D feature points of the new image that have not appeared before will be transformed to the same coordinate as new landmarks. The procedure of visual SLAM is depicted in Figure 4-2: the initial camera model and the observed visual features in the image are put into a pre-defined world reference frame. These 3D feature points become landmarks. The pose of the following camera models with respect to the world can be estimated by observing enough existing landmarks, and their own features will also be reconstructed into the world reference frame to be used as landmarks.

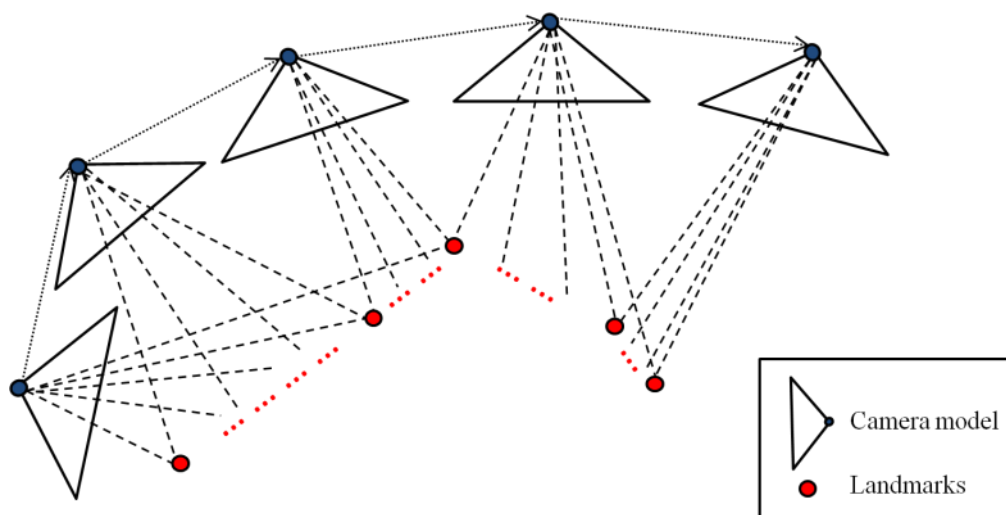


Figure 4-2: An ideal (without noise and error) procedure of visual SLAM.

If a large area (e.g. several rooms or a long street) is covered by the input image data, the number of landmarks can become very large as well. Instead to compare all

landmarks one by one, the visual features of a new input image will only compare with a portion of them. For example, that portion may include the features in the image's immediate predecessors, since there are overlaps between adjacent frames and, thus, there is a relatively high possibility that some identical visual features will be observed. The RGBD frame can provide 3D coordinates of each feature with respect to its own camera system. Thus a set of 3D-to-3D correspondences can be established by performing feature matching between two visual images, and this will be used to estimate their relative camera motions. Since the camera poses of the old images have already been estimated, the new camera pose can be obtained by multiplying their relative camera motions to the older one's camera pose (see expressions (3.2 – 3.4)).

Although comparing with an immediate predecessor in the sequence will result in a relatively high possibility to find feature correspondences, noise and other inaccuracies are usually contained in the observed measurements (*i.e.* the feature points and their depth). If the pose estimation of each camera only relies on its predecessor, the errors will accumulate rapidly and lead to failure of SLAM. In order to avoid accumulation of error, a graph-based approach is applied where SLAM is formulated into an intuitive graph form (a.k.a. pose graph) whose nodes correspond to the camera poses of input frames and whose edges represent constraints between the poses. The core aim of using a pose graph is to find the spatial configuration of the nodes in order to keep them consistent with the measurements modelled by the edges (Grisetti *et al.*, 2010). In this case, the node of new input data will not only rely on its immediate predecessors, but will compare with a series of selected candidates which may potentially match. If the relative transformation between two nodes can be determined, this transformation will

be used as a constraint edge linking the nodes. The optimal pose is estimated due to the configuration which minimise the error of all those constraints on that pose node, referred to as graph optimisation. The related mathematical models are introduced in Section 4.1.1.

Based on the above approaches, an ROS system **RGBDSLAM v2**, which implements a graph-based RGBD-SLAM, is applied to achieve a robust 3D reconstruction in this section. In order to be used in later online sessions, the output data and parts of the processing procedures of this system have been customised and modified. The design and implementation of RGBD-SLAM used in the present research is described in Section 4.1.2. The form of the generated results is presented in Section 4.1.3.

4.1.1. Graph-based SLAM

Grisetti *et al.* (2010) state that the SLAM problem is usually described by probabilistic formulation with a consideration of the noise inherent in the sensor measurement. The camera pose along the moving trajectory can be denoted by an arbitrary initial position \mathbf{x}_0 and a sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, \dots, \mathbf{x}_T\}$, conditioned by odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, \dots, \mathbf{u}_T\}$ which indicate the relative 3D transformations between camera models. The transition model $\mathbf{p}(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)$ represents the probability that the camera pose at time t is in \mathbf{x}_t given that the camera pose at time $t-1$ was in \mathbf{x}_{t-1} and the odometry measurement between them is \mathbf{u}_t . If the SLAM process is only based on vision sensors, the measurements of $\mathbf{u}_{1:T}$ cannot be acquired directly but can be

estimated based on the sensor's data of environment, denoted by $\mathbf{s}_{1:T} = \{\mathbf{s}_1, \dots, \mathbf{s}_T\}$, which represents the observations (*i.e.* captured images) of the workspace. The probability of performing the observation \mathbf{s}_t only depends on the camera pose \mathbf{x}_t and the constructed map \mathbf{m}_t (constituted by landmarks) up to time t ; thus the observation model is $\mathbf{p}(\mathbf{s}_t | \mathbf{x}_t, \mathbf{m}_t)$. The eventual aim here is solving the conditional probability of each camera pose \mathbf{x}_t given all the measurements and constructing the whole map of the environment. This problem can be abstracted to a graph-based SLAM where the camera poses are represented by nodes \mathbf{x}_i and the raw sensor measurements are replaced by "virtual measurements" edges \mathbf{z}_{ij} which connect the nodes \mathbf{x}_i and \mathbf{x}_j in a graph and contains a probability distribution over the relative locations of the two poses, conditioned to their mutual measurements, as shown in Figure 4-3 where each circle represents a camera pose node \mathbf{x}_i and the edges connect the nodes are spatial constraints arising from observations \mathbf{s} .

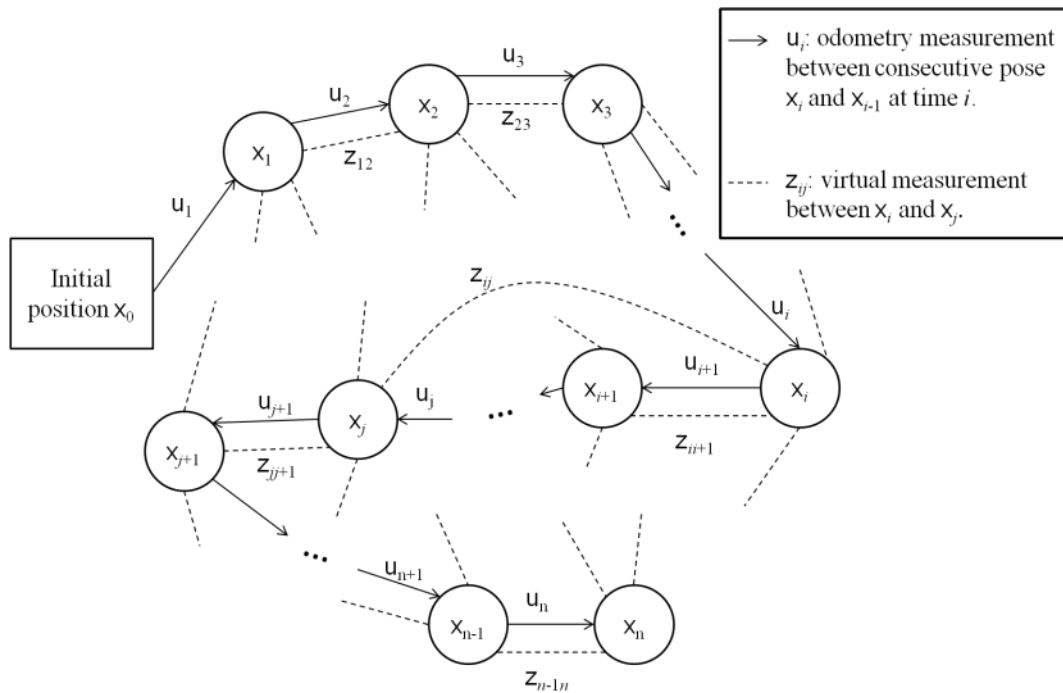


Figure 4-3: A pose-graph representation of a SLAM process.

The virtual measurement is a transformation that makes the observations acquired from \mathbf{x}_i maximally overlap with the observation acquired from \mathbf{x}_j . Assume the predicted measurement which meets the observations of \mathbf{x}_i and \mathbf{x}_j is \hat{z}_{ij} . The *error function* of this edge can be defined as

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = z_{ij} - \hat{z}_{ij}(\mathbf{x}_i, \mathbf{x}_j) \quad (4.1)$$

where z_{ij} is the real measurement of transformation between \mathbf{x}_i and \mathbf{x}_j . The edges also have another property – *information matrix* Ω_{ij} , which interprets the strength of the link/constraint (*i.e.* the weight of this edge). Then the negative log-likelihood $\mathbf{F}(\mathbf{x})$ of all the observations is defined as:

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i, j \rangle \in C} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, z_{ij})^T \Omega_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, z_{ij}) \quad (4.2)$$

where \mathbf{x} is a vectors with all camera poses as its parameters, C is a set of pairs of indices $\langle i, j \rangle$ for which a constraint/edge z_{ij} exists. Therefore the configuration of the nodes \mathbf{x}^* that minimises $\mathbf{F}(\mathbf{x})$ is found by solving the following equation:

$$\mathbf{x}^* = \arg \min_{\mathbf{x}} \mathbf{F}(\mathbf{x}) \quad (4.3)$$

The goal here is configuring all camera poses together to meet the observations as much as possible. Because noise and error always exist within the virtual measurement, some observations may lead to ambiguities of resulting in wrong edges between different poses which do not really exist. Trying to identify more constraints between the nodes from their observations can effectively reduce the side effect caused by the wrong edges, thus making the estimation result more reliable. This process is also known as a graph optimisation problem. In the present proposed system, this problem is dealt with by g2o

(Kümmerle *et al.*, 2011) – an Open Source framework for optimizing graph-based nonlinear error functions. Further detail is given in Section 4.1.2.4.

4.1.2. System implementation

RGBDSLAM v2 is implemented an ROS system for mobile robot applications, as well as being launched on Ubuntu Linux system with an ROS client. The GUI of RGBDSLAM v2 is shown in Figure 4-4: the application is trying to reconstruct an office scene from a set of RGBD data. The first row of the window is visualising the reconstructed dense point cloud and the 3D trajectory of the RGBD camera movement. The four sub-windows in the second row from left to right illustrate the current processing of the RGB image; the associated depth image; the representation of detected features and their responses; the representation of optical flow of the features against the last processed image (the red arrows represent inliers and the blue represents outliers).

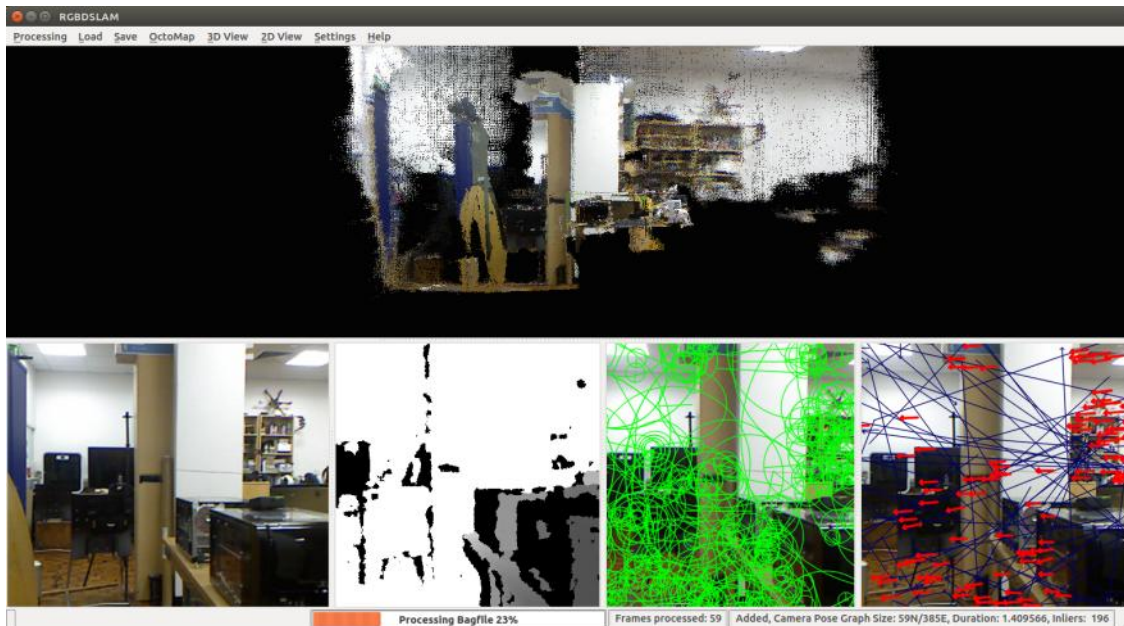


Figure 4-4: The GUI of RGBDSLAM v2.

One of the ROS packages *openni_launch / freenect_launch*, is used to launch the OpenNI driver and publishes the topics of the Kinect data in real time. These topics – colour images, depth images and camera information (including intrinsic camera parameters) – are being subscribed by another ROS node *rgbdslam* – the main process of this system. Alternatively the published messages can also be recorded into a *ROS bag* for later play back. The sequence diagram of the whole session is depicted in Figure 4-5.

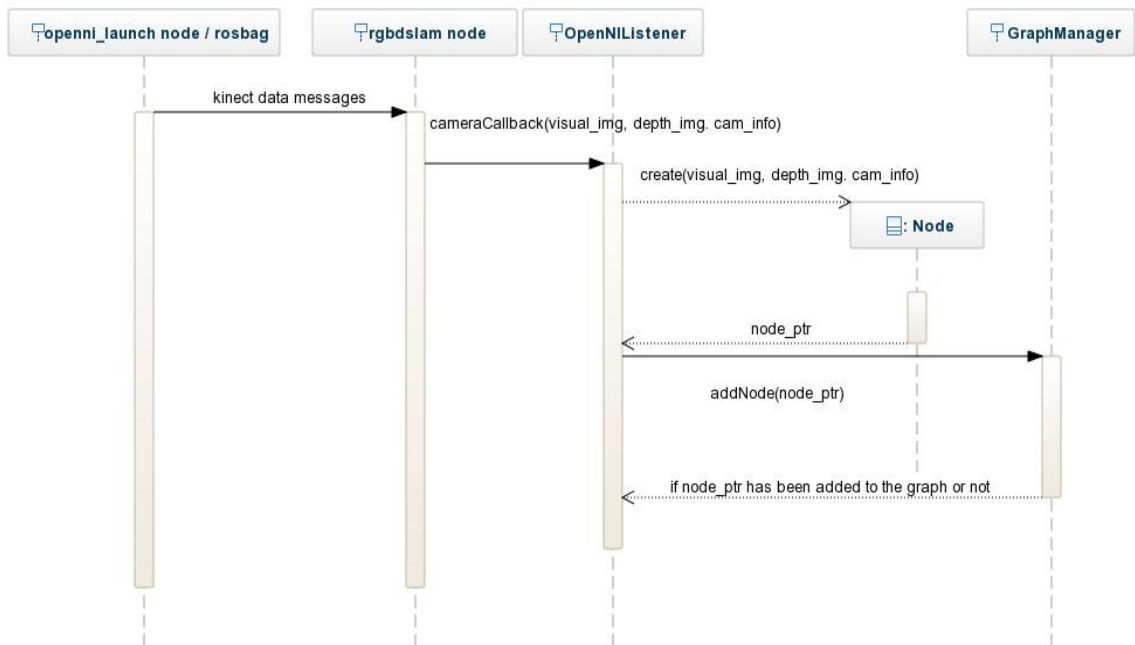


Figure 4-5: The sequence diagram of RGBDSLAM v2.

The *rgbdslam* node will create the instances of class *OpenNIListener* (hereinafter referred to as listener) and *GraphManager* (hereinafter referred to as graph manager) once it launches. The listener will track the specific topics published by a real device or the recorded ROS bag, as mentioned above. At each step of time (labelled by the timestamp of ROS), the synchronous messages are picked up from the topics of colour

images, depth images and camera information and bounding together for creating an instance of class *Node*. Each instance of *Node* (hereinafter referred to as node) encapsulates an RGBD input frame, which will be described in Subsection 4.1.2.1.

The nodes created by the listener are then delivered to the graph manager to generate a graph representation of the map which contains the corresponding camera pose of each input node. A world reference coordinate system will be defined first. In order to simplify the process somewhat, the camera reference frame of the first node x_0 is usually taken as the (world) reference; otherwise the relative transformation between the world and the first node should be given in advance. To estimate the camera pose of the new incoming node, the graph manager compares it with some previous nodes that have already been added to the graph, to decide if the new node will be added or not. The candidates with which the comparison is undertaken are selected via the strategy defined in Subsection 4.1.2.2.

The system will compare the colour image of the new node with the selected old ones. If there are enough inlier correspondences between two frames, a set of 3D-to-3D coordinate correspondences can be determined. These correspondences are used to calculate the relative transformation between the nodes for data association, which is described in Subsection 4.1.2.3.

The ultimate pose estimation of the new node relies on graph optimisation, where the estimated transformations to other nodes are used as constraints. Here RGBDSLAM v2 employs the g2o framework to solve the problem, introduced in Subsection 4.1.2.4.

4.1.2.1 Node construction

The node is created from a synchronised pair comprising of a RGB image and a depth image with known intrinsic camera parameters. It contains the spatial structure of the view being observed with respect to a local camera coordinate system described in Section 2.2.1. When a new node is being constructed, visual features are extracted from the colour image by using one of the feature detection methods (e.g. SIFT, SURF, ORB and so on; the detail of feature detection and matching will be discussed in Section 4.3.1). Then their corresponding 3D positions to the own-camera coordinate system can be calculated from the associated depth image and the intrinsic parameters. Assume the 3D coordinate (X, Y, Z) corresponds the image point (u, v) , then:

$$\begin{cases} Z = \mathbf{dep}(u, v) * s_d \\ X = (u - c_x) * \frac{Z}{f_x} \\ Y = (v - c_y) * \frac{Z}{f_y} \end{cases} \quad (4.4)$$

where $\mathbf{dep}(u, v)$ is the corresponding depth value of the pixel (u, v) , and S_d is the depth scaling factor – for Kinect it returns the real distance and hence the value of S_d is 1. The intrinsic camera parameters, which include principle point (c_x, c_y) and focal length f_x, f_y introduced in Section 2.2.1.1, are provided by the topic of camera information which can be previously set through camera calibration. In fact, (4.4) can be considered as an inverse of the expression (3.6). Therefore a node will store the 2D image coordinates and the 3D camera coordinates of extracted features with the descriptors used for matching. After being added to the graph, each node structure will be correlated to the

g2o data structure *Vertex* and *Edge* which will represent the camera pose estimation and the relative transformations between nodes respectively, described in Section 4.1.2.4.

4.1.2.2 Neighbour and loop closure search

Individual estimation of a node is generally noisy. Thus, for increasing the robustness of map construction and reducing drift errors, it is desirable to establish a reasonable number of constraint edges between nodes. In theory, all possible edges can be identified by applying brute-force comparison to all nodes to produce a dense graph. However this linearly increases the computational expense of both feature matching and graph optimisation with the number of comparisons, and the costing will become unacceptable as the graph becomes larger. In practice a sparse graph is preferred, which only adds necessary edges to the graph to ensure accuracy and efficiency. As discussed above, the nearest predecessors have a relatively high possibility of matching the new node, but can be more easily affected by accumulating error. For this reason, it is desirable to find the virtual measurements (*i.e.* the relative spatial transformation) from the current processing node to earlier nodes (*i.e.* loop closures). Loop closure detection is a classical problem in SLAM when attempting to recognise a previously-visited known place. In this case, if the new node can match a previous node, which means there is sufficient overlap between their observations (images), a loop closure is considered found.

RGBDSLAM v2 employs a strategy for selecting three different types of candidate nodes to compare, called *Predecessors*, *Neighbours* and *Random Samples*. Predecessors refer the immediate predecessors of the new node, and the other two are used for

determining much older loop closures. The k_P last added nodes will firstly be selected into the set of Predecessors. Neighbours refer to the geodesic/graph neighbourhood of the last added node (*i.e.* the direct predecessor of the new node), which means that the nodes under the set of Neighbours should be able to be visited from the last nodes through the established edges in limited depth. This is attained by computing a minimal spanning tree of the limited depth of the graph where the last added node is used as the root. The k_N nodes from the neighbourhood will be drawn randomly into Neighbours with a bias towards earlier frames for finding older loop closures. Other k_{RS} candidates of the last type were randomly sampled from the reminder nodes in the original version of RGBDSLAM v2. However, if the image sequence is sampled at a high rate and the speed of camera motion is relatively slow, it will be not very effective to randomly select candidates among all nodes for loop closing, due to the duplicated content existing among the adjacent images. Instead, a set of **keyframe** nodes is selected for sampling. In the present proposal, the keyframes are selected during the graph construction. Once the pose of new added node is estimated, the system will calculate the displacement distance and the rotation magnitude between the new node and the last

keyframe. Assume the transformation matrix between two nodes is
$$\begin{bmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

the displacement distance is $\sqrt{t_0^2 + t_1^2 + t_2^2}$ and the rotation magnitude is $\arccos\left(\frac{r_{00} + r_{11} + r_{22} - 1}{2}\right)$. If the camera motion changes over a given threshold (e.g.

0.2m for translation or 60° for rotation), or there is no valid transformation can be estimated between the new node and any old keyframe, the new node cannot match any

old keyframes, the new node will be selected as a new keyframe. By sampling from the subset of keyframes, it can avoid comparing the new node with a set of similar nodes, and increase the possibility to detect unknown loop closures. All three types of candidates have been shown in Figure 4-6.

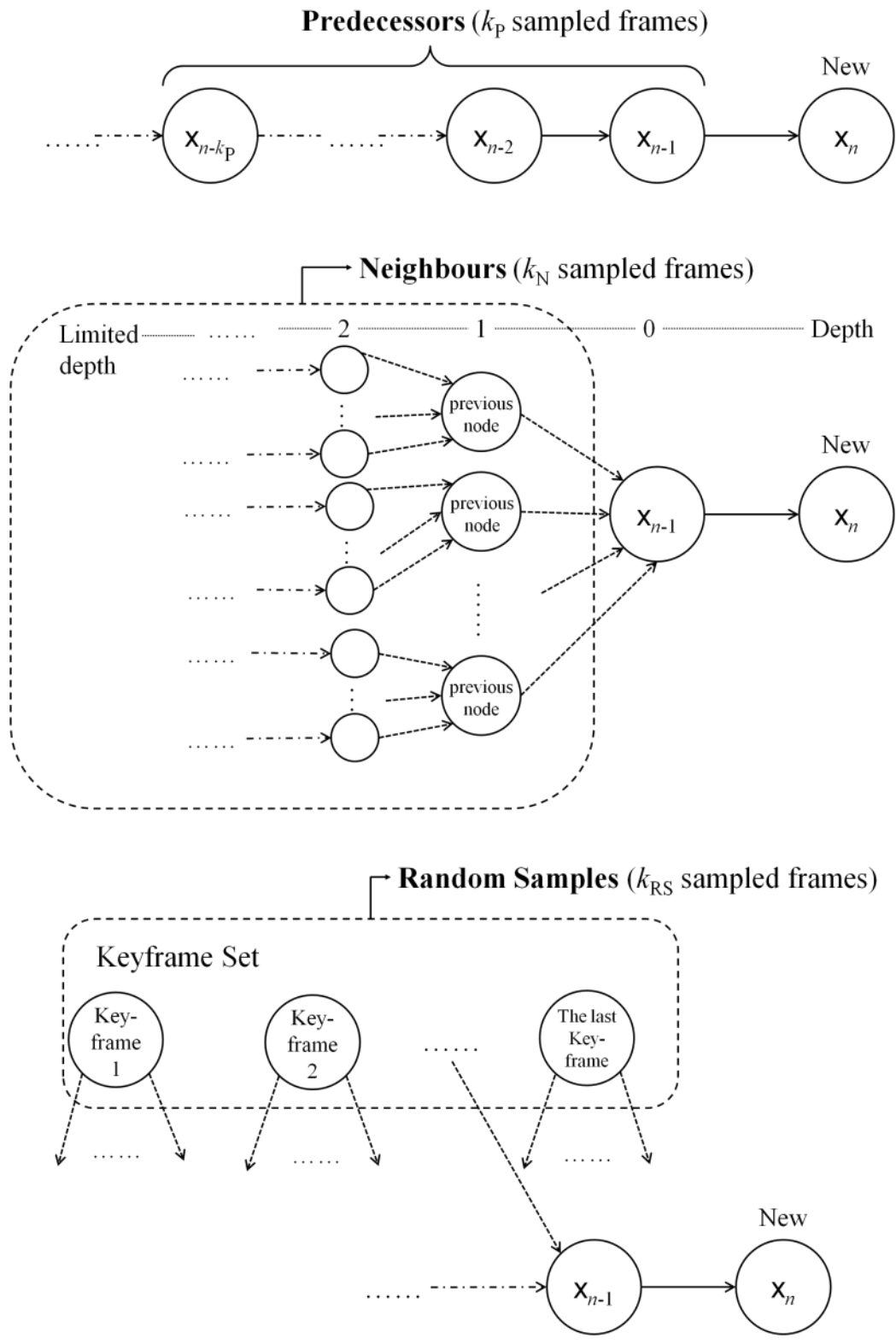


Figure 4-6: The strategy of RGBDSLAM v2 for loop closure detection by selecting three types of candidates from older Nodes.

4.1.2.3 3D-to-3D transformation

In order to estimate the camera pose, the new node will compare with the candidates selected in Section 4.1.2.2 one by one. A feature correspondence is related to same 3D point in space, but with different 3D coordinates with respect to their camera coordinate systems respectively, as shown in Figure 4-7. Note that space points do not change the position, but the camera coordinate system was rotating and translating along the camera motion with respect to the world.

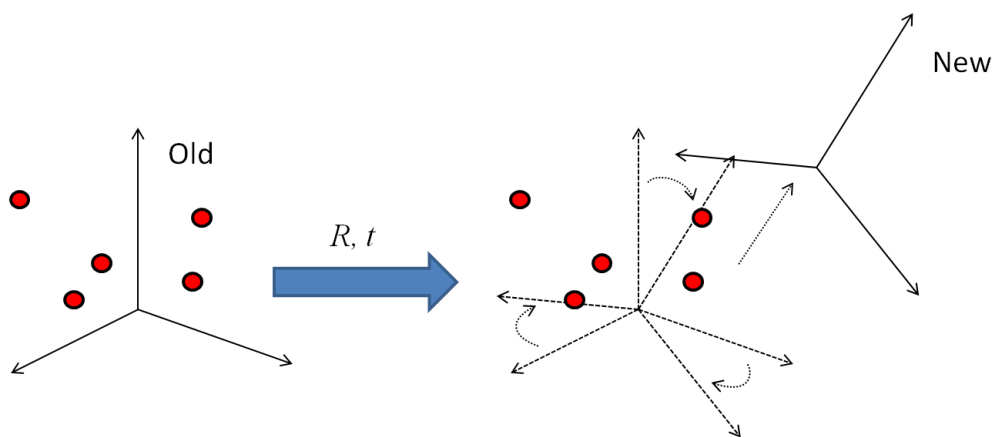


Figure 4-7: The coordinates of same space points are changed due to the different reference frames.

There is a rigid transformation between these 3D coordinate correspondences as stated in expression (3.1), which also indicates the relative transformation between these two camera coordinate systems. The rigid transformation can be estimated from 3D-to-3D correspondences as follows: assume the feature points of the i -th match between the old node and the new node are denoted by p_{old}^i and p_{new}^i respectively, and their corresponding 3D point coordinates are denoted by P_{old}^i and P_{new}^i respectively. The rigid transformation stated in (3.1) can be decomposed into a rotation component R and a translation component t . Thus in this case, (3.1) is expressed as (4.5).

$$P_{\text{old}}^i = RP_{\text{new}}^i + t \quad (4.5)$$

where the 3x3 matrix R and 3-vector t need to be determined. A common way to solve this is using SVD (Besl & McKay, 1992; Eggert et al., 1997). To implement this, two centroids of n matched coordinates P_{old} and P_{new} are calculated first:

$$\begin{aligned} \text{centroid}_{\text{old}} &= \frac{1}{n} \sum_{i=1}^n P_{\text{old}}^i \\ \text{centroid}_{\text{new}} &= \frac{1}{n} \sum_{i=1}^n P_{\text{new}}^i \end{aligned} \quad (4.6)$$

Then the translation component is removed by bringing both centroids to the origin. The re-centred coordinates are defined in (4.7).

$$\begin{aligned} P_{\text{old}}^i &= P_{\text{old}}^i - \text{centroid}_{\text{old}} \\ P_{\text{new}}^i &= P_{\text{new}}^i - \text{centroid}_{\text{new}} \end{aligned} \quad (4.7)$$

In order to solve the optimal rotation component between P'_{old} and P'_{new} , the least squares error criterion given by (4.8) should be minimised.

$$\begin{aligned} \Sigma^2 &= \sum_{i=1}^n \left\| P_{\text{old}}^i - RP_{\text{new}}^i \right\|^2 \\ &= \sum_{i=1}^n \left(P_{\text{old}}^{i\text{T}} P_{\text{old}}^i + P_{\text{new}}^{i\text{T}} P_{\text{new}}^i - 2P_{\text{old}}^{i\text{T}} RP_{\text{new}}^i \right) \end{aligned} \quad (4.8)$$

This equation is minimised when the last term is maximised, which can be represented as $\text{trace}(RM)$. The 3x3 correlation matrix M is defined by:

$$M = \sum_{i=1}^n P_{\text{new}}^i P_{\text{old}}^{i\text{T}} \quad (4.9)$$

Applying SVD to M as

$$M = USV^{\text{T}} \quad (4.10)$$

where U and V are 3x3 unitary matrix and S is a 3x3 real positive matrix. The optimal rotation that maximises the trace(RM) can be obtained from U and V , as shown below:

$$R = VU^T \quad (4.11)$$

With optimal R , the optimal t can be calculated from (4.5), and then the desire transformation matrix between the old node and new node is obtained.

Consider that the mismatch errors may exist and affect the accuracy of transformation, RANSAC referred in Section 2.2.3.1 is used here to remove the outlier correspondences. First a subset of correspondences is randomly selected and used to estimate an initial transformation. Then this transformation will be used to test the remaining correspondences. If enough correspondences satisfy the transformation, these correspondences will be involved in estimating a new transformation which will be used to test other correspondences. Otherwise another subset of correspondences will be randomly selected and the above steps will be repeated. This will be performed iteratively until the number of inliers no longer changes or a pre-defined maximum iteration times have been arrived at.

The graph manager will add the new node into the graph if any valid transformation is established. There may be several transformations from the new node to different old nodes. An initial camera pose estimation related to the world coordinate is estimated by multiplying the relative transformation to the known old camera pose (see equation (3.4)). Furthermore, RGBDSLAM v2 employs g2o framework to decide the ultimate pose of each nodes, known as graph optimisation.

4.1.2.4 Graph optimisation

RGBDSLAM v2 employs a g2o framework which provides Vertex and Edge data structure to handle the pose graph. The graph manager tries to establish the edges between the nodes, passing them to the g2o optimiser to generate a globally consistent map. The association relation between Vertex, Edge and the custom class Node mentioned above is given in Figure 4-8.

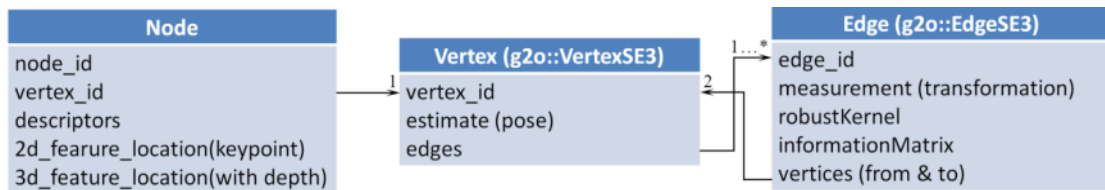


Figure 4-8: The class graph between the custom class Node of RGBDSLAM v2 and the class of Vertex and Edge used for graph-based optimisation within g2o framework.

The poses of camera nodes are stored in the instances of Vertex, and are restrained by virtual measurements (*i.e.* the estimated transformations to other nodes), which are held by Edge instances. The graph representation here is identical with the form in Section 4.2.1, where the edge z_{ij} connects the nodes x_i and x_j . Therefore the g2o optimiser performs a minimisation of the non-linear error function which has been shown in (4.2) and (4.3). It should be noted that the estimations with large errors may impede the accuracy of the created graph. These edges are pruned after the initial convergence and then the optimisation is restarted once again. When a node is added, the optimisation process is called to process the new graph. Considering the efficiency, all old nodes are set as fixed. Thus the optimiser only adjusts the pose of new one to satisfy the constraints from others. At the end of the SLAM process, global optimisation is called to manage the final form of map reconstruction.

4.1.3. Output data

The work period of the **original** version of RGBDSLAM v2 application has been finished after the final global graph optimisation. It supports the export of the estimated camera trajectory as a text file and the reconstructed dense 3D point cloud as a PCD file. The former is particularly important for building the reference template, since in an online tracking session the query images will be compared with a portion of the colour images used in 3D reconstruction to locate them with respect to the reference. The file format of trajectory file is given in Appendix F. The reconstructed dense 3D point cloud involves the geometric information of the target environment. The world cloud points are calculated via the RGBD frames and their estimated camera poses, which can transform 3D coordinates of each local camera reference frame to a unified world reference frame. The dense point cloud will be used as a virtual environment for assisting the application developer to understand the scene structure – thus they can insert augmentation in respect of the scene. However, in order to be used as the reference for an online tracking session, the 3D points that have been correlated with recognisable visual features and descriptors should be specified (*i.e.* the 3D-to-2D correspondences of detected visual features and their corresponding descriptors contained within the images used for reconstruction, as shown in Figure 4-8). Therefore the additional information that will be used for template training is stored as several customised XML files. These are human-readable and simple to use for the cases that only require one-time read/write. The present proposal only saves all the nodes at the end of the reconstruction session. These files include the summary of graph and all camera nodes (see Appendix F for detail). All these output files will further be used for template training, the detail will be presented in Chapter 5.

4.2. Structure from motion

When compared with SLAM which normally follows a “real-time” recursive improvement, making use of adjacent relation of images in the sequence, SfM can handle unordered and uncontrolled image collection (e.g. the images captured by various camera devices with different intrinsic parameters) by applying either batch or incremental solutions (as mentioned in Section 2.2.3.1) for geometric structure reconstruction. Since the incremental SfM methods are more flexible for both online and offline situations, Subsection 4.2.1 describes a basic workflow of an incremental reconstruction method. The case studies of SfM-based 3D reconstruction application VisualSfM are introduced in Subsection 4.2.2. Other related discussions and evaluations are given in Subsection 4.2.3.

4.2.1. Incremental SfM

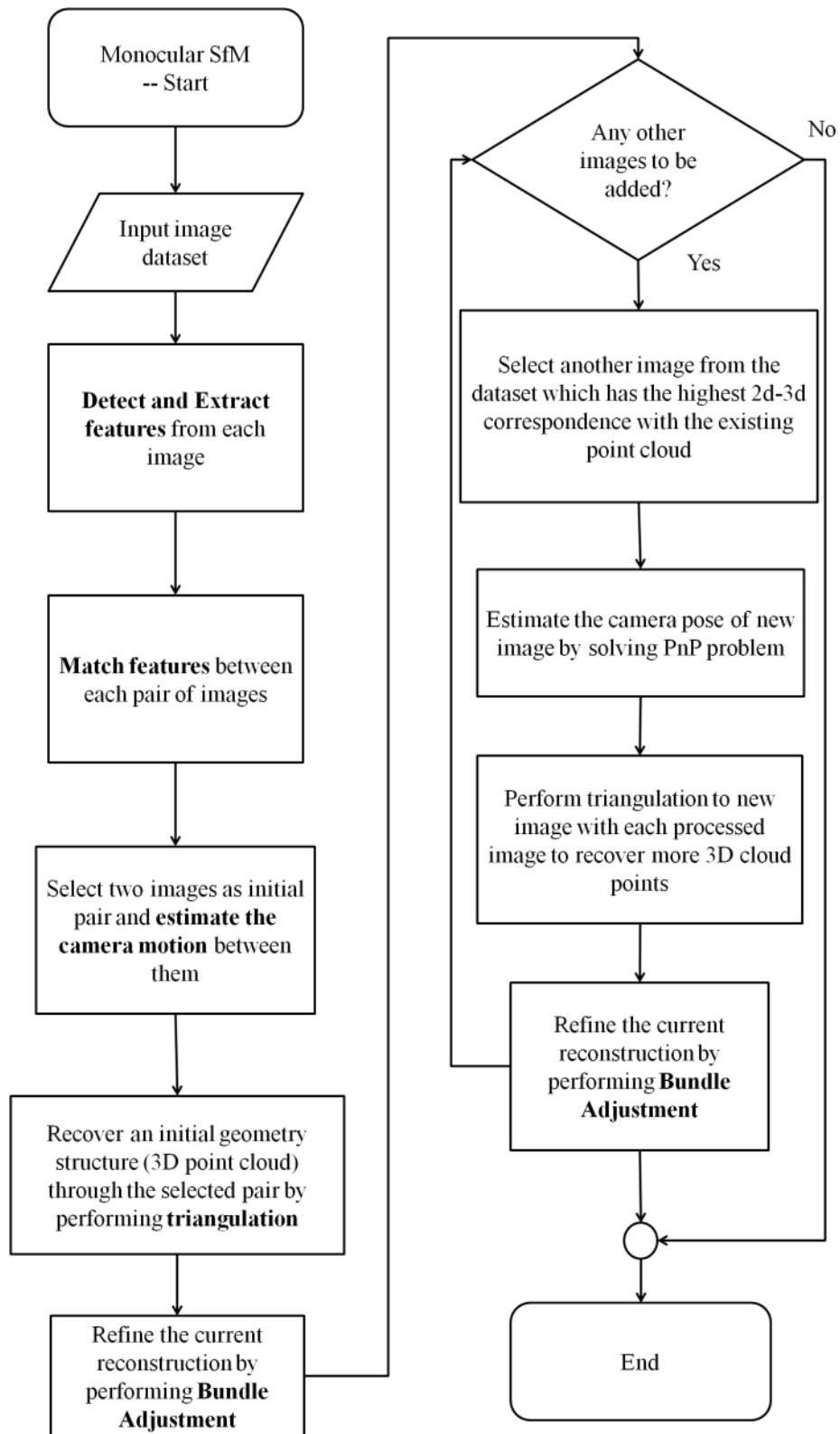


Figure 4-9: The flowchart of monocular SfM-based sparse 3D reconstruction.

The basic flow of a monocular SfM is shown in Figure 4-9. A monocular SfM system usually takes a set of images with multiple overlaps as input data. The images are assumed to be captured by calibrated camera devices, thus the intrinsic parameters should be provided by users. From the outset, the incremental SfM system will perform feature detections and extractions to all images in the input dataset, and then the images will undergo pair-wise for feature matching. To build an initial geometric structure reconstruction, two images are selected from the dataset as initial two-view. Snavely et al. (2008) stated that the choice of initial image pair is necessary to establish a robust 3D reconstruction, and the selected two images should have a large number of correspondences and a reasonably large baseline to perform triangulation – this is because a small baseline will cause high depth ambiguity, and the erroneous 3D points will corrupt the map and ruin tracking (Herrera *et al.*, 2014). The baseline can be obtained from the motion between two camera models, but the camera motions in monocular SfM are unknown and they also need to be estimated through the image data. The specific process of two-view motion estimation based on epipolar geometry is presented in Subsection 4.2.1.1 and a linear triangulation method is presented in Subsection 4.2.1.2. Then the initial reconstruction with a form of sparse point cloud can be built. The point cloud is constituted of the 3D points obtained by back-projecting the feature points from the matched images to the world reference coordinate system. The system then sets out to find visual feature correspondences between each of the remaining images in the dataset and the recovered point cloud to estimate camera motions, and further camera poses in respect of the world. Hence the new 3D points recovered from these correspondences can be gradually added to the cloud. This

incremental reconstruction process is presented in Subsection 4.2.1.3. Since errors may exist during the camera motion/pose estimation, an SfM system usually applies bundle adjustment to refine the 3D reconstructed result. This optimisation adjusts the back-projected position of the recovered cloud points and the camera pose of processed images to minimise the sum of re-projection errors; this is presented in Subsection 4.2.1.4. Bundle adjustment is usually performed after a new image and its contributions (*i.e.* new recovered cloud points) are added to ensure the robustness and stability of the reconstruction process.

4.2.1.1 Epipolar geometry and camera motion estimation

Epipolar geometry is usually used for estimate the camera motions between the images by using 2D-to-2D correspondences. It describes a setting where two pinhole cameras are looking at a same object or scene from two distinct positions, and is similar to the configuration of the human eyes when fixating on an object. In Figure 4-10, the two cameras, \mathbf{C}_0 and \mathbf{C}_1 , are observing same 3D world point P_W from different views and an epipolar constraint can be found between them.

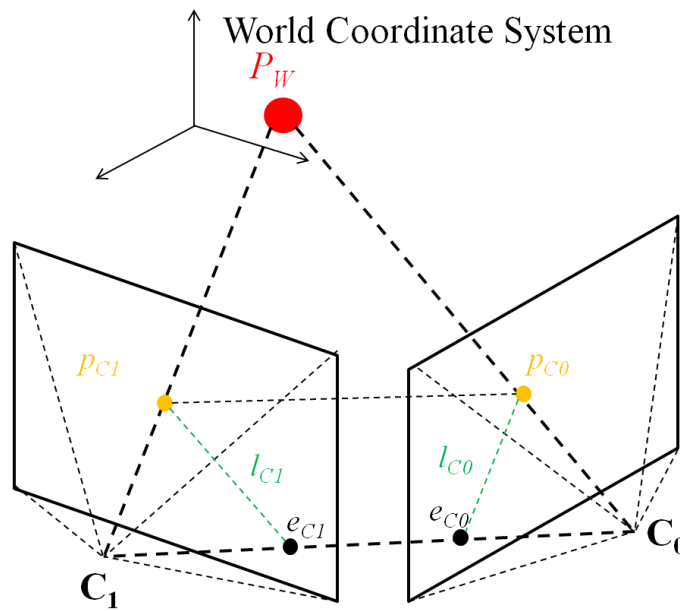


Figure 4-10: Epipolar geometry constraints between two pinhole camera models.

Since there is a displacement between the cameras \mathbf{C}_0 and \mathbf{C}_1 , their optical centres can be projected onto a point within each other's image plane. These projected points are called *epipolar points* and are denoted by e_{C_0} and e_{C_1} respectively. The line segment $p_{C_0} - e_{C_0}$ is the *epipolar line* of the camera \mathbf{C}_0 and is denoted by l_{C_0} . It is actually the projection of the space line which connects the 3D point P_W and the optical centre of \mathbf{C}_1 on the \mathbf{C}_0 's image planes. Symmetrically, the line $p_{C_1} - e_{C_1}$ is the epipolar line of the camera \mathbf{C}_1 as well as the projection of the space line from P_W to \mathbf{C}_0 's optical centre, denoted by l_{C_1} .

Assume an arbitrary space point P_W has 2D projections p_{C_0} and p_{C_1} on \mathbf{C}_0 and \mathbf{C}_1 respectively. These two projective points can be represented with same homogeneous coordinates form expressed in expression (3.6), i.e. $w_{C_0} \cdot p_{C_0} = w_{C_0} \cdot (u_{C_0} \quad v_{C_0} \quad 1)^T$

and $w_{C_1} \cdot p_{C_1} = w_{C_1} \cdot (u_{C_1} \ v_{C_1} \ 1)^T$ respectively. Through epipolar constraints, the corresponding points p_{C_0} and p_{C_1} should satisfy the condition given in (4.12).

$$p_{C_1}^T F p_{C_0} = 0 \quad (4.12)$$

where F is a unique 3×3 rank 2 homogeneous *fundamental matrix* (Hartley & Zisserman, 2003). This equation implies that $F p_{C_0}$ defines the corresponding epipolar line l_{C_1} , and p_{C_1} should lie on it. The fundamental matrix between images only depends on their point correspondences and thus it can be calculated from them. Being of a rank two matrix, F can be estimated given at least seven correspondences by several methods (e.g. the 7-point algorithm (Hartley & Zisserman, 2003), RANSAC, or Least Median of Squares (Rousseeuw, 1984)). In the practice programme process with the OpenCV library, the fundamental matrix can be obtained by passing raw matched correspondences to the function *findFundamentalMat*. The valid F will ensure that most of the correspondences meet the condition in (4.12), and the remainder which do not satisfy the epipolar constraints are considered as outliers.

Suppose the camera models of selected initial two-view in SfM process are denoted by C_0 and C_1 , the visual feature correspondences between them are already computed (the detail of feature detection and matching will be given in Section 4.3.1), and the fundamental matrix between them can be estimated by using their matched correspondences. But in order to obtain the camera motion, another useful matrix – the essential matrix is introduced. Similar to the fundamental matrix, the essential matrix imposes a geometry constraint between the matched homogeneous normalised coordinates in two images, and also implies the position and orientation information of

both cameras in space. In fact, the essential matrix is the specialisation of the fundamental matrix to the case of normalised image coordinates, and it can be obtained from the fundamental matrix by using the intrinsic matrices of the cameras:

$$E = K_{C_1}^T F K_{C_0} \quad (4.13)$$

where K_{C_0} and K_{C_1} denote the 3x3 camera intrinsic matrices of two images respectively.

If the images are captured by same camera device, they will share same intrinsic matrix.

Assume the initial image pair has normalised 3x4 camera extrinsic matrices

$$T_{C_0} = [I | 0] \text{ and } T_{C_1} = [R | t],$$

which indicates the camera poses and can be used to map world coordinates to camera coordinates as shown in equation (3.1). Since the first camera matrix T_{C_0} is assumed to be fixed and canonical (*i.e.* no rotation and no translation), the desired camera motion between them is equivalent to T_{C_1} , the rotation component R and translation component t of which are included in the corresponding 3x3 essential matrix E :

$$E = [t]_{\times} R \quad (4.14)$$

where $[t]_{\times}$ represents the skew-symmetric matrix of 3D translation vector t (see Appendix E) . The R and t can be extracted from E by using singular value decomposition:

$$E = [t]_{\times} R = U * \text{diag}(1,1,0) * V^T \quad (4.15)$$

where U and V are 3x3 unitary matrix. Hartley & Zisserman (2003) indicates that there are 4 possible solutions for T_{C_1} :

$$T_{C_1} = \begin{cases} [UWV^T | +u_3] & (1) \\ [UWV^T | -u_3] & (2) \\ [UW^T V^T | +u_3] & (3) \\ [UW^T V^T | -u_3] & (4) \end{cases} \quad (4.16)$$

where $W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, u_3 is the third column of U . There is only one solution that can

make the recovered 3D point exist in front of both cameras, and the real T_{C_1} can be found by testing all solutions with an arbitrary single point. Since the first camera is set to the fixed one in this case, its camera reference frame is the equal of the world reference frame, and the camera matrix T_{C_1} and the coordinates of recovered space points will be all based on this coordinate system. Alternatively if another world reference frame is desired to be used, the relative transformation from the first camera frame to it should be given.

4.2.1.2 Linear triangulation

Linear triangulation is one of the most common triangulation methods and is based on Hartley *et al.* (1992). Note that the equations presented below can only define a space point up to an indeterminate scale factor due to the inherent scale factor ambiguity of SfM. Assume a 3D world point P_W is being projected to two pixel coordinates p_{C_0} and p_{C_1} on the image planes of C_0 and C_1 respectively by using expression (3.5) (*i.e.* the setup shown in Figure 4-10). Let $A = KT$ in (3.5) where A takes a form of a 3x4

camera matrix, and it can denoted by $A = \begin{bmatrix} A_0^T & A_1^T & A_2^T \end{bmatrix}^T$, where A_i^T represents the i -th row vector of A . Then the relationships between P_W and p_{C_0} , p_{C_1} can be depicted in the following equations:

$$w_{C_0} \cdot \begin{bmatrix} x_{\text{pix-C}_0} \\ y_{\text{pix-C}_0} \\ 1 \end{bmatrix} = A_{C_0} P_W \Rightarrow \begin{cases} w_{C_0} x_{\text{pix-C}_0} = (A_{C_0})_0^T P_W \\ w_{C_0} y_{\text{pix-C}_0} = (A_{C_0})_1^T P_W \\ w_{C_0} = (A_{C_0})_2^T P_W \end{cases} \quad (4.17)$$

$$w_{C_1} \cdot \begin{bmatrix} x_{\text{pix-C}_1} \\ y_{\text{pix-C}_1} \\ 1 \end{bmatrix} = A_{C_1} P_W \Rightarrow \begin{cases} w_{C_1} x_{\text{pix-C}_1} = (A_{C_1})_0^T P_W \\ w_{C_1} y_{\text{pix-C}_1} = (A_{C_1})_1^T P_W \\ w_{C_1} = (A_{C_1})_2^T P_W \end{cases} \quad (4.18)$$

Substituting w_{C_0} and w_{C_1} with $(A_{C_0})_2^T P_W$ and $(A_{C_1})_2^T P_W$ in (4.17) and (4.18) respectively:

$$\begin{cases} x_{\text{pix-C}_0} \cdot (A_{C_0})_2^T P_W = (A_{C_0})_0^T P_W \\ y_{\text{pix-C}_0} \cdot (A_{C_0})_2^T P_W = (A_{C_0})_1^T P_W \\ x_{\text{pix-C}_1} \cdot (A_{C_1})_2^T P_W = (A_{C_1})_0^T P_W \\ y_{\text{pix-C}_1} \cdot (A_{C_1})_2^T P_W = (A_{C_1})_1^T P_W \end{cases} \quad (4.19)$$

It can be found that (4.19) is a four linear inhomogeneous system and P_W only has three unknowns. Thus ideally the P_W can be solved when its correspondence measurements from two images and the related camera matrices are available. These are supplied by finding feature correspondences and camera motion estimations as described in the previous section. Note that the linear triangulation described here is a very basic one which does not consider the effect of noisy data mentioned in Section 2.2.3.1. In practice, the errors caused by the mismatch will be compensated in the optimisation stage by using bundle adjustment, which will be described in Section 4.2.1.4.

4.2.1.3 Incremental reconstruction

Once the geometric structure of the scene in the initial two-view has been recovered, other images in the input dataset can be processed based on the initial reconstruction. Their camera poses and more geometry information of the targets can also be recovered. The next image to add should share enough matched features to the recovered cloud points, then these 3D-to-2D coordinated correspondences between them can be used to estimate the new camera pose by solving the PnP problem. In the practice programme process, the OpenCV library provides a *solvePnP* function which supports solving the PnP problem by using the Levenberg-Marquardt iterative method, P3P or E-PnP methods, which have been reviewed in Section 2.2.3.1. After the camera pose has been found, the correspondences (which do not yet exist in the current point cloud) between the new image and the old processed images will be triangulated and back-projected to the world coordinates by using the same approach described in Section 4.1.2.2. Since not every point on the images will be back-projected to the reference frame (this makes sense because in the tracking session, only feature points are able to be tracked), the final result of the reconstruction is a sparse point cloud.

4.2.1.4 Reconstruction optimisation

As mentioned in Section 2.2.3.1, SfM usually applies a bundle adjustment to produce optimal solutions for recovered cloud point coordinates and the estimated pose of added cameras. If the approximated 3D coordinates are re-projected onto the images, the projections should be very close to their original 2D coordinates. Moreover, a 3D point is not restricted by only one 2D image point, but two or more from different images with the respective estimated camera matrices. A local optimal solution of point

positions and camera poses may lead to a large global error, and bundle adjustment tries to find a best set-up for all data being fitted to a global optimisation.

More specifically, a bundle adjustment makes use of the constraints between the 3D points and the camera matrices to minimise the sum of re-projection errors. Assume there are total m 3D points and n cameras. T_i represents the camera extrinsic matrix of the i -th camera and P_j represents the j -th 3D point. If P_j is visible in the i -th camera, then its originating point is denoted by p_{ij} , and the re-projected point with T_i is denoted by $\mathbf{RP}(P_j, T_i)$. Then the re-projected error of P_j to the i -th camera is defined as:

$$\mathbf{e}_{ij} = \left\| p_{ij} - \mathbf{RP}(P_j, T_i) \right\| \quad (4.20)$$

And the objective function to minimise can be written as:

$$\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \omega_{ij} \mathbf{e}_{ij}^2 \quad (4.21)$$

where ω_{ij} is an indicator variable, it equals 1 when P_j is visible in the i -th camera, otherwise it equals 0. The parameters for attaining this minimum include a 3-vector of each 3D point and 6DOF for each camera matrix; the total number of parameters involved is $3m+6n$. Theoretically this could inflate to a huge problem with the increasing of the parameters, where the general complexity of bundle adjustment for SfM will be $O(m+n)^3$. However, this case has a block-sparse structure because the projection of 3D point on a certain camera is independent to the other points and cameras. Accordingly, there is a sparse version bundle adjustment – SBA (Lourakis & Argyros, 2009) which is capable of dealing with this kind of problem efficiently and reduces the complexity to $O(mn + n^3)$.

4.2.2. VisualSfM

VisualSfM is a comprehensive command-line/GUI application for incremental 3D reconstruction which features SiftGPU for feature detection and matching and Multicore Bundle Adjustment for refinement. The application provides a good balance of process between speed and accuracy, and has been used in several AR related research projects to generate 3D reference models (such as Simões *et al.* (2013) and Kurihara & Sagawa (2014) who utilise VisualSfM for 3D reconstruction of industrial facilities). Although VisualSfM is not totally open-sourced, it stores the intermediate data and results in files which contain camera parameters, image features, descriptors and matching list, together with the final result of restored sparse point clouds. These output files can provide sufficient visual and geometric information of the target environment and is used for template/database training of AR applications in the presented proposal. Specifically, the following steps can be performed by using VisualSfM for 3D reconstruction:

- 1) Let the user select a set of images (with or without detected visual features) from files as input. If the image features are not given, VisualSfM will perform feature detections by using default SiftGPU for each image and save the results in SIFT files (the detail of a SIFT file is given in Appendix G).
- 2) Let VisualSfM perform pair-wise matching for all images if the default feature descriptors (*i.e.* SIFT-like descriptors) are used. The results will be saved in several MAT files (the detail of a MAT file is given in Appendix G). Otherwise the similar format MAT files with the list of feature correspondences of each

image should be imported by user.

- 3) Let VisualSfM incrementally recover sparse point clouds (the process of bundle adjustment is integrated) from the input images by using the features saved in step (1) and correspondences saved in step (2). The intermediate parameters and final results will be saved in a NVM file (the detail of an NVM file is given in Appendix G). The reconstructed dense model will be saved in a PLY file (Bourke, 2009).

The default feature detector and extractor used in VisualSfM is based on SiftGPU – an implementation of the SIFT algorithm which can detect SIFT features and process SIFT descriptors efficiently by utilising GPU computation. Once the input image dataset has been specified by user, VisualSfM will automatically perform feature detection to all images and save these features separately to several SIFT files by each image. Each SIFT file contains a list of detected features of one image and the corresponding descriptor vectors. These feature information sets will be used by VisualSfM for matching keypoints between each pair of images to select the best initial two-view for subsequent 3D reconstruction, as mentioned previously in Section 4.2.1. Alternatively, if the user wishes to make use of customised feature detection and representation algorithms, VisualSfM can still perform feature matching to those SIFT-like descriptors if they have been stored in the required SIFT format. Otherwise, the user should specify the feature matching results for each image in the required MAT format by using an API provided by Wu (2011). Actually the MAT file only contains the indices of matched feature – it does not matter what kind of feature or descriptor is used. With the input images and their feature correspondences, VisualSfM will build a sparse point

cloud model. In practice, some images may not be successfully registered to the constructed model due to the absence of features or matched correspondences with other images.

VisualSfM generates an NVM file which contains the list of camera parameters (both intrinsic and extrinsic), the reconstructed sparse 3D cloud points and their originating 2D measurements. The approaches of how to utilise the output data of VisualSfM to generate template and reference database for an online tracking session will be described in Chapter 5. VisualSfM also supports the population of a dense point cloud model by using multi-view stereo techniques on sparse reconstruction results (as shown in Figure 4-11), which will be used as visual reference for developer to set up the augmented scene in the proposed development framework, as described in Chapter 6.

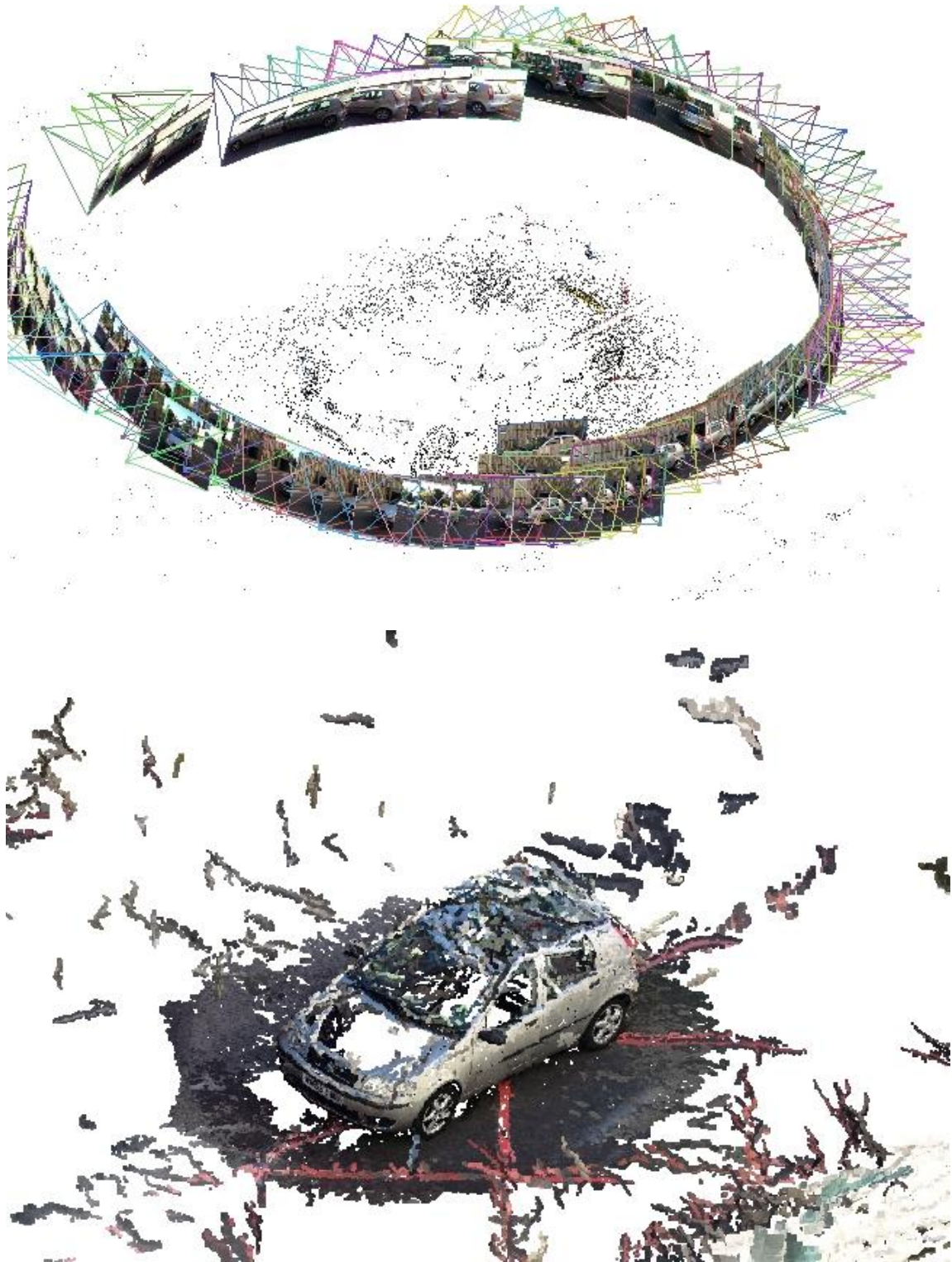


Figure 4-11: The sparse point cloud reconstruction of a car generated by VisualSfM (upper) and the dense point cloud reconstruction based on the VisualSfM result (lower).



4.3. Reconstruction evaluations





Since the reconstructed 3D structure information of target environment and the camera pose of each engaged colour image will be used as templates (reference map) in the online AR application for user tracking, The results should be strictly consistent with the real world (*i.e.* the ground truth data which defines the real location of each object in space) as much as possible. As mentioned at the beginning of this Chapter, the 3D reconstruction process here refers to creating a set of sparse 3D cloud points from the 2D images, and each cloud point is associated with a distinguishable visual feature. The real values of these keypoints (associated spatial positions and visual features) are not easy to measure and collect by other kinds of sensor as mentioned in Section 2.3.3, which is why the vision-based 3D reconstruction approaches are applied for this aim. Furthermore, to compare the ground truth points with their reconstruction results, they need to be tightly coupled with the visual information (e.g. feature description) in the first instance. Then it will be possible to find their corresponding cloud points to compare, but during the process of association and matching, mismatch noise may occur, which cannot be used for evaluation. **Therefore, the estimated camera pose which is independent of the visual information, is compared to ground truth instead here for evaluating the accuracy of the reconstruction results.** As reviewed in Section 2.3.3, the ground truth camera trajectory can be obtained from a non-visual motion-capture system ground truth. The accuracy of pose estimation can also reflect the performance of the 3D reconstruction due to the fact that the estimations of cloud points and camera pose are **interdependent** during the reconstruction process.

Two public datasets containing ground truth camera trajectory and calibrated intrinsic

parameters were tested in this stage for accuracy evaluation – four RGBD sequences of indoor environment published by Sturm *et al.* (2012) and two RGB image sets of outdoor environment published by Strecha *et al.* (2008). An outdoor RGBD dataset is not available since, in the main, Kinect RGBD data can only be collected under indoor environmental conditions due to the limitation of RGBD camera mentioned in Section 2.2.3.2. Thus the RGBD-SLAM based approach will only be tested on indoor RGBD datasets. In contrast, the SfM-based approaches can deal with the colour images contained within both RGBD and RGB datasets. The information of public datasets used for test is shown in Table 4-1.

Table 4-1: The public datasets used in the present research.

RGBD datasets (indoor, 640 x 480)	Sequence 'freiburg1_desk'		Duration: 23.40s (577 frames) Duration with ground-truth: 23.35s Ground-truth trajectory length: 9.263m Avg. translational velocity: 0.413m/s Avg. angular velocity: 23.327deg/s Trajectory dim.: 2.42m x 1.34m x 0.66m
	Sequence 'freiburg1_floor'		Duration: 49.87s (1242 frames) Duration with ground-truth: 44.27s Ground truth trajectory length: 12.569m Avg. translational velocity: 0.258m/s Avg. angular velocity: 15.071deg/s Trajectory dim.: 2.30m x 1.31m x 0.58m

	Sequence 'freiburg1_360'		Duration: 28.69s (756 frames) Duration with ground-truth: 28.70s Ground-truth trajectory length: 5.818m Avg. translational velocity: 0.210m/s Avg. angular velocity: 41.600deg/s Trajectory dim.: 0.54m x 0.46m x 0.47m
	Sequence 'freiburg3_long_office_household'		Duration: 87.09s (2585 frames) Duration with ground-truth: 87.10s Ground-truth trajectory length: 21.455m Avg. translational velocity: 0.249m/s Avg. angular velocity: 10.188deg/s Trajectory dim.: 5.12m x 4.89m x 0.54m
RGB datasets (outdoor, 3072 x 2048)	fountain-P11		Size: 11 Avg. translational displacement: 1.695m Avg. rotation angle: 61.231 deg
	castle-P30		Size: 30 Avg. translational displacement: 4.762m (excluded the huge first leap of 24.012m) Avg. rotation angle: 63.020 deg

Since both RGBDSLAM v2 and VisualSfM allow users to set which methods to use for feature detection, description and matching methods, the performance of several visual features reviewed in Section 2.2.2 are firstly tested and evaluated in Subsection 4.3.1. Then two of the best-performing methods SiftGPU and ORB features are selected to be used for offline 3D reconstruction with both approaches on the public datasets mentioned above. The results are presented in Subsection 4.3.2. All the tests were carried on a laptop powered by an Intel® i7-4510 CPU at 2.00GHz x 4, with 16GB of RAM and Nvidia GeForce GT 750M Graphics Card.

4.3.1. A small study on visual features

The quality of 3D reconstruction and the user-tracking stage described in Chapter 5 strongly depends on the performance of visual feature detection and matching. Intuitively the system detects the distinguishable local visual features within an image, refers to them as feature points or keypoints, and abstracts them into quantitative descriptions (*i.e.* feature descriptors). Thus Keypoints from different images with similar descriptors are considered to be matched and referred to as ‘correspondences’ above. In fact, feature matching is a nearest descriptor retrieval problem and whichever method is chosen only impacts the execution time. The accuracy of matching is based on robust feature detection and representation. Robust features should be distinctive, repeatable and invariant to various changes – in particular scale and rotation. With the exception of the robustness, performance speed also needs attention. Although a real-time performance is not required in the offline session, the similar features will be used as templates in the time-constrained AR application which demand real-time operation.

A trade-off between accuracy and speed should be considered. Several feature detectors and extractors have been reviewed in Section 2.2.2. The most representative blob-like feature methods – **SIFT** and **SURF**; the corner detector with binary descriptor method – **ORB** and a fast implementation of SIFT with GPU computation – **SIFTGPU** are tested and discussed below. The test is carried out with the RGB dataset [fountain-P11] of Strecha *et al.* (2008) which consists of 11, 3072 x 2048 high resolution images of the same fountain captured from different angles. The test is also carried out with a set of the **down-sampled** images of [fountain-P11] with a 720 x 480 resolution as comparison by considering that normal webcams and the Kinect sensor can provide images at this level. The SIFT, SURF and ORB methods tested here are OpenCV 2.4.9 implementation with default parameters (for the SURF detector the recommend Hessian threshold in OpenCV documentation is 300~500, thus 400 is used; 64-D and 128-D SURF descriptors are both tested).

The first experiment focuses on the **processing time** of feature detection and extraction. Without limiting the maximum number of features to detect, the average number of detected keypoints and the average processing time for feature detection and extraction with each method for **11** full-size and down-sampled images are summarised in Table 4-2. Note the qualities of all average values below were expressed with *relative standard deviations (RSD)*, *i.e.* the ratio of standard deviation to mean value.

Table 4-2: A performance test on different feature detector – extractors (bracketed are %RSD).

11 images of [fountain-P11]		OpenCV				SiftGPU
		SIFT (128-D)	SURF (64-D)	SURF (128-D)	ORB(256 bits = 32-D)	(GLSL, 128-D)
3072 x 2048 pixels (full- size)	Avg. No. features	4968 (±22%)	6357 (±16%)		12224 (±43%)	<u>16511</u> (±5%)
	Avg. detection time (ms)	1894 (±2%)	594 (±5%)	616 (±6%)	<u>53</u> (±13%)	Avg. detection & extraction (ms): 302 (±5%)
	Avg. extraction time (ms)	1634 (±3%)	748 (±19%)	726 (±18%)	<u>122</u> (±13%)	
720 x 480 pixels (down- sampled)	Avg. No. features	1620 (±20%)	825 (±25%)		<u>5225</u> (±19%)	3679 (±11%)
	Avg. detection time (ms)	147 (±7%)	53 (±11%)	50 (±12%)	<u>12</u> (±25%)	Avg. detection & extraction (ms): 64 (±10%)
	Avg. extraction time (ms)	167 (±11%)	81 (±30%)	82 (±27%)	<u>27</u> (±15%)	

Consider that to same image the processing time will depend on some degree of the number of detected features, it can be found that ORB detector – descriptor extracted relatively more features with less time, and SiftGPU comes second. These two performed significantly faster than SURF and SIFT. The original SIFT is the slowest, and its processing time with both full-size and down-sampled images are unacceptable

for real-time application. The choice of the number of dimensions for the SURF descriptor has little effect on processing time. It also can be found that the largest numbers of keypoints are detected by using ORB and SiftGPU detectors. SURF and SIFT also provide sufficient keypoints.

Then the **robustness** of each kind of descriptor extracted above will be measured via the accuracy when using them for feature matching. The images in [fountain-P11] are compared with each other in a pair-wise fashion to find keypoint correspondences (which makes sense since it is actually performed at the beginning of SfM system described in Section 4.2). For 11 images in [fountain-P11] there will be **55** comparisons to perform. When two images are being compared, the sets of descriptors from two images are referred to as **query data** and **train data** respectively. For histogram-based SIFT-like descriptors, FLANN introduced in Section 2.2.2.2 is the preferred technique. The main idea involves structuring the descriptor vectors of train image, and then the descriptors of query image will only compare with a partition of the descriptors on train image to efficiently find the one with the nearest distance as a match. However, this kind of structuring does not fit to the binary descriptor since there is a binary value on each bit of vector and it is unnecessary to cluster them into different groups. Thus, for ORB descriptors, the simplest brute-force search is much more efficient and will be used. The FLANN and brute-force matching methods have already been implemented in OpenCV and will be used here. For the SiftGPU, both the FLANN matcher supported by OpenCV and its own encapsulated matching methods – *SiftMatchGPU* – will be tested. The OpenCV FLANN matcher does not really judge which two descriptors from these two sets are matched, but does a k -NN search to find the nearest $k=2$ descriptor in

train for each query. Then a ratio test will make sure that a match is kept only if the

distance ratio **ratio** = $\frac{\mathbf{d}_{\text{NN1}}}{\mathbf{d}_{\text{NN2}}}$ between the closest neighbour (\mathbf{d}_{NN1}) and the second

closest neighbour (\mathbf{d}_{NN2}) is below a certain threshold. For SIFT style feature matching

0.6 – 0.8 is suggested in Lowe (2004) hence **0.7** was used here for SIFT and SURF

descriptors. Moreover, cross-matching was also applied, which will match train

descriptors with the query set and *vice versa* –, and only common matches are kept as

the raw matching results. The cross-check test can be enabled for OpenCV brute-force

matcher also by setting the related argument to true, but in this case *k*-NN search is not

available and it should be noted that the distance metric used in SIFT-like and binary

descriptors are very different. Much refined results can be achieved by removing the

mismatches by checking geometric consistency with RANSAC. The main idea is

estimating a transformation (epipolar or homography) between the matched keypoints

to filter out the outliers. The homography estimation only works with the features on a

planar object as shown in Figure 4-12, which is often used in plane marker-based AR

systems.

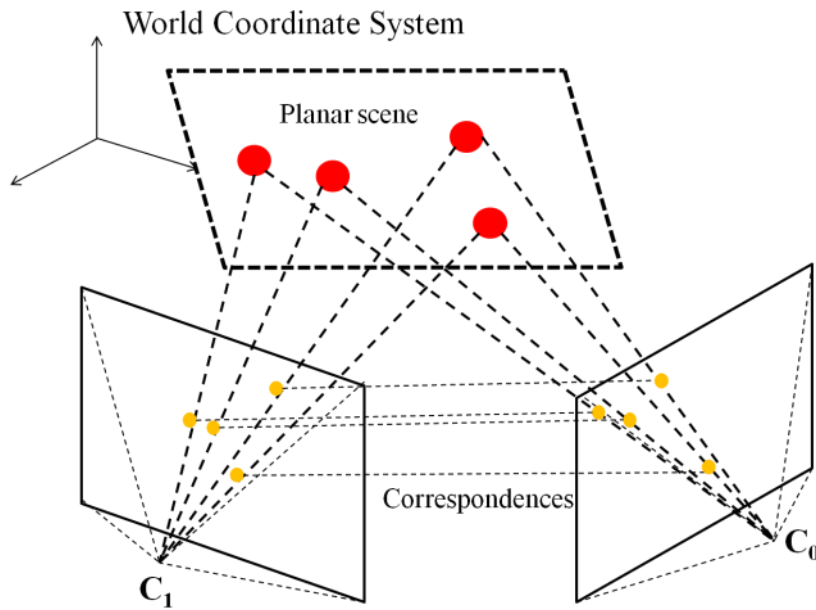


Figure 4-12: Homography correspondences between two images.

In this case, epipolar is more suitable. For the purpose of evaluating the different descriptors used in feature matching, the fundamental matrices between each pair of images are extracted from the ground truth camera projection matrices $A = KT$ where K and T has the same definitions as in expression (3.5). The fundamental matrix between two 3×4 camera projection matrices A and A' can be calculated as:

$$F = [A'c]_{\times} A' A^+ \quad (4.22)$$

where c is the camera centre of A , defined by $Ac = 0$; A^+ is the 4×3 pseudo-inverse matrix of A that $A A^+$ equals to identity matrix I . The match of two images coordinates which meets the formula (4.12) will be considered as a good match.

The matches pass the cross-check and ratio test will be considered as initial matching results. Here the accuracy will be assessed by a **good match rate** –, *i.e.* the ratio between the number of good matches and initial matches in which the good matches are kept by ground truth fundamental matrices (provided along with the dataset in Strecha

et al. (2008)). Another factor to consider is the ratio between the number of good matches and the minimum number of features detected in the pair of images to compare –, *i.e.* the repeatability of detected features (Canclini *et al.*, 2013). The processing time of matching is also considered. The average processing time of each comparison, the good match ratio and the repeatability for each feature descriptor are given in Table 4-3. Since the ground truth fundamental matrices cannot be used for down-sampled images, only full-size images were evaluated in this experiment.

Table 4-3: A performance test on different matching methods for different types of feature (bracketed are %RSD).

55 pair-wise comparisons of [fountain-P11]	OpenCV				SiftGPU	
	SIFT (FLANN)	SURF (64-D, FLANN)	SURF (128-D, FLANN)	ORB (Brute-force)	FLANN	SiftMatch GPU
Avg. matching time of per comparison (ms)	327 (±32%)	371 (±37%)	408 (±26%)	1902 (±59%)	2186 (±60%)	111 (±10%)
Avg. good match ratio	0.88 (±18%)	0.79 (±32%)	0.74 (±36%)	0.21 (±82%)	0.90 (±17%)	0.54 (±64%)
Avg. repeatability	0.12 (±114%)	0.09 (±128%)	0.06 (±138%)	0.12 (±82%)	0.14 (±112%)	0.03 (±110%)

The good match ratio reflects how well the descriptor represents a feature. The repeatability of detected features implies the distinctiveness and robustness of the descriptor. It can be found that SiftGPU and OpenCV SIFT descriptors have the relatively higher performance than other methods. However, consider that SiftGPU using default settings detected much more features than OpenCV SIFT (as shown in

Table 4-2). It took more processing time for matching each pair of images when OpenCV FLANN matcher was used, but with regard to average good match ratio and repeatability, these two are very close. On the other hand, if SiftGPU made use of its integrated matching function SiftMatchGPU, it could process feature matching in very efficient way. Nevertheless, the relatively lower good match ratio and the lowest repeatability of this method indicate that a lot of good matches kept by FLANN matcher were eliminated here. This may suggest that the slower OpenCV FLANN matcher, providing plenty of reliable correspondences, is more applicable to the offline 3D reconstruction in the present proposed system; in contrast, the efficient SiftGPU matcher can be used in the online stage.

The performance of OpenCV SURF descriptor followed the SIFT methods. It is interesting to see that the 64-D descriptor performs better than the 128-D, not only on processing time, but also on good match ratio and repeatability. The good match ratio of OpenCV ORB descriptor with brute-force matcher seems like the worst. However the repeatability of detected features was at the same level as SIFT, which might imply that the high mismatch rate was caused by OpenCV brute-force matcher rather than ORB descriptor. The major cause of slower processing time is because that the brute-force method was applied for ORB descriptor matching, which has the complexity of $O(mn)$ when searching m query data through n train data.

In practice, too many keypoints contribute little to the improvement of the system performance but will increase the computational costs of detection, matching and particularly the RANSAC approximation of estimating transformation via

correspondences. Although there is no request of time limit in the offline stage, a lower cost solution with the same accuracy is preferred. Therefore by considering that both ORB and SiftGPU detectors detected a mass of keypoints in last two experiments, the maximum number of features to detect was limited to **5750** (the average number of features detected by OpenCV SIFT and SURF detectors) in the next experiment. The matching methods applied were same to the last experiment and the performance of each approach was still evaluated with good match ratio, repeatability of detected features and processing time. The results are shown in Table 4-4.

Table 4-4: The performance test on ORB and SiftGPU which has limited the number of features to detect (bracketed are %RSD).

55 pair-wise comparisons of [fountain-P11]	Limited maximum number of detected features = 5750		
	OpenCV ORB (Brute-force)	SiftGPU	
		FLANN	SiftMatchGPU
Avg. matching time of per comparison (ms)	324 (±6%)	399 (±40%)	<u>116</u> (±16%)
Avg. good match ratio	0.19 (±86%)	<u>0.89</u> (±16%)	0.61 (±51%)
Avg. repeatability	0.10 (±93%)	<u>0.13</u> (±109%)	0.12 (±103%)

It can be seen that, although the performance of ORB and SiftGPU with FLANN matcher on good match ratio and repeatability has decreased marginally than the result shown in Table 4-3, the average matching time was improved significantly by limiting the number of detected features. The processing time of ORB was even slightly better than OpenCV SIFT and SURF. With regard to SiftMatchGPU, the processing time was

nearly unaffected and the average good match ratio increased somewhat, but due to the reduction of detected features, the repeatability of detected features arrived to the similar level to the other methods.

Overall, the present research will take a balance by testing SiftGPU, which is as robust as the original SIFT and faster than SURF, and ORB, which is implemented with a totally different mechanism to the SIFT-like algorithms, but capable of achieving an efficient and reasonable performance without GPU computation. The 3D reconstruction results and related discussion are shown in next section.

4.3.2. The evaluation of 3D sparse reconstruction based on camera pose estimation

Two 3D reconstruction/mapping methods described above were tested for their performance when creating reference templates for AR tracking and registration from two perspectives: 1) evaluating the accuracy of the estimated camera pose/trajectory of the input images by comparing them to the ground truth, *i.e.* the evaluation metrics regularly used for SLAM system; 2) testing if the accuracy of estimated camera pose is qualified for a precise AR registration – thus the reconstructed result can be a qualified reference for the following online AR tracking.

The two methods were firstly tested on indoor RGBD sequences [freiburg1_desk], [freiburg1_floor], [freiburg1_360] and [freiburg3_long_office_household] provided by Sturm *et al.* (2012) and evaluated mainly through their benchmark described in Section

2.3.3. Each sequence consists of a continuous set of 640 x 480 synchronous RGB and depth image pairs collected within an office-like indoor environment with a sampling rate at 30 frames per second and the related camera information (intrinsic camera parameters). The property of each sequence is given below:

- [freiburg1_desk] is a short sequence which contains several sweeps over a number of desks in a typical office environment;
- [freiburg1_floor] represents a sweep over the planar floor which contains several salient knotholes and is considered easy to track by authors;
- the camera motion of [freiburg1_360] was moved quickly up and down vertically with a 360 ° turn in the horizontal plane, and most of the colour images are blurred as a result of fast camera motion;
- [freiburg3_long_office_household] has the longest duration, as the camera was moved slowly and steadily around an office desk with many textural and structural features, and the end of the trajectory overlaps with the beginning.

Sturm *et al.* (2012) provide the camera ground truth trajectory of each sequence, thus the accuracy of reconstructed template can be assessed by comparing the camera pose estimation of each frame inside the sequence to the corresponding ground truth. The **RMSE** (Root Mean Squared Error) was used here to measure the absolute trajectory error between estimation and ground truth in quantity as Sturm *et al.* (2012) suggested in their benchmark. The RMSE is also known as *root mean squared deviation* which assesses the quality of an estimator by measuring the average of the deviations between the estimator and what is estimated, and has the same units as the quantity being estimated. In order to calculate the RMSE between the reconstructed result and ground

truth, the estimated camera poses were firstly associated with the ground truth trajectory by matching the timestamps (which are involved in the output file specified in Appendix F). Since the reference frame of the ground truth is different from the world reference frame used during the reconstruction, the estimation should align to the ground truth by finding translation and rotation transformation between two frames. The camera centre contained within pose can be considered as a 3D coordinate, thus there is set of 3D-to-3D correspondences between the estimation and ground truth, which can be used with the SVD method introduced in Section 4.1.2.3 to solve the transformation and align the estimated data to the ground truth frame. The difference between the aligned estimation and ground truth is computed as a vector of translational error, where the i -th element is calculated as:

$$e_{\text{trans}_i} = \sqrt{(x_{\text{aligned}_i} - x_{\text{groundtruth}_i})^2 + (y_{\text{aligned}_i} - y_{\text{groundtruth}_i})^2 + (z_{\text{aligned}_i} - z_{\text{groundtruth}_i})^2} \quad (4.23)$$

and the translational RMSE can be obtained as:

$$\text{RMSE}_{\text{tans}} = \sqrt{\frac{\sum_{i=1}^n e_{\text{trans}_i}^2}{n}} \quad (4.24)$$

The smaller RMSE is the better accuracy of the reconstruction achieves. Here the RMSE is measured in meters.

Since the method of SiftGPU detector – descriptor with FLANN matcher is believed to possess the optimum comprehensive performance (through the test discussed in the previous section), this combination was applied for both reconstruction methods along with the efficient OpenCV ORB detector – descriptor and brute-force matcher. These

two approaches were tested under the same conditions – the maximum number of features to detect was limited to **600** for both approaches on 640 x 480 image sets since it has been found in Section 4.3.1 that detecting more features did not contribute more to the performance. On the contrary, it took more processing time and the result was not superior to those with fewer features (see Table 4-3 and Table 4-4 for comparison). RGBDSLAM v2 was firstly tested on the RGBD dataset. The total processing time and the RMSE on each sequence with both SiftGPU and ORB approaches are recorded in Table 4-5. The RMSE was divided into ‘**before** and **after** final optimisation’ to see whether applying a **global g2o optimisation** would improve the reconstruction results. As mentioned in Section 4.1.2.4, a g2o optimiser was called every time after adding a new node for performing optimisation and it allows user to choose which nodes should be set as fixed during the optimisation. Only setting the first node as fixed will perform a global optimisation which ideally gives the optimal result, but when there are so many nodes and constraint edges in graph, the system time becomes really costly. Therefore, all added nodes except the latest one would be set fixed to perform local optimisation during the reconstruction, and a global optimisation would only be performed at the final stage of the process for efficiency. Actually, the strategy of performing global optimisation on each new added frame was tested on a sequence [freiburg1_desk] to compare with the case of using local optimisation. It cost 5274s for the whole reconstruction whilst applying local optimisation to the same condition only took 711s (see total processing time with SiftGPU on [freiburg1_desk] in Table 4-5). However the RMSE of results were the same, to 0.022m well.

Table 4-5: The performance test of RGBDSLAM v2 on different RGBD sequences.

RGBD sequence (640 x 480)	No. RGBD frames (full 30fps)	Feature detectors and matchers							
		SiftGPU + OpenCV FLANN				OpenCV ORB + Brute-force			
		Total processing time (s)	RMSE (m)	RMSE (m) after final global optimisation	Max. error (m)	Total processing time (s)	RMSE (m)	RMSE (m) after final global optimisation	Max. error (m)
[freiburg1_desk]	577	711	<u>0.022</u>	<u>0.022</u>	<u>0.075</u>	<u>467</u>	0.027	0.024	0.08
[freiburg1_floor]	1222	1528	<u>0.032</u>	<u>0.028</u>	<u>0.108</u>	<u>1210</u>	0.040	0.030	<u>0.108</u>
[freiburg1_360]	745	797	<u>0.071</u>	<u>0.057</u>	<u>0.122</u>	<u>566</u>	0.071	0.060	0.155
[freiburg3_long_of fice_household]	2487	3681	<u>0.024</u>	0.033	0.07	<u>3084</u>	0.055	<u>0.032</u>	<u>0.06</u>

It can be found that the accuracy of results with SiftGPU method was, in general, better than those with the ORB method through the value of RMSE before final optimisation. But by performing final global optimisation, the RMSE with ORB can be improved to the same level of those with SiftGPU. On the other hand, the processing time of ORB was still more efficient than SiftGPU.

VisualSfM was tested on the colour images contained within the above RGBD dataset. In order to compare with RGBDSLAM v2 in similar condition, the same configurations of feature detectors and descriptors were used in VisualSfM. For SiftGPU, the descriptors extracted during the process of RGBDSLAM v2 were exported to .sift files (see Appendix G) and imported to VisualSfM for further feature matching and sparse reconstruction. For ORB which is not supported by VisualSfM natively, the pair-wise matching results by OpenCV brute-force matcher were exported and used by VisualSfM for performing reconstruction. However, it should notice that there was extra processing time shown in Table 4-6 for exporting and importing the features or matching results for VisualSfM.

In contrast to RGBSLAM v2, which requires sequence frames as input, the input data of VisualSfM is considered as unordered and each image in the dataset will be compared with the others to select the best initial two-view, based on a large number of correspondences and a reasonable large baseline (as mentioned in Section 4.2.1). The successive images with small camera movement between the adjacent frames are to be avoided, since the system will take too much time execute pair-wise matching, but the baseline between these adjacent frames are too small for use during triangulation. For

this reason, some adjacent frames with small differences inside the sequences were skipped. This step should have been automatically processed by system but the present VisualSfM does not support this function. **Therefore how many frames should be selected to skip is depending on the baseline distance, average camera translational velocity and angular velocity** of each sequence indicated in Table 4-1. The baseline distance was chosen based on the RGB sensor’s intrinsic parameter and the depth range of the Kinect. The relationship between them is given in Navab & Unger (2010) as below:

$$\text{Depth} = \frac{\text{Baseline} * \text{Focal length}}{\text{Disparity}} \quad (4.25)$$

where the disparity measures the displacement of a point between the two images. Consider that all Sturm *et al.* (2012) RGBD sequences are about indoors office-like environment, the baseline distance should have been 0.18 for measuring the maximum depth of 2m (with the smallest disparity 1 pixel) and thus the skip step of each sequence was ranging from 10 to 20. However, the reconstruction results of some sequences were not very ideal with selected data skip step due to the effect of non-uniform motion and big angular velocity. A ‘gap’ might appear between the selected frames and VisualSfM would create a couple of separated models –, which cannot be used as the reference map for AR tracking. In order to solve the problem, the smaller data skip steps were tested and skipping 5 frames (*i.e.* 6fps relative to the original 30 fps sequence) was found to be able to produce relatively proper reconstruction for most of the sequences with the exception of the [freiburg1_360], which completely failed in the reconstruction by Visual SfM. The reconstruction result of each sequence is displayed in Table 4-6. It is also important to notice that due to the ambiguity of the scale in SfM,

a scale factor should be specified to align the estimation with the ground truth, which is totally random and can be calculated through the alignment.

Table 4-6: The performance test of VisualSfM on RGB data contained within the RGBD dataset (bracketed are total processing time without extra time for exporting & importing files).

RGBD sequence (640 x 480)	No. RGB frames (data skip step = 5)	Feature detectors and matchers: SiftGPU + VisualSfM matcher						
		Feature detection (s)	Exporting & Importing feature descriptors (s)	Pair-wise matching & Sparse reconstruction (s)	Total processing time (s)	Scale	RMSE (m)	Max. error (m)
[freiburg1_desk]	123/613	4	42	742	<u>788 (746)</u>	0.36	<u>0.028</u>	<u>0.065</u>
[freiburg1_floor]	249/1242	8	84	2150	<u>2242 (2158)</u>	0.60	<u>0.032</u>	<u>0.089</u>
[freiburg3_long_of fice_household]	517 / 2585	18	185	4383	<u>4586 (4401)</u>	1.06	<u>0.017</u>	<u>0.049</u>

RGBD sequence (640 x 480)	No. RGB frames (data skip step = 5)	Feature detectors and matchers: OpenCV ORB + Brute-force						
		Feature detection & pair-wise matching (s)	Exporting & Importing matches (s)	Sparse reconstruction (s)	Total processing time (s)	Scale	RMSE (m)	Max. error (m)
[freiburg1_desk]	123/613	161	903	1243	2307 (1404)	0.21	0.120	0.962
[freiburg1_floor]	249/1242	618	2812	3413	6843 (4031)	0.37	0.100	0.641
[freiburg3_long_of fice_household]	517 / 2585	2813	9329	3942	16084 (6755)	0.10	0.824	3.125

From the perspective of processing time, RGBDSLAM v2 with both SiftGPU and ORB approaches was significantly faster than VisualSfM even the extra time of importing and exporting files for VisualSfM was not taken into account. Besides, it can be found that exporting and importing SiftGPU descriptors took ten times longer than feature detection and extraction. Moreover, the time spent on VisualSfM for importing ORB matching results has increased significantly over the number of frames, which was unacceptably costly. From the perspective of RMSE, the results of VisualSfM with SiftGPU method were close or even better than the results of RGBDSLAM v2 with the same configuration, but with ORB, the RMSE values were too large to be considered as good results.

Outdoor RGB dataset [fountain-P11] which features a short path around a fountain, and [castle-P11] – a larger area surrounded by a castle building, were also tested on VisualSfM. The images are of high resolution (3072 x 2048) and captured discretely. The results are shown in Table 4-7.

Table 4-7: The performance test of VisualSfM on outdoor RGB sequences.

RGB sequence (3072x2048)	No. RGB images	Feature detectors and matchers: SiftGPU + VisualSfM matcher					
		Feature detection (s)	Pair-wise matching & Sparse reconstruction (s)	Total processing time (s)	Scale	RMSE (m)	Max. error (m)
[fountain-P11]	11	45	98	<u>143</u>	10.37	0.011	0.02
[castle-P30]	30	89	782	<u>871</u>	6.45	<u>0.342</u>	<u>0.895</u>

RGB sequence (3072x2048)	No. RGB images	Feature detectors and matchers: OpenCV ORB + Brute-force					
		Feature detection & pair-wise matching (s)	Sparse reconstruction (s)	Total processing time (s)	Scale	RMSE (m)	Max. error (m)
[fountain-P11]	11	133	27	160	5.4	<u>0.009</u>	<u>0.015</u>
[castle-P30]	30	1557	677	2234	3.5	51.578	173.510

It can be found that the processing time with the SiftGPU approach was still faster than that with the ORB approach. VisualSfM could handle both relatively smaller areas [fountain-P11] and larger areas [castle-P11] with the SiftGPU approach. However, with ORB approach it only worked well with [fountain-P11] but failed to perform with [castle-P11] (see Figure 4-17 below).

Consider that only the value of RMSE cannot visually reflect the quality of reconstructed map, the **trajectory comparison diagrams** between the estimations and ground truth are presented below for each sequence with different methods. The trajectories were presented from a top view by looking down along the axis of z, y, and x for each row. The **black lines** represented the ground truth trajectories; **the green lines** represented the estimations; and the difference between two trajectories was represented by some **red line segments** linking the estimation and ground truth camera locations with same timestamps.

Figure 4-13 shows the trajectory comparison diagrams of RGBD sequence [freiburg1_desk]. The map generated by each method is very close to the ground truth except VisualSfM (ORB), which also had a higher RMSE value.

Figure 4-14 shows the trajectory comparison diagrams of RGBD sequence [freiburg1_floor]. The estimated camera poses of each method (except VisualSfM (ORB)) were basically close to the ground truth until an obvious 'gap' appeared on the y-z trajectory alignment diagrams at the end. In fact, the last few frames inside [freiburg1_floor] were captured at high speed and these colour images are more blur

than others, which may cause the poor mapping results.

Figure 4-15 shows the trajectory comparison diagrams of RGBD sequence [freiburg1_360]. VisualSfM was failed to generate a valid model in this case. With regard to RGBDSLAM v2, the diagrams show that both approaches represented a complete failure although the RMSE values might not seem big in comparison to that of other RGBD sequences. This is most likely caused by the poor quality of the colour images which have so much blur, resulting in that the low confidence of detected keypoints and their 2D locations.

Figure 4-16 shows the trajectory comparison diagrams of RGBD sequence [freiburg3_long_office_household]. Similar to the result of [freiburg1_desk], all methods produced an accurate map except VisualSfM (ORB).

Figure 4-17 shows the trajectory comparison diagrams of RGB image sets [fountain-P11] and [castle-P11]. It can be found from the 3D trajectory comparison diagrams that there were few displacements along the z axis in these two cases, so only the x-y trajectory alignment diagrams are presented. The reconstruction was performed by VisualSfM. For small set [fountain-P11], both SiftGPU and ORB approaches worked well. But for [castle-P11], only the SiftGPU approach could reconstruct an accurate map whilst the ORB approach failed again.

[freiburg1_desk]_xyz_trajectory

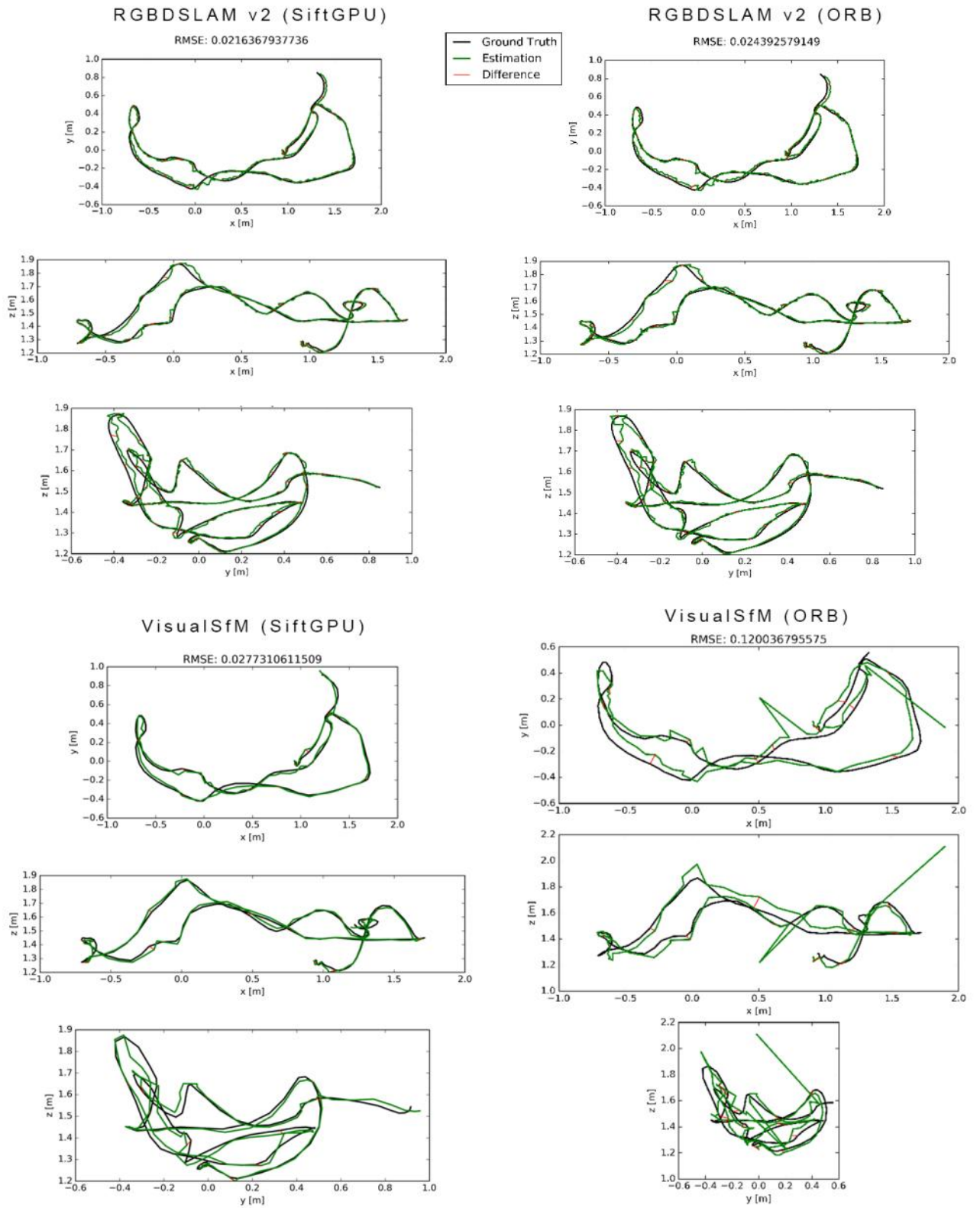


Figure 4-13: The trajectory comparison diagrams for the results of [freiburg1_desk].

[freiburg1_floor]_xyz_trajectory

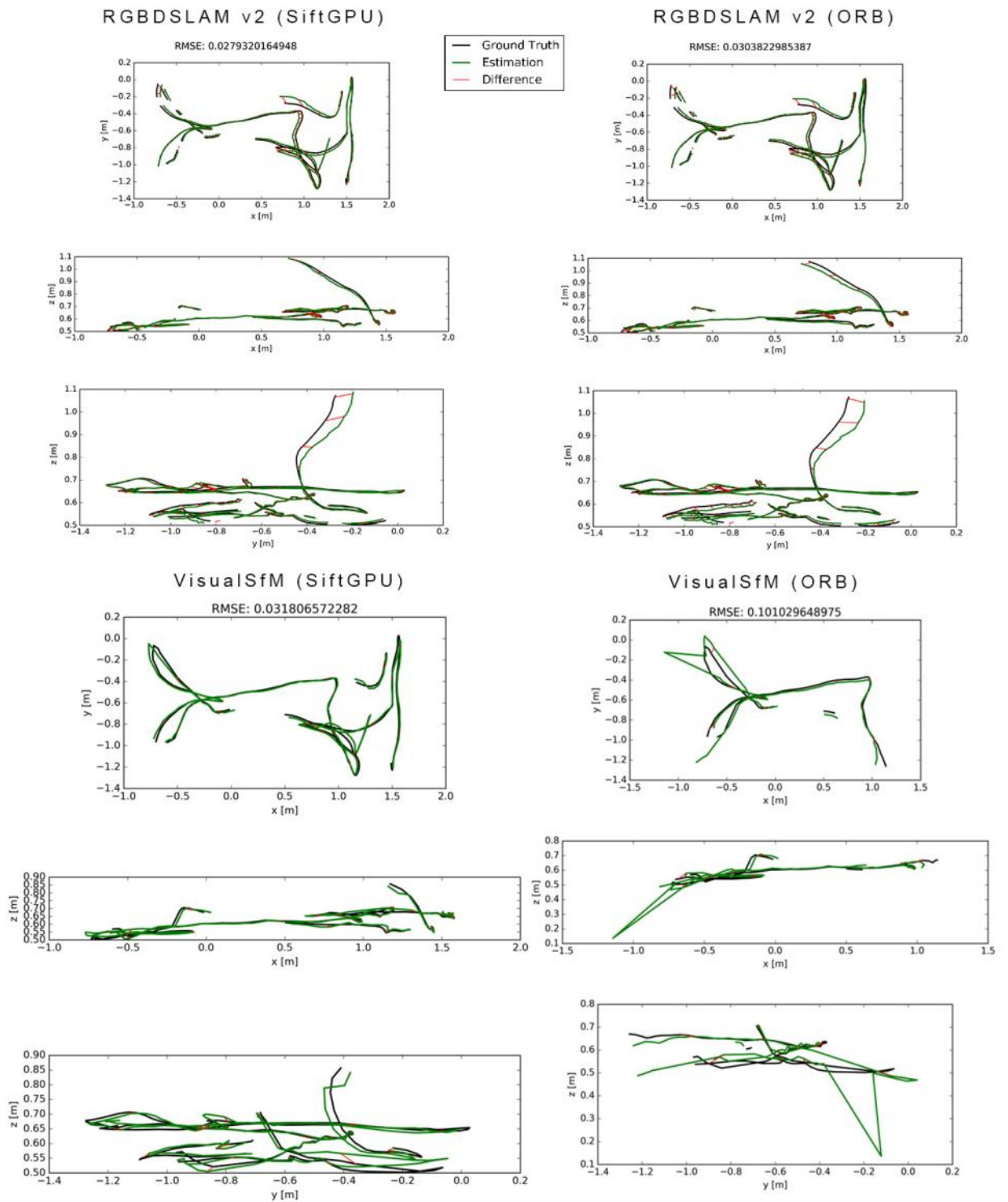


Figure 4-14: The trajectory comparison diagrams for the results of [freiburg1_floor].

[freiburg1_360]_xyz_trajectory

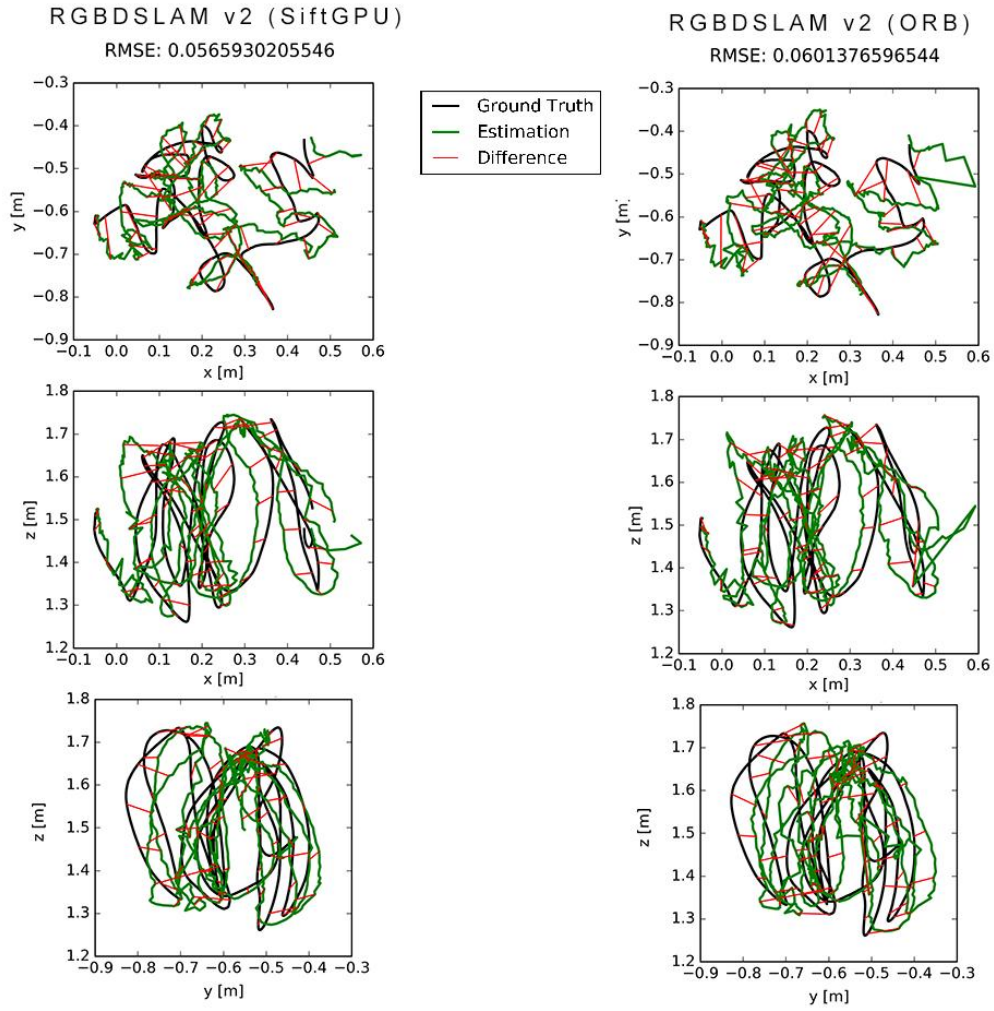


Figure 4-15: The trajectory comparison diagrams for the results of [freiburg1_360].

[freiburg3_long_office_household]_xyz_trajectory

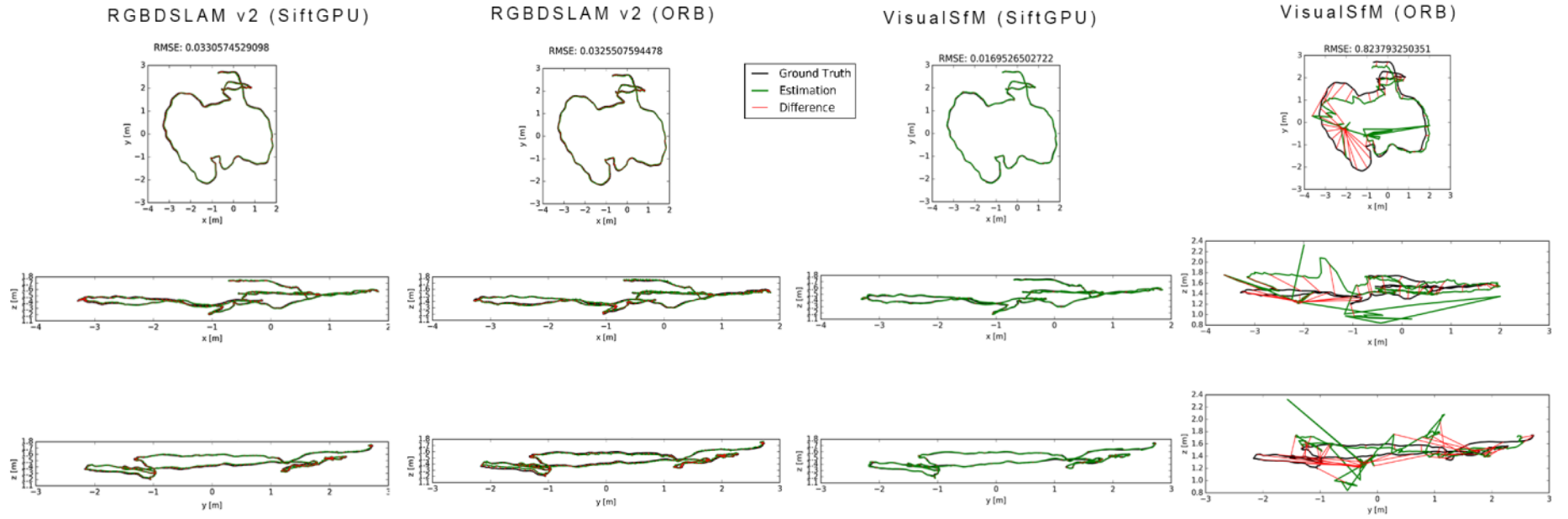


Figure 4-16: The trajectory comparison diagrams for the results of [freiburg3_long_office_household].

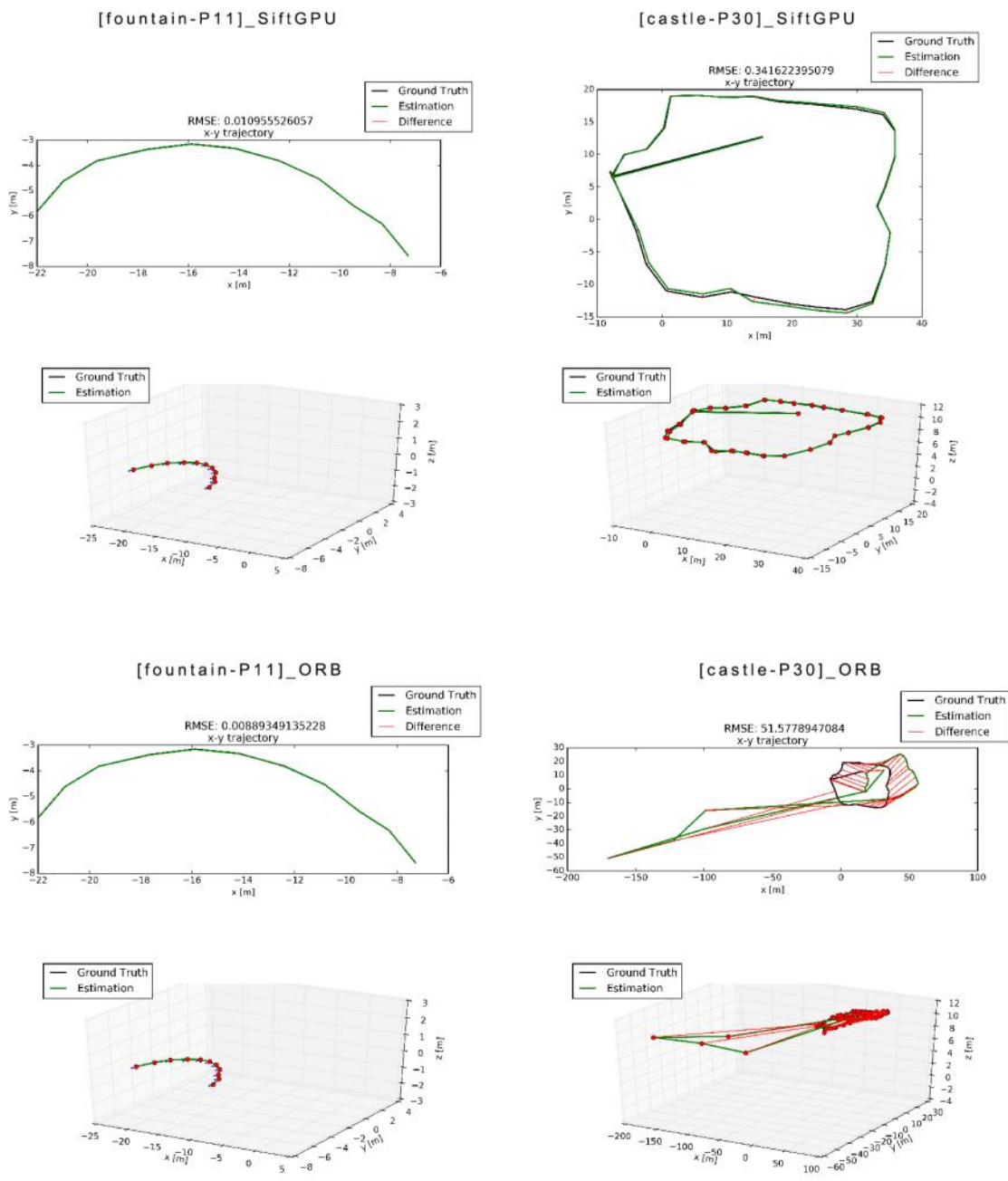


Figure 4-17: The trajectory comparison diagrams for the results of [fountain-P11] and [castle-P11].

In order to ensure that the reconstructed map can be used for AR registration, a simple AR browser implemented with OpenCV (see Section 6.2.1 for detail) was used for displaying pose estimation visually. Objectively, the overlapping ratio between the AR boarding box projected by ground truth and estimation camera poses could be calculated as reference for evaluation. However this method requires the boarding box always be seen in the viewport, but it is unavailable in most tested sequences except [freiburg3_long_office_household] in which the camera always focused on an office desk in the middle. For example, the reconstructed results of [freiburg3_long_office_household] generated by RGBD with SiftGPU shown in Table 4-5 had a mean overlapping rate of 82% with a variance of 0.03, but without comparison and the subjectively measurement opinions from real user, it does not make sense.

4.4. Conclusion

In this chapter, two 3D reconstruction methods – RGBD-SLAM and SfM – are described and applied in the offline session of the present proposal, to create references for the AR user tracking process. These two methods can handle different types of input data (*i.e.* continuous ordered RGBD dataset, or unsorted, discrete RGB datasets). The developers can choose one of them by considering which types of device they wish to use. In the present proposed system, use of the existing applications RGBDSLAM v2 (Endres *et al.*, 2014) and VisualSfM (Wu, 2011; 2013) has been described. Their performance with two representative visual features – SiftGPU (Wu, 2007) and ORB (Rublee *et al.*, 2011) – is tested on several RGBD and RGB public datasets. The

performance of RGBDSLAM v2 with SiftGPU and ORB methods is almost the same, although the SiftGPU method performed a slightly better than the ORB method. The results reflect that the input dataset used for vision-based 3D reconstruction should avoid involving poor-quality images, such as blurred images caused by drastic camera motions. The precision of VisualSfM with SiftGPU method is superior to the same configuration of RGBDSLAM v2, although it takes more time for processing, but with the ORB method the situation is quite different. VisualSfM only supports SiftGPU methods natively, thus the ORB features need to be detected and matched separately. Although VisualSfM allows users to import the feature matches themselves, the expensive time-consuming of file importing is unacceptable. Moreover, most results of VisualSfM using ORB feature matches are not ideal to restore a reliable reference for the subsequent AR online session. In fact, the acceptable accuracy for 3D reconstruction/mapping in AR is still unclear. The subjective visual evaluation requires real users to be involved to draw a conclusion using general quantitative criteria (such as mean opinion scores (Knoche *et al.*, 1999)). This may also suggest that the objective visual evaluation metrics and benchmark datasets for 3D reconstruction/mapping on AR reference template use need to be designed and created in the future.

Chapter 5 Vision-based Template Training and User Tracking

Following the offline 3D reconstruction process presented in the last chapter, the proposed system will convert the output results to reference models which will be used for user tracking in AR application, namely template training. The actual purpose of the training stage is to let the system ‘learn’ and ‘remember’ the scene to augment. Thus, when the application user is “re-visiting” the places which have been previously stored in database, the system will recognise the places based on their appearance. This is also known as **loop closing** in SLAM. Then the system can estimate the direction in which the user camera is looking and how the AR content should be displayed to the user. This process is expected to be performed during the AR application running time and is referred to as the **Online Session** hereinafter.

The main task in the online session, in brief, is retrieving an image from the trained database which has the closest appearance to the online input, and attempting to estimate the user pose with respect to the reference map based on it. The work flow of this retrieval procedure is depicted in Figure 5-1, and more detail is discussed in the following paragraphs.

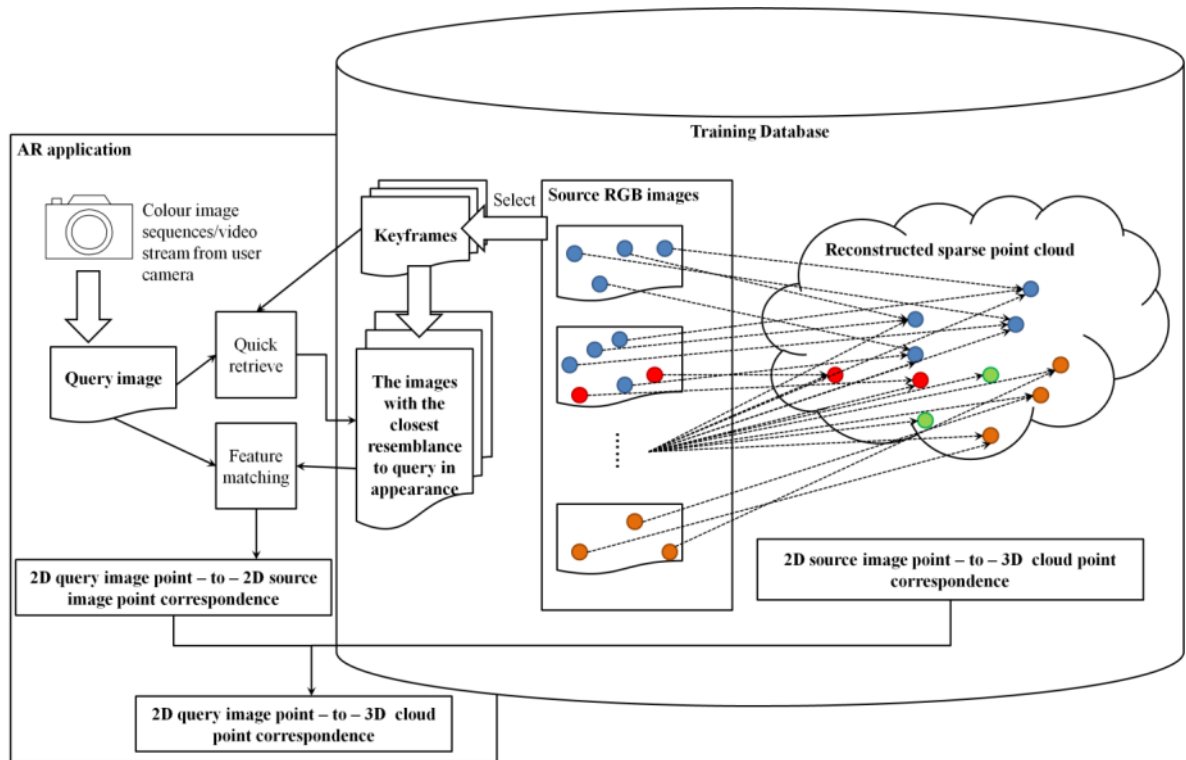


Figure 5-1: The work flow of the proposed online session to query input images from the trained database.

In current proposal, only a standard, off-the-shelf **monocular camera** is used as the input sensor, although an option of using RGBD data for 3D reconstruction has been offered to the application developers in the offline session. From a practical standpoint, the price of some RGBD cameras is not too high for ordinary people to afford, but it is still an optional accessory for potential application users who may not want to buy one. In addition, the usage of the RGBD camera is generally limited to indoor environments and some of the devices even require an external power supply to work (such as the Microsoft Kinect 1.0). In contrast, the monocular cameras have been integrated into many smaller, portable devices (such as smartphones) and are, therefore, more likely to be accepted by the general public. In the proposed framework, no matter which approaches presented above are used in the offline session for 3D reconstruction, their output data will be trained and unified to purely vision-based templates, and, by only

using 2D colour images as the input to the AR runtime. This will be sufficient for the system to deal with the user tracking task. Specifically, the system will try to find several 3D-to-2D coordinate correspondences between the sparse cloud points recovered in the offline session and the image points extracted in the online session. These 3D points share the same feature descriptors with their originated 2D keypoints. Thus, in principle, the desired 3D-to-2D correspondences during the online tracking process can be obtained by matching the feature descriptors between the restored cloud point and the input image keypoints. However, some keypoints possess very similar feature descriptions (appearance), but actually do not come from a same place in the real world, leading to the wrong correspondences. This kind of mismatch errors can be properly avoided by considering the nature of the spatial distribution of the 3D points: some of features are more likely to be observed together because they are generated by common objects, others are not. Some of these relations have already been indicated in the train image data of the 3D reconstruction. Hence, the application input images will be compared with a selected subset of the train images from the training database instead of the unorganised point cloud. The 3D points are grouped and associated with the train images which have their observations. From the above, the training stage will associate three properties with each train images in database: 1) a list of detected keypoints and their feature descriptors; 2) the mapping between the 2D image coordinates and the 3D world coordinates of the keypoints; 3) the estimated camera pose. When the AR application is processing an input image as query, the system will firstly retrieve a subset of train images from the database which have the closest appearance to the query, then it will attempt to identify the 2D-to-2D correspondences between the query and these images, and further to establish the 3D-to-2D

correspondences. However, how to efficiently retrieve the candidates that have the closest appearance to the real-time input image of the application becomes another problem.

Some of the train images used for the 3D reconstruction may contain a large body of repetitive content, especially when continuous images are used. It makes no sense to undertake comparison with all of them one by one to find a close match one as this is very inefficient too. Therefore, a set of **keyframes** was selected during the offline session as the representatives which covers the most features of the 3D reconstruction but with less overlap between each other, as defined in Section 4.1.2.2. The online session will use these keyframes as templates for loop closure detection. Due to the continuity of input data where the adjacent images usually have similar appearances, the loop of following-up input images can usually be closed by conducting comparison with their predecessors' matching results. However, there are **two cases** requiring the system to perform a full search through all keyframes. Firstly, when an AR application is started up, the first few images acquired should always compare with all keyframes until the first loop closure is detected, due to the lack of the knowledge of the initial pose of the user camera. Secondly, when the user tracking has failed, the following-up images acquired should be compared with all keyframes until the system re-locates them again. Although the images with too many repetitive contents have already been removed from the set of keyframes, the reconstruction of wide areas will generate plenty of keyframes for representing different places. The comparisons based on local feature matching are still very time-consuming, which is unacceptable in real-time AR applications. It is necessary to avoid performing precise feature matching between query

and keyframes in the image retrieval stage. A coarse but more efficient comparative method is desired to find the candidates with the highest likelihood to match the query, and then the feature matching will be performed to check if there are sufficient correspondences between the query and the selected candidates to establish valid transformations.

For these reasons, a bag of words (BoW) based approach has been adopted in the present research for encoding each image into a global descriptor vector – or *signature*. Each element of the signature vector is associated with a visual word, specifying how important this word is to the image. The visual words are obtained by clustering several selected local visual features and these will have same format as the feature descriptor used. The collection of all visual words forms a vocabulary and any image can look up the visual features via the vocabulary to encode itself as a one-dimensional BoW signature. The process of vocabulary training and the image signature coding will be described in Section 5.1, which takes account of both the histogram-based descriptors (SIFT-like) and the binary descriptors (ORB). After encoding the image data, *openFABMAP* (Glover *et al.*, 2012) will be applied to quickly select the keyframes with the highest likelihood to match the query image acquired in the online session. This will be introduced in Section 5.2. The complete real-time camera pose estimation process of image sequences with the above methods will be presented in Section 5.3 and the evaluation and commentary on the results are given in Section 5.4.

5.1. Bag of words and vocabulary

For vision-based AR applications, a crucial task is tracking the user with respect to a pre-built reference map by using the image sequence acquired from the user's camera and the offline trained database. This is known as a loop closure detection problem in SLAM which engages to identify an old place from the new input based on the built map. As mentioned in Section 4.1.2.2, in order to perform a graph-based optimisation for 3D reconstruction robustly, the loops to old nodes need to be found as constraints for each new added node. The candidates to compare are basically selected from the direct predecessors and their neighbours due to the topological structure of the graph. In addition, the keyframes which are used as appearance-based thumbnails of different locations in the reference map are also considered. However, the initial state of the end user who initiates the application in the online session is totally random, thus only keyframes can be used for detecting loop closure. This can be considered as an image retrieval problem where the system is attempting to search the user query images amongst the keyframes. The simplest technique for this is to compare the query with all of the representative thumbnail images one by one, but the cost of performing feature matching can be really expensive and inefficient, as mentioned above. Suppose each image contains an average of n features, and there are m keyframes in a database to compare for searching for a query. In such a case, the total cost of feature comparisons will be $m * n^2$. Unlike the offline session, the online session expects to achieve real-time processing so that the time to process each image should be near to the time to acquire it. Therefore the one-to-one feature matching should be avoided. In order to improve the efficiency of image retrieval, the bag of words method is used in the present research. A visual vocabulary is trained based on a set of local virtual features selected from a

specific environment (such as an office-like indoor environment or an urban outdoor environment). All detected features in the training data are grouped in different clusters due to the distance measurement used in feature matching between the descriptor vectors. Each cluster is represented by its central vector with the mean value of the clustered descriptors, referred to as a visual word. All of these visual words make up the vocabulary, which can be used to abstract an image to a single signature vector S , the length of which equals the vocabulary size $|v|$. In order to encode an image with n features to a signature, at most $|v| * n$ feature comparisons will be performed. Then, to rank the amount of the similarities between the query and the m keyframes, another m comparisons based on signature are required. Since the size of the vocabulary $|v|$ is typically far less than the total number of features $m * n$ contained within the keyframes, the required times of feature matching during the image retrieval are effectively reduced.

The detail of the visual word clustering for two kinds of local visual features – SIFT-like and ORB is described in Subsection 5.1.1, and the generation of the signatures of images based on visual vocabulary is presented in Subsection 5.1.2.

5.1.1. Visual word clustering

One of the most common used clustering algorithms is *k-means clustering* (Steinbach *et al.*, 2000). Assume each feature descriptor used for vocabulary training is denoted by a data point. The k-means clustering aims to partition these data points into k clusters, where each point will be grouped into the cluster with the nearest cluster centroid. Once all points are assigned to the respective clusters, the mean value of the data points in a

cluster will be calculated and used as the new cluster centroid. Then all data points will be re-partitioned to the new clusters to calculate their respective mean values for updating another set of new centroids. This process will be performed iteratively until the assignments of each point no longer change, which means the algorithm has converged. The speed of convergence in k-means clustering is mainly based on the choice of the initial cluster centroids. An effective approach, k-means++ (Arthur & Vassilvitskii, 2007), is applied to deal with the initial values of the cluster centroids as follows:

- 1) Choose the first cluster centroid uniformly at random from the data points.
- 2) For each data point, compute the distance between it and the nearest centroid, which has already been chosen.
- 3) Choose a new cluster centroid from the remaining data points with a probability proportional to the squared distance from the nearest existing cluster centroid of each point.
- 4) Repeat 2) and 3) until k centres have been chosen.

Although the initial selection in the k-means++ takes extra time, the following process of k-means will converge quickly which actually lowers the computation time.

Theoretically, the max size of vocabulary equals the total number of the features contained within the keyframes when each different feature descriptor forms a cluster by itself. The more visual words (clusters) are generated, the more precise signature vector (with more elements) can be used to represent images. But it also takes more time for encoding. Although there is no mandatory requirement for choosing the

number of cluster, the time of encoding input images during the online session should meet the real-time requirement. It mainly depends on what type of visual feature is used and how long the matching method will take to compare feature descriptors.

In order to generate the visual words through a set of feature descriptors by using k-means clustering, the distance and mean values of the feature descriptors used must be defined first. There are two feature extraction methods tested in the present research: SIFTGPU with 128-dimensional histogram-based SIFT-like descriptors and ORB with 256 bits binary descriptors.

In actual programming, the 128-D SIFT-like descriptor vectors can be represented simply by using a floating point array. The distance between two vectors is defined by Euclidean distance (see Appendix D), and the mean value is literally the average of a set of vectors. On the other hand, the Hamming distance is used to measure the difference between the binary descriptors. The 256 binary bitset is usually stored as an *Unsigned Char* data array of length 32 in the computer, and the Hamming distance between two bitsets can be calculated by using bitwise XOR, and counting the bits that differ. An example of the Hamming distance is given in Figure 5-2.

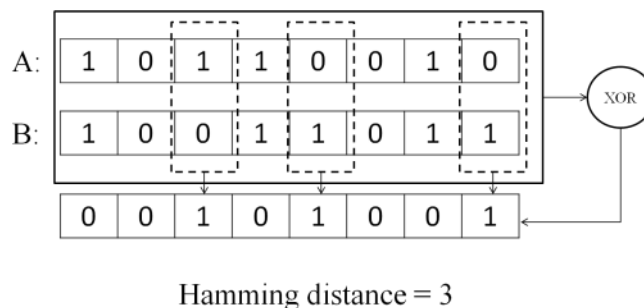


Figure 5-2: an example of the Hamming distance between two 8 binary bitsets. The hamming distance equals to the number of total differences between the two bitsets.

The mean value of a set of binary bitsets is determined by testing each bit value amongst the whole set: if more than half of the members have ‘1’ on a bit, the mean value will set the corresponding bit to ‘1’, otherwise the bit will be set to ‘0’.

An alternative vocabulary generation method for SIFT-like descriptors is recommended by Cummins & Newman (2011) who use a fixed Euclidean distance threshold for cluster partition. This method will randomly pick a descriptor first as one of the initial cluster centres. The Euclidean distance between this first initial cluster centre and each of the other descriptors will be inspected sequentially. If the distance exceeds the threshold, this descriptor will be considered as another initial cluster centres, and the following descriptors will be compared with all these selected initial cluster centres to determine if they can be an initial cluster centre or not. Once all initial cluster centres are determined, the rest of the descriptors will be assigned to the centre with the nearest distance to them respectively. The mean value of each cluster will be calculated as the final cluster centre used as visual word. The pseudo codes of this algorithm are presented in Table 5-1.

Table 5-1: The BoW cluster algorithm for SIFT-like descriptors suggested by Cummins & Newman (2011)

```
// Suppose descriptors_[] is an unsorted array of the feature descriptor to cluster. The first descriptor is
picked up as the first initial cluster centre and pushed onto a stack initial_centres_.
initial_centres_.push(descriptors_[0]);
For each d_ in descriptors_:
    min_distance = MAX_DISTANCE;
```

```

For each ic_ in initial_centres_:

    min_distance = max(min_distance, distance (d_, ic_));

End for (initial_centres_)

    // If all distances between a descriptor and existing initial cluster centres exceed the pre-defined
    // threshold value cluster_threshold_, this descriptor will be treated as a new initial cluster centre and
    // pushed onto initial_centres_.

    If min_distance > cluster_threshold_:

        initial_centres_.push(d_);

    End if
End for (descriptors_)

// Assign the descriptors d_ to its nearest initial cluster centre c_.

For each d_ in descriptors_:

    For each ic_ in initial_centres_:

        If ic_ is the nearest initial cluster to d_:

            clusters_[ic_].push(d_);

        End if

    End for (initial_centres_)

End for (descriptors_)

// Recalculate the average of each cluster as the new cluster centre (i.e. visual word) and push them onto
vocabulary_ as final output.

For each c_ in clusters_:

    visual_word = average(c_);

```

```
vocabulary_.push(visual_word);  
End for (clusters_)
```

Cummins & Newman (2011) argue that the randomly chosen cluster centres of k-mean tend to lie largely within the densest region of the feature space due to the fixed number of partitions (*i.e.* ‘k’ clusters), but the metric distances between the clusters are variant, which may cause tiny variations between generated words through the dense region.

5.1.2. BoW signatures

The visual vocabulary is used for abstracting images to signature vectors $S = (w_1, \dots, w_{|v|})$, where $|v|$ is the number of visual words in the vocabulary and the variable w_q corresponds to the q -th word of the vocabulary, reflecting the importance (a.k.a *weight*) of this word to the image. Thus an image can be represented by a one-dimensional array instead of a set of high-dimensional feature descriptors. In order to generate a signature, the image should look up the vocabulary for assigning the detected local features to the visual words with the nearest distance. This requires at most $|v| * n$ comparisons between the image features and the visual words if the vocabulary is unstructured. A reasonably large vocabulary size can ensure the accuracy of the signatures for interpreting the images with different contents. Although the comparison times are superior to the one-to-one direct feature matching between the images, it would still take a long time for coding the image signatures when a mass of words is involved. In order to further reduce the time of looking-up a word, the vocabulary is organised to a *hierarchical k-means tree*. The original k-means clustering has a flat structure and each

partition is independent of each other. In contrast, the hierarchical k-means tree represents the partitions using a tree structure, as shown in Figure 5-3.

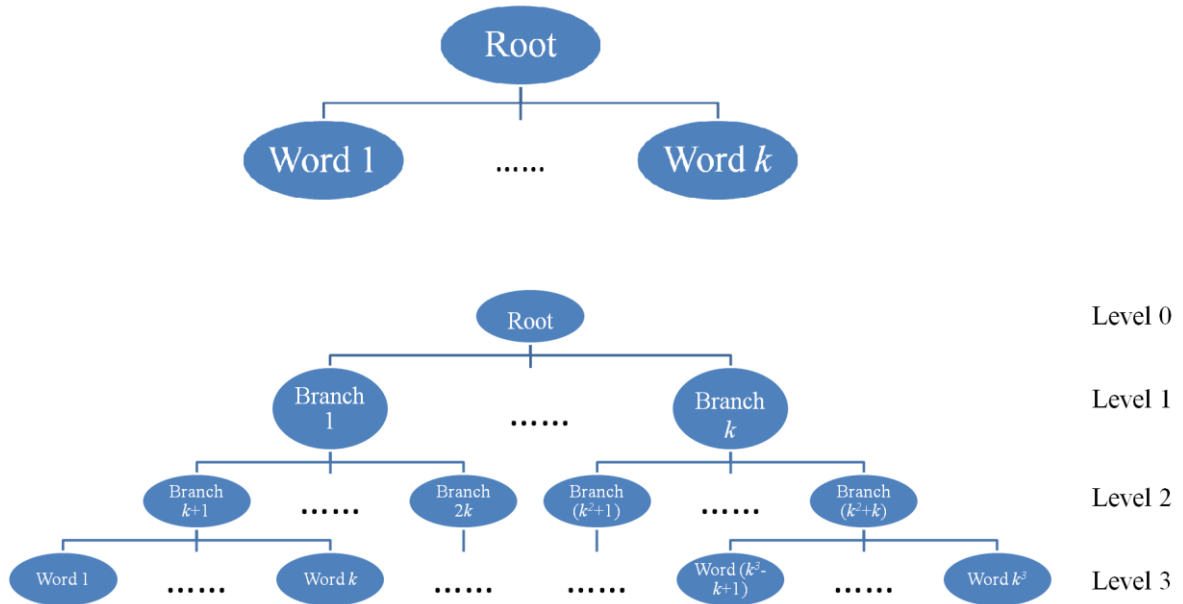


Figure 5-3: K-means clustering (upper) and a 3-level depth hierarchical k-means tree (lower).

Each level of the tree divides the data points contained within their parent node into k subsets by using standard k-means clustering. The recursion terminates when the dataset is divided into single data points or a given depth level has been reached. There are two arguments that should be carefully considered in hierarchical k-means clustering: the

number of the cluster centroids k and the depth level of the tree l . A total of $\frac{k^{(l+1)} - 1}{k - 1}$

nodes are used to represent the centroids of each sub cluster, and the k^l leaf nodes, which, at the deepest level, represent the visual words. Although it will take extra time to build the hierarchical tree during the offline training session, the comparison times of searching a word has reduced from comparing all k^l words to traversing $k * l$ branch

nodes.

After assigning all image features to the respective visual words, the weight of each word to the image can be assessed. A common weighting method in image retrieval is term frequency-inverted document frequency (tf-idf) (Sivic & Zisserman, 2003). For BoW, the term frequency indicates the number of times a word occurs in an image as a simple proportional representation, and the inverted document frequency is an inverse function of the number of the training image data in which a word occurs, which quantifies the specificity of a word. A standard tf-idf weight value w_i of the i -th word in the vocabulary is calculated as:

$$w_i = \frac{n_i}{N} \log \frac{M}{m_i} \quad (5.1)$$

where n_i is the number of occurrences of the i -th word in the query image, N is the total number of features (assigned to the visual words) in the query image, M is the total number of training images, and m_i is the number of the training images where the i -th word appears. The tf-idf weight value of each word to the encoding image is held by the corresponding element in the signature vector.

5.2. Image retrieval and loop closing

The aim of image retrieval is to determine whether there are some template images in database that have similar appearance to the query image. In the present proposal, image retrieval is used to decide if the new observation of the user camera in the online session

originates from places within the reference map, or from unknown places. However, the decision of whether loop closures have been found may be misled by perceptual aliasing, which refers to different places having similar appearance in the environment. As stated in Cummins & Newman (2008), even though two images share many features, they still can be false matches due to perceptual aliasing. Cummins & Newman (2008) deals with the issue by proposing a *Fast Appearance-Based Mapping* (FAB-MAP) system, which utilises BoW and extends the approach by learning a generative model offline from a set of training data, capturing the fact that certain combinations of visual words tend to co-occur (as mentioned above). They have proved that the FAB-MAP approach can effectively recognise places and reject the mismatch errors, which also has a reasonable computational cost for online loop closure where the map contains several thousand places. Based on these features, the FAB-MAP will be applied in the online session to decide whether the input images are captured from a known place in database. Glover *et al.* (2012) implement an Open Source toolbox openFABMAP integrated into OpenCV which will be used in the proposed system. The relative algorithms and detailed process are described below.

5.2.1. Chow-Liu tree

A key feature of FAB-MAP is that it models the dependence of feature co-occurrence in the environment. Because the visual words of the same object are likely to appear or disappear together, when matching queries to the templates with these words, the likelihood will be higher when all words are observed, as opposed to only a partial set. The full distribution of visual word co-occurrence is learned by using a *Chow-Liu*

dependency tree which is calculated from a training dataset (e.g. a collection of the images which have similar visual features with the reconstructed scene but are not completely identical). Consider a distribution $\mathbf{P}(S)$ of $|v|$ discrete variables of signature vectors $S = (w_1, \dots, w_{|v|})$, the parameters of which are expected to be learned from the training data. If $\mathbf{P}(S)$ is a general distribution without a special structure, the space needed to represent the distribution increasing exponentially in $|v|$, which quickly becomes intractable when the size of $|v|$ increases. To deal with it, another distribution $\mathbf{Q}(S)$ is generally used to approximate $\mathbf{P}(S)$ which possesses some special structure that makes it tractable to work with, such as an extreme structural constraint – naive Bayes approximation, restricting each variable must be independent of all others. The similarity between $\mathbf{P}(S)$ and $\mathbf{Q}(S)$ is defined by the *Kullback–Leibler (KL) divergence*:

$$\mathbf{D}_{\text{KL}}(\mathbf{P}, \mathbf{Q}) = \sum_{x \in X} \mathbf{P}(x) \log \frac{\mathbf{P}(x)}{\mathbf{Q}(x)} \quad (5.2)$$

where the summation is carried out over all possible states in the distribution. The KL divergence is zero when $\mathbf{P}(S)$ and $\mathbf{Q}(S)$ are identical and strictly larger otherwise. The Chow-Liu algorithm approximates $\mathbf{P}(S)$ by the closest tree-structured *Bayesian network* $\mathbf{Q}(S)_{\text{opt}}$ in the sense of minimising the KL divergence, requiring less severe constraint than the naive Bayes approximation. Bayesian networks are probabilistic graphical models that represent a set of random variables and their conditional dependencies via a directed acyclic graph. The graphical model of naive Bayes approximation only has $|v|$ nodes and does not have any edges between them. In contrast, the tree-structured $\mathbf{Q}(S)_{\text{opt}}$ is determined by considering the complete graph with $|v|$ nodes and $\frac{|v| * (|v| - 1)}{2}$ edges, where the edge (w_i, w_j) between the node w_i and w_j has *mutual information* $\mathbf{I}(w_i, w_j)$ as its weight:

$$\mathbf{I}(w_i, w_j) = \sum_{w_i \in \Omega, w_j \in \Omega} \mathbf{p}(w_i, w_j) \log \frac{\mathbf{p}(w_i, w_j)}{\mathbf{p}(w_i)\mathbf{p}(w_j)} \quad (5.3)$$

where the summation is carried out over all possible states of w , *i.e.* $w > 0$ if the word occurs or $w = 0$ otherwise. $\mathbf{p}(w_i)$ and $\mathbf{p}(w_i, w_j)$ can be calculated from the frequency of word occurrence and co-occurrence in training data. Mutual information measures the degree to which knowledge of the value of one variable predicts the value of another. It is zero if two variables are independent and strictly larger otherwise. The Chow-Liu algorithm forms a minimum spanning tree which maximises information entropy and will have the same structure as $\mathbf{Q}(S)_{\text{opt}}$, where the dependencies between variables with little mutual information are omitted and the corresponding variables are approximated as independent. The joint distribution over word occurrence is contained within the Chow-Liu tree.

The openFABMAP package supports the build of a Chow-Liu tree from the set of signatures of training data with Class *ChowLiuTree*. The structure of the tree produced is stored in a $4 \times |v|$ matrix held by *cv::Mat* structure of the OpenCV. The q -th column corresponds to the q -th word node w_q , where the first row stores the parent node index of the word p_q ; the second row stores the unconditional probability of the word occurrence $\mathbf{p}(w_q > 0)$; the third and fourth rows store the conditional probabilities of the word occurrence, given its parent $\mathbf{p}(w_q > 0 | w_{p_q})$. The Chow-Liu tree then will be used in the main process of FAB-MAP for estimating *observation likelihood*, in other words, the probability distribution over the observation of BoW signature given the possible locations.

5.2.2. Location representation and likelihood

The 3D reconstruction in the Offline Session can be divided to several discrete locations based on selected keyframes. Assume there are m keyframes. The reference map can be denoted by $\mathbf{L} = (\mathbf{L}_1, \dots, \mathbf{L}_m)$. Each keyframe is associated with a BoW signature and its camera pose with respect to the world reference frame. Theoretically, this signature indicates the appearance representation of the observation that can be acquired in a corresponding pose. However the word observation w_q on the keyframes are actually noisy measurements of the existence of the underlying scene element e_q , which may or may not exist at that location. It is easy to find that each word does not make an equal contribution to recognise a location, which means the similarity between two images should not be measured via a simple distance metric between their signature vectors. FAB-MAP associates each location with an appearance model which indicates the belief about the existence of each scene element at the location:

$$\mathbf{L}_i: \left\{ \mathbf{p}(e_1 = 1 | \mathbf{L}_i), \dots, \mathbf{p}(e_{|V|} = 1 | \mathbf{L}_i) \right\} \quad (5.4)$$

where each individual observation probability $\mathbf{p}(e_q = 1 | \mathbf{L}_k)$ is estimated by modelling the reliability of visual word detection $\mathbf{p}(w_q | e_q)$ and the prior knowledge of how common a scene element e_q is in the environment:

$$\mathbf{p}(e_q = 1 | \mathbf{L}_i) = \frac{\mathbf{p}(w_q | e_q = 1)\mathbf{p}(e_q = 1)}{\sum_{s_e \in \{0,1\}} \mathbf{p}(w_q | e_q = s_e)\mathbf{p}(e_q = s_e)} \quad (5.5)$$

The core Class `FabMap` of `openFABMAP` will ask for detector model recall ($\mathbf{p}(w > 0 | e = 1)$) and detector model precision ($\mathbf{p}(w > 0 | e = 0)$) as input parameters $PzGe$ and $PzGNe$ used to account for detector noise and precision.

FAB-MAP calculates observation likelihood via word co-occurrence model stored in the offline produced Chow-Liu tree. The probability distribution of signature S given the location L_i can be calculated as:

$$\mathbf{p}(S | L_i) = \mathbf{p}(w_r | L_i) \prod_{q=1}^{|v|-1} \mathbf{p}(w_q | w_{p_q}, L_i) \quad (5.6)$$

where w_r is the root of the tree, w_{p_q} is the parent of w_q in the tree and

$$\mathbf{p}(w_q | w_{p_q}, L_i) = \sum_{s_e \in \{0,1\}} \mathbf{p}(w_q | e_q = s_e, w_{p_q}) \mathbf{p}(e_q = s_e | L_i) \quad (5.7)$$

Specifically consider the binary states of w_q and w_{p_q} as s_q and s_{p_q} , then

$\mathbf{p}(w_q = s_q | w_{p_q} = s_{p_q}, L_i)$ can be calculated as:

$$\begin{aligned} \alpha_1 &= \mathbf{p}(w_q = s_q) * \mathbf{p}(w_q = \bar{s}_q | e_q = 0) * \mathbf{p}(w_q = \bar{s}_q | w_{p_q} = s_{p_q}) \\ \beta_1 &= \mathbf{p}(w_q = \bar{s}_q) * \mathbf{p}(w_q = s_q | e_q = 0) * \mathbf{p}(w_q = s_q | w_{p_q} = s_{p_q}) \\ \alpha_2 &= \mathbf{p}(w_q = s_q) * \mathbf{p}(w_q = \bar{s}_q | e_q = 1) * \mathbf{p}(w_q = \bar{s}_q | w_{p_q} = s_{p_q}) \\ \beta_2 &= \mathbf{p}(w_q = \bar{s}_q) * \mathbf{p}(w_q = s_q | e_q = 1) * \mathbf{p}(w_q = s_q | w_{p_q} = s_{p_q}) \end{aligned} \quad (5.8)$$

$$\mathbf{p}(w_q = s_q | w_{p_q} = s_{p_q}, L_i) = \frac{\mathbf{p}(e_q = 0 | L_i) * \beta_1}{\alpha_1 + \beta_1} + \frac{\mathbf{p}(e_q = 1 | L_i) * \beta_2}{\alpha_2 + \beta_2}$$

Actually the only quantified appearance representation of locations in the present proposal are keyframes. Thus suppose every detected word in the keyframes exactly corresponds to a real feature in the scene (*i.e.* $\mathbf{p}(w > 0 | e = 0, L_i) = 0$). The localisation likelihood $\mathbf{p}(S_{\text{query}} | L_i)$ is equivalent to the comparison likelihood between query and

the i -th keyframe. FABMAP applies a vote strategy based on BoW for calculating the match probability between each query and the keyframes. The term $\mathbf{p}(w_q | w_{p_q}, L_i)$ in equation (5.6) represents the likelihood that the q -th word existing in the i -th keyframe. $\mathbf{p}(w_q | w_{p_q}, L_i)$ was replaced with a restricted model for enabling an efficient implementation using inverted index (*i.e.* the inverse mapping from words to images). If the q -th word was not previously observed in some locations, the related probability is denoted by $\mathbf{p}(w_q | w_{p_q}, L)|_0$ which shares a single common value for all these locations.

Then, converting them to log-likelihoods, where if the q -th word was observed in L_i , the weights of the votes this word casts for the i -th keyframe is $\log\left(\frac{\mathbf{p}(w_q | w_{p_q}, L_i)}{\mathbf{p}(w_q | w_{p_q}, L)|_0}\right)$,

otherwise it equals 0. For each observed word on query, the inverted index is used to retrieve the list of the keyframes in which it occurs and the corresponding likelihood will be updated by adding up the word log-likelihoods.

One more thing to note is that it is entirely possible for the application user to move out of the range of 3D reconstruction, which means the user camera pose at that moment cannot be recovered through the reference map in database, but the acquired image may have similar features with some keyframes. In order to avoid false matching, openFABMAP utilises the set of training data to evaluate a likelihood of the query observation from an unknown location. It firstly calculates the comparison likelihood between query and each image used for training then generates an ‘average likelihood’ which represents the probability of the query matching a virtual location with average appearance. If the average likelihood is high, it means that most of the words in the

query image are very common in the environment which may cause perceptual aliasing. If no other likelihood is higher than the average likelihood, the retrieval of query image is considered to have failed. The subsequent input images will reset to “initial” state with no prior knowledge.

For the subsequent input after a successful matching, it will have a high probability to match the same keyframe with its predecessor as discussed in Section 4.1.2.2. If RGBDSLAM v2 was used for 3D reconstruction, it would select and store keyframe information sequentially from the input image sequence. Thus the adjacent keyframes represent the adjacent places of the reconstructed environment, which means if the best match of the query image acquired at time t is considered as the i -th keyframe, then the query image acquired at time $t + 1$ is likely to match one of the keyframes $\{i - 1, i, i + 1\}$. Therefore the proposed system will try to match the subsequent with these three candidates first, if no valid estimation can be found then the system will apply FAB-MAP. This is more efficient than encoding the signature of each query and computing the match probabilities between it and all keyframes. VisualSfM does not select keyframes during the 3D reconstruction, so an additional keyframes selection needs to be carried out offline by using the estimated camera pose of each input source image. The images are sorted by their position and orientation and the sequential keyframes of adjacent places are selected based on the same rule defined in Section 4.1.2.2.

5.3. Implementation

The whole template training process and the usage of the database data are shown in Figure 5-4.

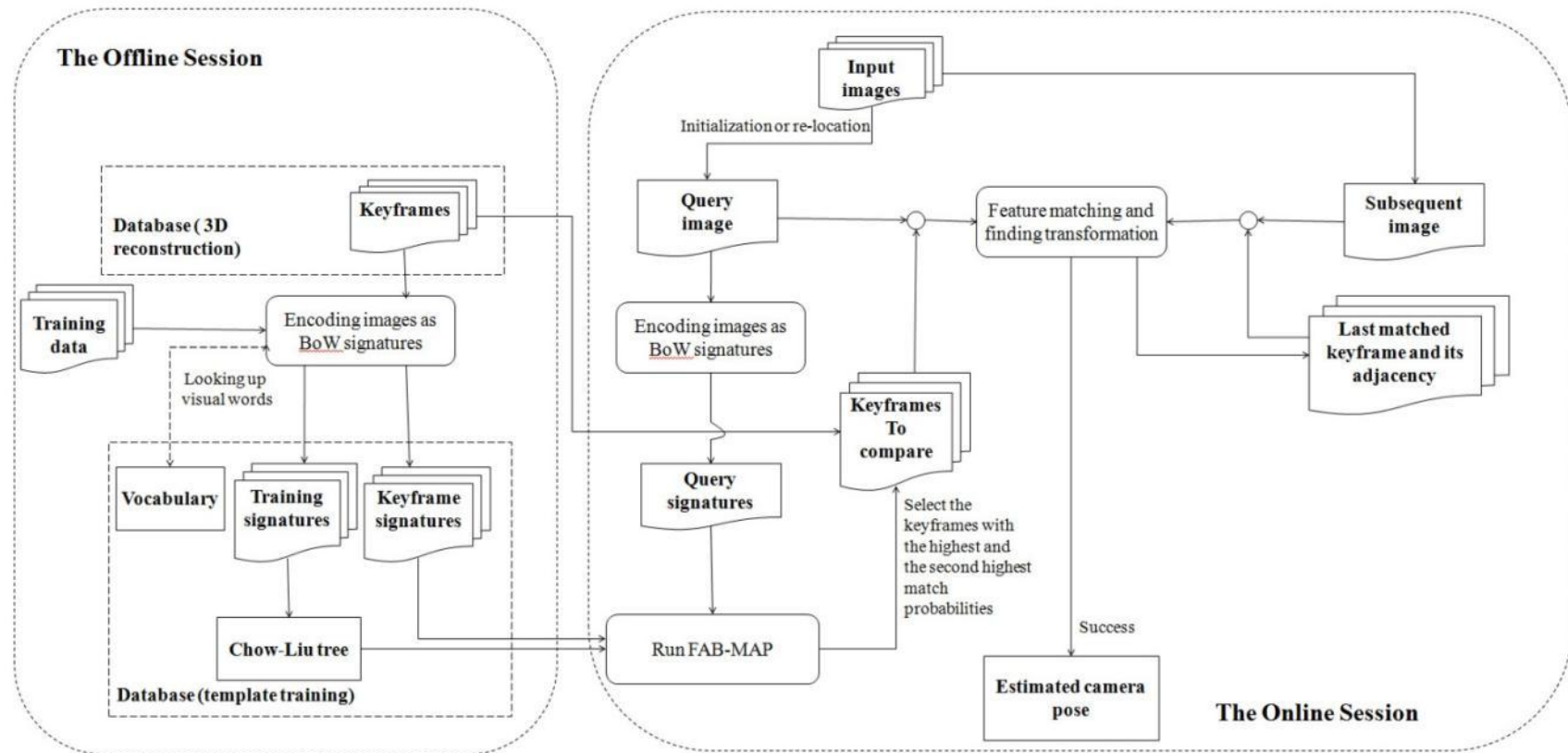


Figure 5-4: The whole process of offline template training and how the online session makes use of the data contained within the trained database for user camera pose estimation.

A visual vocabulary should be generated and a set of train images needs to be used to study the co-occurrence law of these words to some specific environments. For a complete online SLAM process which does not rely on any pre-known template in the environment, the selection of the training data should be very careful, since both keyframes and query input are acquired from an unknown environment online. The training data used must be general enough and cover many different cases to deal with a similar but unknown environment. However, the proposed case is focusing on a template-based AR system, where the 3D reconstruction provides the geometric information and the selected keyframes provide the visual information of the target environment. The source images used for 3D reconstruction can be directly used as the training data to make the studying stage focus more on this specific environment, and generally their visual features will be used to generate vocabulary. But in order to further improve the precision of the generated visual words, the unreliable features are not expected to be involved thus only those features which have been successfully matched for finding the relative transformation and were considered as inliers (*i.e.* the inlier features generated by RGBDSLAM v2 and the sparse cloud points generated by VisualSfM) are used.

The offline training encodes the training data and the selected keyframes as sequences of signatures with their feature descriptors and the visual vocabulary. The training signatures are used to build a Chow-Liu tree as described in Section 5.2.1. All of these data will be loaded onto the system when online stage starts. The system will perform feature detection and extraction on each input image with the same detector – descriptor used for 3D reconstruction. Due to the unknown location of the first input image, the

system will encode it as a BoW signature and apply FAB-MAP method described in Section 5.2.2 to look for the keyframe which has the highest probability to match the query. The keyframe with the second highest probability is also selected in the case that a valid transformation cannot be found between the query and the highest one. The precise feature matching is performed between the query and the selected candidates to establish 3D-to-2D coordinate correspondences. If there are enough remaining inliers after solving the PnP problem mentioned in Section 4.2.1.3, a valid transformation is considered to have been found. Since the output of RGBDSLAM v2 provides the 3D feature points with respect to each local camera reference frame, so the estimated transformation needs to be multiplied with the camera pose of the matched keyframe from the database to obtain the pose of user viewport with respect to the world reference frame. On the other hand, VisualSfM provides the mapping between the world 3D coordinates and the 2D keypoints, thus the camera pose of query can be directly obtained. Once the system successfully locates the user camera to the reference map, the last matched keyframe and its adjacency (*i.e.* the previous and the next keyframes in sequence) will be considered as the candidates to match for the following input images. If the subsequent image cannot find any valid transformation to these candidates, the system will repeat the same steps which have applied to the first query. If still no valid transformation can be found or FAB-MAP judges this query as a new place with the highest probability, the system will consider that the user has been moving out of the scope of the constructed reference map and it will move to processing the next input as a first unknown query until the system re-locates the user.

5.4. Results and evaluations

The results and evaluations presented here focus on two parts: a small study on the effects of using SiftGPU and ORB BoW with different parameter settings inside FAB-MAP for image retrieval; and applying those techniques in the proposed framework for template training, and then inspecting the performance by the accuracy of online tracking result.

The first experiment made use of a selection of Glover *et al.* (2010)'s St. Lucia suburb streets dataset. As shown in Figure 5-5, 25 frames were selected from one of the 640 x 480 video sequences collected at 08:45am as test data, and another 25 corresponding frames captured from the similar locations of the test data at 08:45 but 3 weeks later, were used as query data. The aim was to retrieve the correct corresponding image of each query image from the test data, and both frame-to-frame local feature matching and FAB-MAP with different BoW described above were tested for performance comparison on processing time and precision.



Figure 5-5: first row: 3 (out of 25) frames in test data; second row: the corresponding frames of the first row in query data which were taken at same place, same time but 3 weeks later.

As mentioned in Section 2.3.3, two regularly used evaluation metrics for information retrieval are precision and recall. However in this experiment, there was only one matched frame for each query in the test data, thus only the precision rates will be presented. The so-called frame-to-frame local feature matching method was just performing an exhaustive search by comparing query with each frame in the test data, and selected the one with the largest number of matches. The processing time and precision with SiftMatchGPU (which has been proven more efficient than the FLANN matcher in Section 4.3.1) for SiftGPU descriptors and the brute-force matcher for ORB descriptors are shown in Table 5-2.

Table 5-2: The processing time and retrieval precision of frame-to-frame local feature matching method by using SiftGPU and ORB features (bracketed are %RSD).

625 frame-to frame comparisons between query and test data (640 x 480)	SiftGPU (SiftMatchGPU)	ORB (Brute-force)
Total processing time (ms)	3085	3375
Avg. matching time of per comparison (ms)	5 ($\pm 18\%$)	5 ($\pm 11\%$)
Avg. time on per query (ms)	123	135
Retrieval precision rate	96%	44%

It can be found that both matchers cost about 5ms for each frame-to-frame feature matching. The retrieval precision with SiftMatchGPU was much higher than that with the brute-force matcher, which also implies the lower good match ratio of the brute-force matcher for ORB in Table 4-3. Consider the proposed online session, assume the standard sampling rate of a real-time input video is 30 fps, there are only 0.03 seconds processing time available for each input frame. If the frame-to-frame feature matching methods presented above are used to search the closet keyframes in the database, it will only allow 6 comparisons between the frames – which are not enough for larger reference maps with more keyframe. In fact, the reconstructed map of [freiburg1_desk] in Section 4.3.2 – a small scene around an office desk – even contained 21 keyframes. Besides, finding the keyframe with the closest appearance to the input is just a preparatory work for estimating the camera pose, which also requires more processing time. Therefore FAB-MAP with BoW was applied for an efficient retrieval. For SiftGPU descriptors, the OpenCV 2.4.9 implementation of Cummins & Newman (2011)'s vocabulary generation method with a fixed distance threshold was used (see Table 5-1). A smaller threshold value will generate a larger vocabulary, and three different threshold values were tested here for generating vocabularies with different

size. On the other hand, the hierarchical k-means clustering (see Figure 5-3) with different k was used for generating vocabularies for ORB descriptors. The vocabularies and Chow Liu tree were trained on 127 non-overlapping frames selected from other video sequences inside the St. Lucia dataset. Both test and query data were encoded into signatures for FAB-MAP to calculate the similarity. The processing time and precision with different feature descriptors and vocabularies are shown in Table 5-3.

Table 5-3: The processing time and retrieval precision of FAB-MAP with different feature descriptors and vocabularies.

25 query from 25 test data (640 x 480)		SiftGPU		
Threshold		0.7	0.75	0.8
Vocabulary size		5382	2183	891
Processing time (ms)	Encoding	300	180	10
	FAB-MAP	50	100	30
	Avg. time on per query	14	11	2
Retrieval precision rate		52%	<u>56%</u>	32%

25 query from 25 test data (640 x 480)		ORB		
k		8	7	6
Vocabulary size		4096	2401	1296
Processing time (ms)	Encoding	390	330	270
	FAB-MAP	20	20	20
	Avg. time on per query	16	14	12
Retrieval precision rate		56%	<u>68%</u>	52%

For both feature descriptors, the larger vocabulary took more time for encoding the signatures. However, even with the largest vocabularies (5382 for SiftGPU and 4096 for ORB) that have been tested, the averaged processing time on each query was much less

than that of the frame-to-frame methods in Table 5-2. The precision might be affected by vocabulary size and the train data. It can be found that the larger vocabulary did not guarantee a better precision. With regard to the effect of the train data, it is still unclear according to the present research and is considered as a future work.

In order to further inspect whether the performance of FAB-MAP with BoW can satisfy the proposed online AR tracking, the results generated in Section 4.3.2 are used in the next experiment. The test was mainly performed on indoor RGBD datasets [freiburg1_desk] which has achieved a relatively high accuracy on 3D reconstruction procedure. Sturm *et al.* (2012) provide another sequence [freiburg1_desk2] centred on the same desk scene but recorded at different time, and a further sequence [freiburg1_room], which starts with the desk scene but continues around the wall of the room and finally close the loop with the desk scene. The vocabulary is generated via the inlier features of [freiburg1_desk] which were detected and stored during the 3D reconstruction stage. The Chow Liu tree was trained on 20 non-overlapping frames selected from [freiburg1_room]. The keyframes of [freiburg1_desk] selected during the 3D reconstruction were used as test data, and the whole sequences of [freiburg1_desk] and [freiburg1_desk2] were used as query in online session. The purpose was trying to perform pose estimation of each query frame in near real-time through the reference map by matching the keyframes. In addition, to check if the vocabulary and the train data generated from a specific environment (e.g. office-like in this case) can be generically used for other similar environment, the sequence [freiburg3_long_office_household] – which focuses on a different office desk – was also tested. In this case, the vocabulary and train data of [freiburg1_desk] was used with

test data of the keyframes of [freiburg3_long_office_household] to query other images inside the sequence. The results with different feature descriptors are presented in Table 5-4 below, the parameters with the highest precision reflected in Table 5-3 were used for generating vocabularies.

Table 5-4: The test results of the proposed online tracking stage on two office-like environments, [freiburg1_desk] and [freiburg3_long_office_household], which have been restored in the offline session as presented in Section 4.3.2 (bracketed are %RSD).

Detector – descriptor	SiftGPU		
Vocabulary size	1385 (BoW distance threshold = 0.75)		
Test data	freiburg1_desk		freiburg3_long_office_household
No. keyframes	21		35
Query data	freiburg1_desk	freiburg1_desk2	freiburg3_long_office_household
No. frames	613	640	2585
No. pose estimations	605	403	<u>2581</u>
Avg. time on per frame (ms)	65(±20%)	<u>85 (±28%)</u>	74 (±22%)
RMSE (m)	<u>0.035</u>	<u>0.051</u>	0.076
Detector – descriptor	ORB		
Vocabulary size	2401 ($k = 7$)		

Test data	freiburg1_desk		freiburg3_long_office_household
No. keyframes	25		38
Query data	freiburg1_desk	freiburg1_desk2	freiburg3_long_office_household
No. frames	613	640	2585
No. pose estimations	<u>610</u>	<u>404</u>	2577
Avg. time per frame (s)	<u>58 (±36%)</u>	85 (±51%)	<u>62 (±26%)</u>
RMSE (m)	0.057	0.079	<u>0.073</u>

The trajectory comparison diagrams between the estimations and ground truth which have been used in Section 4.3.2 are also shown here, as shown in Figure 5-6, Figure 5-7 and Figure 5-8.

[freiburg1_desk]_xyz_trajectory

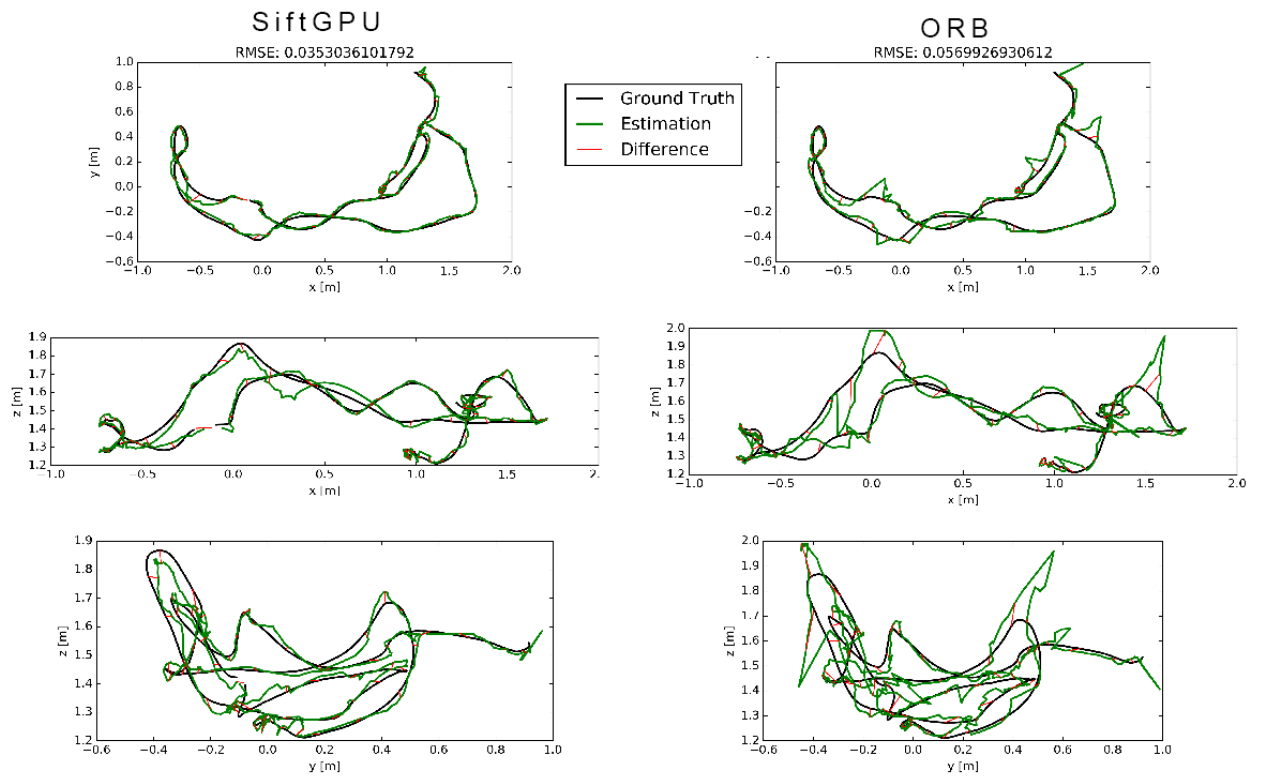


Figure 5-6: The trajectory comparison diagrams for the online tracking results of [freiburg1_desk].

[freiburg1_desk2]_xyz_trajectory

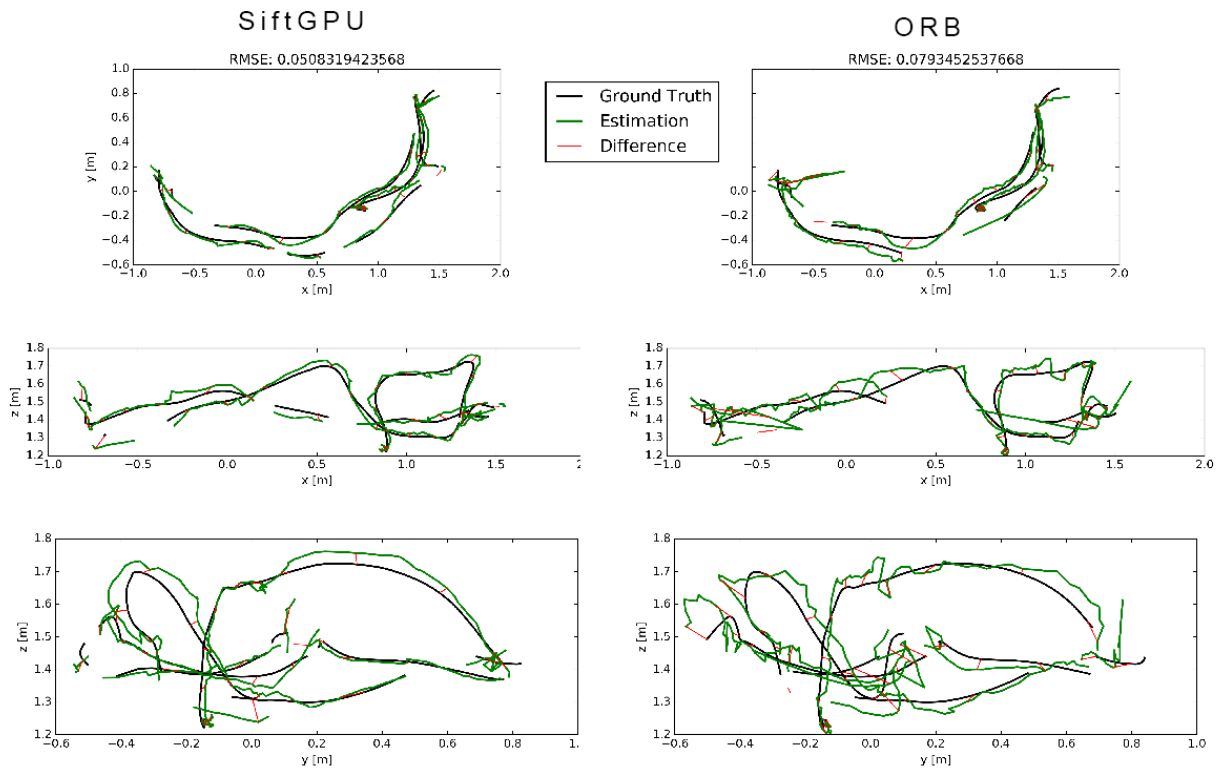


Figure 5-7: The trajectory comparison diagrams for the online tracking results of [freiburg1_desk2].

[freiburg3_long_office_household]_xyz_trajectory

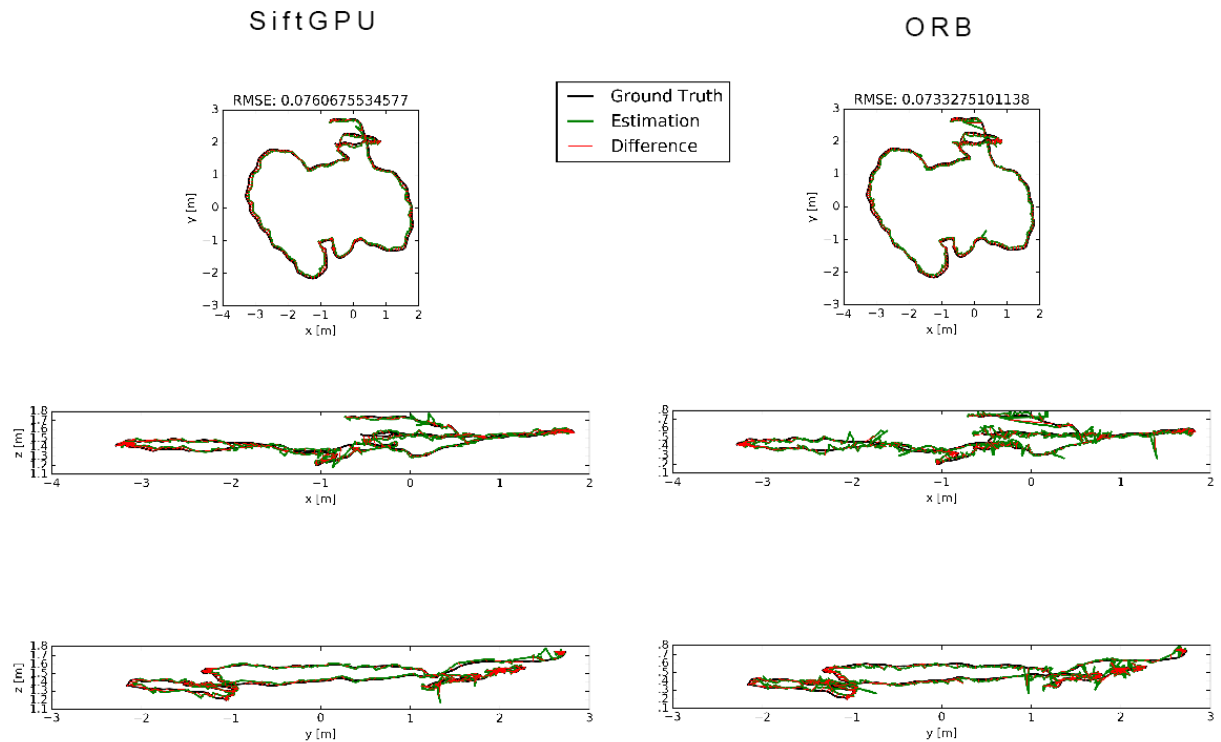


Figure 5-8: The trajectory comparison diagrams for the online tracking results of [freiburg3_long_office_household].

In the first case, the original [freiburg1_desk] dataset, which had been used for creating the reference template of the scene, was used for performing the AR tracking test. When SiftGPU-based approaches were used, the estimated trajectory is relatively accurate and stable, although a few frames were failed to be retrieved. Statistically, when ORB-based methods were used, the number of pose estimations and their RMSE value are similar to those with SiftGPU. However, the trajectory comparison diagrams show more slight displacements between the estimation and ground truth. In the second case, the [freiburg1_desk2] dataset was captured from the same environment of the

[freiburg1_desk] but with somewhat different shooting angles and settings. The majority of camera poses of the frames can be recovered via the trained database. No matter which method (with SiftGPU or with ORB) was used, the trajectory comparison diagrams show many inconsistencies between the estimation and ground truth. The failed frames can be divided into the following conditions: 1) the images are blurry (primarily caused by rapid camera motions); 2) the settings of the workspace were changed; 3) the camera was moving out of the scope of the reference map. The results of [freiburg3_long_office_household] are more or less same to the [freiburg1_desk]. The proposed rapid image retrieval method worked well in this case, proving that the vocabulary and training data of a certain environment can be applied to another environment with similar appearance. Therefore from the perspective of the application developers, they should not be required to train visual vocabulary and training data for use of FAB-MAP themselves. The vocabulary and training data of different environment should be supported within the development framework.

The best processing rates were 15 fps with SiftGPU and 17 fps with ORB on retrieving [freiburg1_desk], where was almost no “lost” condition that demands a reset during the tracking. Generally, the proposed rapid image retrieval method with SiftGPU can retrieve the query image from the trained database effectively, but the tracking results with ORB were not always good. The pose estimation results in the online session are not as accurate as those in the offline session, since there are no optimisation methods applied by considering the limitation of the processing time in a real-time application.

5.5. Conclusion

In this chapter, a pure vision-based online AR tracking session, based itself on the reconstruction results generated in Chapter 4, is presented. The present research mainly focuses on rapidly retrieving the online input images of an AR application from the trained database to meet the real-time requirement. The BoW (Sivic & Zisserman, 2003) methods are used for abstracting an image to a single signature vector with a visual vocabulary, which can be generated from either SIFT-like feature descriptors or ORB descriptors. Then, the vocabulary and a pre-trained dataset of a certain environment are used with the FAB-MAP (Cummins & Newman, 2008) approach to score the matching likelihood between the query and a set of keyframes contained within the database. It has been shown that this method with both SiftGPU and ORB approaches can retrieve the query image from the trained database effectively in near real-time, but the accuracy is still less than satisfactory. The optimisation methods applied in the offline session are not performed in this online session due to the limitation of the processing time, causing unstable and less accurate pose estimation results. Even so, it can be learned from the test results that blurred input image and the relatively small change to the original scene will result in failure of the present proposed tracking method. Similar to the reconstruction results in Chapter 4, what accuracy for AR tracking is acceptable is unclear, which is also considered as a future work to be solved with involving real users.

Chapter 6 Conceptual User Interface and Use Case

Scenario

This chapter will firstly describe the conceptual user interface for the proposed markerless AR development framework (Section 6.1). The basic process of how application developers would interact with the framework is described. Since the proposed UIs are still at a conceptual stage of development, the usability evaluations of the AR system mentioned in Section 2.1.3 have not been carried out, but are considered as suitable for future study. However, an AR project being planned at the time of writing for an AR exhibit at the National Marine Aquarium is discussed. It will give an overview of applying the proposed work as part of a real use case scenario and to gain feedback from real users. An analysis of how the proposed system will fit to the use case followed by an initial plan of user-centred usability evaluation is given in Section 6.2.

6.1. Conceptual user interface

The design of the user interfaces within the proposed framework focuses on two kinds of end users: the developer who will interact with the application development framework and the final user who will interact with the developed application. The application developer will execute the offline session of 3D reconstruction and template training, described in the corresponding sections of Chapters 4 and 5, but they are not required to have any understanding of how internal process worked. Essentially, this is a *black-box* style system in which the developer only needs to provide source input data

(*i.e.* the RGB or RGBD dataset of a target environment) and to set up the augmentation information with the system output (*i.e.* the 3D reconstructed reference map of the target environment). The system will automatically generate the database for subsequent application use. In terms of the AR application user, the present proposal has not considered what kind of human-interaction experience the developer can provide to their user. This will be a future aim for any ongoing research. The effort described in this chapter mainly focuses on the technical implementation of an AR development framework with a GUI tool for setting up virtual information (particularly the 3D model) with respect to the reconstructed environment (Section 6.1.1), and the augmentation registration and display processes of AR browsers (Section 6.1.2).

6.1.1. Development framework

The development framework described here, and shown in Figure 6-1, is targeted at the higher-level developer who desires to implement a simple markerless AR application for a specific environment. The developer needs to provide the visual information (images or videos) to the system for offline 3D reconstruction and template training. The RGBD sequential dataset will be processed by RGBDSLAM v2 and the RGB dataset will be processed by VisualSfM. The internal techniques have been expressed in Chapter 4. Since RGBDSLAM v2 and VisualSfM are independent software packages, a *shell script* file, in essence a text file with a sequence of command which can be interpreted and executed in UNIX-like system, is used as a universal launcher to start the desired program and pass the corresponding arguments (e.g. the source file of input dataset and the file path of output). Consider that the process of vocabulary training expressed in Section 5.1 requires a dozen of general but representative image sets of

various types of environments, such as an indoor office or outdoor urban environment as input, but the input dataset provided by the developer is focused on a specific place and it is not reasonable to let them collect more images from other places for generating vocabulary. Therefore, the trained visual vocabulary is trained by the lower-level developer and provided together with this proposed framework together. The complete 3D reconstruction and template training session is executed automatically and does not require any developer intervention.

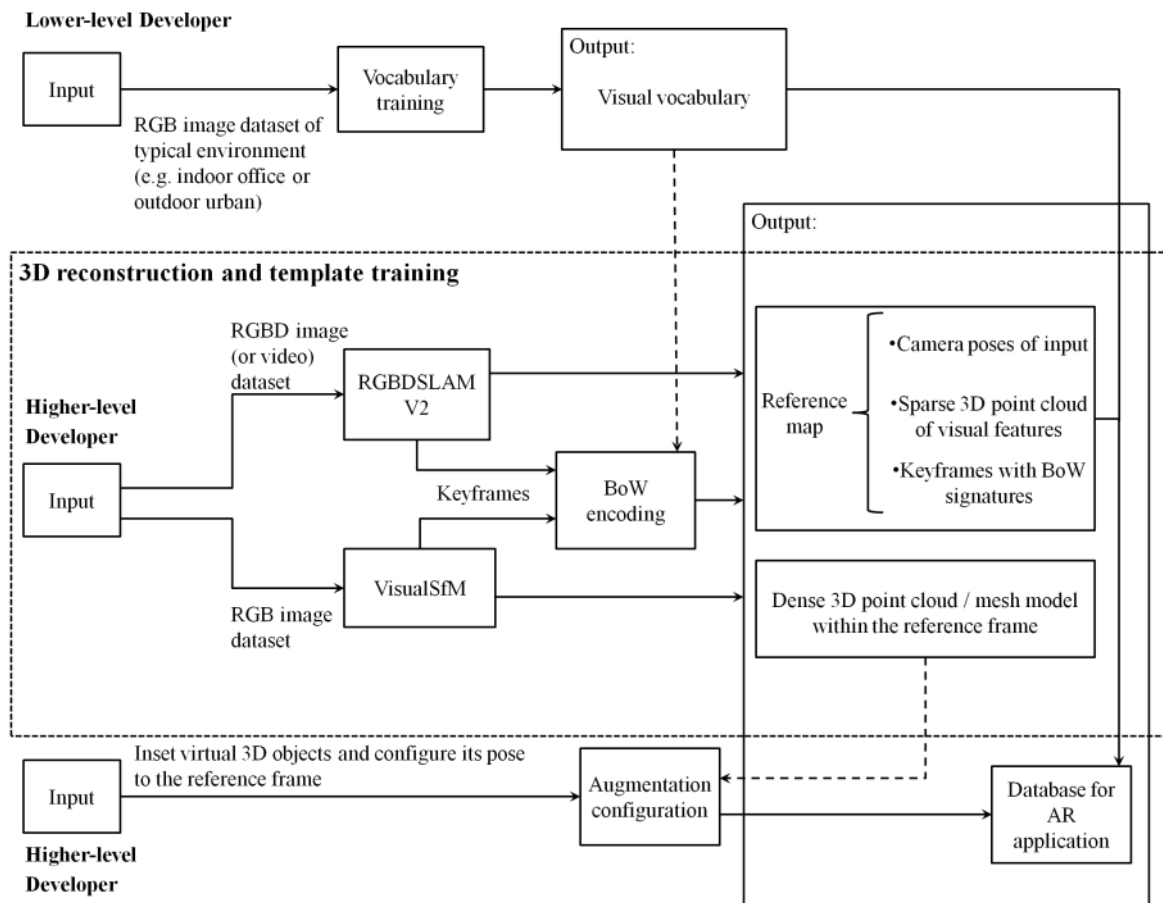


Figure 6-1: The input-process-output diagram of a 3D reconstruction-based AR development framework, equates to the offline session mentioned previously.

Both RGBDSLAM v2 and VisualSfM support the generation of dense colour point cloud models from images, although this is not altogether necessary for the AR tracking

aim which only requires the spatial positions of feature points in the world as reference. A successful 3D reconstruction will result in a dense point cloud model very similar to the real scene, which is more intuitive and meaningful than the sparse model from the perspective of a human viewer (for an example see Figure 4-11). Thus the dense model is used as reference in an augmentation configuration process for the developer to insert virtual information (3D objects in particular). This process allows the developer to set up the pose and scale of the virtual models with respect to the world reference frame in which the dense model is located. This can be undertaken with a simple GUI tool contained in the proposed development framework. The GUI tool is mainly implemented with the *QtGui* module of the *Qt* software framework (Blanchette & Summerfield, 2006) which provides a set of GUI elements such as windows, buttons, and sliders to create GUI-based application, and OpenGL which is used for drawing the 3D display window. As shown in Figure 6-2, this GUI tool includes a menu bar, a display window and a 3D object control panel. The menu item 'I/O' allows developer to:

- select a reconstructed dense point cloud from file (with the format of PCD or PLY, which can be handled through the Point Cloud Library (see Appendix B) as a visual reference;
- insert a custom 3D object (typically a mesh model within a OBJ file supported by various 3D graphics application, such as *3ds Max*, *Blender* and *SketchUp*) to the same coordinate system of the dense model as augmentation;
- save the configuration result (the pose and the scale of inserted 3D model after configuring) to a text file.

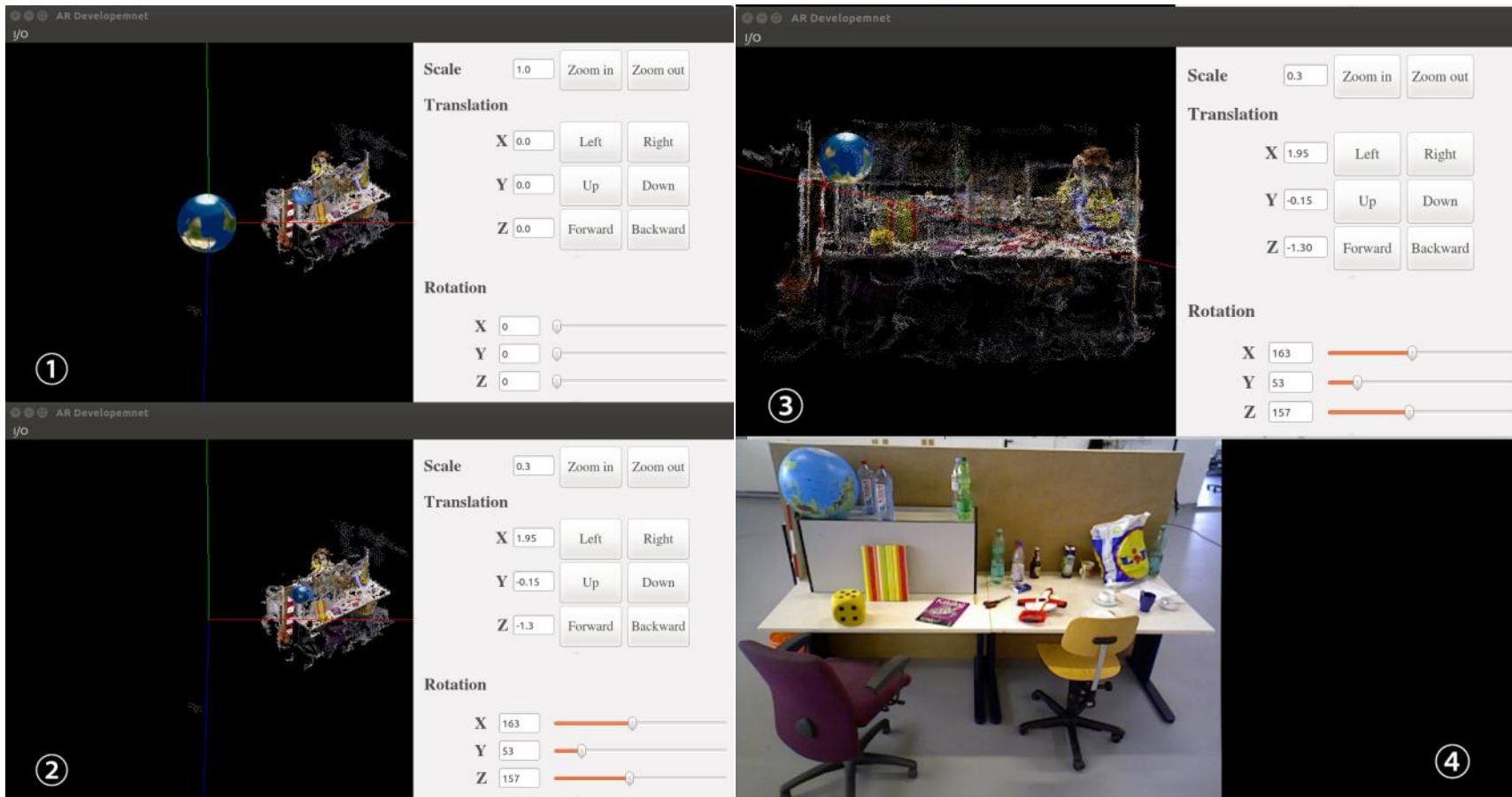


Figure 6-2: The simple GUI tool contained within the AR development framework.

The display window allows the developer to view a virtual space of the environment to augment, which includes 3D axes (red for 'X', green for 'Y', blue for 'Z') of the reference frame, the loaded dense reconstruction and the inserted object. Using a mouse, the developer can scale, rotate, or move the viewport of the display window to observe this virtual space and any model from different perspectives. The dense reconstruction with respect to the reference frame (*i.e.* 3D axes) is fixed, which is equivalent to the relation between the sparse reconstruction and the world reference frame. In order to display augmentation during the online session, the relative position and orientation of the virtual object with respect to the world reference frame should be defined. Thus the 3D object control panel allows the developer to adjust the pose and scale of the inserted object in the reference frame by using the dense reconstruction as visual reference. For example, as shown in Figure 6-2, the screenshot ① presents the dense point cloud of an office desk scene (generated from the VisualSfM reconstruction result of the [freiburg3_long_office_household] dataset). The tricolour axes represent the origin of the reference frame. The inserted 3D model 'earth' was initially located at the origin. In the screenshot ②, the position, orientation and scale of the 'earth' with respect to the reference frame has been manually adjusted with the 3D object control panel, making the 'earth' model overlap the 'real earth model' in the dense reconstruction. It can be seen from the viewport of the display window shown in the screenshot ③, which was changed to the same perspective with one of the RGB image of the source dataset shown in the ④, the inserted 'earth' model and the real earth model locate in the same location with same orientation and scale. These configuration parameters were then stored to the database and would be used to set up the augmentation with respect to the reference map when the AR application is running. The generating process of the AR

view is described in next section.

6.1.2. AR registration and display

The user camera pose is calculated at the frequency of the online processing rate of each input frame, determined by the methods to be used, which have been described in Chapters 4 and 5. They are used to register AR objects onto the input image, and these synthetic scenes will finally be displayed in front of the application user. For the displays of the desired virtual information, the 3D position and orientation with respect to the reference map should be given so that it can be hypothetically put into the real world, as the function of the GUI tool described in Section 6.1. The registration process utilising the camera parameters include the camera pose, which implies the relative position between the virtual things and the camera, and the intrinsic parameters which can be used to project the virtual items with a correct perspective onto the image plane and to render and display the final augmented images. The practice methods used in applications for AR registration and display are described in the following paragraphs.

In order to register augmentation to the original image, the intrinsic camera parameters, the estimated camera poses, and the pose of virtual objects with respect to the world reference frame are required to perform the 3D-to-2D projection which has been described in Section 3.2.3 (see expression (3.5)). The intrinsic camera parameters of the camera device are pre-defined in the system; the camera pose of each query image is solved through the approaches introduced in previous chapters, and the framework should allow the developer to define the inserted virtual information and its pose with

respect to the training map, which will be described in Section 6.2. In this section, a simplified AR browser implemented with OpenCV library is demonstrated. This simple browser will deliver an intuitive appreciation of the camera pose estimation results and how they can be used for AR registration.



Figure 6-3: The OpenCV augmented views of part of the RGB frames contained within the dataset [freiburg3_long_office_household].

As an example, shown in Figure 6-3, the colour sequence within [freiburg3_long_office_household] and its pose estimation generated by RGBDSLAM v2 (see Section 4.3) are used here to simulate the AR registration. Assume the inserted information is a bounding box (wireframe) aligned to the border of the outside face of the white box. The pose and scale of this planar bounding box with respect to the world reference frame can be identified first by using the augmentation configuration GUI tool described in Section 6.1. It can then be transformed into the different camera reference frames from the world through the estimated camera poses, and finally be projected to the corresponding 2D image planes. To hold the coordinate vectors and transformation matrices, `cv::Mat` is used. The coordinate transformation presented in expression (3.5) is

implemented by matrix multiplication supported by `cv::Mat`. It can be seen in Figure 6-3 that the bounding box was projected and drawn on the proper position on the different images with correct perspectives.

It is feasible for OpenCV to draw the contours or wireframes of simple polygons on the background images by linking up the projected points. However, the limitation is that OpenCV does not really support the functions of rendering 3D graphical models with complex shape, structure and texture, which are required by some visual AR applications where there is an expectation that 3D models can be merged into the real environment as seamlessly as possible. For this reason, an alternative OpenGL AR browser is also implemented.

OpenGL provides all the basic functions to create the required rendering mentioned above. It has also been used in the proposed GUI tool for the developer to set up the pose and scale of the virtual object with respect to the reference frame. In fact, the view display in the GUI tool's display window, as shown in Figure 6-2, and used here for creating an AR browser is quite the same, except that the GUI tool's display window allows developers to change the perspective of observation themselves, where the field of view of AR browser depends on the current user camera pose estimation. This process can be described simply as follows. The AR application firstly acquires a view of the environment from the user camera. The system compares this view with the reference map within the database and decides if the current user camera pose with respect to the reference map can be identified. If it can be, the system will calculate if the augmentation object defined by the application developer can be observed from the

current camera's view by performing 2D projection. If it can be, the projection will be merged to the original input view as the augmented view which would then be displayed to user. The specific implementation process of this OpenGL AR browser is given in Appendix B. The OpenGL AR views based on the AR configuration example shown in Figure 6-2 are shown in Figure 6-4 below. It can be seen, the real earth model in the original scene is overlapped with a virtual earth model.

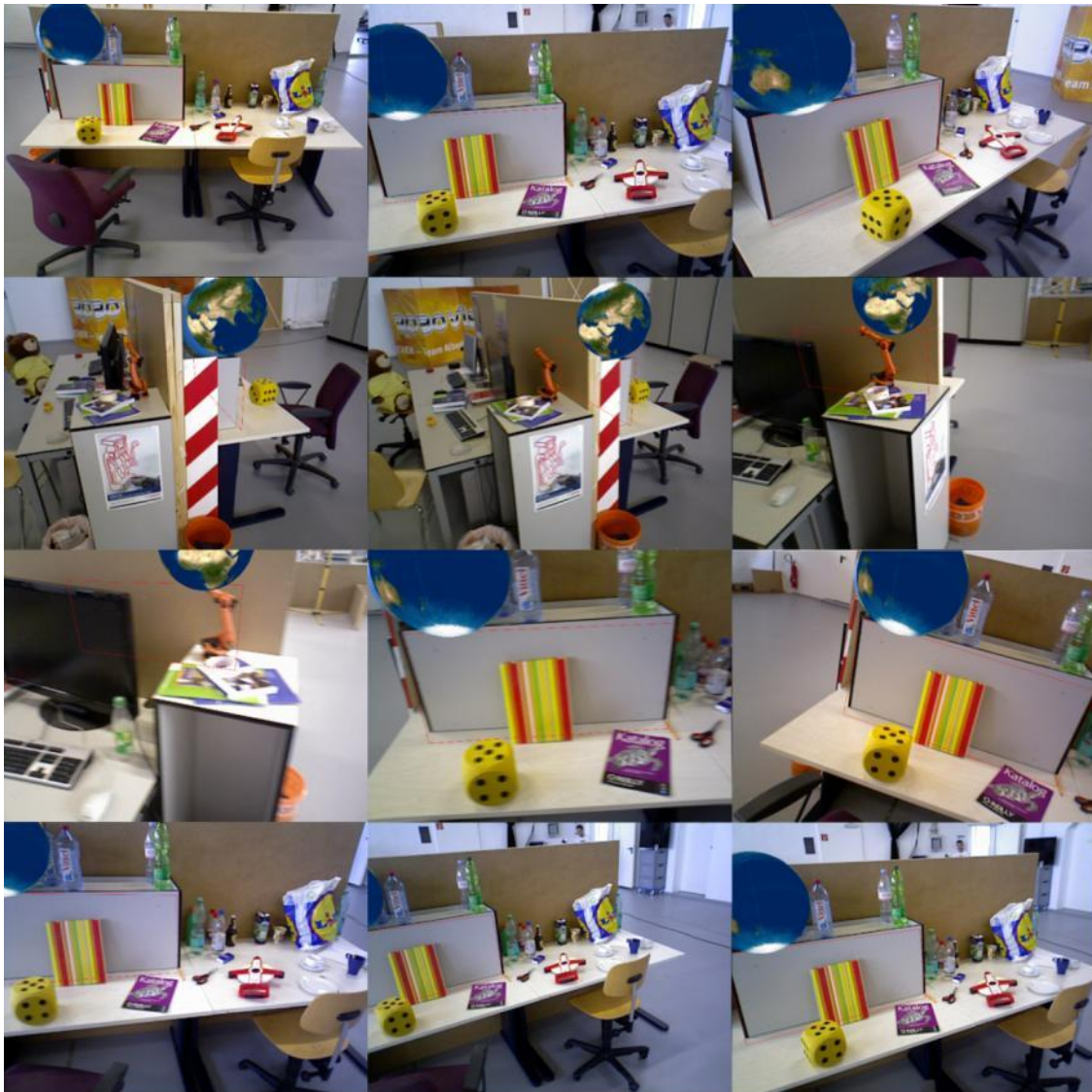


Figure 6-4: The OpenGL augmented views of part of the RGB frames contained within the dataset [freiburg3_long_office_household].

6.2. Use case scenario

As mentioned in Section 2.1.3, a system can be evaluated from both a technical perspective and an end user perspective. The former can refer to the stability of the system performance and the latter can be assessed from the capability of the system to meet the end user's goal(s). Both perspectives are also considered to be the most important factors for the AR application developers in the evaluation of a development tool (see questionnaire results reported in Section 3.1). The technologies used in this proposed work have been tested with their corresponding evaluation metrics, as presented in Chapter 4 and Chapter 5. However, for a proper usability evaluation of the system or the user interface of the system which connects human and computer for interaction, real users must, of course, be involved. Therefore, an AR project in collaboration with the National Marine Aquarium in Plymouth is, at the time of writing, being planned, and is presented here as a potential use case. The background and requirement analysis of this use case, and how it would be used for the future usability evaluation of the proposed framework is discussed in the following sections.

6.2.1. Background and requirements



Figure 6-5: The National Marine Aquarium.

The National Marine Aquarium is the largest public aquarium in the UK welcoming 300,000 visitors per year. Recently (2017), they have been cooperating with the **Human Interface Technologies (HIT)** Team of the University of Birmingham to address how to apply AR technology on mobile platforms to improve the experience of their visitors. The basic idea involves the superimposition of a number of animated virtual models of a selection of cetacean marine mammals over the main public function zone in front of the Eddystone Reef Tank (Figure 6-6; see also Footnote 7). As shown in Figure 6-7, the target area is of a two-tier architectural style. The tank and function zone are located at the lower floor with an upstairs passage around them. The cetacean models, as shown in Figure 6-8, are hung from the ceiling, over the public function zone, and can be observed by the visitors from both downstairs and upstairs.



Figure 6-6: The Eddystone Reef Tank and its function zone.

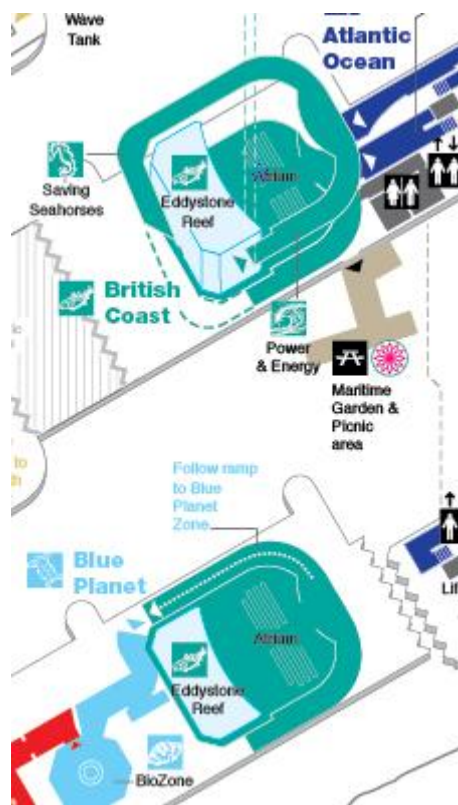


Figure 6-7: Two-tier architecture of the Eddystone Reef Tank area.



Figure 6-8: The upstairs passage and physical marine animal models hung from the ceiling.

It is intended to display the AR versions of the mammals in place of these physical models, and to make them accessible to visitors as applications on their own mobile devices (smartphones or tablets). Thus, as previously presented in Table 2-1, this case relates to a specific location-based exhibit and requires accurate registration for the display of AR models correctly onto and within features of the environment. Therefore the proposed framework presented in this thesis offers a potential solution for use in this project. The HIT Team application developers can make use of either RGB or RGBD cameras to collect the visual information around the physical models hung over the function zone. It would be useful to capture the data from both upstairs and downstairs, enabling the created reference map to be used for user tracking on both floors. Then the developers can deliver these data to the development framework described in Section 6.1 above. One of the 3D reconstruction methods applied in Chapter 4 will create the

reference template and the dense model of the target environment based on the input data. The keyframes generated during the reconstruction will be trained and encoded into BoW signatures and be stored in an application database for efficient online AR tracking. After generating the database, the GUI tool shown in Figure 6-2 will allow the developers to insert the 3D marine animal models with respect to the reconstructed model of the target environment. Then an AR application will recognise the NMA Eddystone Reef area based on the appearance of the physical models hung from the ceiling and display the AR marine animals and animations superimposed within the area and the results can be published to the application final users.

However, more specific requirements of the application – in other words, the educational aims it expects to achieve and how the application should interact with the visitors to support such an educational aim – are still (at this stage) unclear. This is a task that needs to be solved by the application designers and developers, which is very necessary to the proposed framework to create further functions and UI features to support their development. For this reason, the development process of this project will be tracked in the near-term future for getting the direct opinions and feedback from the developers in time.

6.2.2. Usability evaluation plans

The usability evaluations of the proposed framework first focus on the higher-level developers – in this case, the developers of the desired aquarium application from the HIT team. In order to support their development, it needs to communicate with them

frequently to determine their design and goal. The proposed framework may be modified and improved several times during their development to meet their requirements. Another potential solution of this project is to use Vuforia's multiple-marker SDK, which is now considered as an alternative selection of application development by the HIT developers. Thus it will be useful to ask them to compare the usability of these two development tools. The objective evaluation metrics to evaluate these two methods on processing rate, performance and capacity of handling the target environment will also be designed as future study aims.

The completed application then will be tested and evaluated by the visitors of the NMA with a questionnaire relating to several system performance criteria, such as the quality and stability of the AR view, the practicality of the user interface and the interaction experience.

6.3. Conclusion

In this chapter, the techniques and implementations of the conceptual user interfaces are presented. The entire offline database training session described in the previous chapters is integrated into a black-box style system and the application developer can interact with it by using a general user interface in the form of shell script. A simple GUI tool based on QtGUI and OpenGL is provided for developers to insert virtual 3D objects and to set up their pose and scale with respect to a previous reconstructed dense model. Further functionality is expected to be provided within this framework in the future to support developers in customising the GUI and user interaction of their AR applications.

The augmentation configuration will then be stored in the database and used in an AR application for rendering and displaying the virtual objects in their proper locations. OpenGL is mainly used for AR registration and display, with the camera pose estimations generated in the online session. It produces a reasonable visual result, but more complex situations of the scene should be considered in future research, such as the effect of occlusion, lighting or shadow.

In order to evaluate the usability of the proposed framework, real use case scenarios with real users are expected to be involved in the future. Currently an AR project based on cooperation between the National Marine Aquarium and the Human Interface Technologies Team of the University of Birmingham is being planned as a use case in which to apply the proposed framework. Technically, the framework is quite capable of handling the case, but future work needs to focus on the developers' requirements and feedback during the development. Meanwhile, an alternative AR SDK will be applied by the developers to solve the same problem. The evaluation metrics will be designed for evaluation and these two methods will be compared both subjectively and objectively. The final application user feedback will also be used to evaluate the usability of the proposed framework and subsequent development of the application.

Chapter 7 Conclusions and Future Work

The present research set out to propose and demonstrate a conceptual markerless AR framework system by taking account both of the needs of the *application developers* who wish to apply AR techniques for general development, and those of the *traditional users* who experience and interact with the end results of the AR application. The proposed framework system is divided into two process sessions: an offline session for reference template training, based on the 3D reconstruction techniques, and an online session for AR registration and display by tracking the user camera through the trained database. The techniques and implementations of this system are mainly discussed in Chapters 4, 5 and 6. The performance of the system and its component processes were tested and evaluated with several public datasets of different environments. The overall conclusions of the research are presented in Section 7.1 and suggestions for further research are suggested in Section 7.2.

7.1. Conclusions

The recent use of marker-less, natural feature-based tracking methods in AR research projects and in the development of application SDKs generally fall into two cases. In the first case, the lower-level developers (*i.e.* researchers) tend to apply different methods to design and implement AR applications for solving specific issues, which cannot be directly used by the higher-level (*i.e.* non-specialist) developers for general purposes. In the second case, some of the recent AR SDKs started to support markerless tracking for higher-level development, but, to the author's knowledge, most of them only allow the user to create the reference map of a small workspace and to set up the

augmentation in an online session, which is not convenient for the developer, who may wish to implement some more complicated functions (such as designing a GUI) for their applications.

The present research attempted to solve the above problems by proposing a markerless AR framework system for both the higher-level developers and the final users of their applications. In order to provide a robust and reliable reference template for the online application system to track the user's viewport and to perform an accurate AR registration, two effective offline 3D reconstruction applications – RGBD-SLAM v2 (Endres *et al.*, 2014) and VisualSfM (Wu, 2011; 2013) – were integrated into the proposed system. The former accepts the continuous ordered RGBD dataset which can be obtained from a RGBD camera, such as Microsoft's Kinect, but only works for indoor environment. The latter can handle the unsorted, discrete RGB datasets which can be captured by an ordinary camera and can be applied in both indoor and outdoor environments. Both two types of input sensor are portable and reasonably low-cost, allowing the developer to choose which one they prefer. Since there is no limitation of processing time in an offline session, both applications feature relatively expensive optimisation methods to refine the reconstruction results. The performance of these two applications with two visual features – SiftGPU (Wu, 2007) and ORB (Rublee *et al.*, 2011) – was tested on several RGBD and RGB public datasets (Strecha *et al.*, 2008; Sturm *et al.*, 2012). If the colour frames contained within the source dataset were of a good quality (without too much blur), both RGBD-SLAM v2 and VisualSfM with SiftGPU were found to generate a robust map of the target scene. However, with ORB, the results of RGBD-SLAM v2 were similar to those found with SiftGPU, whilst the

results of VisualSfM all failed, except in the case of [fountain-P11], which only included 11 frames captured along a short path. The test results also suggested that the datasets with a lot of blurred images were not ideal for use in creating the reference map. This is because the necessary coordinates and feature descriptions of these images are not clear and reliable, which may cause mismatch errors and wrong reconstructions. From the perspective of the developers, when they are collecting the source data from the target scene, they should avoid rapid camera motions.

Following the reconstruction process, whether the results are generated by RGBD-SLAM v2 or VisualSfM, they are unified into the same format and stored into database. By considering the real-time requirement of the AR application, if the user is viewing a scene restored previously and stored in the database, the system should rapidly locate the user's viewport with respect to the reference map stored within the database. Since the 3D reconstruction methods used in the present proposal can handle a large space (with many source images used), a rapid image retrieval strategy is applied to find the keyframe with the highest likelihood of matching the query image from the database, exploiting the BoW (Sivic & Zisserman, 2003) and FAB-MAP (Cummins & Newman, 2008) methods. Two types of visual vocabulary for SIFT-like and ORB feature descriptors were tested. Both SiftGPU and ORB approaches can retrieve the query image from the trained database effectively in near real-time, but the stability and accuracy of the online estimations were not very ideal although the similar camera pose estimation method of the offline session was used in the online session. This is because the expensive optimisation methods cannot be applied here for the requirement of real-time performance. Currently, if the application users can maintain a stable camera view,

with a relatively slow movement around the restored scene, the system can still generate a reasonably AR view, although drift and vibration was still found to occur on occasions. Even when they moved the camera out of the reference map and returned shortly thereafter, the system could relocate them. However, the performance of the online session has only been tested on a few datasets with similar environmental qualities. The robustness of the online pose estimation process needs to be improved and tested on different environments and situations.

The user interfaces proposed as part of the present research mainly focused on the development framework, which provides a general user interface in the form of a shell script for the application developers to apply their chosen methods with their input dataset. The dense point cloud model generated in the 3D reconstruction process is presented in a simple GUI tool, used as the visual reference for the developers to set up the augmentation (*i.e.* 3D objects). This GUI tool allows the developer to insert virtual 3D models, and to configure their pose and scale with respect to the restored environment. The augmentation configuration is saved into the trained database and used in the online application for AR registration. For the application users, the present proposal provides an OpenGL AR browser to view the augmentation via the developers' configuration and online pose estimations. The 3D models can be rendered and displayed correctly with all methods described above, but the effects of occlusion, lighting and shadow have not been dealt with as yet. Moreover, more complex functions and interaction are expected to be supported as part of the future plans for the development framework, enabling the developers to design and customise their application for different aims.

Currently, the biggest problem facing both 3D reconstruction-based template training and online user tracking is that the accuracy for acceptable visual performance of AR cannot be measured without real users. Similarly, the usability of user interfaces also needs to be evaluated with end users. Therefore, applying the proposed work in real use cases and involving end users for further system evaluation are considered as the most important goals to achieve in the future.

7.2. Future research and development

Further improvements on the achievements of the present research mainly focus on three directions of effort: 1) improve the performance of the online vision-based user-tracking; 2) apply the proposed framework in a real use case described in Section 6.2 and refine the functions of the development framework based on the developers' feedback; and 3) design the visual performance and usability evaluation parameters for the proposed framework based on real users.

Without the optimisation methods used in the offline session, the performance of the online pose estimation is currently neither suitably robust nor stable. An inexpensive approach needs to be developed to refine the online pose estimation without increasing the processing time. In addition, most of the CV techniques used in the present proposal were designed for the SIFT-like feature. It will be useful to explore their applications for the ORB feature, enabling developers to benefit from its lower-computational-cost and totally open-sourced nature.

The current proposal has presented a conceptual design of an offline development framework. Although at the current stage of development, it only allows the developers to input the source data for creating the reference map of a target environment and configure a simple augmentation to display, it can, potentially, involve more functions (e.g. customising the GUI for their application, designing the forms of interaction for their application users, or adding animated AR content) for meeting the developers' demands, which will be further exploited by tracking and communicating on the development process of the developers and designers of the real use case being planned in Section 6.2. The real users (both higher-level developers and application end users) need to be involved for determining the evaluation metrics on AR visual performance and system usability of the proposed framework.

PUBLICATION

- Jingjing Xiao, Rustom Stolkin, Yuqing Gao, Ales Leonardis. “Robust fusion of colour and depth data for RGB-D target tracking using adaptive range-invariant depth models and spatio-temporal consistency constraints”. IEEE TIP, submitted.

REFERENCES

- [1] Abbas, S.M. and Muhammad, A. (2012) **Outdoor rgb-d slam performance in slow mine detection.** *In Proceedings of Robotics*; Proceedings of ROBOTIK 2012; 7th German Conference on. 1-6. VDE.
- [2] Alahi, A., Ortiz, R. and Vandergheynst, P. (2012) **Freak: Fast retina keypoint.** *In Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2012 IEEE Conference on. 510-517. IEEE.
- [3] Alshamari, M. and Mayhew, P. (2008) **Task design: Its impact on usability testing.** *In Proceedings of Internet and Web Applications and Services*, 2008. ICIW'08. Third International Conference on. 583-589. IEEE.
- [4] Amin, D. and Govilkar, S. (2015) Comparative study of augmented reality SDKs. *Int. J. Comput. Sci. Appl.(IJCSA)*, 5: (1).
- [5] Angeli, A., Filliat, D., Doncieux, S., et al. (2008) Fast and incremental method for loop-closure detection using bags of visual words. *IEEE transactions on robotics*, 24: (5): 1027-1037.
- [6] Arthur, D. and Vassilvitskii, S. (2007) **k-means++: The advantages of careful seeding.** *In Proceedings of Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. 1027-1035. Society for Industrial and Applied Mathematics.
- [7] Atkinson, K.E. (2008) **An introduction to numerical analysis.** John Wiley & Sons.
- [8] Azuma, R. (1997) A Survey of Augmented Reality. *Presence: Teleoperators and Virtual Environments*, 6: (4): 355-385.
- [9] Baeza-Yates, R. and Ribeiro-Neto, B. (1999) **Modern information retrieval.** ACM press New York.
- [10] Baggio, D.L. (2012) **Mastering OpenCV with Practical Computer Vision Projects.** Packt Publishing Ltd.
- [11] Bahl, P. and Padmanabhan, V.N. (2000) **RADAR: An in-building RF-based user location and tracking system.** *In Proceedings of INFOCOM 2000*. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. 775-784. Ieee.2
- [12] Bay, H., Ess, A., Tuytelaars, T., et al. (2008) SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110: (3): 346-359.
- [13] Besl, P.J. and McKay, N.D. (1992) **Method for registration of 3-D shapes.** *In Proceedings of Robotics-DL tentative*. 586-606. International Society for Optics and Photonics.
- [14] Bichlmeier, C., Ockert, B., Kutter, O., et al. (2008) **The visible korean human phantom: Realistic test & development environments for medical augmented reality.** *In Proceedings of International Workshop on Augmented environments for Medical Imaging including Augmented Reality in Computer-aided Surgery (AMI-ARCS 2008)*, New York, USA.
- [15] Blanchette, J. and Summerfield, M. (2006) "A Brief History of Qt". *C++ GUI Programming with Qt 4*. Prentice-Hall xv-xvii.
- [16] Blum, J.R., Greencorn, D.G. and Cooperstock, J.R. (2012) **Smartphone sensor reliability for augmented reality applications.** *In Proceedings of International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*. 127-138. Springer.
- [17] Bolles, R.C., Baker, H.H. and Marimont, D.H. (1987) Epipolar-plane image analysis: An approach to determining structure from motion. *International journal of computer vision*, 1: (1): 7-55.
- [18] Boonsriwai, S. and Apavatjirut, A. (2013) **Indoor WIFI localization on mobile devices.** *In Proceedings of Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 2013 10th International Conference on. 1-5. IEEE.
- [19] Bostanci, E., Kanwal, N., Ehsan, S., et al. (2013) User tracking methods for augmented reality. *International Journal of Computer Theory and Engineering*, 5: (1): 93.
- [20] Bouguet, J.-Y. (2004) Camera calibration toolbox for matlab.
- [21] Bourke, P. (2009) Ply-polygon file format. **Dostupné z www:** <http://paulbourke.net/dataformats/ply>.
- [22] Bradski, G. (2000) The opencv library. *Doctor Dobbs Journal*, 25: (11): 120-126.
- [23] Brown, D.C. (1966) Decentering distortion of lenses. *Photogrammetric Engineering*, 32: (3): 444-462.
- [24] Bruder, G., Steinicke, F., Valkov, D., et al. **Augmented virtual studio for architectural exploration.** *In Proceedings*. Citeseer.
- [25] Bryson, M. and Sukkarieh, S. (2008) Observability analysis and active control for airborne SLAM. *IEEE Transactions on Aerospace and Electronic Systems*, 44: (1): 261-280.
- [26] Buczkowski, A. (2016) **How accurate is your smartphone's GPS in an urban jungle?** [online].

- <http://geoawesomeness.com/how-accurate-is-your-smartphones-gps-in-an-urban-jungle/> [Accessed April 2017]
- [27] Burrus, N. (2012) **Kinect Calibration** [online]. <http://nicolas.burrus.name/index.php/Research/KinectCalibration> [Accessed August 2016]
- [28] Byröd, M. and Åström, K. (2010) **Conjugate gradient bundle adjustment**. In *Proceedings of European Conference on Computer Vision*. 114-127. Springer.
- [29] Cai, Z., Han, J., Liu, L., et al. (2017) RGB-D datasets using microsoft kinect or similar sensors: a survey. **Multimedia Tools and Applications**, 76: (3): 4313-4355.
- [30] Calonder, M., Lepetit, V., Strecha, C., et al. (2010) "BRIEF: Binary Robust Independent Elementary Features". In Daniilidis, K.;Maragos, P. & Paragios, N. (Eds.) **Computer Vision – ECCV 2010**. Springer Berlin Heidelberg 778-792.
- [31] Canclini, A., Cesana, M., Redondi, A., et al. (2013) **Evaluation of low-complexity visual feature detectors and descriptors**. In *Proceedings of Digital Signal Processing (DSP)*, 2013 18th International Conference on. 1-7. IEEE.
- [32] Canny, J. (1986) A computational approach to edge detection. **IEEE Transactions on pattern analysis and machine intelligence**, (6): 679-698.
- [33] Carmigniani, J. and Furht, B. (2011) "Augmented Reality: An Overview". In Furht, B. (Ed.) **Handbook of Augmented Reality**. Springer 3-46.
- [34] Carozza, L., Tingdahl, D., Bosché F., et al. (2014) Markerless vision-based augmented reality for urban planning. **Computer-Aided Civil and Infrastructure Engineering**, 29: (1): 2-17.
- [35] Cawood, S. and Fiala, M. (2008) **Augmented Reality: A Practical Guide**. Pragmatic Bookshelf.
- [36] Chang, W. and Qing, T. (2010) **Augmented Reality System Design and Scenario Study for Location-Based Adaptive Mobile Learning**. In *Proceedings of Computational Science and Engineering (CSE)*, 2010 IEEE 13th International Conference on. 20-27 11-13 Dec. 2010.
- [37] Chen, J., Zou, L.-h., Zhang, J., et al. (2009) The comparison and application of corner detection algorithms. **Journal of multimedia**, 4: (6): 435-441.
- [38] Chen, P. (2004) **Chap. 3: Geometric Transformations** [online]. <http://www.di.ubi.pt/~agomes/cg/teoricacg/03e-transformations.pdf> [Accessed August 2016]
- [39] Chen, Z. (2003) Bayesian filtering: From Kalman filters to particle filters, and beyond. **Statistics**, 182: (1): 1-69.
- [40] Cheng, Q. (2015) **Determining Principles for the Development of Virtual Environments for Future Clinical Applications**. DOCTOR OF PHILOSOPHY (Ph.D.), The University of Birmingham.
- [41] Choi, J., Jang, B. and Kim, G.J. (2011) Organizing and presenting geospatial tags in location-based augmented reality. **Personal and Ubiquitous Computing**, 15: (6): 641-647.
- [42] Choi, S., Kim, T. and Yu, W. (1997) Performance evaluation of RANSAC family. **Journal of Computer Vision**, 24: (3): 271-300.
- [43] Chum, O. and Matas, J. (2005) **Matching with PROSAC-progressive sample consensus**. In *Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. 220-226. IEEE.1
- [44] Comport, A.I., Marchand, E. and Chaumette, F. (2003) **A real-time tracker for markerless augmented reality**. In *Proceedings of Mixed and Augmented Reality*, 2003. Proceedings. The Second IEEE and ACM International Symposium on. 36-45 7-10 Oct. 2003.
- [45] Corke, P., Lobo, J. and Dias, J. (2007) An introduction to inertial and visual sensing. **The International Journal of Robotics Research**, 26: (6): 519-535.
- [46] Courtney, J., Magee, M.J. and Aggarwal, J.K. (1984) Robot guidance using computer vision. **Pattern Recognition**, 17: (6): 585-592.
- [47] Crandall, D.J., Owens, A., Snavely, N., et al. (2013) SfM with MRFs: Discrete-continuous optimization for large-scale structure from motion. **IEEE Transactions on pattern analysis and machine intelligence**, 35: (12): 2841-2853.
- [48] Crossley, M. (2015) "A guide to coordinate systems in Great Britain". In Survey, O. (Ed.) D00659 v2.4 ed.
- [49] Cui, Y., Schuon, S., Chan, D., et al. (2010) **3D shape scanning with a time-of-flight camera**. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2010 IEEE Conference on. 1173-1180. IEEE.
- [50] Cummins, M. and Newman, P. (2007) **Probabilistic appearance based navigation and loop closing**. In *Proceedings of Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2042-2048. IEEE.
- [51] Cummins, M. and Newman, P. (2008) FAB-MAP: Probabilistic localization and mapping in the

- space of appearance. **The International Journal of Robotics Research**, 27: (6): 647-665.
- [52] Cummins, M. and Newman, P. (2011) Appearance-only SLAM at large scale with FAB-MAP 2.0. **The International Journal of Robotics Research**, 30: (9): 1100-1123.
- [53] Curless, B. (1999) From range scans to 3D models. **ACM SIGGRAPH Computer Graphics**, 33: (4): 38-41.
- [54] Del Bimbo, A. (1999) **Visual information retrieval**. Morgan and Kaufmann.
- [55] Dellaert, F., Seitz, S.M., Thorpe, C.E., et al. (2000) **Structure from motion without correspondence**. In *Proceedings of Computer Vision and Pattern Recognition*, 2000. Proceedings. IEEE Conference on. 557-564. IEEE.2
- [56] DeMenthon, D.F. and Davis, L.S. (1992) **Model-based object pose in 25 lines of code**. In *Proceedings of European conference on computer vision*. 335-343. Springer.
- [57] DeSouza, G.N. and Kak, A.C. (2002) Vision for mobile robot navigation: A survey. **IEEE Transactions on pattern analysis and machine intelligence**, 24: (2): 237-267.
- [58] Dong, p.T. (2013) A review on image feature extraction and representation techniques. **International Journal of Multimedia and Ubiquitous Engineering**, 8: (4): 385-396.
- [59] Dünser, A. and Billinghurst, M. (2011) "Evaluating augmented reality systems". **Handbook of augmented reality**. Springer 289-307.
- [60] Eggert, D.W., Lorusso, A. and Fisher, R.B. (1997) Estimating 3-D rigid body transformations: a comparison of four major algorithms. **Machine Vision and Applications**, 9: (5-6): 272-290.
- [61] Endres, F., Hess, J., Engelhard, N., et al. (2012) **An evaluation of the RGB-D SLAM system**. In *Proceedings of Robotics and Automation (ICRA)*, 2012 IEEE International Conference on. 1691-1696. IEEE.
- [62] Endres, F., Hess, J., Sturm, J., et al. (2014) 3D Mapping with an RGB-D Camera. **Robotics, IEEE Transactions on**, 30: (1): 177-187.
- [63] Engel, J., Schöps, T. and Cremers, D. (2014) **LSD-SLAM: Large-scale direct monocular SLAM**. In *Proceedings of European Conference on Computer Vision*. 834-849. Springer.
- [64] Ferreira, F., Veruggio, G., Caccia, M., et al. (2012) Real-time optical SLAM-based mosaicking for unmanned underwater vehicles. **Intelligent Service Robotics**, 5: (1): 55-71.
- [65] Fiala, M. (2005) **ARTag, a fiducial marker system using digital techniques**. In *Proceedings of Computer Vision and Pattern Recognition*, 2005. CVPR 2005. IEEE Computer Society Conference on. 590-596. IEEE.2
- [66] Fischler, M.A. and Bolles, R.C. (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. **Communications of the ACM**, 24: (6): 381-395.
- [67] Fjeld, M. (2003) Introduction: Augmented reality-usability and collaborative aspects. **International Journal of Human-Computer Interaction**, 16: (3): 387-393.
- [68] Fukuda, T., Zhang, T. and Yabuki, N. (2014) Improvement of registration accuracy of a handheld augmented reality system for urban landscape simulation. **Frontiers of Architectural Research**, 3: (4): 386-397.
- [69] Furness, T.A. (1969) "The Application of Head-Mounted Displays to Airborne Reconnaissance and Weapon Delivery". **Symposium for Image Display and Recording**. U.S. Air Force Avionics Laboratory, Wright-Patterson AFB.
- [70] Furukawa, Y. and Ponce, J. (2010) Accurate, dense, and robust multiview stereopsis. **Pattern Analysis and Machine Intelligence, IEEE Transactions on**, 32: (8): 1362-1376.
- [71] Galarza, A.I.R. and Seade, J. (2007) **Introduction to classical geometries**. Springer Science & Business Media.
- [72] Gao, X.-S., Hou, X.-R., Tang, J., et al. (2003) Complete solution classification for the perspective-three-point problem. **IEEE Transactions on pattern analysis and machine intelligence**, 25: (8): 930-943.
- [73] Geng, J. (2011) Structured-light 3D surface imaging: a tutorial. **Advances in Optics and Photonics**, 3: (2): 128-160.
- [74] Glover, A., Maddern, W., Warren, M., et al. (2012) **Openfabmap: An open source toolbox for appearance-based loop closure detection**. In *Proceedings of Robotics and Automation (ICRA)*, 2012 IEEE International Conference on. 4730-4735. IEEE.
- [75] Glover, A.J., Maddern, W.P., Milford, M.J., et al. (2010) **FAB-MAP+ RatSLAM: Appearance-based SLAM for multiple times of day**. In *Proceedings of Robotics and Automation (ICRA)*, 2010 IEEE International Conference on. 3507-3512. IEEE.
- [76] Gonzalez, R.C. and Woods, R.E. (2008) Digital image processing. **Nueva Jersey**.

- [77] Grisetti, G., Kummerle, R., Stachniss, C., et al. (2010) A tutorial on graph-based SLAM. **Intelligent Transportation Systems Magazine, IEEE**, 2: (4): 31-43.
- [78] Guo, J., Wang, Y., Chen, J., et al. (2009) **Markerless tracking for augmented reality applied in reconstruction of Yuanmingyuan archaeological site**. *In Proceedings of 11th IEEE International Conference on Computer-Aided Design and Computer Graphics*. 324-329.
- [79] Handa, A., Whelan, T., McDonald, J., et al. (2014) **A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM**. *In Proceedings of Robotics and automation (ICRA), 2014 IEEE international conference on*. 1524-1531. IEEE.
- [80] Harris, C. and Stephens, M. (1988) **A combined corner and edge detector**. *In Proceedings of Alvey vision conference*. 50. Citeseer.15
- [81] Harris, C.G. and Pike, J. (1988) 3D positional integration from image sequences. **Image and Vision Computing**, 6: (2): 87-90.
- [82] Hartley, R., Gupta, R. and Chang, T. (1992) **Stereo from uncalibrated cameras**. *In Proceedings of Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. 761-764. IEEE.
- [83] Hartley, R. and Zisserman, A. (2003) **Multiple view geometry in computer vision**. Cambridge university press.
- [84] Hartley, R.I. and Sturm, P. (1997) Triangulation. **Computer vision and image understanding**, 68: (2): 146-157.
- [85] Hassner, T. and Basri, R. (2006) **Example based 3D reconstruction from single 2D images**. *In Proceedings of 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*. 15-15. IEEE.
- [86] Haugstvedt, A.C. and Krogstie, J. (2012) **Mobile augmented reality for cultural heritage: A technology acceptance study**. *In Proceedings of Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*. 247-255 5-8 Nov. 2012.
- [87] Haykin, S.S. (2001) **Kalman filtering and neural networks**. Wiley Online Library.
- [88] Heare, D. and Baker, M.P. (1998) "Computer Graphics (C Version)". New Jersey: Prentice Hall International Inc.
- [89] Heikkilä, J. and Silvén, O. (1997) **A four-step camera calibration procedure with implicit image correction**. *In Proceedings of Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. 1106-1112. IEEE.
- [90] Henry, P., Krainin, M., Herbst, E., et al. (2010) **RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments**. *In Proceedings of In the 12th International Symposium on Experimental Robotics (ISER. Citeseer*.
- [91] Herling, J. and Broll, W. (2011) "Markerless Tracking for Augmented Reality". *In Furht, B. (Ed.) Handbook of Augmented Reality*. Springer 255-272.
- [92] Herrera, C.D., Kim, K., Kannala, J., et al. (2014) **Dt-slam: Deferred triangulation for robust slam**. *In Proceedings of 3D Vision (3DV), 2014 2nd International Conference on*. 609-616. IEEE.1
- [93] Herrera, D., Kannala, J. and Heikkilä J. (2011) **Accurate and practical calibration of a depth and color camera pair**. *In Proceedings of International Conference on Computer analysis of images and patterns*. 437-445. Springer.
- [94] Hildreth, E.C. and Ullman, S. (1982) "The measurement of visual motion". DTIC Document.
- [95] Holmdahl, T. (2015) **BUILD 2015: A closer look at the Microsoft HoloLens hardware** [online]. <https://blogs.windows.com/devices/2015/04/30/build-2015-a-closer-look-at-the-microsoft-hololens-hardware> [Accessed April 2017]
- [96] Horenstein, H. (2005) **Black and white photography: a basic manual**. Little Brown & Company.
- [97] Hsu, R.-L., Abdel-Mottaleb, M. and Jain, A.K. (2002) Face detection in color images. **IEEE Transactions on pattern analysis and machine intelligence**, 24: (5): 696-706.
- [98] Huang, Y., Jiang, Z., Liu, Y., et al. (2011) "Augmented Reality in Exhibition and Entertainment for the Public". *In Furht, B. (Ed.) Handbook of Augmented Reality*. Springer 707-720.
- [99] Idris, F. and Panchanathan, S. (1997) Review of image and video indexing techniques. **Journal of visual communication and image representation**, 8: (2): 146-166.
- [100] Ingrand, F., Lacroix, S., Lemai-Chenevier, S., et al. (2007) Decisional autonomy of planetary rovers. **Journal of Field Robotics**, 24: (7): 559-580.
- [101] Jansen, B.J. and Rieh, S.Y. (2010) The seventeen theoretical constructs of information searching and information retrieval. **Journal of the American Society for Information Science and Technology**, 61: (8): 1517-1534.
- [102] Jevremovic, V. and Petrovski, S. (2012) **MUZZEUM - Augmented Reality and QR codes**

enabled mobile platform with digital library, used to Guerrilla open the National Museum of Serbia. *In Proceedings of Virtual Systems and Multimedia (VSMM)*, 2012 18th International Conference on. 561-564 2-5 Sept. 2012.

[103] Johnson, L., Levine, A., Smith, R., et al. (2010) "Simple augmented reality". **The 2010 Horizon Report**. ERIC 21-24.

[104] Jordan, P., Socrate, S., Zickler, T., et al. (2009) Constitutive modeling of porcine liver in indentation using 3D ultrasound imaging. **Journal of the mechanical behavior of biomedical materials**, 2: (2): 192-201.

[105] Juan, L. and Gwon, O. (2009) A comparison of sift, pca-sift and surf. **International Journal of Image Processing (IJIP)**, 3: (4): 143-152.

[106] Kakuta, T., Oishi, T. and Ikeuchi, K. (2008) **Development and Evaluation of Asuka-Kyo MR Contents with Fast Shading and Shadowing.** *In Proceedings of International Society on Virtual Systems and MultiMedia (VSMM 2008)*. 254-260.

[107] Kato, H. and Billinghurst, M. (1999) **Marker tracking and HMD calibration for a video-based augmented reality conferencing system.** *In Proceedings of Augmented Reality*, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on. 85-94 1999.

[108] Kaufmann, H. (2004) **Geometry education with augmented reality.** PhD Dissertation thesis, Vienna University of Technology.

[109] Ke, Y. and Sukthankar, R. (2004) **PCA-SIFT: A more distinctive representation for local image descriptors.** *In Proceedings of Computer Vision and Pattern Recognition*, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on. II-506-II-513 Vol. 502. IEEE.2

[110] Kersten, T. and Lindstaedt, M. (2012) Image-based low-cost systems for automatic 3D recording and modelling of archaeological finds and objects. **Progress in cultural heritage preservation**, 1-10.

[111] Kim, J.-H. and Chung, M.J. (2003) **SLAM with omni-directional stereo vision sensor.** *In Proceedings of Intelligent Robots and Systems*, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on. 442-447. IEEE.1

[112] Klein, G. and Murray, D. (2007) **Parallel tracking and mapping for small AR workspaces.** *In Proceedings of Mixed and Augmented Reality*, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on. 225-234. IEEE.

[113] Klein, G. and Murray, D. (2009) **Parallel tracking and mapping on a camera phone.** *In Proceedings of Mixed and Augmented Reality*, 2009. ISMAR 2009. 8th IEEE International Symposium on. 83-86. IEEE.

[114] Knoche, H., De Meer, H.G. and Kirsh, D. (1999) **Utility curves: Mean opinion scores considered biased.** *In Proceedings of Quality of Service*, 1999. IWQoS'99. 1999 Seventh International Workshop on. 12-14. IEEE.

[115] Kolstee, Y. and van Eck, W. (2011) **The augmented Van Gogh's: Augmented reality experiences for museum visitors.** *In Proceedings of Mixed and Augmented Reality - Arts, Media, and Humanities (ISMAR-AMH)*, 2011 IEEE International Symposium On. 49-52 26-29 Oct. 2011.

[116] Kondo, D., Goto, T., Kouno, M., et al. (2004) **A virtual anatomical torso for medical education using free form image projection.** *In Proceedings of Proceedings of 10th International Conference on Virtual Systems and MultiMedia (VSMM2004)*. 678-685.

[117] Konolige, K. and Garage, W. (2010) **Sparse Sparse Bundle Adjustment.** *In Proceedings of BMVC*. 1-11. Citeseer.

[118] Koschan, A. and Abidi, M. (2005) Detection and classification of edges in color images. **IEEE Signal Processing Magazine**, 22: (1): 64-73.

[119] Košecká, J., Li, F. and Yang, X. (2005) Global localization and relative positioning based on scale-invariant keypoints. **Robotics and Autonomous Systems**, 52: (1): 27-38.

[120] Kosecka, J., Zhou, L., Barber, P., et al. (2003) **Qualitative image based localization in indoors environments.** *In Proceedings of Computer Vision and Pattern Recognition*, 2003. Proceedings. 2003 IEEE Computer Society Conference on. II-3-II-8 vol. 2. IEEE.2

[121] Kostaras, N.N. and Xenos, M.N. (2009) **Assessing the usability of augmented reality systems.** *In Proceedings*.

[122] Krevelen, D.W.F.v. and Poelman, R. (2010) A Survey of Augmented Reality Technologies, Applications and Limitations. **The International Journal of Virtual Reality**, 9: (2): 1-20.

[123] Kümmerle, R., Grisetti, G., Strasdat, H., et al. (2011) **g2o: A general framework for graph optimization.** *In Proceedings of Robotics and Automation (ICRA)*, 2011 IEEE International Conference on. 3607-3613. IEEE.

[124] Kurihara, T. and Sagawa, H. (2014) **Markerless Camera Tracking for Complex Structures such**

as Plant Facilities. *In Proceedings of IEEE ISMAR 2014 Workshop on Tracking Methods & Applications.*

[125] Labbe, M. and Michaud, F. (2014) **Online global loop closure detection for large-scale multi-session graph-based slam.** *In Proceedings of Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on.* 2661-2666. IEEE.

[126] Labudzki, R. and Legutko, S. (2011) Applications of machine vision. *Manuf. Ind. Eng.* 2: 27 - 29.

[127] Lamon, P., Nourbakhsh, I., Jensen, B., et al. (2001) **Deriving and matching image fingerprint sequences for mobile robot localization.** *In Proceedings of ICRA.* 1609-1614.

[128] Langley, R.B. (1998) RTK GPS. *GPS World*, 9: (9): 70-76.

[129] Lau, D. (2013) **The Science Behind Kinects or Kinect 1.0 versus 2.0** [online]. http://www.gamasutra.com/blogs/DanielLau/20131127/205820/The_Science_Behind_Kinects_or_Kinect_10_versus_20.php [Accessed April 2017]

[130] Lee, G.A., Dunser, A., Kim, S., et al. (2012) **CityViewAR: A mobile outdoor AR application for city visualization.** *In Proceedings of Mixed and Augmented Reality (ISMAR-AMH), 2012 IEEE International Symposium on.* 57-64. IEEE.

[131] Lee, K. (2012) Augmented reality in education and training. *TechTrends*, 56: (2): 13-21.

[132] Lee, T. and Hollerer, T. (2007) **Handy AR: Markerless inspection of augmented reality objects using fingertip tracking.** *In Proceedings of Wearable Computers, 2007 11th IEEE International Symposium on.* 83-90. IEEE.

[133] Lemaire, T., Berger, C., Jung, I.-K., et al. (2007) Vision-based slam: Stereo and monocular approaches. *International journal of computer vision*, 74: (3): 343-364.

[134] Leutenegger, S., Chli, M. and Siegwart, R.Y. (2011) **BRISK: Binary robust invariant scalable keypoints.** *In Proceedings of 2011 International conference on computer vision.* 2548-2555. IEEE.

[135] Levenberg, K. (1944) A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2: (2): 164-168.

[136] Levoy, M., Pulli, K., Curless, B., et al. (2000) **The digital Michelangelo project: 3D scanning of large statues.** *In Proceedings of Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* 131-144. ACM Press/Addison-Wesley Publishing Co.

[137] Liao, H., Inomata, T., Sakuma, I., et al. (2010) Three-dimensional augmented reality for mr-guided surgery using integral videography auto stereoscopic-image overlay. *IEEE transactions on biomedical engineering*, 57: (6): 1476-1486.

[138] Lieberman, H., Paternò, F., Klann, M., et al. (2006) "End-user development: An emerging paradigm". *End user development.* Springer 1-8.

[139] Lindeberg, T. (1998a) Edge detection and ridge detection with automatic scale selection. *International journal of computer vision*, 30: (2): 117-156.

[140] Lindeberg, T. (1998b) Feature detection with automatic scale selection. *International journal of computer vision*, 30: (2): 79-116.

[141] Lisin, D.A., Mattar, M.A., Blaschko, M.B., et al. (2005) **Combining local and global image features for object class recognition.** *In Proceedings of 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops.* 47-47. IEEE.

[142] Liu, Y. and Zhang, H. (2013) **Performance evaluation of whole-image descriptors in visual loop closure detection.** *In Proceedings of Information and Automation (ICIA), 2013 IEEE International Conference on.* 716-722. IEEE.

[143] Livingston, M.A., Rosenblum, L.J., Brown, D.G., et al. (2011) "Military applications of augmented reality". *Handbook of augmented reality.* Springer 671-706.

[144] Livingston, M.A., Rosenblum, L.J., Julier, S.J., et al. (2002) "An augmented reality system for military operations in urban terrain". DTIC Document.

[145] Longuet-Higgins, H.C. (1987) A computer algorithm for reconstructing a scene from two projections. *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, MA Fischler and O. Firschein, eds, 61-62.

[146] Lourakis, M. and Argyros, A.A. (2005) **Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment?** *In Proceedings of Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1.* 1526-1531. IEEE.2

[147] Lourakis, M.I. and Argyros, A.A. (2009) SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36: (1): 2.

[148] Lowe, D.G. (1999) **Object recognition from local scale-invariant features.** *In Proceedings of Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on.* 1150-1157 vol.1152 1999. 2

- [149] Lowe, D.G. (2004) Distinctive image features from scale-invariant keypoints. **International journal of computer vision**, 60: (2): 91-110.
- [150] Lowe, D.G. (2005) "Demo software: Sift keypoint detector".
- [151] Lu, C.-P., Hager, G.D. and Mjolsness, E. (2000) Fast and globally convergent pose estimation from video images. **IEEE Transactions on pattern analysis and machine intelligence**, 22: (6): 610-622.
- [152] Lu, F. and Milios, E. (1997) Globally consistent range scan alignment for environment mapping. **Autonomous robots**, 4: (4): 333-349.
- [153] Ma, Y., Soatto, S., Kosecka, J., et al. (2012) **An invitation to 3-d vision: from images to geometric models**. Springer Science & Business Media.
- [154] Mallet, A., Lacroix, S. and Gallo, L. (2000) **Position estimation in outdoor environments using pixel tracking and stereovision**. In *Proceedings of Robotics and Automation*, 2000. Proceedings. ICRA'00. IEEE International Conference on. 3519-3524. IEEE.4
- [155] Marquardt, D.W. (1963) An algorithm for least-squares estimation of nonlinear parameters. **Journal of the society for Industrial and Applied Mathematics**, 11: (2): 431-441.
- [156] Marr, D. and Hildreth, E. (1980) Theory of edge detection. **Proceedings of the Royal Society of London B: Biological Sciences**, 207: (1167): 187-217.
- [157] Marr, D. and Poggio, T. (1976) "Cooperative computation of stereo disparity". **From the Retina to the Neocortex**. Springer 239-243.
- [158] Martinez, H. and Bandyopadhyay, P. (2014) **Analysis of Four Usability Evaluation Methods Applied to Augmented Reality Applications**.
- [159] Mataboni, P.J. (1992) "Mobile robot guidance and navigation system". Google Patents.
- [160] Mikolajczyk, K. and Schmid, C. (2004) Scale & affine invariant interest point detectors. **International journal of computer vision**, 60: (1): 63-86.
- [161] Mikolajczyk, K. and Schmid, C. (2005) A performance evaluation of local descriptors. **IEEE Transactions on pattern analysis and machine intelligence**, 27: (10): 1615-1630.
- [162] Mikolajczyk, K., Tuytelaars, T., Schmid, C., et al. (2005) A comparison of affine region detectors. **International journal of computer vision**, 65: (1-2): 43-72.
- [163] Miksik, O. and Mikolajczyk, K. (2012) **Evaluation of local detectors and descriptors for fast feature matching**. In *Proceedings of Pattern Recognition (ICPR)*, 2012 21st International Conference on. 2681-2684. IEEE.
- [164] Milgram, P., Takemura, H., Utsumi, A., et al. (1994) **Augmented reality: A class of displays on the reality-virtuality continuum**. In *Proceedings of Photonics for industrial applications*. 282-292. International Society for Optics and Photonics.
- [165] Minati, G., Abram, M. and Pessa, E. (2009) "Processes of Emergence of Systems and Systemic Properties: Towards a General Theory of Emergence". World Scientific.
- [166] Mor, L., Levy, R.M. and Boyd, J.E. (2012) "Augmented reality for virtual renovation". **Proceedings of the second international ACM workshop on Personalized access to cultural heritage**. Nara, Japan, ACM.
- [167] Mousavi, V., Khosravi, M., Ahmadi, M., et al. (2015) The Performance Evaluation of Multi-Image 3d Reconstruction Software with Different Sensors. **The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences**, 40: (1): 515.
- [168] Muda, A.K., Choo, Y.-H., Abraham, A., et al. (2014) **Computational Intelligence in Digital Forensics: Forensic Investigation and Applications**. Springer.
- [169] Muja, M. and Lowe, D.G. (2009) Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. **VISAPP (1)**, 2.
- [170] Muja, M. and Lowe, D.G. (2012) **Fast matching of binary features**. In *Proceedings of Computer and Robot Vision (CRV)*, 2012 Ninth Conference on. 404-410. IEEE.
- [171] Navab, N. and Unger, C. (2010) Rectification and disparity.
- [172] Neider, J. and Davis, T. (1993) **OpenGL programming guide: the official guide to learning OpenGL, release 1**. Addison-Wesley Longman Publishing Co., Inc.
- [173] Newcombe, R.A., Izadi, S., Hilliges, O., et al. (2011) **KinectFusion: Real-time dense surface mapping and tracking**. In *Proceedings of Mixed and augmented reality (ISMAR)*, 2011 10th IEEE international symposium on. 127-136. IEEE.
- [174] Newman, P., Cole, D. and Ho, K. (2006) **Outdoor SLAM using visual appearance and laser ranging**. In *Proceedings of Proceedings 2006 IEEE International Conference on Robotics and Automation*, 2006. ICRA 2006. 1180-1187. IEEE.
- [175] Nielsen, J. (1994a) **Usability engineering**. Elsevier.
- [176] Nielsen, J. (1994b) **Usability inspection methods**. In *Proceedings of Conference companion on*

- Human factors in computing systems. 413-414. ACM.
- [177] Nielsen, J. and Molich, R. (1990) **Heuristic evaluation of user interfaces**. In *Proceedings of Proceedings of the SIGCHI conference on Human factors in computing systems*. 249-256. ACM.
- [178] Nistér, D. (2004) An efficient solution to the five-point relative pose problem. **IEEE Transactions on pattern analysis and machine intelligence**, 26: (6): 756-770.
- [179] Nistér, D. (2005) Preemptive RANSAC for live structure and motion estimation. **Machine Vision and Applications**, 16: (5): 321-329.
- [180] Nistér, D., Naroditsky, O. and Bergen, J. (2006) Visual odometry for ground vehicle applications. **Journal of Field Robotics**, 23: (1): 3-20.
- [181] Núñez, M., Quirós, R., Núñez, I., et al. (2008) **Collaborative augmented reality for inorganic chemistry education**. In *Proceedings of WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*. WSEAS.
- [182] Pagliari, D. and Pinto, L. (2015) Calibration of kinect for xbox one and comparison between the two generations of Microsoft sensors. **Sensors**, 15: (11): 27569-27589.
- [183] Pandey, M., Nair, M.P., Wadhwa, M., et al. (2014) Augmented Reality based on Image Processing. **International Journal Of Engineering And Computer Science**, 3: (5): 5924-5929.
- [184] Papagiannakis, G., Ponder, M., Molet, T., et al. (2002) **LIFEPLUS: Revival of life in ancient Pompeii**. In *Proceedings of International Society on Virtual Systems and MultiMedia (VSMM 2002)*.
- [185] Park, D.K., Jeon, Y.S. and Won, C.S. (2000) **Efficient use of local edge histogram descriptor**. In *Proceedings of Proceedings of the 2000 ACM workshops on Multimedia*. 51-54. ACM.
- [186] Patsadu, O., Nukoolkit, C. and Watanapa, B. (2012) **Human gesture recognition using Kinect camera**. In *Proceedings of Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*. 28-32. IEEE.
- [187] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011) Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, 12: (Oct): 2825-2830.
- [188] Pressigout, M. and Marchand, E. (2006) **Hybrid tracking algorithms for planar and non-planar structures subject to illumination changes**. In *Proceedings of Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*. 52-55. IEEE Computer Society.
- [189] Prewitt, J.M. (1970) Object enhancement and extraction. **Picture processing and Psychopictorics**, 10: (1): 15-19.
- [190] Pulli, K., Baksheev, A., Korniyakov, K., et al. (2012) Real-time computer vision with OpenCV. **Communications of the ACM**, 55: (6): 61-69.
- [191] Quigley, M., Conley, K., Gerkey, B., et al. (2009) **ROS: an open-source Robot Operating System**. In *Proceedings of ICRA workshop on open source software*. 5. Kobe, Japan.3
- [192] Rabbi, I. and Ullah, S. (2014) **3D Model Visualization and Interaction Using a Cubic Fiducial Marker**. In *Proceedings of International Conference on Augmented and Virtual Reality*. 381-393. Springer.
- [193] Reimann, A. (2015) **Intrinsic calibration of the Kinect cameras** [online]. http://wiki.ros.org/openni_launch/Tutorials/IntrinsicCalibration [Accessed August 2016]
- [194] Reitmayr, G. and Drummond, T.W. (2006) **Going out: robust model-based tracking for outdoor augmented reality**. In *Proceedings of Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on*. 109-118 22-25 Oct. 2006.
- [195] Reitmayr, G. and Schmalstieg, D. (2004) **Collaborative augmented reality for outdoor navigation and information browsing**. na.
- [196] Remondino, F. and El-Hakim, S. (2006) Image-based 3D modelling: A review. **The Photogrammetric Record**, 21: (115): 269-291.
- [197] Rieman, J., Franzke, M. and Redmiles, D. (1995) **Usability evaluation with the cognitive walkthrough**. In *Proceedings of Conference companion on Human factors in computing systems*. 387-388. ACM.
- [198] Riisgaard, S. and Blas, M.R. (2003) SLAM for Dummies. **A Tutorial Approach to Simultaneous Localization and Mapping**, 22: (1-127): 126.
- [199] Roberts, G.W., Evans, A., Dodson, A., et al. (2002) **The use of augmented reality, GPS and INS for subsurface data visualization**. In *Proceedings of FIG XXII International Congress*. 1-12.
- [200] Roberts, L.G. (1963) **Machine perception of three-dimensional soups**. Massachusetts Institute of Technology.
- [201] Roden, T.E., Parberry, I. and Ducrest, D. (2007) Toward mobile entertainment: A paradigm for narrative-based audio only games. **Science of Computer Programming**, 67: (1): 76-90.
- [202] Rolland, J.P., Yohan Baillot and Goon, A.A. (2001) "A survey of tracking technology for virtual

- environments.". *In* Barfield, W. & Caudell, T. (Eds.) **Fundamentals of Wearable Computers and Augmented Reality**. Lawrence Erlbaum.
- [203] Rosin, P.L. (1999) Measuring corner properties. **Computer vision and image understanding**, 73: (2): 291-307.
- [204] Rosten, E. and Drummond, T. (2006) "Machine learning for high-speed corner detection". **Proceedings of the 9th European conference on Computer Vision - Volume Part I**. Graz, Austria, Springer-Verlag.
- [205] Rosten, E., Porter, R. and Drummond, T. (2010) Faster and better: A machine learning approach to corner detection. **IEEE Transactions on pattern analysis and machine intelligence**, 32: (1): 105-119.
- [206] Rousseeuw, P.J. (1984) Least median of squares regression. **Journal of the American statistical association**, 79: (388): 871-880.
- [207] Rublee, E., Rabaud, V., Konolige, K., et al. (2011) **ORB: An efficient alternative to SIFT or SURF**. *In* *Proceedings of Computer Vision (ICCV)*, 2011 IEEE International Conference on. 2564-2571 6-13 Nov. 2011.
- [208] Rui, Y., Huang, T.S. and Chang, S.-F. (1999) Image retrieval: Current techniques, promising directions, and open issues. **Journal of visual communication and image representation**, 10: (1): 39-62.
- [209] Russell, M. (2012) **11 Amazing Augmented Reality Ads** [online]. <http://www.businessinsider.com/11-amazing-augmented-reality-ads-2012-1?op=1&IR=T/#t-a-porter-makes-storefronts-interactive-1> [Accessed October 2016]
- [210] Rusu, R.B. and Cousins, S. (2011) **3d is here: Point cloud library (pcl)**. *In* *Proceedings of Robotics and Automation (ICRA)*, 2011 IEEE International Conference on. 1-4. IEEE.
- [211] Santana-Fernández, J., Gómez-Gil, J. and del-Pozo-San-Cirilo, L. (2010) Design and implementation of a GPS guidance system for agricultural tractors using augmented reality technology. **Sensors**, 10: (11): 10435-10447.
- [212] Schall, G., Wagner, D., Reitmayr, G., et al. (2009) **Global pose estimation using multi-sensor fusion for outdoor augmented reality**. *In* *Proceedings of Mixed and Augmented Reality*, 2009. ISMAR 2009. 8th IEEE International Symposium on. 153-162. IEEE.
- [213] Schleicher, D., Bergasa, L.M., Ocaña, M., et al. (2009) Real-time hierarchical outdoor SLAM based on stereovision and GPS fusion. **IEEE Transactions on Intelligent Transportation Systems**, 10: (3): 440-452.
- [214] Schöning, J. and Heidemann, G. (2015) **Evaluation of multi-view 3D reconstruction software**. *In* *Proceedings of International Conference on Computer Analysis of Images and Patterns*. 450-461. Springer.
- [215] Sharifi, M., Fathy, M. and Mahmoudi, M.T. (2002) **A classified and comparative study of edge detection algorithms**. *In* *Proceedings of Information Technology: Coding and Computing*, 2002. Proceedings. International Conference on. 117-120. IEEE.
- [216] Sherstyuk, A., Vincent, D., Berg, B., et al. (2011) "Mixed reality Manikins for Medical Education". **Handbook of Augmented reality**. Springer 479-500.
- [217] Shilov, A. (2016) **Intel and Google Equip Smartphones with 3D Cameras and Computer Vision** [online]. <http://www.anandtech.com/show/9940/intel-and-google-equip-smartphones-with-3d-cameras-and-computer-vision> [Accessed April 2017]
- [218] Simões, F.P.M., Roberto, R.A., Figueiredo, L.S., et al. (2013) **3D tracking in industrial scenarios: a case study at the ISMAR tracking competition**. *In* *Proceedings of Virtual and Augmented Reality (SVR)*, 2013 XV Symposium on. 97-106. IEEE.
- [219] Sivic, J. and Zisserman, A. (2003) **Video Google: A text retrieval approach to object matching in videos**. *In* *Proceedings of Computer Vision*, 2003. Proceedings. Ninth IEEE International Conference on. 1470-1477. IEEE.
- [220] Smith, S.M. and Brady, J.M. (1997) SUSAN—a new approach to low level image processing. **International journal of computer vision**, 23: (1): 45-78.
- [221] Snavely, N., Seitz, S.M. and Szeliski, R. (2008) Modeling the world from internet photo collections. **International journal of computer vision**, 80: (2): 189-210.
- [222] Sobel, I. and Feldman, G. (1968) A 3x3 isotropic gradient operator for image processing. **a talk at the Stanford Artificial Project in**, 271-272.
- [223] Sola, J., Monin, A., Devy, M., et al. (2008) Fusing monocular information in multicamera SLAM. **IEEE transactions on robotics**, 24: (5): 958-968.
- [224] Sonka, M., Hlavac, V. and Boyle, R. (2014) **Image processing, analysis, and machine vision**. Cengage Learning.
- [225] Starner, T., Schiele, B. and Pentland, A. (1998) **Visual contextual awareness in wearable**

- computing.** *In Proceedings of Wearable Computers*, 1998. Digest of Papers. Second International Symposium on. 50-57. IEEE.
- [226] Steinbach, M., Karypis, G. and Kumar, V. (2000) **A comparison of document clustering techniques.** *In Proceedings of KDD workshop on text mining.* 525-526. Boston.400
- [227] Stone, R.J. (1996) "A Study of the Virtual Reality Market". Department of Trade and Industry.
- [228] Strasdat, H., Montiel, J. and Davison, A.J. (2010) **Real-time monocular SLAM: Why filter?** *In Proceedings of Robotics and Automation (ICRA)*, 2010 IEEE International Conference on. 2657-2664. IEEE.
- [229] Strasdat, H., Montiel, J.M. and Davison, A.J. (2012) Visual SLAM: why filter? **Image and Vision Computing**, 30: (2): 65-77.
- [230] Strecha, C., von Hansen, W., Van Gool, L., et al. (2008) **On benchmarking camera calibration and multi-view stereo for high resolution imagery.** *In Proceedings of Computer Vision and Pattern Recognition*, 2008. CVPR 2008. IEEE Conference on. 1-8. Ieee.
- [231] Stricker, M.A. and Orengo, M. (1995) **Similarity of color images.** *In Proceedings of IS&T/SPIE's Symposium on Electronic Imaging: Science & Technology.* 381-392. International Society for Optics and Photonics.
- [232] Sturm, J., Engelhard, N., Endres, F., et al. (2012) **A benchmark for the evaluation of RGB-D SLAM systems.** *In Proceedings of Intelligent Robots and Systems (IROS)*, 2012 IEEE/RSJ International Conference on. 573-580. IEEE.
- [233] Sun, R., Sui, Y., Li, R., et al. (2011) **The design of a new marker in augmented reality.** *In Proceedings of International Conference on Economics and Finance Research Singapore*, 129-132.
- [234] Swain, M.J. and Ballard, D.H. (1991) Color indexing. **International journal of computer vision**, 7: (1): 11-32.
- [235] Szeliski, R. (2010) **Computer vision: algorithms and applications.** Springer Science & Business Media.
- [236] Temeltas, H. and Kayak, D. (2008) SLAM for robot navigation. **IEEE Aerospace and Electronic Systems Magazine**, 23: (12): 16-19.
- [237] Thoeni, K., Giacomini, A., Murtagh, R., et al. (2014) A comparison of multi-view 3D reconstruction of a rock wall using several cameras and a laser scanner. **The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences**, 40: (5): 573.
- [238] Torr, P.H. and Zisserman, A. (2000) MLESAC: A new robust estimator with application to estimating image geometry. **Computer vision and image understanding**, 78: (1): 138-156.
- [239] Torralba, A., Murphy, K.P., Freeman, W.T., et al. (2003) **Context-based vision system for place and object recognition.** *In Proceedings of Computer Vision*, 2003. Proceedings. Ninth IEEE International Conference on. 273-280. IEEE.
- [240] Triggs, B., McLauchlan, P.F., Hartley, R.I., et al. (1999) **Bundle adjustment—a modern synthesis.** *In Proceedings of International workshop on vision algorithms.* 298-372. Springer.
- [241] Ullman, S. (1979) **The interpretation of visual motion.** Massachusetts Inst of Technology Pr.
- [242] Ullman, S. (1996) **High-level vision: Object recognition and visual cognition.** MIT press Cambridge, MA.
- [243] Van Krevelen, D. and Poelman, R. (2007) *Augmented Reality: Technologies, Applications, and Limitations.*
- [244] Visual Geometry Group, U.o.O. (2004) **Multi-view and oxford colleges building reconstruction** [online]. <http://www.robots.ox.ac.uk/~vgg/data/data-mview.html> [Accessed April 2017]
- [245] **Vuforia** [online]. <https://www.vuforia.com/> [Accessed April 2017]
- [246] Wang, J., Zha, H. and Cipolla, R. (2006) Coarse-to-fine vision-based localization by indexing scale-invariant features. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, 36: (2): 413-422.
- [247] White, M., Petridis, P., Liarokapis, F., et al. (2007) Multimodal mixed reality interfaces for visualizing digital heritage. **International Journal of Architectural Computing**, 5: (2): 322-337.
- [248] Wilson, J.R. (1999) Virtual environments applications and applied ergonomics. **Applied Ergonomics**, 30: (1): 3-9.
- [249] Won, C.S., Park, D.K. and Park, S.-J. (2002) Efficient use of MPEG-7 edge histogram descriptor. **Etri Journal**, 24: (1): 23-30.
- [250] Wu, C. **SiftGPU manual** [online]. <http://cs.unc.edu/~ccwu/siftgpu/manual.pdf> [Accessed August 2016]
- [251] Wu, C. (2007) "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)".
- [252] Wu, C. (2011) VisualSfM: A visual structure from motion system.

- [253] Wu, C. (2013) **Towards linear-time incremental structure from motion**. *In Proceedings of 3D Vision-3DV 2013*, 2013 International Conference on. 127-134. IEEE.
- [254] Wu, C., Agarwal, S., Curless, B., et al. (2011) **Multicore bundle adjustment**. *In Proceedings of Computer Vision and Pattern Recognition (CVPR)*, 2011 IEEE Conference on. 3057-3064. IEEE.
- [255] Wu, Y. and Huang, T.S. (1999) **Vision-based gesture recognition: A review**. *In Proceedings of International Gesture Workshop*. 103-115. Springer.
- [256] Yan, C.C., Xie, H., Zhang, B., et al. (2015) Fast approximate matching of binary codes with distinctive bits. *Frontiers of Computer Science*, 9: (5): 741-750.
- [257] Yang, M.-H., Kriegman, D.J. and Ahuja, N. (2002) Detecting faces in images: A survey. *IEEE Transactions on pattern analysis and machine intelligence*, 24: (1): 34-58.
- [258] Zhang, C. and Zhang, Z. (2014) "Calibration between depth and color sensors for commodity depth cameras". *Computer Vision and Machine Learning with RGB-D Sensors*. Springer 47-64.
- [259] Zhang, Z. (2000) A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22: (11): 1330-1334.
- [260] Zhou, F., Duh, H.B.-L. and Billinghamurst, M. (2008) **Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR** *In Proceedings of 7th IEEE/ACM International Symposium on*. IEEE, 2008. ISMAR 2008.
- [261] Zhu, X., Cao, Q., Yokoi, H., et al. (2016) **Large Scale Indoor 3D Mapping Using RGB-D Sensor**. *In Proceedings of International Conference on Intelligent Robotics and Applications*. 313-321. Springer.
- [262] Zoellner, M., Keil, J., Drevensek, T., et al. (2009) **Cultural Heritage Layers: Integrating Historic Media in Augmented Reality**. *In Proceedings of 15th International Conference*. 193-196. IEEE.

APPENDICES

A. AR application development & requirement audience survey

The aim of this survey is to identify the requirements and concerned factors of developers (and potential developers) for Augmented Reality (AR) application development.

* Required

1. Please state your occupation: *

2. Do you know what Augmented Reality (AR) is? *

(Mark only one oval)

- Yes
- Uncertain
- No

Explanation: Augmented Reality (AR) refers to a live view of the physical, real world environment which is augmented by using computer generated information (i.e. sound, video, graphics or text information). Recent popular AR applications

include Nintendo's Pok émon GO.

3. Have you ever used any AR applications? *

(Mark only one oval)

- Yes
- Uncertain – (skip to question 5)
- No – (skip to question 5)

4. Please name or describe one or two of the AR applications you have used: *

5. Have you ever had a thought to apply AR in your own work or project? *

(Mark only one oval)

- Yes, I have already made an AR application/system.
- Yes, but it's still just a thought or idea at the moment.
- No, but it may interest me later.
- No, I'm not interested at all. – (skip to question 24)

6. Which areas would you like to develop AR for? *

(Tick all that apply)

- Education or training
- Research
- Tour guide for exhibitions or tourist attractions
- Retail or advertisement

- Entertainment (e.g. mobile game)
- Other: _____

7. Have you ever developed an AR Application/System? *

(Mark only one oval)

- Yes, I have developed an application for my own use.
- Yes, I have developed an AR application for a third party.
- No, but I have asked other people to develop one for use by me. – (skip to question 11)
- No. – (skip to question 18)

8. Please describe your AR Application/System (for developer): *

9. How satisfied were you with your AR development experience? *

(Mark only one)

	1	2	3	4	5	
Very dissatisfied						Very satisfied

10. Please specify why you were satisfied or dissatisfied: *

– (skip to question 14)

11. Please describe your AR Application/System: *

12. How satisfied were you with your AR application? *

(Mark only one)

	1	2	3	4	5	
Very dissatisfied						Very satisfied

13. Please specify why you were satisfied or dissatisfied:*

– (skip to question 18)

14. During the development, what factors do you consider the most important when evaluating an AR development toolkit? Please rate the importance of the following factors. *

(Mark only one per row)

	Not at all important	Slightly important	Important	Fairly important	Very important	No opinion
Proper documentation support						
Simple and intuitive user interface						
Cross-platform						
The provided functions (e.g. tracking, rendering methods) meet the end users' goals						
Robust performance						
Open source						
Reasonable price						

15. If you have other considerations, please specify below and rate the importance (5 for Very important and 1 for Not at all important):

16. During the development, have you had experience with the following situations?

Did you consider them as barriers of development? *

(Mark only one per row)

	Not a barrier	Somewhat of a barrier	Moderate barrier	Extreme barrier	I have never encountered this situation.
Poor documentation support					
Difficult-to-use user interface					
No support for multiple platforms					
The provided functions (e.g. tracking, rendering methods) did not meet the end users' goals					
Unstable performance					
Totally closed					
Unacceptable price					

17. If you have encountered other unfavourable situations, please specify below and

rate the level of difficulty (4 for Extreme barrier and 1 for Not a barrier):

Please consider the following factors, select the options that best describes the needs of your (expected/realised) AR case.

18. The general environment of your AR application is *

(Mark only one oval)

- Indoor
- Outdoor
- Both
- Uncertain

19. Is your application designed for a specific location (e.g. a heritage site)? *

(Mark only one oval)

- Yes
- Uncertain
- No

20. The tracking method used in your application is *

(Mark only one oval)

- Marker-based Tracking, based on artificial image markers or models.
- Markerless-based Tracking, based on natural features of the original environment (e.g. SLAM-based).

- Markerless-based Tracking, based on geographic information (e.g. GPS/INS-based).
- Hybrid Tracking.
- Uncertain.
- Other: _____

21. Does your application require an accurate AR registration (i.e. the AR contents always align to the background environment tightly in proper perspective)? *

(Mark only one oval)

- Yes
- Uncertain
- No

22. Aside from AR tool kits, what factors do you consider most important when evaluating a software development toolkit in general? If applicable, please rate the importance of the following factors.

(Mark only one per row)

	Not at all important	Slightly important	Important	Fairly important	Very important	No opinion
Proper documentation support						
Simple and intuitive user interface						
Cross-platform						

The provided functions (e.g. tracking, rendering methods) meet the end users' goals						
Robust performance						
Open source						
Reasonable price						

23. If you have other considerations, please specify below and rate the importance (5 for Very important and 1 for Not at all important):

24. If you have any other thoughts about AR, please write it below:

B. Software and development supports

An AR system can be developed with the assistance of several existing software frameworks and development tools. Lower-level development libraries and tools make it easier for researchers to program and realise AR frameworks with their own methods. Some popular and powerful software libraries are given below.

ROS

Robot Operating System (ROS) is a set of software libraries and tools for robot software development, although it has been more generally used in many other domains. The goals of ROS include solving the *code reuse* issues resulting from the growing number of robot types and their widely varying hardware, and managing complexity and facilitating rapid prototyping of software for experiments (Quigley *et al.*, 2009). To that end, ROS contains many Open Source implementations of common robotics functionality and algorithms, referred to as *ROS packages*. Many CV-relevant packages are available for ROS due to the close connection between the CV and robotics community (e.g. CV-based SLAM). Another useful feature of ROS is providing standard operating system services like package management and passing message between nodes/processes (the running processes of ROS are represented in a graphical architecture, thus each executable file within an ROS package is treated as a node). ROS nodes use an *ROS client library* (e.g. *roscpp* for c++ language and *rospy* for Python scripting language) to communicate with other nodes by publishing or subscribing to a topic, as well as writing and calling a service. An example of the implementation of an ROS application RGBDSLAM v2 is described in Section 4.1.2.

OpenCV

OpenCV is a library of programming functions aimed at providing the tools needed to solve CV problems (Pulli *et al.*, 2012). It is written in C++ and its primary interface is in C++, but also provides interfaces or wrappers for many other languages, such as Python and C#. It contains a mixture of low-level image-processing functions and high-level algorithms which are very useful for vision-based AR processing such as camera calibration, described in Section 2.2.1.1. Many popular feature extraction and matching methods are also described in Section 2.2.2. OpenCV also provides useful functions for dealing with epipolar geometry, the PnP problem, and many Open Source works provided by third parties. The detailed use of OpenCV for building the AR system in the present research will be presented in the following chapters. OpenCV is also used as the primary vision package in ROS.

PCL

Point Cloud Library (PCL) is an open-source library of algorithms for point cloud processing tasks and 3D geometry processing. It is written in C++. The library contains state-of-the-art algorithms for: filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation (Rusu & Cousins, 2011). It is used to hold the 3D points measured (by 3D camera) or recovered (by 3D reconstruction) with point cloud datasets, and the library offers a few general purpose tools for the user to interact with or visualise these point clouds. The point cloud data can be stored within a PCD file used inside PCL.

OpenGL

Open Graphics Library (OpenGL) is a cross-language, cross-platform API to graphics hardware (Neider & Davis, 1993). As mentioned in the last section, it can be used to interact with the GPU and enable GPU-accelerated computing. But it also can be used for rendering 2D and 3D vector graphics. Baggio (2012) notes that, although there are several commercial and Open Source 3D-engines (such as *Unreal Engine*, *Ogre* and *Unity 3D*) which are well qualified for the rendering task, they are all based on either OpenGL or Microsoft Direct3D. However, Direct3D is only supported on the Windows platform, thus OpenGL is more appropriate for building cross-platform rendering systems. OpenGL is usually used for creating a display window (viewport) for rendering and visualising 3D objects in many systems. The implementation of an OpenGL AR browser used in the proposed framework mentioned in Section 6.1.2 is described below.

- **2D background mapping**

To render an AR scene, the original input image is used as the background, and the virtual 3D models are projected onto it. To set up the background plane, OpenGL generates a 2D texture with the image and linearly maps it onto the whole quad viewport of the AR browser. Since the background plane does not have depth, an orthographic projection shown in 6.1 is used to project it onto the screen.

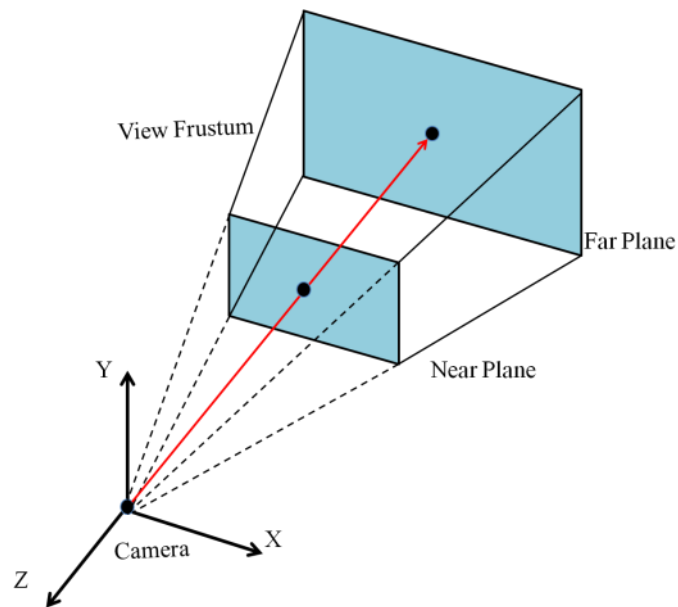
$$\begin{bmatrix} -\frac{2}{width} & 0 & 0 & 1 \\ 0 & -\frac{2}{height} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (\text{B.1})$$

- **3D graphics display**

The process with OpenGL to produce a 3D scene on the screen is very similar to taking a photograph with a camera (Neider & Davis, 1993), and involves several transformations quite like those of 3D-to-2D process depicted in Figure 3-13. The most important relationship that needs to be identified is the relative position between the camera and the scene or models to capture. More specifically, OpenGL has a fixed *right-handed coordinate system* (RHS) as world reference frame to describe them. The viewport of an OpenGL window is equivalent to the camera view which is referred as eye space, and the local space of the models in the world is referred as object space. The definition of axes and origins of these two spaces are the same as those of the world space by default, which means that, when the viewer is looking at the viewport, the positive X-axis is pointing right, the positive Y-axis is pointing up and the positive Z-axis is pointing out of the screen. In the case of the OpenGL display window of the GUI tool shown in Figure 6-2, it seems that developer was adjusting the viewpoint to observe the scene from different perspectives in screenshots ② and ③, but actually the eye space of viewer was never moved. The identical visual perception of moving a camera in one direction with respect to the world can be generated by moving the so-called ‘fixed’ dense reconstruction (with the reference axes) and the inserted object in the opposite direction. Similarly when the proposed AR application system is creating the content for its browser, the viewport is set fixed while the augmentation object (referred as ‘model’ below) is being moved according to the corresponding pose estimation. This takes benefit from the matrix-manipulation command provided by OpenGL to directly operate the model by using the estimated pose matrix produced in the tracking stage.

Although OpenGL also allows the user to move the viewpoint by using *gluLookAt* command, it is more superfluous, since the pose matrix needs to be decomposed into three required vectors for describing where the camera centre is, which direction the camera is looking at and which direction is up (i.e. the positive Y-axis of eye space).

As described in expression (3.5), in order to perform 3D-to-2D camera projection, two matrices are needed. With the exception of the view matrix (*i.e.* camera pose matrix), which is loaded to the Model-View matrix of OpenGL to operate the pose of model from the world space to the eye space, the projection matrix is also loaded to the Projection matrix of OpenGL to determine the final view for display on the screen. The format of the projection matrix used by OpenGL is a somewhat different with that shown in (3.6). OpenGL perspective will project the model to a clip space to simulate the limited field of view of camera device, which specifies a view frustum as shown below.



The clip space of OpenGL.

The view frustum is defined by a near plane and a far plane, which is perpendicular to the negative Z-axis of the eye space. Only the vertices located on the inside of the frustum will be rendered. Otherwise anything that falls outside this range is clipped and cannot be seen. Thus the projection matrix is given by:

$$\begin{bmatrix} -\frac{2 * f_x}{width} & 0 & \frac{2 * c_x}{width} - 1 & 0 \\ 0 & \frac{2 * f_y}{height} & \frac{2 * c_y}{height} - 1 & 0 \\ 0 & 0 & \frac{z_{Near} + z_{Far}}{z_{Near} - z_{Far}} & \frac{2 * z_{Near} * z_{Far}}{z_{Near} - z_{Far}} \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad (B.2)$$

where f_x and f_y are focal length, (c_x, c_y) is the principle point, $width$ and $height$ are related to the screen size, and z_{Near} and z_{Far} are the inverted z coordinates of the near and far clipping planes. The focal length and the principle point here are identical to the intrinsic camera parameters of the input camera sensor, thus the 2D projection of the virtual object can be created as well as the input image. The final transform maps the projections from the clip space to the viewport, which defines the size and shape of the display area on the screen. This can be set up with *glViewport* function which is managed by OpenGL automatically. The basic flow of AR view display with OpenGL commands is presented below.

```
// Since the input image should be drawn as background in the viewport, the depth test need to be
disabled for the moment.

glDisable(GL_DEPTH_TEST);

// Draw the background image
.....

glEnable(GL_DEPTH_TEST);

// Load the projection matrix
```

```
glMatrixMode(GL_PROJECTION);  
  
glLoadMatrixf(projection_matrix);  
  
// Load the pose matrix (transform from the world space to the eye space)  
  
glMatrixMode(GL_MODELVIEW);  
  
glLoadIdentity();  
  
glMultMatrixd(pose_matrix);  
  
// render the 3D models  
  
.....
```

C. Conversions between rotation matrices and quaternions

Three dimensional rotations can be represented by 3x3 orthogonal matrices:

$$R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \quad (\text{C.1})$$

which are similar to the component R shown in expression (3.1). Although the form of matrix is very convenient to perform any transformation to points denoted by a column vectors in Euclidean space, the nine entries contained in R are redundant for output storage referred in Section 4.1.3. Since there are only 3 rotational DOFs, essentially three parameters are required to represent a 3D rotation. Except 3x3 matrixes, another two common ways are **euler angles** and **quaternions**. Euler angles use three angle values -- roll, pitch and yaw related to three axes to describe the rotation. There is no redundancy but it has gimbal lock problem, which loses one DOF when the axes of two are driven into a parallel configuration. By contrast, quaternion representation is based on complex numbers which is composed of one real element and three complex elements (*i.e.* four parameters in total), given in (A.2).

$$\mathbf{q} = q_w + q_x \mathbf{i} + q_y \mathbf{j} + q_z \mathbf{k} \quad (\text{C.2})$$

where \mathbf{i} , \mathbf{j} and \mathbf{k} are imaginary numbers that meet the following conditions:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \quad (\text{C.3})$$

Quaternions are less intuitive than euler angles but they avoid the problem of gimbal lock. In this research, to store the camera poses estimated in the 3D reconstruction session described in Chapter 4. The 4x4 transformation matrices are converted into 7-vectors, which are composed of translational 3-vector and 4-vector $(q_w \quad q_x \quad q_y \quad q_z)$

of rotation quaternion, and then they are written into the data files. In turn, in online session described in Chapter 5, the system will read the reference map from the data files and interpret the 7-vectors into 4x4 transformation matrices again. The conversions between the matrices and quaternions are given below.

A pure rotation matrix R should be special orthogonal, i.e. $\mathbf{det}(R) = 1$. Then the matrix R can be converted to a quaternion \mathbf{q} using this basic form:

$$\begin{aligned}
 q_w &= \frac{\sqrt{1 + r_{00} + r_{11} + r_{22}}}{2} \\
 q_x &= \frac{r_{21} - r_{12}}{4q_w} \\
 q_y &= \frac{r_{02} - r_{20}}{4q_w} \\
 q_z &= \frac{r_{10} - r_{01}}{4q_w}
 \end{aligned} \tag{C.4}$$

The equivalent matrix of a quaternion can be obtained as follow:

$$\begin{aligned}
 r_{00} &= q_w^2 + q_x^2 - q_y^2 - q_z^2 \\
 r_{01} &= 2q_xq_y - 2q_wq_z \\
 r_{02} &= 2q_xq_z + 2q_wq_y \\
 r_{10} &= 2q_xq_y + 2q_wq_z \\
 r_{11} &= q_w^2 - q_x^2 + q_y^2 - q_z^2 \\
 r_{12} &= 2q_yq_z - 2q_wq_x \\
 r_{20} &= 2q_xq_z - 2q_wq_y \\
 r_{21} &= 2q_yq_z + 2q_wq_x \\
 r_{22} &= q_w^2 - q_x^2 - q_y^2 + q_z^2
 \end{aligned} \tag{C.5}$$

D. Distance metrics

Consider two vectors $\mathbf{a} = (a_1, a_2, \dots, a_n)$ and $\mathbf{b} = (b_1, b_2, \dots, b_n)$ in an n -dimensional real vector space \mathbb{R}^n .

Euclidean distance

The Euclidean distance is the straight-line distance between two points in Euclidean space. The associated norm is called the Euclidean norm, also known as L_2 norm or L_2 distance, defined in (B.1).

$$\mathbf{d}_2(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (\text{D.1})$$

Manhattan distance

The Manhattan distance is the sum of lengths on each coordinate axis. It is also known as L_1 norm or L_1 distance, defined in (B.2).

$$\mathbf{d}_1(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_1 = \sum_{i=1}^n |a_i - b_i| \quad (\text{D.2})$$

E. Skew-symmetric matrix representation of cross product

3x3 skew symmetric matrices can be used to represent cross products as matrix multiplications. Consider vectors $\mathbf{a} = (a_1 \ a_2 \ a_3)^T$, then the cross product matrix of \mathbf{a} is defined as:

$$[\mathbf{a}]_{\times} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \quad (\text{E.1})$$

F. RGBD-SLAM output files

Camera trajectory file

Each line of the generated trajectory file contains the translation and orientation of the camera in the world reference frame at a certain time. The format of each line is **'timestamp tx ty tz qx qy qz qw'** where the [timestamp] denotes the time of Kinect data being published, the [tx ty tz] gives the estimated camera centre and the [qx qy qz qw] gives the estimated orientation in form of a unit quaternion of the optical centre of the colour camera with respect to the world. The conversion between quaternion and 3x3 rotation matrix is given in Appendix C. The translation vector t can be calculated from camera centre c by using $t = -Rc$.

XML files of graph summary and nodes

The summary file of generated graph contains:

- the number of camera nodes: how many RGBD frames have been used as nodes during the reconstruction;
- the list of node pairs restrained by edges: which node pairs have enough feature correspondences for estimating a valid transformation between them (stored as <Node_id1, Node_id2>);
- the list of keyframes and their associated nodes: which nodes are selected as representative keyframes due to the rule described in Section 4.1.2.2 and which nodes are associated (have edges) with them (stored as <Keyframe_node_id, <Associated_node_ids> >);
- the list of inlier features and their originating feature points: so-called inlier

features come from the RANSAC inlier correspondences used for estimating transformations between nodes (as mentioned in Section 4.1.2.3), which can be also interpreted as 3D space points and their originating 2D-to-2D correspondences can be considered as the projections on different images (stored as $\langle \text{Inlier_id}, \langle \text{Node_id}, \text{Feature_id} \rangle \rangle$);

Nodes are stored separately. Each node file contains: node_id , vertex_id , 2D coordinates of features, 3D coordinates of features and visual descriptors, as shown in Figure 4-8. In addition, a reverse-index from each visual feature to an inlier feature stored in the summary file is also recorded, as $\langle \text{Feature_id}, \text{Inlier_id} \rangle$.

G. VisualSfM file formats

SIFT

The default feature detector and extractor used in VisualSfM is SiftGPU. The API of SiftGPU supports to save the detected feature measurements and their 128-D SIFT descriptor vectors to a *Lowe's ASCII format SIFT files* (Wu; Lowe, 2005) for each processed image and using the extension '.sift'. VisualSfM always generates binary format SIFT files as intermediates which are not human readable but much more time and space-saving than ASCII format in file writing and reading. If the ASCII format SIFT files are imported to VisualSfM, they will be converted to binary format SIFT files automatically.

```
# .sift (Lowe's ASCII format)
<Number of features> < Dimensionality of descriptor vectors >
# <List of features>
#<Feature>
<Row> <Column> <Scale> <Orientation>
< Descriptor vector >
#</ Feature >
.....
#< Feature >.....#</ Feature >
# </List of features>
```

MAT

VisualSfM performs pair-wise matching between all input images and stores the feature

matches in binary format MAT files for each processed image and using the extension ‘.mat’. The MAT format file of each image only contain a list of the indices of its matching images, match counts and the correspondences of matched feature indices, which means it is irrelevant to what kind of features and descriptors were used for matching. It can be custom generated by using "*SfM->Pairwise Matching->Import Feature Matches*" in VisualSfM. The input txt file that contains all the feature matches should follow the format below:

match file (*.txt)
Match file = <List of Image-Match>
Image-Match = <image1_path> <image2_path> <# of matches>
<List of 0-based feature indices in image1>
<List of 0-based feature indices in image2>

NVM

VisualSfM saves intermediate parameters during the sparse reconstruction as N-View Match (NVM) files, which contain camera list, 3D point list and the associate PLY files.

The format description is given below:

.nvm
NVM_V3 # file version header
<Reconstructed model>
<Number of cameras (Integer)>
<List of cameras>

```

#<Camera>

<Image path (String)>

<Focal length (Float)>

<Rotation quaternion WXYZ (Float[4])> # or <Rotation matrix (Float[9])>

<Camera Centre (Float[3])>

<Radial distortion (Float)>

0 # 0 indicate the end

#</Camera>

.....

#<Camera>.....#</Camera>

# </List of cameras>

<Number of 3D points (Integer)>

# <List of points >

#<Point>

<3D position XYZ(Float[3])>

<Point colour RGB (Integer[3])>

<Number of 2D measurements (Integer)>

# <List of measurements>

# <Measurement>

<Image index (Integer)> # start from 0

<Feature index (Integer)> # start from 0

<2D position xy(Float[3])>

# </Measurement>

```



```

.....
# <Measurement>.....# </Measurement>

# </List of measurements>

#</ Point >

.....

#<Point>.....#</ Point >

# </List of points >

# <Empty Model>

<Number of unregistered images>

# <List of unregistered images>

<Image path (String)> <Focal length (Float)> 1 0 0 0 0 0 0 0

.....

<Image path (String)> <Focal length (Float)> 1 0 0 0 0 0 0 0

# </List of unregistered images>

# 0 camera and 0 point to indicate the end

0

0

#the last part of NVM file points to the PLY files

< Number of associated PLY files>

< Model-index that has PLY>

```