Pictorial Markers with Hidden Codes and Their Potential Applications

By

ROBERT (HUY) LE

Department of Computer Science
School of Engineering, Computer and Mathematical Sciences
AUCKLAND UNIVERSITY OF TECHNOLOGY

A dissertation submitted to the Auckland University of Technology in accordance with the requirements of the degree of MASTER OF COMPUTER AND INFORMATION SCIENCES.

MAY 22, 2018

Primary Supervisor: Dr. Minh Nguyen

Secondary Supervisors: Steffan Hooper

"Augmented Reality Will Be As Big As the iPhone."

Tim Cook, CEO Apple Inc.

Preface

First of all, I am beholden to my primary supervisor Dr. Minh Nguyen for his dedication and willingness to help me with this research and being a good friend to me throughout the whole year. I really could not finish this thesis in time without his guidance. I would also like to thank my secondary supervisor Steffan Hooper for his help.

I have been fortunate to work along with a small research group under Dr. Minh Nguyen's guidance. I would like to thank all my colleagues, especially to Huy Tran and Lei Qiu who gave me ideas, proof-read my work and offered me help whenever I need. I also would like to thank the AUT staffs and technicians who have provided such a wonderful IT services among various software which are used by me of completing this thesis.

I am also thankful to my beloved family for love and support. Especially to my parents, for their sacrifices and providing in both moral and financial support my decision to go back to study. I am grateful for having such a great family. I am so proud of all of them. Most of all, I thank my dear girlfriend - Annie Nguyen, who has supported me more than anyone. She has always encouraged and reassured me. She has taken great responsibly for housework and let me both work and rest. I thank her with all my heart.

In Auckland, May 22, 2018

TABLE OF CONTENTS

ATTESTATION OF AUTHORSHIP

I, Robert (Huy) Le, hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

SIGNED: ..... DATE: .....22/05/2018.....

List of Publications

This thesis is based on the following original publications which are referred to in the number as Paper 1 - Paper 7. The publications are reproduced with kind permission from the publishers.

1. **H. Le**, M. Nguyen. *"Enhancing Textbook Study Experiences with Pictorial Bar-codes and Augmented Reality"*. In Proceedings of 17th International Conference on Computer Analysis of Images and Patterns (CAIP), Ystad, Sweden, 22-24 August, 2017.

2. **H. Le**, M. Nguyen, H. Tran, W. Yeap. *"Pictorial AR Tag with Hidden Multi-Level Bar-Code and Its Potential Applications"*. In Multimodal Technologies Interact (ISSN 2414-4088). 2017, 1(3), 20; doi:10.3390/mti1030020.

3. M. Nguyen, H. Tran, **H. Le**, W. Yeap. *"Adaptive Stereo Vision System using Portable Low-cost 3D Mini Camera Lens"*. In In IEEE International Conference of Mechatronics Machine Vision Practice (M2VIP), Auckland, New Zealand, 21-23 November, 2017.

4. M. Nguyen, H. Tran, **H. Le**, W. Q. Yan. *"A Tile Based Colour Picture with Hidden QR Code for Augmented Reality and Beyond"*. In the 23rd ACM Symposium on Virtual Reality Software and Technology (VRST), Gothenburg, Sweden, 8-10 November, 2017.

5. M. Nguyen, H. Tran, **H. Le**. *"Exploration of the 3D World on the Internet Using Commodity Virtual Reality Devices"*. In Multimodal Technologies Interact (ISSN 2414-4088). 2017, 1(3), 15; doi:10.3390/mti1030015.

6. M. Nguyen, H. Tran, **H. Le**, W. Q. Yan. *"A Personalised Stereoscopic 3D Gallery with Virtual Reality Technology on Smartphone"*. In Int. Conf. Image Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4 Dec - 6 Dec, 2017.

7. L. Qiu, M. Nguyen, H. Tran, **H. Le**, W. Q. Yan. *"Digital Map using Augmented Reality on Smart Devices: Motivation, Design, and Implementation"*. In Int. Conf. Image Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4 Dec - 6 Dec, 2017.

List of Tables

List of Figures

Figure **Page**

ABSTRACT

Background and Objective: most of the today's Augmented Reality (AR) applications would use either black and white data markers or pictorial (template) markers to allocate, identify, and render the virtual information on the real scene. The data markers, in general, do not deliver meaningful visual information to users; whereas, the template markers suffer from the system processing complexity. However, they both survive until today as each of these markers has many uniquely worthy points which could resolve other's drawbacks.

Methods: in this thesis, we propose several different novel concept designs of AR marker which combine the technical advantages of both data markers and template markers. These markers are capable of embedding either a one-dimensional bar-code or a two-dimensional quick response (QR) code in a coloured figure to enhance AR experiences. They are not only aiming to improve the system performance but also present the useful and meaningful realistic-looking graphical content to the users. Another advantage of our proposed markers is that they could provide self-error detection and correction which could help to recover the lost information. In short, these following markers will be introduced within this thesis:

- *Pictorial Marker with Hidden Bar-code (PMBC)* is capable of hiding a single one-dimensional bar-code in a graphical content using autostereogram theory.
- *Curtain Styled Pictorial Marker (CSPM)* is capable of embedding the two dimensional QR code in a coloured figure and also provides a correct orientation of virtual objects.
- *Envelope Styled Pictorial Marker (ESPM)* is an improved method of CSPM which could provide wider graphical content visualisation and more accurate self-error detection and correction.

Results: the several different experimental exercises have been conducted to qualify our proposed methods on technical performances. We have also presented few proposed marker physical prototypes and produced several working demos. Moreover, our proposed markers have an equivalent system processing performance to it of data markers whereas the visual information remains unchanged.

CHAPTER 1

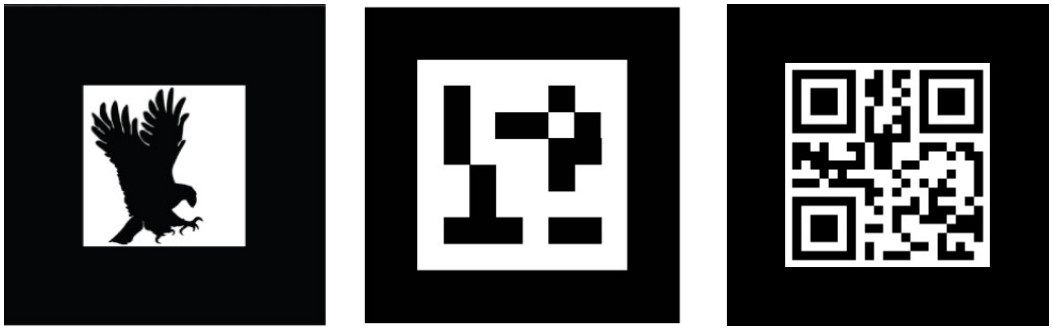
INTRODUCTION

*Parts of this chapter has been published in **paper 1** and **paper 2** listed in publication list*

For decades, researchers have been trying to create natural virtual environments by merging the computer-generated information with the real world. The result is "Augmented Reality" (AR) whereby the virtual information can be easily rendered upon the real environment in real time. Since the first working prototype was introduced in 1960s [1] by Ivan Sutherland and his colleagues, AR has been widely used in many life sectors, such as medical, manufacturing, entertainment, military [2]. The primary process principle of AR applications remains the same which is the identification of detectable targets (could be either data marker or template marker) and virtual information rendering and orientation [3]. There exist many different tools such as tracking, registration and rendering [4], which provide an efficient AR experience. These tools could be implemented quickly with today's powerful technology hardware and well-supported software platforms. However, the major long-term drawbacks are still existed, such as system processing complexity and information presentation capability.

1.1 Current Issues and Motivation

The template markers as shown in Figure (1.1a) are frequently used as they are convenient for detecting and displaying content. They also present some meaningful information (e.g. the flying eager). However, to be useful, it requires an image registration process which could cause a significant increase of database storage when the number of images rises. The recognition process



(a) Template Marker (b) Bar-code Marker (c) QR Marker

Figure 1.1: Template markers (left figure) are often used as they present meaningful information. However, system processing complexity is the primary issue which is holding back their usages in robust and unambiguous applications. On the other hand, data markers (middle and right figure) provide efficient system performance but it presents almost no meaningful visual information.

is also unreliable sometimes due to the undesired similarity between registered images [5]. One major technical issue of using template markers is system performance. Figure 1.2 shows that the performance of data marker (QR code marker) continually remains whereas it gives an exponential growth for template marker when the number of image registration is doubled each time. The reason for this difference is that template matching [6] is a commonly used technique for the identification step. The template matching method compares the marker against with each image in the dataset to find the similarity. The similarity can be calculated based on either sum of squared differences (SSD) or cross-correlation [7].

$$D = \sum d(x_i, y_i)^2 \tag{1.1}$$

The dissimilarity D between template marker and data set image can be found by using SSD formula as shown in the equation (1.1), where i goes through all of pixel locations, x_i and y_i are the values of template marker and data set image at i^{th} pixel accordingly and d is difference between these values. In cross-correlation, the dissimilarity is

$$D = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2} \sqrt{\sum_i (y_i - \overline{y})^2}} \tag{1.2}$$

Where $\overline{x}$ and $\overline{y}$ are the pixel means of template markers and dataset images, respectively. The dissimilarity is then calculated by using either one of the above formula. The data set image which gives the lowest dissimilarity or highest similarity will be selected as the identity of the

template marker. However, the processing time would be increased while the database size is getting larger. Thus, the usability of template markers is mostly considered in less complex AR applications with the small-scale of dataset capacity [8].

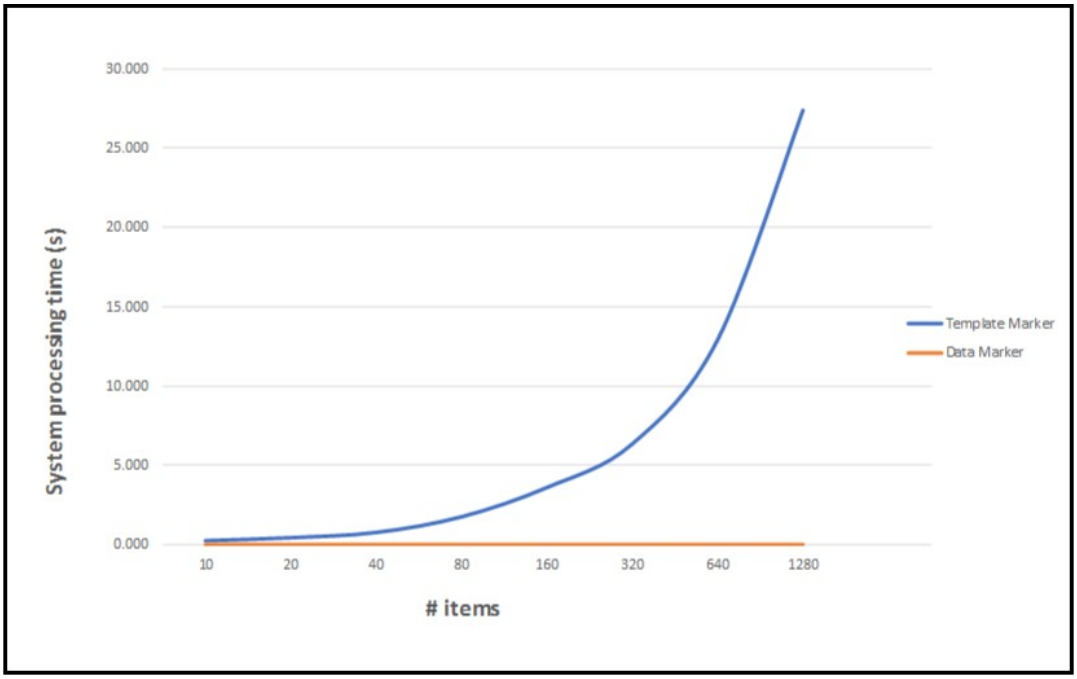


Figure 1.2: Comparison of system processing performance between data marker and template marker.

The other most used AR marker is data marker as shown in Figure (1.1b) and Figure (1.1c). These markers normally consist of black and white data cells surrounding with a thick dark-coloured border. These data cells can be decrypted to a unique binary number which was stored in the application database. Thus, the undesired similarity issue is impossible to exist, and it also helps to improve the system processing performance. As stated above, the template marker's identification technique would consume much larger processing power and time due to the large data set and more complex identification algorithm are required. This makes data marker as a winner in the system processing performance competition as the indexing [9] is the main used technique for the marker identification. Another advantage of using data marker is error detection and correction capability. Hamming codes [10] and Reed–Solomon [11] codes are two methods that are often used for error detection and correction. However, the visual presentation capability is not the strong point of data marker as these data cells provide meaningless information.

Both template and data makers have their robust points and drawbacks as summarised in Table 1.1. The template marker presents useful information, but it faces the problem of system processing complexity and reliability. The data marker on another hand could overcome the

template marker's problem and exhibit the capability of error detection and correction; however, it presents meaningless visual information. Therefore, the new marker design which has the advantageous combination of template and data markers would be an efficient approach.

	Template makers	**Data markers**
Advantages	• Present useful information to users	• Contain unique hidden code • Less detection time • Consume less database storage • Error detection and correction available
Disadvantages	• Consume more database storage • Undesired similarity between images • Longer detection time	• Present non-useful information to users

Table 1.1: Template markers vs data markers in AR applications

1.2 Objectives and Scope

The objective of this work is **to enhance the AR experiences by introducing the new marker design concepts which combine the technical advantages of both data markers and template markers**. They, thus, need to provide not only a useful graphical content presentation, but also the capability to improve system performance and security issue. In short, the proposed markers should be able to present the following novel contributions:

- The capability of hiding numerical codes in the image without significant damaging of the graphical content.
- The capability of detecting/restoring lost information.
- Good system processing performance but it still provides enough visual information to the users.
- An evaluation of the proposed marker based on various experimental exercises including the comparison with template and data marker.
- A superior AR marker that could be used in both research and commercial applications.

In order to accomplish these above objectives, we divide the research problems into the following research questions (RQs):

RQ 1: How to hide different bar-codes and QR code (2D) in template images?

RQ 2: Will the newly proposed marker concepts have the equivalent technical performance to the bar-code marker?

RQ 3: What are the future potential application domains?

1.3 Thesis structure

The rest of this thesis is structured as follows:

A conceptual background review of augmented reality is given in **Chapter 2**: the history, how does augmented reality differ from virtual reality, application domains, some predictions for the future growth. Along with providing few theory previous works, we also present some examples of the augmented reality real-world applications.

Chapter 3 then briefly introduces the image processing techniques and the mathematical framework used throughout the rest of this thesis. These are including basic image segmentation, the notation which is used for coordinate systems, and transformation motions.

Chapter 4 describes the chosen methodology of this thesis which includes: research framework, data collection techniques, and support tools.

Chapter 5 introduces the concept design of pictorial marker with hidden bar-code (PMBC). It briefly describes how to conceal the one-dimensional bar-code into a normal image texture based on the ideas of autostereogram.

Chapter 6 takes us a step further than **Chapter 5** where it shows us how to hide the two-dimensional bar-code (QR code) in a normal image texture in two different ways.

Chapter 7 evaluates the results of several conducted experimental exercises aiming to determine the proposed markers performance applied to various physical prototypes. Furthermore, we present few application demos running on the personal computer platform.

Chapter 8 concludes the body of the thesis with a summary of the contributions made, limitations

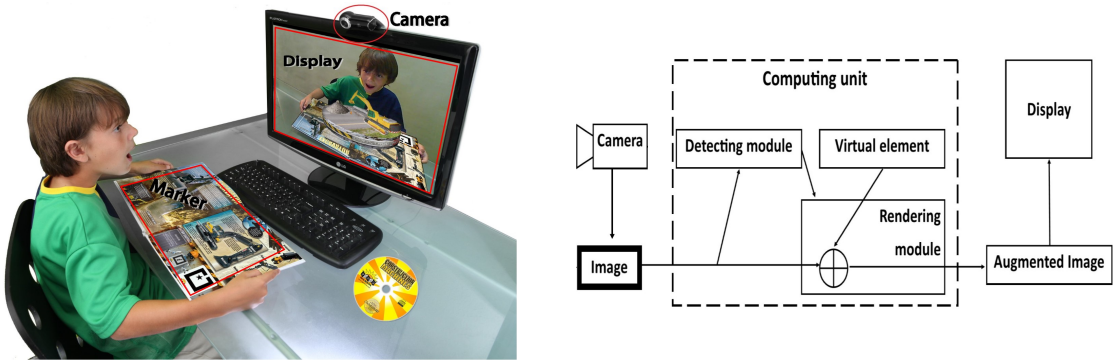
of this thesis and discusses few future plans and improvements. Furthermore, we also provide several appendices describe some of our technical tools, software source codes and application demos used in this thesis

CHAPTER 2

BACKGROUND: THE RISE OF AUGMENTED REALITY

*Parts of this chapter have been published in **paper 1**, **paper 2**, **paper 4**, and **paper 7** listed in publication list*

Augmented Reality (AR) is a reasonably recent, and very fascinating field. AR seems like something straight out of the scientific movies, but it is here today and easily accessible by most of the current information devices. However, the market share of AR is not very large, and most of its applications are just out of prototyping or under development. There are still many challenges and difficulties which have been stopping AR from setting its feet in the higher level. However, it does not mean impossible for us as the fast growing of Information Technology (IT) industry is giving us the opportunities to explore this potential sector. This chapter describes the meaning of AR such as its definition, history, gives examples of the real-world applications of AR (along with images where possible).



(a) Augmented Reality application

(b) Augmented Reality system work-flow

Figure 2.1: Template markers

2.1 What is Augmented Reality?

Augmented Reality (AR) is a field of computer science, which presents a special form of human-computer interaction. The main concept of AR is to replace parts of the reality with additional informations in the real-time [12]. These informations could be computer - generated which would not be perceived directly by human senses. It means that AR aims to keep the users in the real world but to augment their experience with the virtual elements. AR is defined as any system which has the following three characteristics [2]:

- Combines real world and virtual world.
- Is interactive in real time.
- Is registered in three dimensions.

In addition, the Reality-Virtuality continuum which was defined by Paul Milgram [13] shows that AR is lying between real environment and virtual environment as shown in Figure 2.2 . The real world and virtual environment are separated at the two ends of this continuum with the middle part called Mixed Reality (MR). The MR region is made up of two smaller regions which are AR and Augmented Virtuality (AV). AR is staying closer to the region of the real environment at the end of the spectrum. It means that real world is still AR's key perception but augmented by extra computer-generated information.

A basic AR system should consist four main components as shown in Figure 2.1b: (1) a workable camera, (2) computational unit, (3) display component and (4) a detectable marker. The system captures the image via the camera then passes captured image to the processing unit. The processing unit will identify the marker information in the captured image. It then deduces the orientation and the location of the camera, and augments the virtual object on the top of the

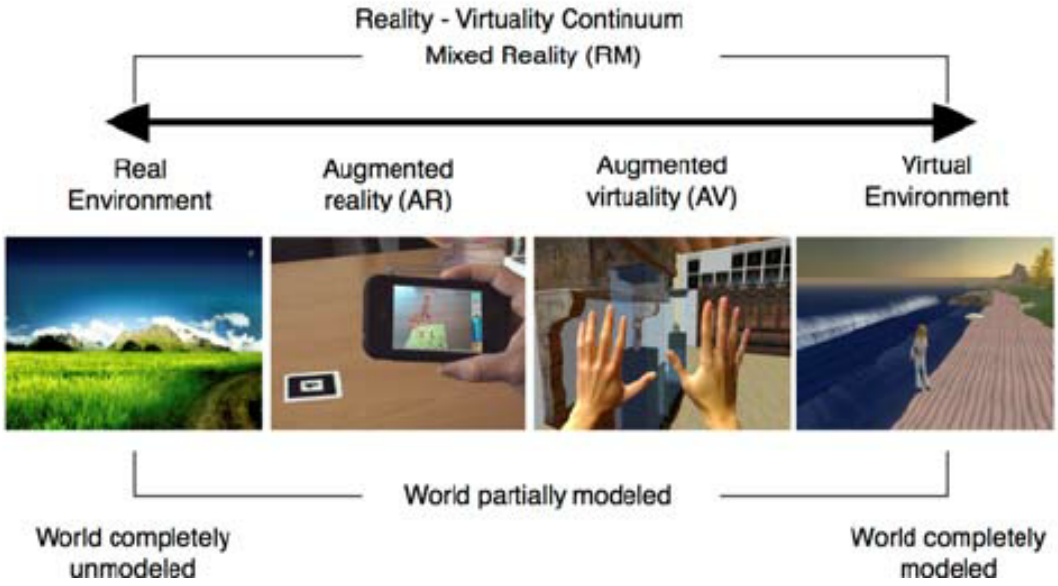


Figure 2.2: Virtuality Continuum was defined by Paul Milgram in 1994 [14].

captured image and displays it on the screen. The AR applications could be design to run on various devices such as PC, mobile or other computation units depending on the usage purposes. Figure 2.1a illustrates an example of a basic AR application which is running on PC platform. The real environment which is the magazine as seen by the web camera and is augmented and displayed on computer screen. The AR system could also access other camera types such as the built-in mobile camera or digital camera if the system were design to run on Android or IOS.

2.2 History and Bright Future

As mentioned above, AR has such a long history when the first computer-based head-mounted display (HMD) was constructed by Ivan Sutherland in 1960s [1] as shown in Figure 2.3. The HMD allows the users to see the 3D graphics through its display optics. At the same time, Bell Helicopter embedded a system on Bell UH-1 combat version that could augment the vision of helicopter pilots via their HMD. It helps the pilots in night guided landing and combat by operating of several different infrared cameras due to the conflict in Vietnam reached its peak. While the virtual reality (VR) became more popular over the years, the technical limitations were still not allowed to distribute the use of AR in a larger area. AR became forgotten while VR had been invested in both financial fund and science research during the 1970s and 1980s by government aids.

Since the early 1990s, AR started to be considered and ready to reappear again due to the fast growth of the advanced technology [15, 16]. Boeing research group described the word of "Augmented Reality" in their research on mounting cables in aeroplanes in 1993 [17]. In the late

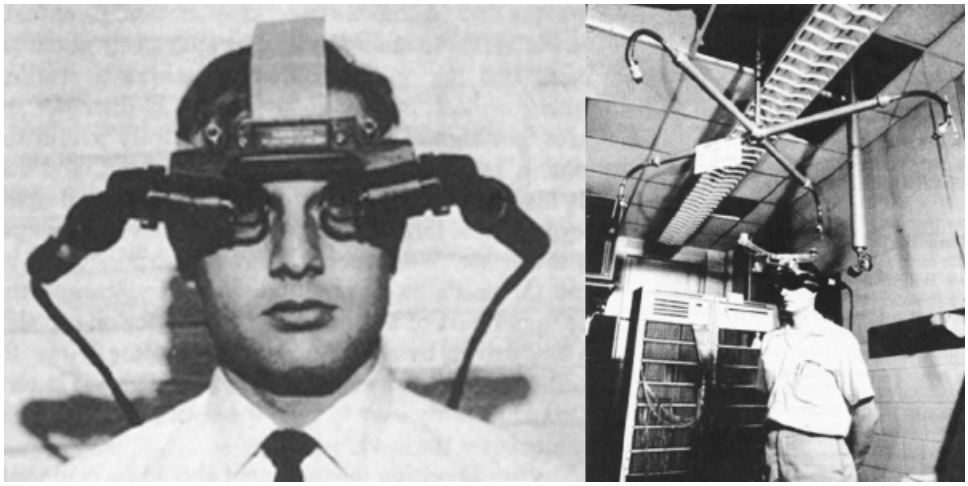


Figure 2.3: The world's first head mounted display

1990s, Columbia University Computer Science research group presented the wearable computer system that can support users in their everyday interactions with the world [18]. Since the expansion of smart-phone market shares in the early 2000s, AR research and development are getting higher public attention to push the custom hardware, and commercial products begin to appear. An annual report from Goldman Sachs [19] has shown that there were over 3 billion dollars which were spent on the AR technology investment and researches for the last two years. The report also predicts that the AR revenue would contribute 80 billion dollars to global GDP in 2025 due to the market's demand. There are already many famous technology companies such as Google, Microsoft, Samsung, Facebook been involved in the competition. Other smaller companies and organisations around the world also work on their own equipment pieces. However, AR is still at the dawn of its age and there are many other potential opportunities are waiting for being discovered.

Information technology related research and advisory company **Gartner, Inc** maintains Gartner hype cycle for the emerging technologies and the future trends. The hype cycle is a graphical chart of the technology life cycle stages. It predicts how the individual technology should be going through from conception to maturity and widely accepted by population [20]. According to Gartner, the hype cycles identifies five overlapping stages:

- **Innovation Trigger:** the technology now is in early proof-of-concept stories with few possible prototypes but no usable existed products. However, there are usually giving the potential spurs on the media interest trigger significant publicity.
- **Peak of Inflated Expectations:** the technology is now catching much attention from the public with few product designs have been implemented successful and unsuccessful.

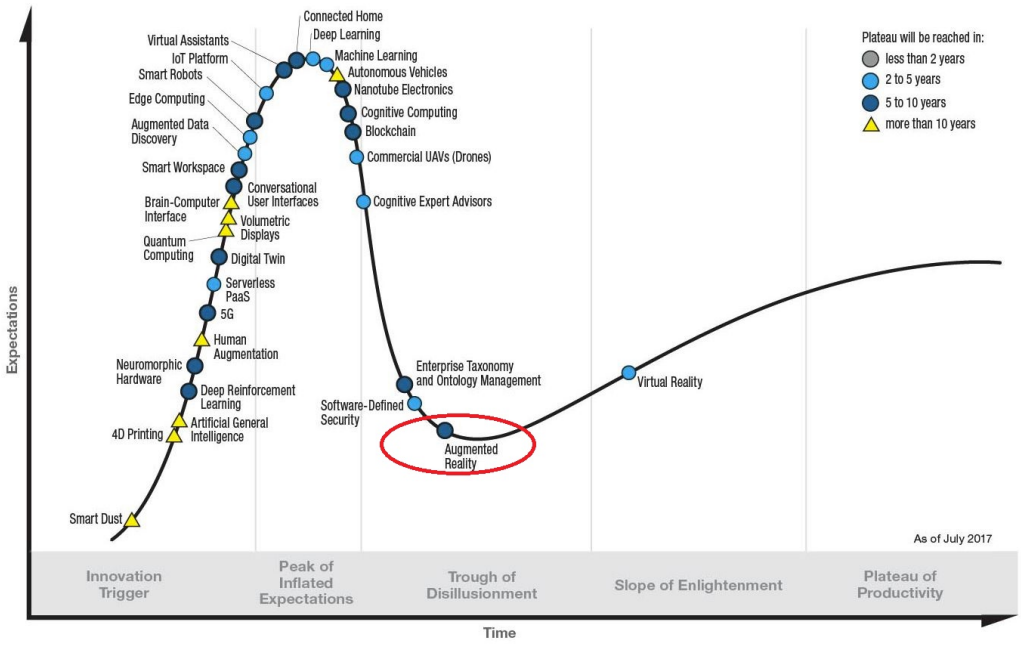


Figure 2.4: Gartner Hype Cycle for Emerging Technologies by July 2017

- **Trough of Disillusionment:** the product implementation failures lead to some public interest wanes. The investments only continue if the technology providers improve the products and it does not mean that the technology is completely crossed out.
- **Slope of Enlightenment:** the technology is now becoming widely understood and can be a benefit to the public in both research and commercial. More investments and producers start to implement and test the products in their own environments.
- **Plateau of Productivity:** the technology becomes widely implemented as many more new products are introduced in the market.

In the Gartner's hype cycle for emerging technologies in July 2017, augmented reality (AR) is at the trough of disillusionment (Figure 2.4). The hype cycle predicts that it would take 5-10 years to reach the plateau of productivity. The AR technology is now falling to the stage where more product implementations and improvements are necessary.

Google is one of the first companies which had entered in the AR race since 2012 when Google glass (Figure 2.5a) was introduced to the public [21]. Google glass acts like a mini HMD which allows the users to experience the AR through the optics on the glass. In early 2016, Microsoft also announced their own AR device called Hologens (Figure 2.5b). It was introduced as a revolutionary connection way in the future as it gives the users more capability to present but interact with the virtual environment [22]. Since the world has been shipping from the age of

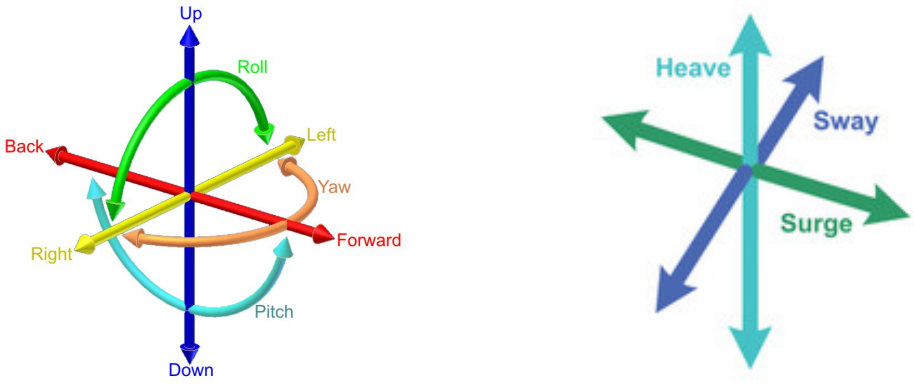
(a) Google glass (b) Microsoft Hololens (c) Apple Iphone X

Figure 2.5: Augmented Reality devices

computer to the age of tablets and mobile devices, the chance for AR technology expanding is more visible than ever before. Most of the mobile devices are now made with the aim of AR usage [23] which means more hardware and software will be developed for AR. In late 2017, Apple introduced the new iPhone X model (Figure 2.5c) which powered by the new A11 bionic 6-core processing chip [24] which helps to boost for AR applications and development.

2.3 How does Augmented Reality differ from Virtual Reality?

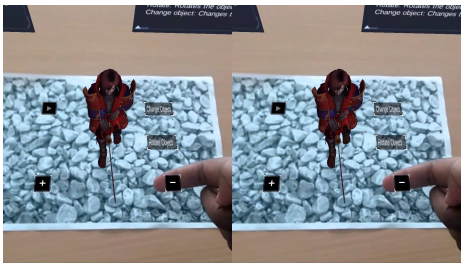
Comparing between AR and VR is like comparing between apple and orange as they are not the same thing and they are designed for different usage purpose although that they share in common is the term “reality”. VR takes users totally out of actual reality, whereas AR enhances the actual reality experiences. VR will completely replace the user’s surrounding environment with the computer-generated world, typically with three-degrees of freedom (3DOF). AR only overlays some additional visual information partially on the real world with six degrees of freedom (6DOF) [25]. 6DOF is the ability to move along three different perpendicular axes (x,y,z) for positions combined with changes in orientation through rotation such as pitch, yaw, and roll (Figure 2.6a) while 3DOF only offers positions detection (Figure 2.6b). Most of the modern technologies usually come with a variety of sensors that automate or easily detect all of those motions [26].



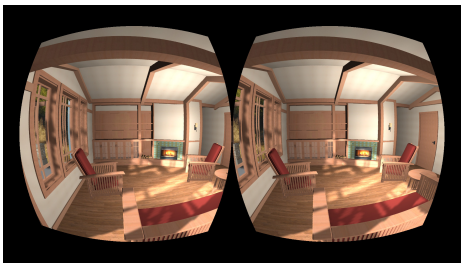
(a) Six degrees of freedom (6DOF) (b) Three degrees of freedom (3DOF)

Figure 2.6: The freedom of movement in Augmented Reality and Virtual Reality

From the presentation perspective, AR is functionally similar to VR, with the key difference being a transparent screen to display both actual views and rendered overlay objects. It means that they might share the same underlying hardware, but present distinctly different experiences to the users. However, AR requires the additional requirements in tracking for the orientation of the virtual objects. Therefore, 6DOF is taken placed to support this extra function. In AR, the system uses location, motion, and orientation sensors to determine the position and orientation of a camera (Figure 2.7a). It then renders the virtual objects as they would appear from the viewpoint of the camera. This step can be easily processed by a mobile, helmet, tablet, personal computer or glasses. On another hand, VR only operates along with a head-mounted display as the presentation is now shifting to the virtual world (Figure 2.7b).



(a) Augmented Reality application



(b) Virtual Reality application

Figure 2.7: Presentations in Augmented Reality and Virtual Reality

2.4 Augmented Reality Application Domains

Since the information revolution started in the 1990s, AR has been exploited in several different areas of our life. With the advantage of new technologies, AR may just be the thing to change our daily lives. Probably, it is the next step in the way we collect, process and interacts with the information [27]. The AR technology had many applications in different fields and kept expanding over time.

2.4.1 Entertainment

When we watch the weather forecast, we usually see the reporters are standing in the front of changing weather maps. In reality, they are always standing in the front of the green screen, and those information are the computer - generated (also called chroma-keying technique [28]. This method helps to reduce the setting up a time and also provides a better way to control the environment as shown in Figure 2.8a. It can be extended to display not only 2D images but also the 3D objects as shown in Figure 2.8b. The Archeoguide project [29] is an augmented archaeological sites tour which allows the tourists to see the full reconstructed historical buildings in 3D

surrounding by the actual real site. Moreover, this application enhances people's imagination about the lost historical sites. They could use their perceptions to experience and feel how the ancient buildings were like in the past. In addition to those applications, AR has also been used for gaming purposes. Figure 2.8c shows the game setup of ARQuake which is an indoor and outdoor AR first-person game [30]. It allows the players to shoot down virtual demons while they are moving through the real world by using a large HMD to interact with virtual objects. However, it is not a brilliant idea when people are focusing on handy and light accessories nowadays. Pokémon GO as shown in Figure 2.8d is an indicative AR based game which addicts many people around the world [31]. Players can use their mobile devices to catch virtual Pokémon characters in the real world and interact with other users via Wi-Fi.

(a) Weather forecast studio set up

(b) Archeoguide project

(c) ARQuake game set-up [32]

(d) Pokémon GO game

Figure 2.8: AR games

2.4.2 Education

The possibilities for teaching and learning provided by AR technology have been increasing. AR recently emerged in the field of education supporting different subject areas such as mathematics (Figure 2.9a), chemistry (Figure 2.9b), physics, etc. The orientation of the virtual objects in the real environment allows students to visualise and understand the complex abstract concepts and spatial relationships [33]. A student can easily experience with phenomena which are not possible to do in the reality [34]. It means that students will have a higher motivated attitude in class to be able to improve their learning effort. In 2001, Mark Billinghurst from Hiroshima City

University developed the Magic Book using AR technology to make reading more captivating [8]. This application used marker based technique to view the augmented virtual object when readers point their "Opera glass" to the markers.

Rainer Malaka and his teammates [35, 36] had built a mobile outdoor AR application which provides learners with another way to learn history through a story-telling game. This application identifies user's current historic location and searches through its database to get the information about the current place. The virtual spirits of the past of the current place appear and tell the users the information about themselves and the place. The idea of this application is giving the learners an opportunity to learn history through live stories telling by a historical virtual person. The outcome shows that the attractiveness of the learners surely increased compared to the traditional way of learning. However, it is not convenient for students to use this system due to the limitation of time which does not usually allow them to be at the historic locations. History study is culturally constructed subject, and most of its facts do not exist anymore. Thus it is not easy to imagine without the help of visual examples.

(a) Augmented Reality Mathematics

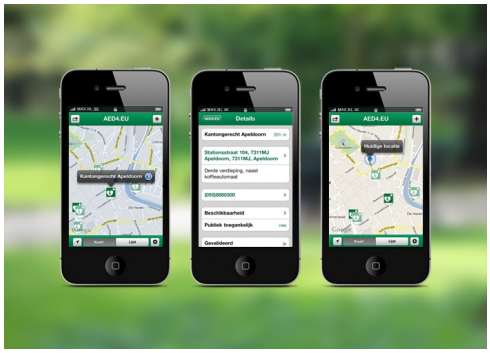
(b) Augmented Reality Chemistry

Figure 2.9: AR education apps

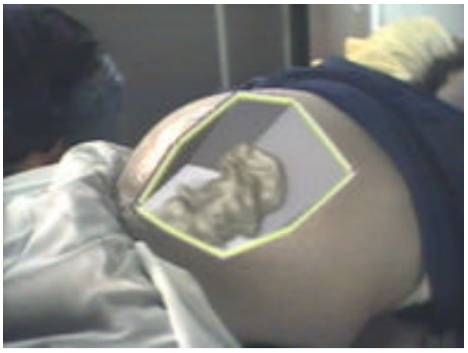
2.4.3 Medical and Health Care

Medical is one of the most common financially invested domains according to the national budget for the medical expenditure report which was described by Amitabh Chandra and Jonathan [37]. It stated that the medical expenditure budget has been increasing since the 1980s across Europe, North America and Oceania. Along with the fast growth of information technology industry, it is not too hard for many applications which could be used to benefit the medical-related purposes.

The suddenly collapsed patients can be saved more quickly by AED4EU mobile application [38] which was designed by Lucien Engelen from University Nijmegen Medical Centre, The Netherlands. The AED4EU application (Figure 2.10a) shows all known automated external defibrillators (AED's) which are currently located near the patients. It can give the patients the actual AED locations, phone number and the address if possible. They then can easily project



(a) AED4EU mobile application



(b) Augmented Reality ultrasound scanner

Figure 2.10: AR medical apps

and contact with the nearest AED to seek for help in case of emergency. Another application for AR in the medical domain is in ultrasound imaging [39] that allows the physician to see directly inside a patient. This system was designed to view the fetus of the pregnant woman's abdomen in the first stage as shown in Figure 2.10b. The system will generate 3D fetus within a pregnant woman's abdomen while the physician moves the scanner around the abdomen. The doctor can also apply this method to the needle guided biopsies where they can view the patient's organ without the need of injection deeply into the skin. The anatomy surgery will become easier and safer where the surgical team has the ability to view the imaging data in real time while they are processing the procedure [40] [41].

2.4.4 Military

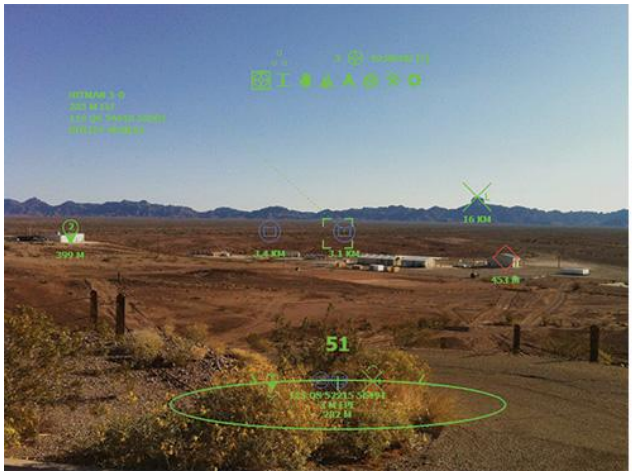
In 1955, the first mockup of the head-up-display (HUD) concept unit along with a sidestick controller was proposed by US Navy's Office [42]. It is usually set up with three primary components: (1) projector unit, (2) combiner, and (3) a computer-generated video display [43]. The fighter pilots do not need to refocus to view the outside after looking at the optically nearer instruments (Figure 2.11a). In mid of the 1980s, the U.S air force introduced the integrated helmet and display sighting system (IHADSS) to be used by the AH-64 Apache pilots [44]. The chopper pilot has a wider field of view with the symbology monocular display located right in the front of his eyes (Figure 2.11b). It also provides the slewable IR imaging sensor which will be slaved along with the pilot's head movements. The pilot only needs to look at the target that he is trying to engage instead of turning the helicopter forwards to it. It helps to increase the pilot survival chance and combat performance. Although HUD was initially designed for military purpose, it is now widely used in commercial aircraft and other civilian applications.

In the ground combat, AR technology can use as a networked communication system which presents the battlefield information onto soldiers goggles in the real time [45]. The commanders

can send the map and important information directly to the view of the soldiers without time delay. The soldiers now do not need to look down to the 2D map or chest-worn computer; they can see the combat objects such as navigation waypoints, friendly or enemy forces as marked overlaid virtual icons on their real-world views. The system can also track exactly which directions and coordinates the soldier is looking at and update the overlaid information accordingly (Figure 2.11c).

(a) HUD of an aircraft fighter

(b) IHADSS



(c) ARC4

Figure 2.11: AR military apps

2.5 Augmented Reality Markers

As described in Section 2.1, the detectable marker is one of the key components that augmented reality (AR) system normally requires. There are many different marker types which are used within AR applications depending on usage purposes and supported software platforms. However, they should be falling in either one of two main categories: (1) template markers and (2) data marker. Each of these marker categories provides different methods to encrypt/ decrypt hidden

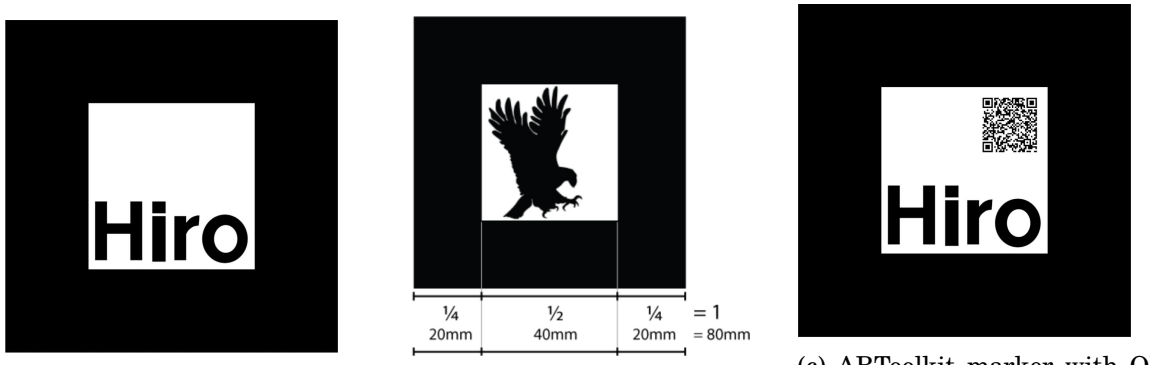
information and has unique advantages and drawbacks. Furthermore, in this section, we discuss few popular marker types which are used within AR application and their characteristics.

2.5.1 ARToolkit Template Marker

ARToolkit marker is the default optical input to the open-source computer tracking library - ARToolkit (Figure 2.12a) which was created by Hirokazu Kato in 1999 [46]. This marker is simply a graphic image content, but it must have the following characteristics:

- The marker must be a square shape.
- The marker border must be continuous in either black or white.
- The border thickness must be 25% of the length of an edge of the marker by default.
- The inner graphical content must be 50% of the marker area, and it could be either black, white or coloured as shown in Figure 2.12b.

The quick response (QR) code can be used within the ARToolkit marker to improve the accuracy and detectability as shown in Figure 2.12c.



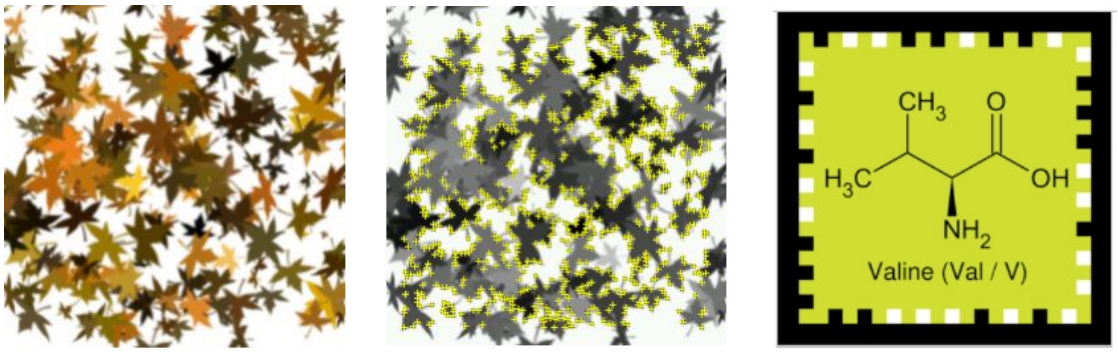
(a) ARToolkit marker (b) ARToolkit marker dimension (c) ARToolkit marker with QR code

Figure 2.12: ARToolkit template marker examples

2.5.2 Vuforia Marker

Vuforia marker is used within one of the most well-known AR development platforms - Vuforia [47]. The Vuforia platform is the key product of Qualcomm Technologies which uses the computer vision-based image recognition techniques to provide the features and capabilities to develop AR applications for Android, iOS and Unity platforms. The marker can be created in two different ways: (1) natural image target, and (2) frame marker. The natural image target (Figure 2.13a) does not need to be black and white regions or recognisable codes and can be any type of coloured figures such as photos, book covers, posters, etc. Each naturally features of the

registered target image will be marked as the unique identity of the image itself (Figure 2.13b). The system will use sophisticated algorithms in order to detect the those stored features of the target image to retrieve its identity. In order to increase the detectability and accuracy of the



(a) Natural image target (b) Image target features (c) Frame marker

Figure 2.13: Vuforia marker examples

marker, the registered image targets are recommended to have the following features:

- The marker should be rich in detail.
- The marker should have a good contrast graphical content.
- The marker should not have repetitive patterns (e.g. checkerboard, identical windows).
- The marker must be either 8- or 24-bit JPG, and PNG formats with the size is not larger than 2 MB.

However in practice, we cannot control the graphical content features, and there are still few drawbacks existed within image processing domain such as undesired similarity and system complexity processing. Thus, Vuforia provides the frame marker (Figure 2.13c) as an acceptable solution to fill in this missing gap. The image frame includes the unique ID which is presented as the encoded binary pattern along the border and independent graphical content. The system then can just use simple data decoding technique to identify the unique marker ID. However, there is the limitation of 512 detectable frame marker for the users to choose from. Therefore, the frame marker is often considered in AR application with small-scale of image capacity.

2.5.3 Data Marker

The data marker is the most used marker in AR application and normally consisting of black and white data cells surrounding with a thick border. A simple binary marker which contains the encoded binary string represented as the black and white squares (Figure 2.14a). This decoded binary string is treated as the unique marker identity (ID). The more encoded data, the larger marker is needed. The advantage of this marker type is less consumed processing time needed as

no image is required to store in the dataset. Another strong point is that most data markers have built-in error detection and correction which will be described in details in Section 3.2.

(a) Binary marker (b) QR code marker (c) DataMatrix marker

(d) PDF147 bar-code

(e) Customized QR code (f) Facebook Messenger code (g) Snapchat code

Figure 2.14: Data marker examples

In order to increase the data storage capacity, the 2D bar-code markers are also considered to be used within the AR application. 2D bar-codes such as: Quick Response (QR) code [48, 49] (Figure 2.14b), Data Matrix [50] (Figure 2.14c), or PDF147 [51] (Figure 2.14d) were initially developed for logistics and traffic control purposes. The DataMatrix can hold up to 3,116 encoded ASCII characters (with extensions). The marker is usually presented as a squared shape with an even number of rows and an even number of columns from 10×10 to 144×144. It uses Reed-Solomon codes for the error detection and recovery with the capability to sustain 30%

damage.

The PDF417 bar-code was invented by Dr. Ynjiun Wang at Symbol Technologies in 1991. The PDF417 is a stacked linear bar-code symbol format widely used in identification cards, transportation system, and inventory management. The PDF stands for Portable Data File. The 417 means that each pattern consists 4 bars and spaces, and that each pattern is 17 units long. In theory, a single PDF bar-code can hold up to 1850 alphanumeric characters, or 1108 bytes of data. The QR code is another famous 2D bar-code invented by the Japanese corporation Denso-Wave in 1994. It can hold different types of characters such as: number, alphabet letters, binary data and Kanji characters which is one of key character types of Japanese writing system. A single QR code can hold up to 7089 alphanumeric characters, 2953 bytes of binary data or 1817 Kanji characters. The QR code can be easily customized for advertisement purposes (Figure 2.14e) or extended to be used as scanable profile ID within the social media and networking such as: Facebook (Figure 2.14f) or SnapChat (Figure 2.14g).

CHAPTER 3

BACKGROUNDS: COMPUTER VISION AND IMAGE PROCESSING TECHNIQUES

*Parts of this chapter has been published in **paper 1** and **paper 2** listed in publication list*

Computer Vision has great potential for Augmented Reality (AR) applications. The computer vision techniques are involving in different processes of AR system such as: target recognition, virtual information presentation and rendering. They normally do not require engineering the real world environment and can be applied easily with any type of video cameras. In this chapter, we will discuss about few computer vision techniques and supported tools which will be employed throughout the remainder of this thesis.

3.1 Stereogram and its Unique Properties

Autostereograms or single image stereograms were invented by Sir David Brewster in 1840s as an improvement over the kaleidoscope, the apparatus which allowed representing 3D shapes [52]. Modern day's stereograms for computers were first introduced by *Tyler et al* in 1990 [53]. Tyler managed to demonstrate the ability to generate depth illusions by using image patterns consisting of a single image random dot stereogram. The stereogram image contains an unlimited range of depth and can create 3D in-depth both above and behind the image plane. Stereograms quickly gained interest from the public in 1990s through the patented application called *"Magic Eye"* [54]. Applications of stereograms range from public-driven media support (3D books or videos [55]) to watermarking [56], vision therapy [57], and crystallography [58]. Many 3D design tools offer to generate stereogram stills or videos from depth map and texture map of a scene or to use a 3D

object as a start up point.

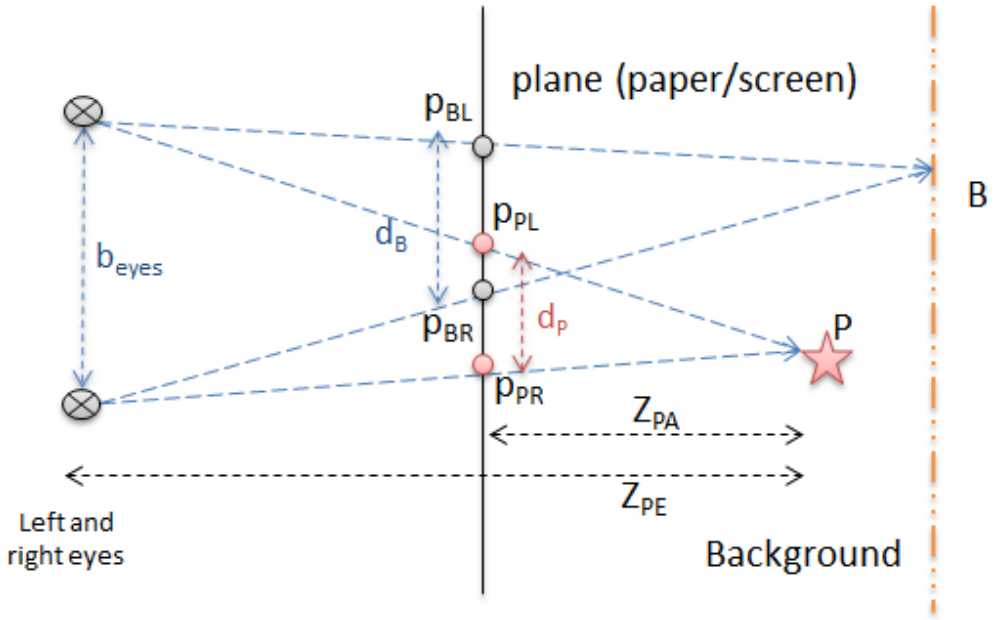


Figure 3.1: Principle behind perceiving a stereogram and how to view it

Figure 3.1 demonstrates the background theory of stereogram. There are two points: B is a point lying on the background surface, and P is the point lying in front of background. Human eyes have a base distance b_{eyes}. To correctly view a stereogram, the two eyes must not focus on the picture surface, but the point behind it and 3D points will appear in our brain if the eyes determine two projected correspondence points which may look similar in shape or color. For instance, points p_{BL} and p_{BR} help to determine position of B, points p_{PL} and p_{PR} help to determine position of P in space. Theoretically, the maximum distance $d_P = p_{PR} - p_{PL}$ between two correspondence points p_L, p_R on stereogram cannot be greater than b_{eyes}: $b_{\text{eyes}} \leqslant d_{\text{P}}$; at $b_{\text{eyes}} = b_{\text{P}}$, point P is at infinity. From the figure, we have:

$$\frac{Z_{\text{PA}}}{Z_{\text{PE}}} = \frac{d_{\text{P}}}{b_{\text{eyes}}} => d_{\text{P}} = \frac{Z_{\text{PA}} \times b_{\text{eyes}}}{Z_{\text{PE}}} \tag{3.1}$$

The distance d_P is different from disparity $d^*{}_P$ in a stereo vision where the smaller disparity, the further the point; but there is a relationship. How do they relate? In stereo vision, we have:

Where $d^*{}_P = p_{PL} - p_{PR} => d^*{}_P \sim d_P$; therefore $d_P \sim -d^*{}_P$; this indicates a direct relationship between stereo vision disparity $d^*{}_P$ and stereogram d_P.

$$Z_{\mathrm{PE}} = f\frac{b_{\mathrm{eyes}}}{d^*_{\mathrm{P}}} \tag{3.2}$$

When viewed it in a particular way (cross-viewing or parallel viewing); a hidden floating object will appear in 3D. We generate a stereogram image by repeating its patterns horizontally across the original image to present a range of 3D depth forms within certain constraints [53]. The distance between repeating patterns defines the depth of virtual point perceived by the viewers. The principle of hiding a 3D profile beneath a flat stereogram is shown in Figure 3.1. The viewers can optically reveal the 3D scene from a stereogram by forcing the two eyes to see two different points on the image surface, or focus on a point behind the surface. Because the image contains repeating patterns, the brain is tricked to fuse them together and thereby creating 3D image.

From the above, to view a stereogram, both eyes should focus on a point above or behind the screen. The viewing directions converged, the repeating texture patterns are horizontally translated and are merged on the top of each other. If identical image patterns are well overlaid (they look identical), the brain identifies that it might be the same point in space, thus, assign to it a position in 3D. Then the depth of any other points is dependent on other relative location of correspondences compared to the previously determined points. Thus, to generate a stereogram, given a 3D profile $Z(x, y)$ sampled on a grid of $M \times N$, a strip of chosen image pattern $P(i, j)$ of sampled size $M \times N$. In short, a stereogram can be generated by adding pixels around a selected pattern to represent all points of the disparity/ depth map [59]. Figure 3.2 displays a stereogram image, created based on two images: a texture image (left stereo image) and a disparity map (depth) which was generated from a stereo reconstruction process.

Retrieving 3D information from the stereogram images, however, is an ill-posed inverse optical problem due to the random nature of matching similarity and the structural ambiguity of repetitive patterns. The reconstruction can be achieved by using stereo matching techniques. The stereogram image can be cut in half to have C_1 and C_2 with C_1 is the left half of the stereogram image and C_2 is the right half of the stereogram image. The disparity/ depth map could then be rebuilt by using any available local, global or semi-global stereo matching process [60]. This reconstructed depth map can be treated as the hidden information which is holding the unique code or number.

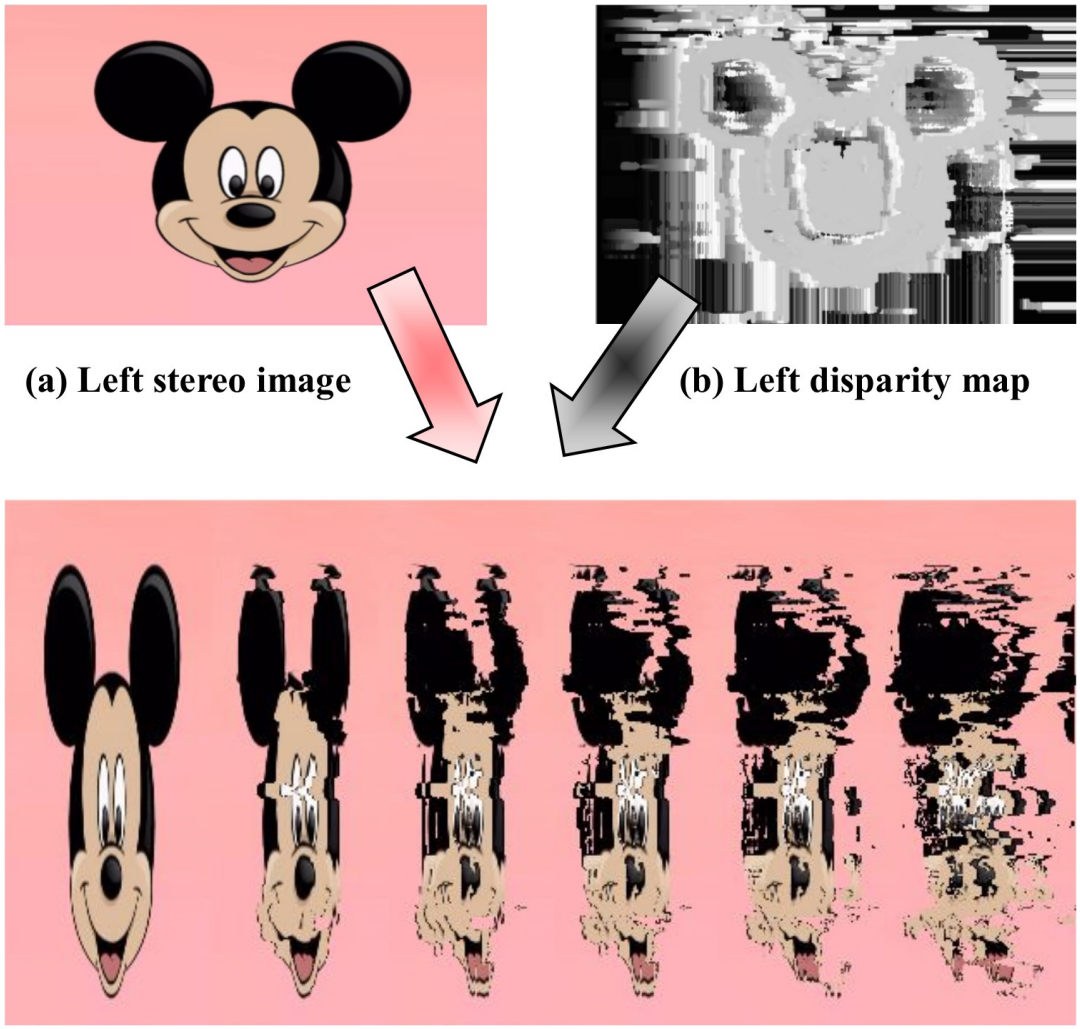


Figure 3.2: Stereogram created from a depth-map and an image

3.2 Marker detection procedure

The marker based detection process includes several different steps in order to identify the marker hidden information. Firstly, the system need to be able to find the outlines of potential markers and determine its boundary. Marker boundary detection could easily be done by many image processing techniques. The system then should also capable to identify the marker information within the confirmed boundary. In addition, there will be different algorithms which are used for the marker information identification based on the marker type. In theory, the marker based detection process consists of the following two main steps:

- Marker boundary detection
- Marker information identification

The execution order of the described steps may differ or the system would merge them into the same algorithm. However, the main concept is usually remaining unchanged. In the rest of this section, we will discuss in details about each step of the marker based detection process.

3.2.1 Marker boundary detection

The intensity image (grey-scale image) needs to be obtained first in the marker boundary detection. The system is necessary to convert the captured image to grey-scale format from RGB format using a well-known technique [61]. The equation 3.3 shows the relationship of gray-scale value and RGB color components.

$$Y = 0.2126R + 0.7152G + 0.0722B \tag{3.3}$$

Next the system will use either thresholding technique [62] to search for the potential marker from the binary image and edges detection method [63] to identify the marker boundary. The thresholding technique is normally using the adaptive thresholding method to determine the illumination changes in the image [64] as shown in Figure 3.3. As long as the image illumination changes are identified, the system can classify which objects are most likely to be the marker area. Depending on the particular requirement, the system may reject the objects that are smaller then defined constant or otherwise are clearly not the marker.

Canny edge detection is the widely used method to detect a wide range of edges in particular image as shown in Figure 3.4. The algorithm can be broken down into five smaller steps as follow:

Figure 3.3: In order to obtain the adaptive thresholding image (right), the original captured image (left) should be converted into grey-scale image (middle).

1. Using Gaussian filter to remove high frequency noise.
2. Compute the image intensity.
3. Apply non-maximum suppression to remove “false” responses to to edge detection.

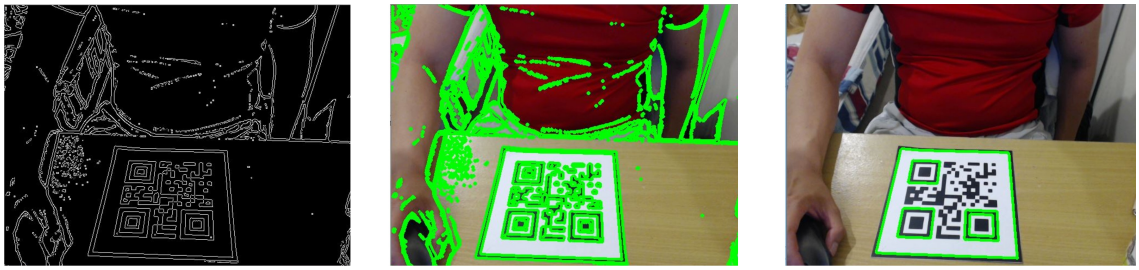


Figure 3.4: An example of edge detection process (left) using threshold image (Figure 3.3) with superimposed contours onto the original captured image (middle) by green color. The right figure shows the remaining detected edges after applying the noise removal algorithm.

4. Apply thresholding using a lower and upper boundary on the gradient values.
5. Using hysteresis to track edges; the weak edges that are not connected to strong edges will be suppressed.

Finally, the boundary of the marker is determined for the future hidden information process.

3.2.2 Marker information identification

As described in Section 2.5, there are many different types of markers available on the market, but they are falling in two main categories which are template markers and data markers. Each marker category provides different way to store and encrypt the hidden information. The hidden information will be encrypted by using either template matching technique (template marker) and data decoding method (data marker).

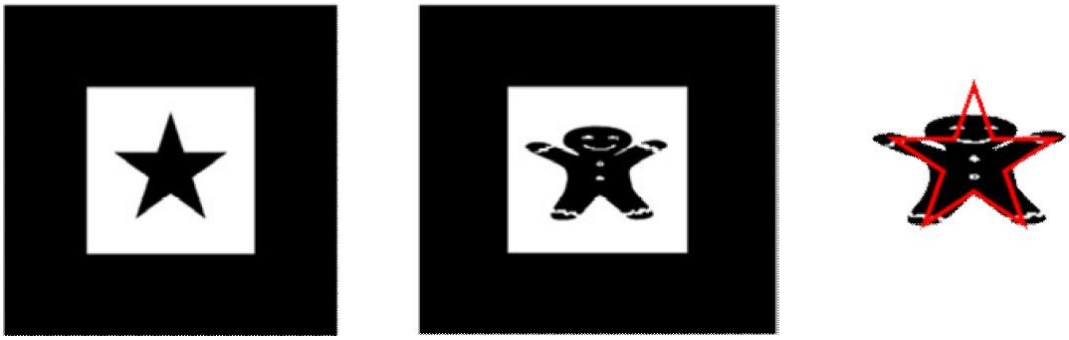


Figure 3.5: An undesired similarity example between marker. They look completely different to human eyes, but the template matching process may confuse them as their presentation areas are almost overlapping.

3.2.2.1 Template matching

Template matching technique is used identify the detected marker identity by compare it with each of sample images which are stored in the database. However, the size, location and orienta-

tion of detected marker are unknown as the detected marker is not warped. The system then will be scale the detected marker to the same size of the sample image and compare them in four different positions according to four possible orientations. This process is repeated till the end of database loop. The sample image that gives the highest similarity value (HSV) is the correct marker. The orientation of detected marker is also defined as the same with the position of the matching sample image. This orientation information could be used for the future display purposes such as the orientation of virtual objects. However, if the HSV is lower than a threshold, the detected marker is rejected. The HSV can be easily calculated based on either sum of squared differences or cross-correlation [7] as described in Equation 1.1 and Equation 1.2. As the system needs to match the detected marker against each of data set sample images four times, it is clear that more time consumed would be needed for the marker identification. Therefore, it will be inefficient in practice if there were a huge data set. Another disadvantage of template matching is that the detection process could give the wrong outcome due to the undesired similarity of images even they look completely different to the human eyes as shown in Figure 3.5. These problems were clearly described in Section 1.1 as the main technical issue this thesis is trying to solve.

3.2.2.2 Data decoding

The data decoding method is normally used for data markers which are usually made up with black and white data cells. Black color is representing "1" in binary number and white for "0". The system will use this principle to get a series of binary values which can be represented as the whole data of the marker. As the binary series are unique, this decoded binary number is the same as a marker ID or marker identity as shown in Figure 3.6. This method is giving an advantage over template matching in term of time processing and undesired similarity of markers.

Another advantage of data decoding is that besides encoding information, it also provides the capability of error detection and correction. This feature is not possible to be used on template marker without damaging the graphical area. As mentioned above, Hamming codes [10] and Reed–Solomon [11] codes are two most used methods for error detection and correction.

The Hamming codes are linear error-correcting codes family which were invented by Richard Hamming in 1950. The algorithm makes use of parity bits which tell whether the number of "1" in particular binary string is even or odd. There will be two type of bit parity: odd parity and even parity. The odd parity equals to one when number of "1" is even, and zero for otherwise. The even parity equals to one when number of "1" is odd, and zero for otherwise. The added parity bit to data is capable to detect the error of one single bit in that binary string. One parity bit can only reveal one single-bit error of entire binary string. Therefore, more than one parity bit

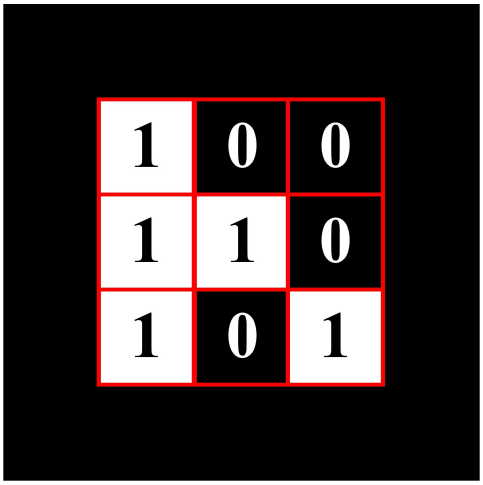


Figure 3.6: Data decoding with the ID is 100110101 in binary or 309 in decimal.

is normally added for binary string error detection and correction. The binary string or data is usually divided into blocks (eg: 4 bits) and each block is encoded separately. The Hamming (7,4) adds three additional parity bits to every block of four data bits in a binary string. It is able to detect all single-bit errors and correct any of them. However, there will not always be a single bit error in practice. The Hamming (8,4) is an extended version of Hamming (7,4) which is suitable for both single error correction and double error detection. In other words, the more parity bits are added, the more errors could be detected and corrected.

The Reed-Solomon codes were invented by Irving Stoy Reed and Gustave Solomon in 1960. The Reed-Solomon codes operate on m-bit symbols whereas the Hamming codes operate on individual bit. The codes are defined as polynomials operating over finite fields. They are designed to detect and correct multiple symbol errors. The number of t check symbols will be added to the original data, the codes are able to detect up to t erroneous symbols, or correct up to $\frac{t}{2}$ symbols. The Reed-Solomon codes are usually used in DVDs, CDs, satellite communications and complexity bar-codes such as quick response code where higher degree of data reliability is priority [65]. Therefore the Hamming codes are often referred when it comes to the error detection and correction for simple data marker.

3.3 Marker pose estimation

In order to render virtual information on the of a physical marker, we need to find the marker pose estimation. This step will be started with determining of the camera position related to the physical marker. The system then can use this information to blend the virtual information into the real world environment (physical marker). In order word, the AR system needs to transfer the 3D coordinates of virtual object in real world to 2D coordinates and presents them on the display

screen. In most of AR system, there are three different coordinate systems or transformations as shown in Figure 3.7: (1) World coordinates, (2) Camera coordinates, (3) Display screen coordinate.

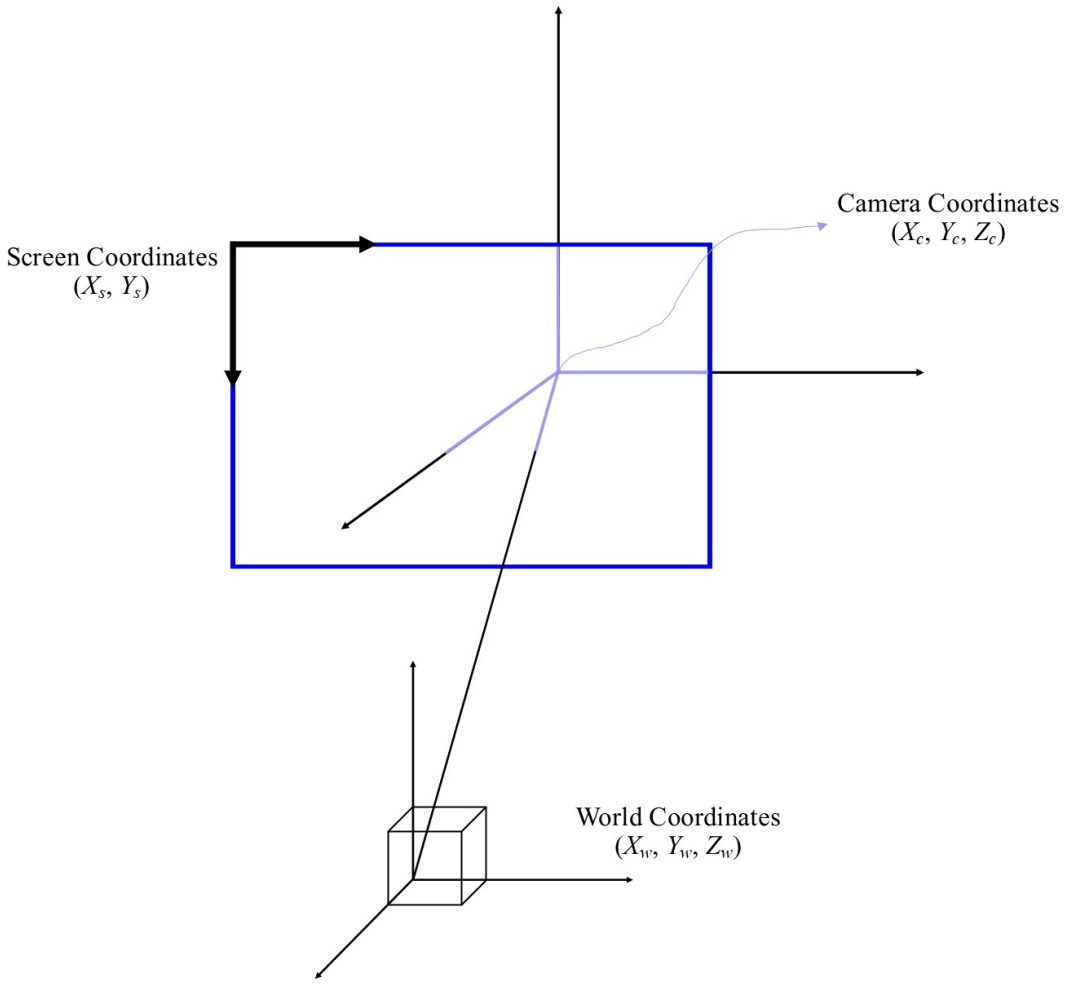


Figure 3.7: Augmented reality coordinate systems

World coordinates (M_W): defines the location of trackable physical marker or virtual information that we would like to render in the real world scene. The world coordinate system is presented as X_w, Y_w, and Z_w in 3D space.

Camera coordinates (M_C): defines the position and orientation (pose) of the video camera that is currently used to view the real world scene. All the physical points of the real world (including the virtual information) scene will be defined relatively to the camera in this transformation.

Display screen coordinates (M_S): defines a projection from the real world coordinates (3D) to

the 2D coordinates which are used to render the pixels on the screen such as HMD or graphic monitor display. However in order to keep the realism of the augmented scene, the highly accuracies of all three geometric transformations (M_W, M_C, and M_S) are required.

3.3.1 Pinhole camera model

The pinhole camera model [66] projects 3D points (3D coordinates of **P**) into the camera screen using a perspective transformation as shown in Figure 3.8. The *optical axis* is presented as the line through the center of focus of the camera **O** and is perpendicular to the camera screen at point **C** (**Z** axis). The *focal length (f)* is the distance between the center of focus of the camera and the camera screen. Let's say that we would like to project the 3D coordinate of point **P** where $\mathbf{P} = [U, V, W]$ on the camera screen. Then the 2D projection of point **P** is the intersection point between camera screen and the line which goes through point **P** and the center of focus of the camera; donated by **p'** $= [x, y]$. In order words, we can express the values of x, y by the following formulas:

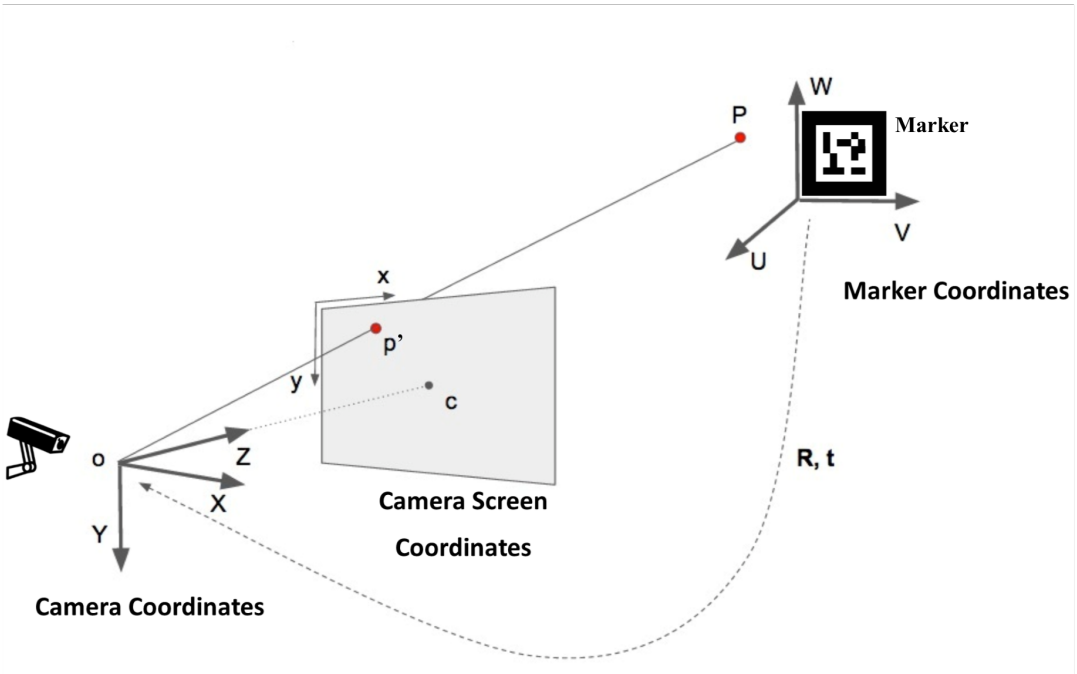


Figure 3.8: Relationship between the marker coordinates and the camera coordinates.

$$x = f\frac{X}{Z} \tag{3.4}$$

$$y = f\frac{Y}{Z} \tag{3.5}$$

3.3.2 Camera parameters

There are two different types of camera parameters which are used to define the relationships between the coordinate systems in most of AR application which are extrinsic camera parameters and intrinsic camera parameters. The Figure 3.9 shows how the camera parameters have been used within the flow of virtual information rendering process.

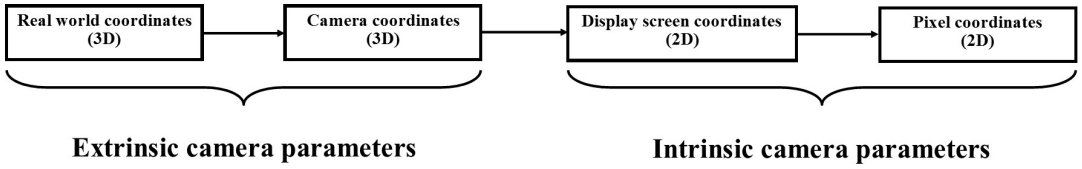


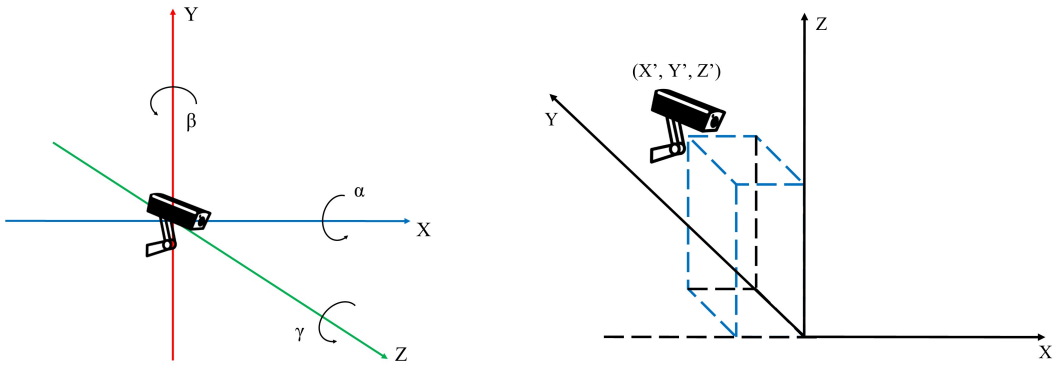
Figure 3.9: The flow of rendering virtual information on the camera screen.

3.3.2.1 Extrinsic camera parameters

The extrinsic camera parameters identify the transformation between the unknown camera coordinate (M_C) and the world coordinate (M_W) which is also including the coordinates of virtual information that we would like to render. The extrinsic camera parameters are external to the camera and may change with respect to the world frame. As mentioned above, AR technology is using six degrees of freedom (6DOF) [25] for rendering and updating the virtual information orientation and location. It means that the virtual information has only two kinds of transformation forms on a static camera, which are translation and rotation. The translation motion occurred when the camera is moved from its current location (X,Y,Z) to a new location (X',Y',Z') in 3D space as shown in Figure 3.10b. The rotation motion which is absorbed when the camera is rotated about the X, Y and Z axes. The camera rotation motion is often represented by using Euler angles [67] (roll, pitch and yaw), or the direction of rotation angles (α, β, γ) as shown in Figure 3.10a.
The transformation T should normally consist of 3×3 rotation matrix R and translation vector t as shown in Equation 3.6

$$T = [R|t] \tag{3.6}$$

Translation motion occurred when camera is moved from its current location (X,Y,Z) to a new location (X',Y',Z') in 3D space. It has three degrees of freedom and represented by vector t which can be calculated as in Equation 3.7. Other motion is rotation, which is absorbed when the camera is rotated about the X, Y and Z axes. Camera rotation motion is often represented by using Euler angles [67] (roll, pitch and yaw), a 3×3 matrix R or a direction of rotation and angle as shown in Equation 3.8.



(a) Rotation (b) Translation

Figure 3.10: The orientations can be expressed with rotation angles (α, β, γ) around axis (X, Y, Z). The new location (X', Y', Z') in 3D space is defined by translations along the axis (X, Y, Z)

$$t = (X' - X, Y' - Y, Z' - Z) = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{3.7}$$

$$R = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix} \tag{3.8}$$

or can be expressed in homogeneous coordinates:

$$T = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \tag{3.9}$$

3.3.2.2 Intrinsic camera parameters

The intrinsic camera parameters are internal and fixed to individual camera properties or setup. In most of the time, the intrinsic camera calibration matrix K [68] needs to be calculated (Equation 3.10) first before starting the tracking process.

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Where f_x and f_y are the camera focal length in the x and y directions. The coordinates of image center point C is donated by (c_x, c_y). s is the axis skew due to projected image distortion. However

in most of nowadays cameras, pixels are often square and columns and rows are straight. Thus, the value of s can be discarded; $s = 0$ and $f_x = f_y$.

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

The radial distortion and tangential distortion have also occurred due to the imperfect camera lenses. In the end, the camera lens distortion is also taking into account in order to get the 2D coordinates of projected point p' (see Equation 3.12 and 3.13).

$$p' = D(K[R|t]M') \tag{3.12}$$

or can be written in matrix form:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = D \left(\begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix} \right) \tag{3.13}$$

where D is the distortion function, (x, y, 1) is the 2D coordinate of projected point p' and (U, V, W, 1) is 3D coordinate of point P is real world. The radial distortion can be easily calculated as Equation 3.14. The position ($x_{corrected}$, $y_{corrected}$) is the corrected output of the old pixel point at (x,y). The k_1, k_2, and k_3 are the coefficients and r is the value of rotation matrix.

$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3.14}$$

Tangential distortion is another type of lens distortion which occurs when the image taking lenses are not perfectly parallel to the display plane. The tangential distortion can be corrected via the Equation 3.15; where p_1, p_2 are the coefficients.

$$\begin{aligned} x_{corrected} &= x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_{corrected} &= y + [2p_2 xy + p_1(r^2 + 2y^2)] \end{aligned} \tag{3.15}$$

In order to solve the lens distortion, we need to find out the values of five distortion coefficients k_1, k_2, k_3, p_1, and p_2 as shown in Equation 3.16. A well-known pattern such as chessboard has been widely used to calculate five coefficient parameters. However, the major problem with these calibration approaches is that the process needs to be done manually in a separate calibration procedure. Thus, the accurate camera calibration is remaining as an open problem of image processing.

$$D_{coefficients} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{bmatrix} \tag{3.16}$$

CHAPTER 4

METHODOLOGY

The research methodology is defined as a systematic way to solve a problem. It shows how and which type of research should be carried out in order to solve the defined problem. The methodology is also telling us which methods should be used to gain the new knowledge [69]. In this chapter, we will discuss the uses of different methodologies and tools in this thesis. The rest of this chapter will be presented in three different sections:

- The research framework used for this thesis.
- The discussion about data collection techniques, analysis and evaluation will be used in this thesis.
- Describe some available tools (including software and hardware) which could be used to for prototype development and data analysis.

4.1 Research framework

The design science research (DSR) [70] will be used as the main research framework of this thesis. The DSR was described as a framework for research process in information system (IS) research. It includes three main components which are the environment, IS research and knowledge base. The Figure 4.1 shows the details of the adopted framework in this thesis.

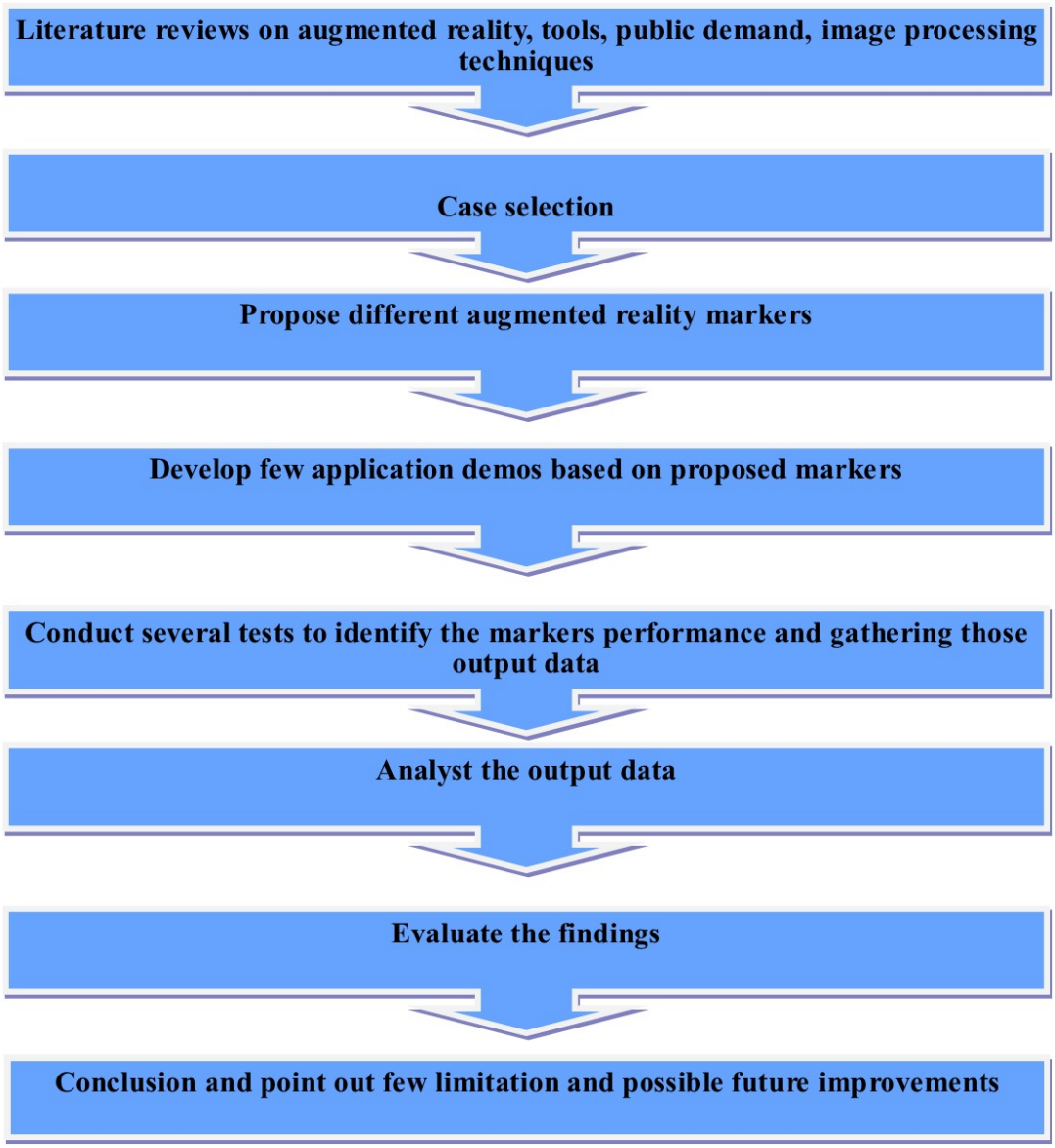


Figure 4.1: Research framework

4.2 Data collection and analysis

The main data collection techniques used in this thesis are application development, experiments set-up and observations. The main reason for choosing the experiments as the main technique instead of other popular methods such as questionnaires or personal interviews is the reliability. The questionnaires technique is not too hard to conduct the nowadays great Internet and social media. However, it is not too easy to make sure that the questions have been answering are accurate and in the most fairly way. Conducting physical experiments and collecting data are taking longer time, but also provide the results with higher accuracy and more reliable. Another

important reason for choosing to conduct the experiments is that they are more likely to be accepted for examination by most of the international academic reviewers. The data collection process will be started with application development. We are planning to introduce three new, different AR marker within this thesis. The details of the markers source code (including creation and hidden code decryption) are presented in the appendices. The several experiments will be then conducted to qualify the markers system performances such as hidden code detectability and error correction. The types of data to collect are times and numbers. All of the experiments will be carried out under the same condition and supported equipment settings.

The most import data analysis aspect in computer science research is to look for the meaning of observed interpretation as well as what is experienced and knowledge gain by the subjects. The article [71] defines qualitative data analysis as "working with the data, organising them, breaking them into manageable units, coding them, synthesising them, and searching for patterns". The data analysis objective is to explore the patterns, concepts and meaning. All the data gathered in these experiments will be saved in a private repository and will be analysed and decoded at the appropriate time. We also conduct the same experiments with other markers such as a bar-code marker, QR code marker or template marker to present a brief performance comparison between them. As the objective of this thesis is presenting the new AR markers which have the characteristic of both template and data marker or the proposed markers should have the equivalent system performance with data marker. Thus, the bar-code and QR marker are the key comparisons in this situation.

4.3 Supported software

Python [72] is an interpreted high-level programming language for general-purpose programming which was created by Guido van Rossum in 1991. Python emphasises code readability which provides a syntax that allows the programmers to express the logic in fewer lines of codes [73]. This advantage is the major reason for us to choose Python as the main programming language for this thesis as it could help us to save time in term of the programming process. Python also supports multiple programming paradigms such as object-oriented, imperative and functional programming styles. It also uses a scripting language as well as non-scripting contexts. Python can be easily packaged into standalone executable applications and use third-party packages for the expansion tools. The following codes are showing how Python code is implemented compared to other famous programming languages such as C++ and Java in order to present the same output:

- Sample code in **Python**:

```
print("Hello, World")
```

- Sample code in **Java**:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello,World");
    }
}
```

- Sample code in **C++**:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello,World";
    return 0;
}
```

Pycharm [74] is an integrated development environment (IDE) which was designed specifically for the Python language (Figure 4.2). Pycharm is free software under academic license and supports different platform such as: Windows, macOS and Linux. Pycharm is chosen as the main IDE for this thesis as it supports different operation systems and provides the development environment with freely cost.

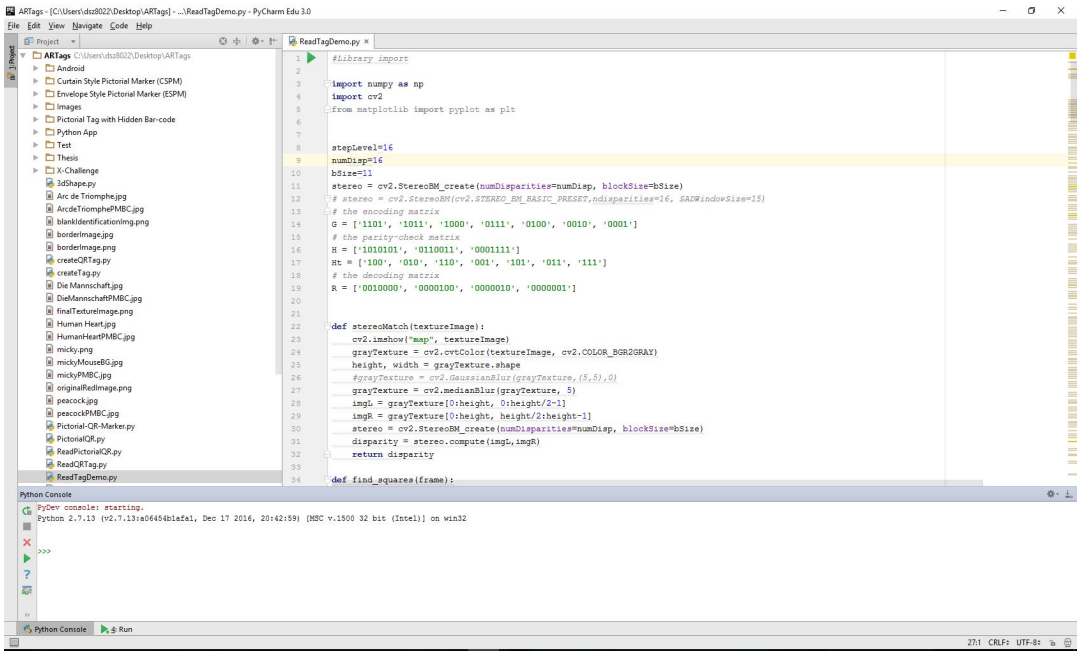


Figure 4.2: Pycharm IDE.

CHAPTER 5

PICTORIAL MARKER WITH HIDDEN BAR-CODE

*Parts of this chapter has been published in **paper 1** and **paper 2** listed in publication list*

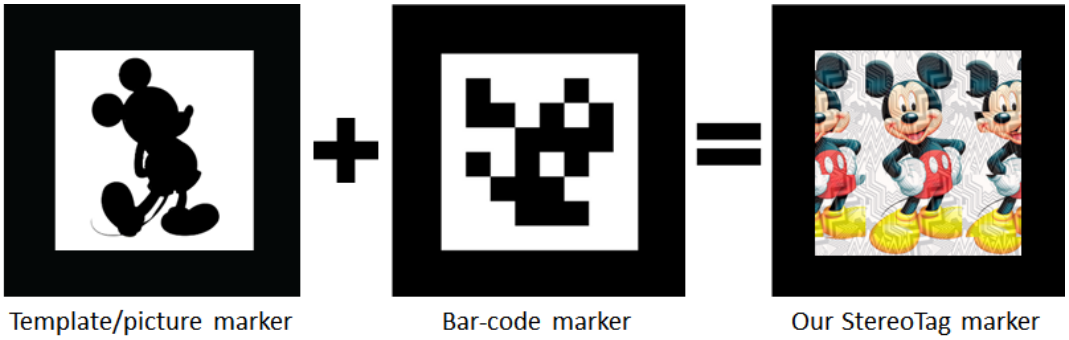
Pictorial marker with hidden bar-code (PMBC) uses the idea of stereogram to conceal a multi-level bar-code optically. The initial design is demonstrated in Figure 5.1b, the detail will be described further in this chapter. The proposed PMBC marker presents some notable advantages over others:

- **Large range of data:** The multi-level bar-code can hold L^N different numbers, with L being the number of levels in each bar, and N, the number of bars.
- **Virtually Pictorial:** The image inside each PMBC marker is made from meaningful illustrations rather than black bars, squares, or dots.
- **Flexibility of Pattern:** The decoded information is independent of image patterns; a broad range of images can be used to encode the same bar-code.

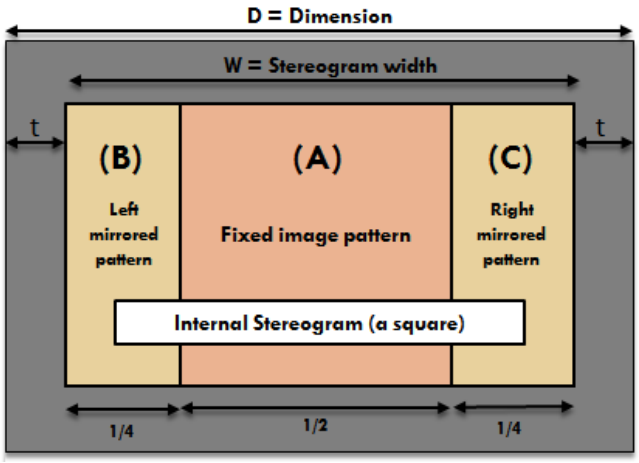
Each PMBC marker is a rectangle with a dimension $D = M \times N$ measured in pixels or millimeters; border thickness t is relatively small (we set $t = 4\%$ of D). The quadrilateral property of the rectangles can be used to detect their four straight lines and four corners; these are used for detecting the marker. The internal image is a stereogram (size $W \times H$) made of three regions. The central area is a fixed image (region A that fills up $\geq 50\%$ of the stereogram) and two repeated patterns on both sides of region A (region B and region C with $\leq 25\%$ of the stereogram each).

Hidden inside each stereogram is a bar-code with many horizontal bars with the same thickness. Each bar is coated with different grey levels between black and white; these levels are used

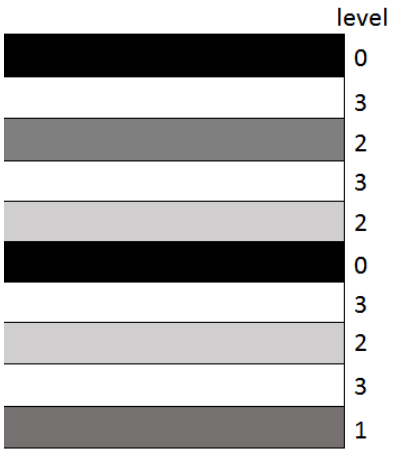
to represent different depth levels inside the stereogram. Figure 5.1c displays an example of 4-level binary bar-code with 10 horizontal bars. Each bar can hold four different levels: 0, 1, 2, 3; corresponding to black, dark grey, light grey, and white. Thus, this barcode can store as many as $4^{10} = 1,048,576$ different numbers.



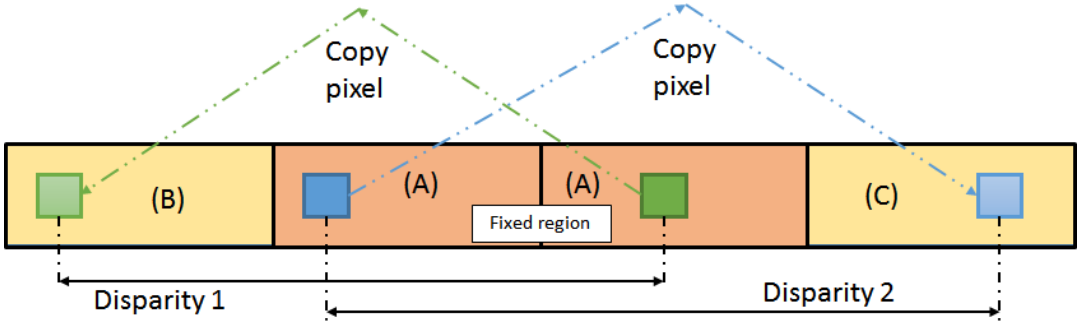
(a) Idea of the proposed PMBC Marker.



(b) Design of proposed PMBC marker.



(c) Multi-level bar-code.



(d) Principle of the PMBC marker creation

Figure 5.1: Design of PMBC marker that optically hides a multi-level bar-code.

Figure 5.2 demonstrates necessary steps of creating our proposed PMBC marker. As described, our PMBC marker has a black border so that it is easily and reliably detectable under various

circumstances. In theory, the internal stereogram of the PMBC marker can encode any 1D barcode such as Code11, Code 32, Code 49, Code 93, Code 128, EAN-8, and EAN-13.

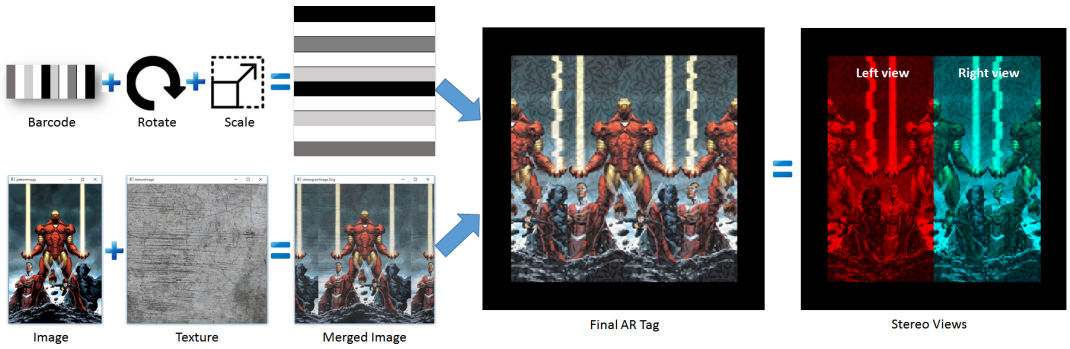


Figure 5.2: The process of creating a PMBC marker from a multi-level bar-code and a picture (Ironman). The middle 50% of the PMBC marker is a fixed illustration of the Ironman figure. The PMBC marker is also a side-by-side stereo image pair (red/cyan painted).

5.1 Multi-level Bar-code

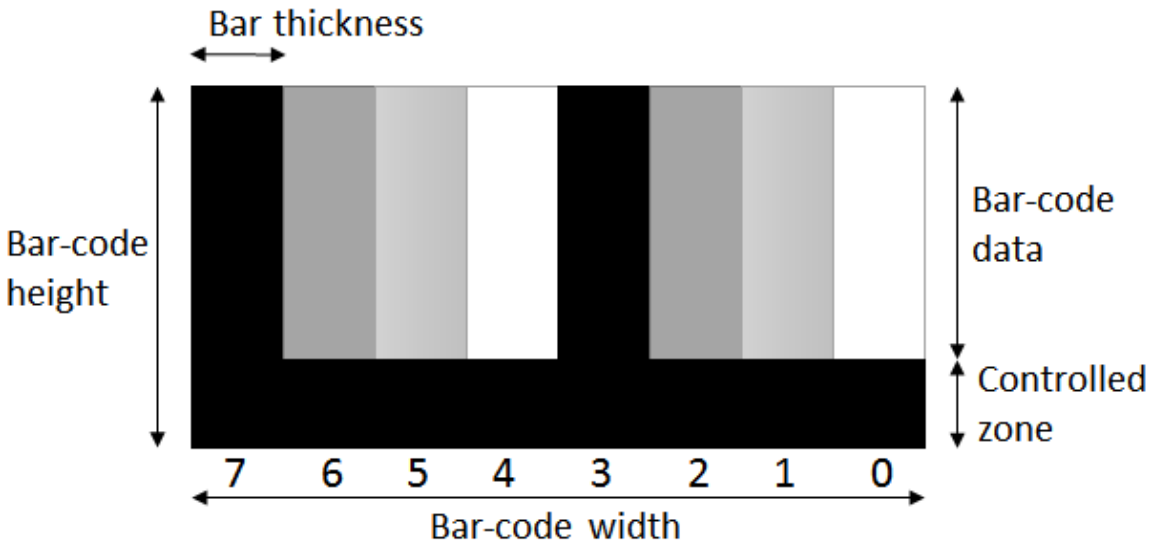


Figure 5.3: Proposed multi-level Bar-code (version 1).

The initial design of multi-level bar-code is shown in Figure 5.3. This multi-level bar-code has a dimension of $W \times H$. It may have many vertical bars; each has thickness T, the data held in each bar is one number, corresponding to depth levels (level 1, 2, 3, behind the surface). We represent it using grey levels (the darker, the lower depth, the brighter, the higher depth); thus, black represents lowest level depth and white represents the highest level depth.

The number A presented by this bar-code is:

$$A = b_{N-1}L^{N-1} + b_{N-1}L^{N-2} + .. + b_1L^1 + b_{k-1}L^0 \tag{5.1}$$

Where b_i is depth level at i^{th} bar, L is number of depth levels at each bar, and N is the number of bars.

The controlled zone is a horizontal bar at the bottom of the bar-code; it is used to self-check the validity of the bar-code and specifies the bar-code version. The controlled bar has the same thickness as a vertical bar, and always have the lowest depth level (black colour). Currently, we define five different versions of this multi-level bar-code as shown in Table 5.1. The Higher version has more vertical bars and more levels presenting in each bar; thus storing a larger range of numbers. For instance, version 0 can only represent numbers between 0 and 63 using 6 bars and two levels each. On the other hand, version 6 may store up to 7.95×10^{25} different numbers using 24 bars and 12 levels each.

Version	Controlled Zone	Number of bars (N)	Depth levels (L)	Presentation Range
0	$\frac{1}{4}$ width	6	2	$0 - 63$
1	$\frac{1}{8}$ width	8	4	$0 - 6.55 \times 10^4$
2	$\frac{1}{12}$ width	12	6	$0 - 2.18 \times 10^9$
3	$\frac{1}{16}$ width	16	8	$0 - 2.81 \times 10^{14}$
4	$\frac{1}{24}$ width	24	12	$0 - 7.95 \times 10^{25}$

Table 5.1: Specification of different versions of multi-level bar-code

5.2 Stereogram Construction

The internal image I_c of PMBC marker has a dimension of $W \times H$. First, we select a $W' \times H'$ illustration image I_i to be placed at the center of the stereogram. Let $s_H = H'/H$ is the ratio between the height of I_i and I_c; pixels of I_c is defined as:

$$I_c(x,y) = I_i(s_H \times x + \frac{(W - W')\%W'}{2}, s_H \times xy) \tag{5.2}$$

Occasionally, image I_c may contain large blank monotone regions like the walls and the tabletop and this could cause stereo correlation to fail. We opt to add artificial textures on I_c to eliminate the blank regions if any. A texture image I_t is chosen and the new image I_s is:

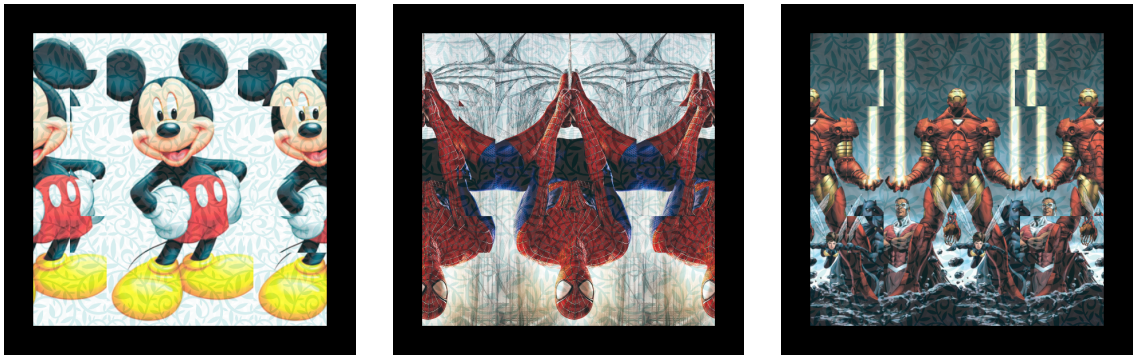
$$I_s(x,y) = (1-k) \times I_c(x,y) + k \times I_t(x,y) \tag{5.3}$$

where k is the transmission coefficient ($0 < k < 1$).

The central 50% region of I_s is kept intact (region A - Figure 5.1b) and will be used to repeat itself on the left and the right of the figure (regions B and C). In other words, pixels with identical/same colours to the left and right of a selected pattern are added with different horizontal shifts according to the disparity map D^*. The detail can be found in the pseudo-code below:

```
for h in range(0, H):
        for w in range(0, H/4):
                k = barcodeLevel[H/L]
                stereogram[h,w + k] = pattern[h,H/2+w]
                stereogram[h,3*H/4+w] = pattern[h,H/4+w+k]
```

The final image is then decorated with a thick black border to generate our proposed marker. As stated, the value stored is independent to the look of our markers. Three examples of the PMBC markers are shown in Figure 5.4. All of them hide the same multi-level bar-code storing number 29,984.



(a) Micky mouse marker. (b) Spider man marker. (c) Iron man marker.

Figure 5.4: Three examples of PMBC markers that conceal the same multi-level bar-code.

5.3 Pre-processing: Decoration of the Marker's Central Image

As discussed, we use the central image regions to build a side-by-side stereo image, so that is capable of storing 3D dense map. In general, feature points are needed to efficiently describe the 3D distance of points on the stereo pair. However, some of the chosen images might lack feature points (corners, distinct pixels), e.g. it contains a sizeable blank background as seen in Figure 5.5-left. Some other image with many horizontally repeating patterns could also affect the 3D stereo reconstruction process.

To tackle the problem, we need to decorate the central image. There are two ways as seen in Figure 5.5-right: (1) combining the image with random colour texture, (2) adding vertical shading illumination to the picture. We prefer the second method as the colour of the central

image is well-kept. However, from our experiments, both the methods work well on reducing the monotone background of the pictures.

Figure 5.5: There are two methods to decorate the inner marker image: (1) adding textures and (2) adding shading gradients

5.4 Data Storage Capacity

The PMBC marker uses the binary numbers (1 and 0) as stored comparable data to replace the original coloured image data. Because binary numbers are simple, they have a limitation on how many items can be stored based on the number of available bits. They absorb fewer spaces in the database and also require shorter processing time than the original data. Assuming that we have to process twenty 24-bit colours of advertisement posters with the average size of 600×1200. There will be $2.06 \times 20 = 41.2$ Megabyte or 4×10^4 Kilobyte spaces need to be reserved in the database as shown in Table 5.2. However, there are only 1-Kilobyte spaces which are more than

thousand times less than the needed amount in case of using 8 bits binary numbers for the same quantity of items. Another advantage of using binary numbers is the dramatic improvement of processing performance when compared with the manual image registration technique. This is because binary number indexing technique does not require the image comparison process. The image similarity issue is also minimized as each stored number is unique.

Marker type	Storage (Kilobyte)	Presentation quantity
8 bits binary number	1	1
12 bits binary number	1.5	1
600 x 1200 24-bit color image	2.06×10^3	1
600 x 1200 32-bit color image	2.75×10^3	1
600 x 1200 48-bit color image	4.12×10^3	1

Table 5.2: Comparison of absorbed storage by different marker types

5.5 Detection and Decryption

5.5.1 Internal Image Detection

Firstly, we need to find closed contours on the pictorial bar-code. Some basic image processing steps are required; they are outlined below:

- Convert the input image from RGB to greyscale
- Perform an adaptive binary thresholding method.
- Detect contours, if there are four vertices, it should be a quadrilateral.
- Apply Perspective Transform [75] to retrieve the internal image of the marker.

Once the border is detected, we can obtain the internal image by the folowing formula for further decoding.

$$\overset{I'}{\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix}} = \overset{map\ matrix}{M} . \overset{I}{\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}} \tag{5.4}$$

where M is a map matrix, I' is the internal image created from the four corners $g(x_i, y_i), i = 0, 1, 2, 3$ of the original image I.

5.5.2 Numerical Decryption and Stereo Reconstruction

The retrieved internal stereogram image of the detected PMBC could be used to rebuild the hidden multi-level bar-code. This step is equivalent to a stereo reconstruction process applied on two stereo images C_1 and C_2; C_1 is the left half of the stereogram and C_2 is the right half of the stereogram. The disparity levels (ranged between 0, d_{MAX}) are known from the width of the internal stereogram.

The intensity-based stereo matching is a complex algorithm; they can be categorized into at least three families below:

- **Local matching algorithms:** Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), and Cross-correlation based block matching stereo algorithm (BMS) [76].
- **Global stereo matching algorithms in 1D (one scan-line at a time):** Dynamic Programming Stereo (DPS) [77], Symmetric Dynamic Programming Stereo (SDPS) [78], and Belief Propagation - 1D version (1DBP) [79].
- **Global stereo matching algorithms in 2D:** Graph Cuts Stereo (GCS) [80].

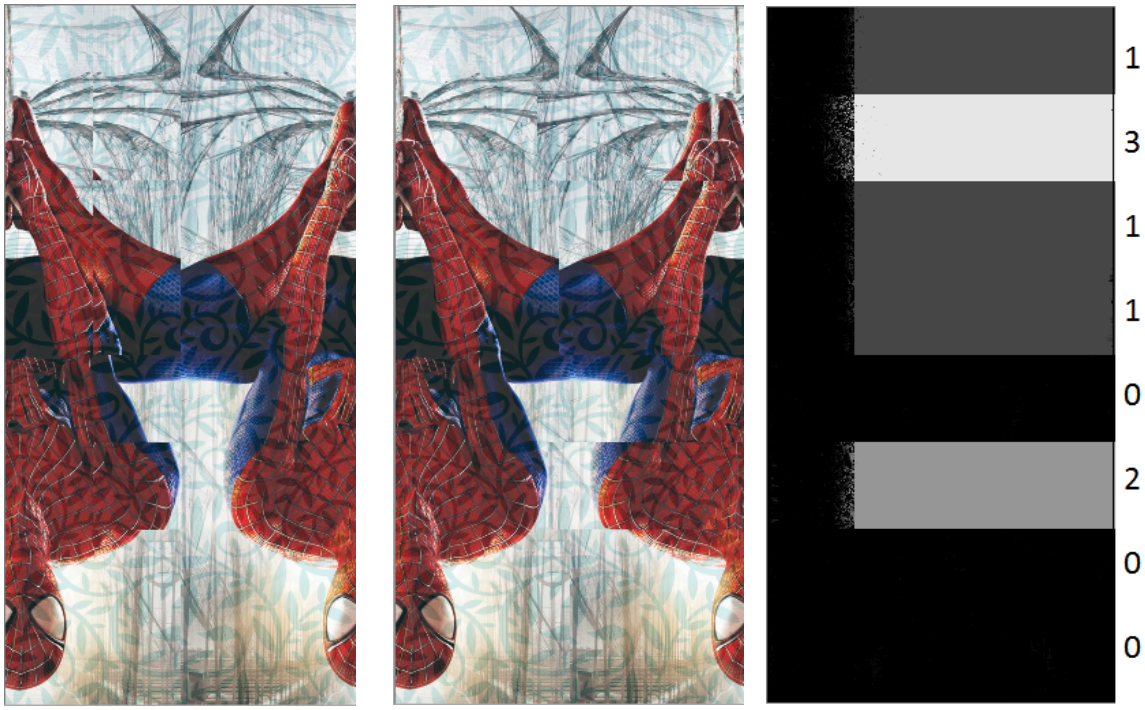
We use a technique that guides DPS using pre-computed BMS depth map, thus restricting and guiding the DPS search for optimal profile related to signals in 2D. Previously being evaluated in [81], **both DPS and BMS are fast but GCS is not**; therefore, to achieve a good speed, we combine BMS and DPS for the guidance method: Stereo SGBM - the semi block matching algorithm. The profiles made from BMS are used as a guiding profile. In addition to its general dynamic programming calculation, quantitative guiding scores are added to DPS's structure so that the method can follow these guiding profiles to establish a more accurate stereo matching. The algorithm uses Dynamic Programming described by Birchfield and Tomasi [77], to make the stereo matching more robust as the computation of disparities is cast as an energy minimisation problem. BMS uses windows to match correspondences; its results are influenced by pixels in multiple scanlines. The cooperation of BMS and DPS should eliminate the effect of the straight strikes made by single scanline reconstruction of DPS. On Middlebury Stereo Evaluation website [82], this SGBM algorithm obtains approx. 76% matching accuracy.

Figure 5.6 displays a disparity map results of some sample AR markers. The disparity range d is set to be between -32 and +31 pixels (64 levels). We calculate the SAD value at each disparity d:

$$SAD_d(x,y) = \sum_{a=-k}^{k} \sum_{a=-k}^{k} |I_l(x+a, y+a) - I_r(x+a+d, y+a)| \tag{5.5}$$

The disparity map is normalised to 0 and 255 and display to users along with the left and right stereo images. Mainly, the brighter the pixel, the closer the 3D point is, and vice-versa. If dynamic

programming fails to obtain a disparity value for a point, it returns a value of -1.



(a) Left stereo image (b) Right stereo image (c) Resulted disparity

Figure 5.6: Left and right stereo images extracted from Spider man marker shown in Figure 5.4b and its disparity map, read from top down: $13110200_4 = 29984_{10}$.

This barcode can be recognised using basic image processing methods, depth level of each bar is averaged, and a unique number can be calculated accordingly using the formula of Equation 5.1.

CHAPTER 6

PICTORIAL MARKERS WITH HIDDEN QUICK RESPONSE CODE

In this chapter, we propose two different types of augmented reality (AR) markers: (1) curtain style pictorial marker (CSPM), and (2) envelope style pictorial marker (ESPM). They still keep the technical advantages to provide not only a useful graphical content presentation but also a capability to improve system performance and security issue. In short, these markers should present the following novel contributions:

- **Large data storage capacity:** The concealed QR code could hold at least 133 items with the lowest version and up to 23,648 items with the highest version [83].
- **Flexibility of Pattern:** The images are approximately similar to each other that can be used without worrying about the undesired similarity as the decoded information and the image patterns are independent.
- **Virtually Pictorial:** The image inside CSPM is made from meaningful illustrations rather than black and white patterns as in data marker which provides no real pictorial information.
- **Error detection and correction capability:** The capability to restore at approx. 30% of codewords.

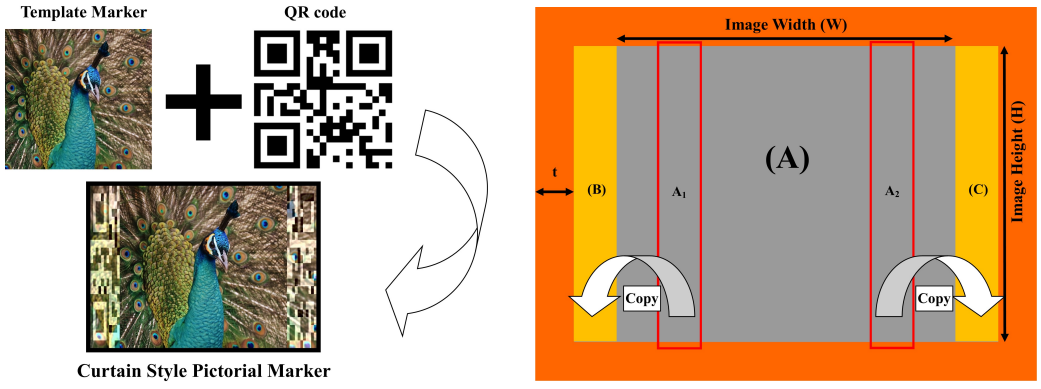
6.1 Curtain Style Pictorial Marker

As mentioned, our proposed curtain style pictorial marker (CSPM) is a newly proposed AR marker design which is capable of presenting not only a realistic-looking coloured graphical content but also encoding quick response (QR) code.

6.1.1 Marker Creation

The construction design of CSPM is shown in Figure 6.1b; it will have a black color border which is used to ease the marker recognition, detection and segmentation. Let's say that we have an image with the dimension of $H \times W$ where H is the height and W is the width. The border thickness t is equal to 2% of the image longest edge or $t = max(H, W) \times 2\%$. The central area A is a fixed original image pattern which occupies 50% of CSPM area. Other two repeated patterns on both sides of region A (region B and region C with 25% each). Let's assume that we need encrypt a unique string: "Peacock" into QR code. Thus, we generate a QR Code image from it, called I_{QR}, and resize it to $\frac{W}{2} \times H$ pixels. Then the two curtain-edged regions (B and C) will be used to hold this binary QR code.

For region B, we read the left half of the QR Code I_{QL}; if the QR block is black, we copy



(a) CSPM ideal

(b) CSPM marker construction design

Figure 6.1: The CSPM ideal is to combine the image pattern and QR code together.

1-to-1 from the central colour image to the curtain-like region. If the block is white, we generate the in the curtain a block with the highest illuminant different. Region C is done similarly, using the right half of the QR Code I_{QR}. The brief idea of CSPM is when the left region is subtracted from the right region, we will receive a binary QR Code. Of course, there are a number of other pre/post image processing operations applied to achieve the best results as seen in the bottom image of Figure 6.1a. Figure 6.3 shows few examples of full created CSPM marker with different QR codes encrypted for each marker.

6.1.2 QR code Decryption

The hidden QR code decoding and virtual object orientation process are demonstrated in Figure 6.2. The procedure includes three steps: (1) internal image detection, (2) QR code regions detection, (3) QR code reconstruction and decoding, and (4) virtual information rendering.

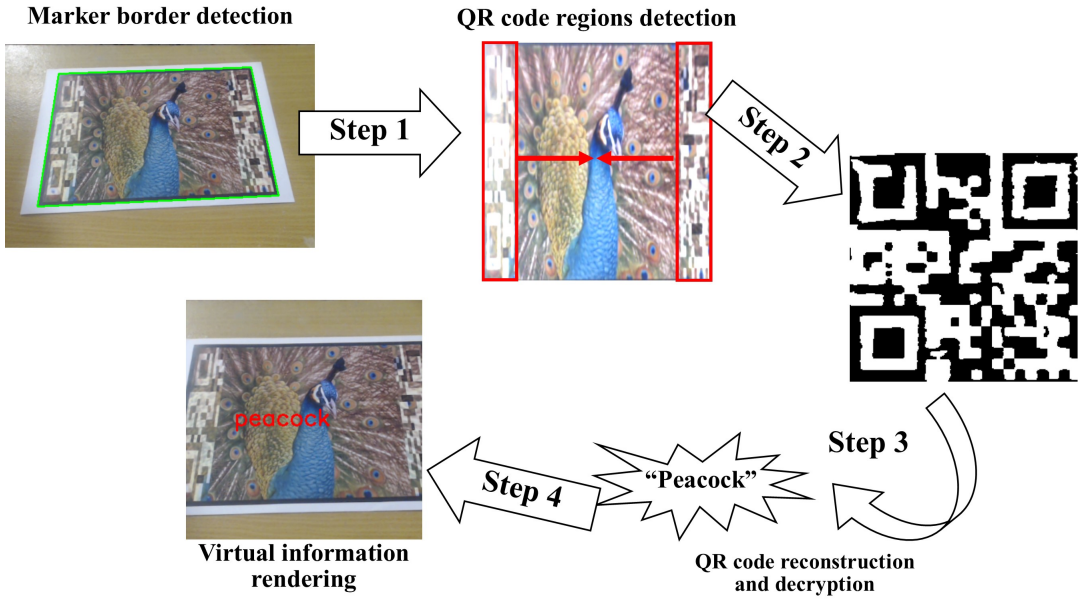


Figure 6.2: CSPM hidden QR code detection and decoding process

6.1.2.1 Internal image detection

The marker border needs to be identified first to retrieve the internal image. This is a common image processing-related problem which could be solved effectively by using Contour Approximation Method [84]. The steps to detect marker border are outlined below:

- Convert the input image from RGB to greyscale
- Perform an adaptive binary thresholding method to detect contours in the image.
- If there are four vertices in the contour, it should be identified as a quadrilateral.
- Apply Perspective Transform [75] to retrieve the internal image of the marker:

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix}^{I'} = \overset{map\ matrix}{M} . \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}^{I} \tag{6.1}$$

where M is a map matrix, I' is the internal image created from the four corners $g(x_i, y_i), i = 0, 1, 2, 3$ of the original image I. Once the internal image is identified, it could be used to obtain further decoding.

6.1.2.2 QR code regions detection

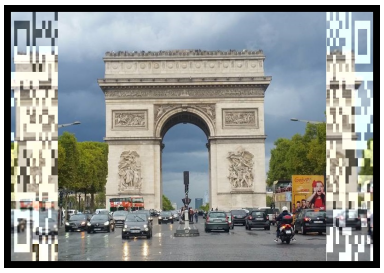
After the internal image is discovered, we cut the image into halves: A and B. For half A, we also cut it into halve: A_L and A_R. To achieve the first half of the QR Code, we perform image subtraction between the two A_L and A_R:

$$A_{diff} = ||A_L - A_R||$$

We do similarly with half B of the image to get B_{diff}. In the end, the two A_{diff} and B_{diff} are combined to reconstruct the full QR sample image. After some further histogram equalisation to compensate the illumination difference, and some thresholding. We will retrieve a readable QR code hidden from the original image.

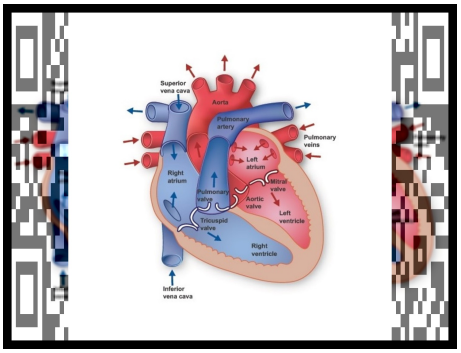
6.1.2.3 QR code reconstruction and decryption

The full detected QR code can now be used to extract the hidden information and present the orientation of the marker. There are many available open-source barcode reading libraries such as Zbar[85] or ZXing. Each of those libraries can read an image frame and automatically detect the QR code. The meaningful text can be then decoded as well as with the code orientation by defining the four corners of the QR code. The system then can use this information to render computer-generated graphics on the marker.



(a) Arc de Triomphe

(b) Die Mannschaft



(c) Human Heart

(d) Peacock

Figure 6.3: Examples of CSPM markers

6.2 Envelope Style Pictorial Marker

Our proposed envelope style pictorial marker (ESPM) is an extended version of CSPM which was described in Section 6.1. ESPM marker is designed to have the independently hidden QR code regions to the central fix image pattern. Thus, it will be able to restore the lost information at the higher rate.

6.2.1 Marker Creation

Assume that we have a square QR code with the dimension $D \times D$, to be mapped on a colour image of a dimension $W \times H$; where W and H are the width and height of the image measured in either pixels or millimetres. In order to map the QR code on the image, we need to rescale the dimension of QR code to the size of the image. The scaling transformation can be calculated as follow:

$$\begin{pmatrix} W \\ H \end{pmatrix} = D \times \begin{bmatrix} S_w & 0 \\ 0 & S_h \end{bmatrix} \tag{6.2}$$

where:

- S_w is the width scaling factor: $S_w = \frac{W}{D}$
- S_h is the height scaling factor: $S_h = \frac{H}{D}$

After that, the rescale QR code is divided into four different diagonally regions to form four triangles located at each edge of the QR code as shown in Step 1 of Figure 6.4b. Now we have two horizontal QR triangles and two vertical QR triangles after the division. Then we need to scale down the height of each triangle to wanted occupation ratio of the original image. Let's say that we only want the QR code to occupy 20% of the original image. Thus, the value of the occupation ratio R should be 0.2. We introduce three different versions of ESPM based on its occupation percentage (as shown in Table 6.1). The scale height of horizontal triangles (h') and vertical triangles (v') can be calculated by equation (6.3) and equation (6.4).

$$h' = \frac{H}{2} \times R \tag{6.3}$$

$$v' = \frac{W}{2} \times R \tag{6.4}$$

where:

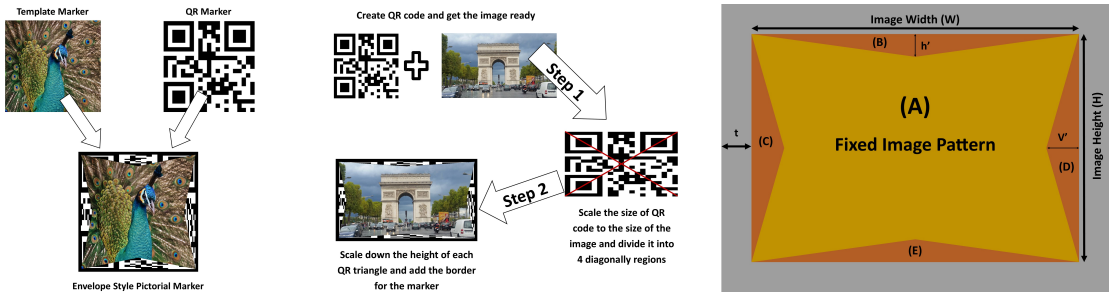
- R is the occupation ratio
- h' is the height of horizontal QR triangles

- v' is the height of vertical QR triangles

After the scaling process, we have for various QR code regions located within the border of the original image as shown as regions (B),(C),(D) and (E) of Figure 6.4c. Each of ESPM versions will have a black-border square which is used for marker recognition and identification. The border thickness t is equal to 2% of the longest edge ($t = \max(H, W) \times 0.02$) of that image.

Version	Presentation percentage	Occupation ratio
20	80%	0.2
18	82%	0.18
15	85%	0.15

Table 6.1: ESPM versions and their characteristics



(a) ESPM Ideal (b) ESPM Creation Steps (c) ESPM Construction Design

Figure 6.4: The design of ESPM also combines the QR code and image pattern together, but the hidden encrypted code regions are independent to the fix image pattern region.

6.2.2 Hidden code Decryption

The hidden QR code decoding and virtual object orientation process are demonstrated in Figure (6.5). The procedure includes three steps: (1) internal image detection, (2) QR code regions rescaling, (3) QR code reconstruction and decoding, and (4) virtual information rendering.

6.2.2.1 Internal image detection

This step is similar to the CSPM internal image detection step which was described in Section 6.1.2.1.

6.2.2.2 QR code regions rescaling

After the internal image is discovered, we need to rescale four QR code regions which are located at the inner image edges for future reconstruction. The internal image needs to be resized to a

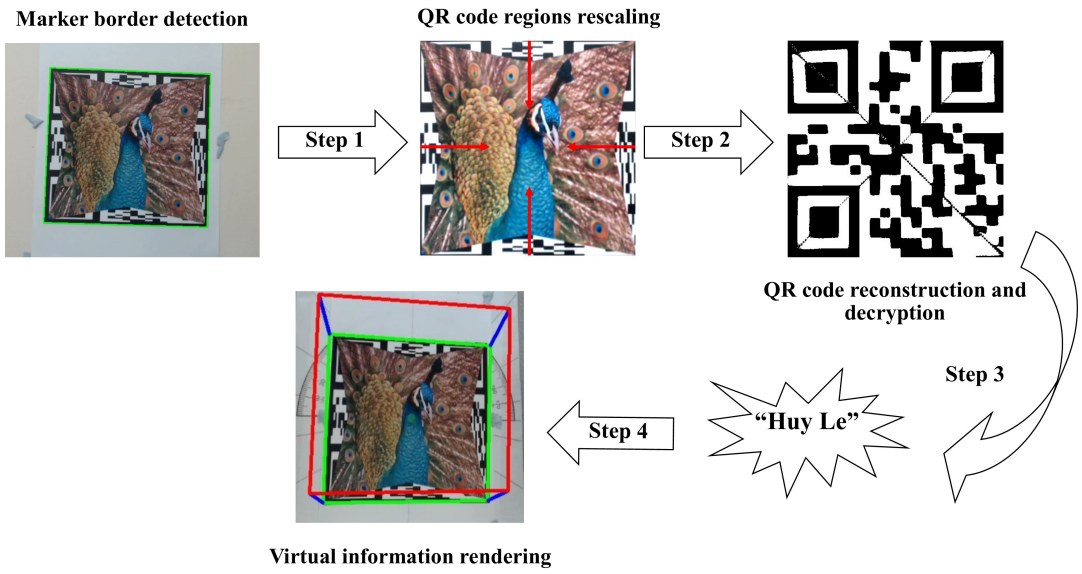


Figure 6.5: ESPM hidden QR code detection and decoding process

suitable dimension which has the same measurement for every edge (520×520 pixels in this case); a QR code is a square shape. Then the resized image will be cut in half in the horizontal direction and vertical direction to make four separate rectangles. The horizontal rectangles can be used to retrieve the top and bottom part of hidden QR code, and the vertical rectangles are for left and right part. Each hidden QR code region can be retrieved as follow:

$$\begin{pmatrix} E \\ H \end{pmatrix} = \begin{pmatrix} E \\ \frac{E}{2} \end{pmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{R} \end{bmatrix} \tag{6.5}$$

where:

- R is the QR code occupation ratio
- E is the edge of the QR code (the longer edge)
- H is the resized height of the QR code

Then, the H will be cropped to the size of edge E to form a square shape for each region. Each region contains an isosceles triangle which consists an individual QR code part. We then could form a complete QR code by merging all four regions as shown in the top-right image of Figure 6.5.

6.2.2.3 QR code reconstruction and decryption

The full detected QR code can now be used to extract the hidden information and present the orientation of the marker. There are many available open-source barcode reading libraries such as Zbar[85] or ZXing. Each of those libraries can read an image frame and automatically detect

the QR code. The meaningful text can be then decoded as well as with the code orientation by defining the four corners of the QR code. The system then can use this information to render computer-generated graphics on the marker.

CHAPTER 7

EXPERIMENTS AND RESULTS

*Parts of this chapter has been published in **paper 1** and **paper 2** listed in publication list*

Conducting experimental exercises against with different scenarios is one of the key techniques to qualify our proposed augmented reality (AR) markers quality. The first experimental exercise is hidden code detectability test. This is a vital step in most AR applications as the AR system must determine the marker's identification accurately to provide the correct virtual information. Therefore, in Section 7.1 we evaluate the hidden code detection performance of proposed markers against different light conditions and effects. Secondly, in Section 7.2, we explore the accuracy of our approach to the failure condition or also known as error correction test. The experimental exercise of system processing performance is also carefully conducted in Section 7.3. Finally, we conclude Section 7.4 with the results showing few samples of AR application running with our proposed marker under different conditions.

We characterise the errors using a physical prototype of our proposed marker with the same camera's settings and environmental conditions for each exercise as shown in Figure 7.1. The camera is located 0.5 meters away from the tested marker with the same supported hardware and software equipment:

- **Computational unit**:
  - Intel(R) Core(TM) i5-4440 CPU @ 3.10GHz
  - RAM 8.00 GB
  - Video card: NVIDIA GeForce GTX 660 - 2048 MB

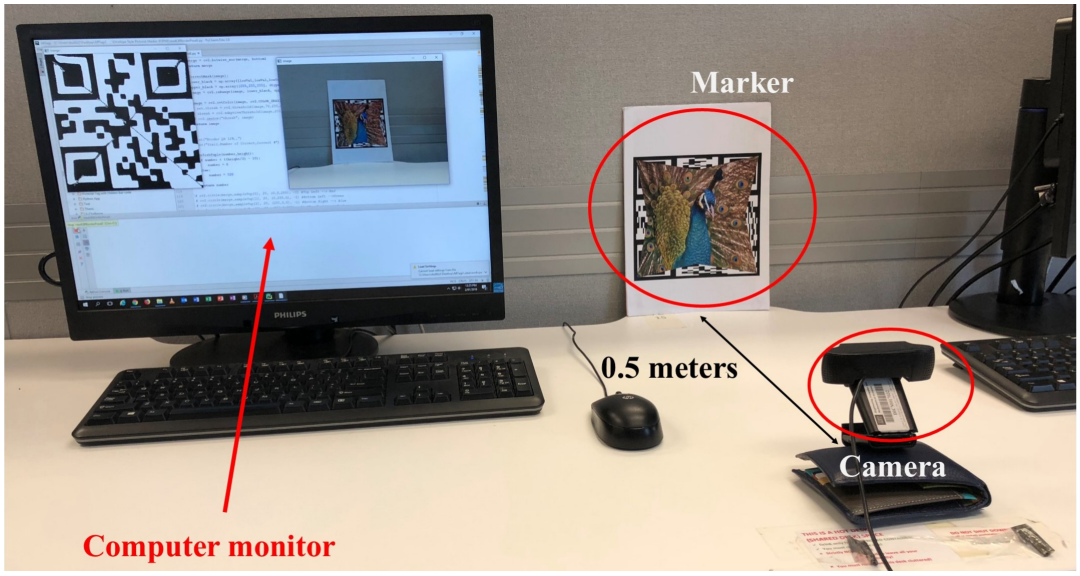
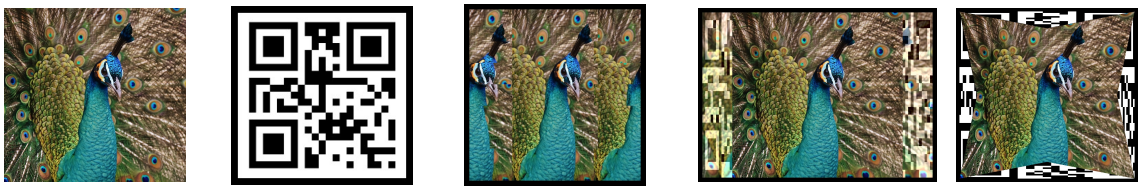


Figure 7.1: Experimental exercises set up.

- **Logitech C920 HD Pro Web Camera**:
  - Full HD 1080p video recorded (up to 1920 x 1080 pixels)
  - Auto focus
- **Software and IDE**:
  - Python
  - Pycharm

In order to make the fair comparison between different markers, we decide to use the same image texture and hidden code ("**Huy Le**") for all of the test markers as shown in the following figures.



(a) Original image (b) QR code (c) PMBC marker (d) CSPM marker (e) ESPM marker

Figure 7.2: Testable markers with the same image texture and hidden code information.

7.1 Hidden Code Detectability

In this experimental exercise, we quantify the detectable accuracy of differently proposed marker against with the result of quick response (QR) code. The decodable target information remains the same for all markers ("**Huy Le**"). For each marker, we follow these steps:

1. Step 1. Each frame captured, scan the marker to identify the hidden information. If the hidden information is correct, add 1 to the counter.
2. Step 2. Repeat the step 1 until it reaches 1000 frames.
3. Step 3. Calculate the percentage of correct information detected over 1000 frames.
4. Step 4. Repeat above steps for five times (trails).
5. Step 5. Calculate the percentage average.

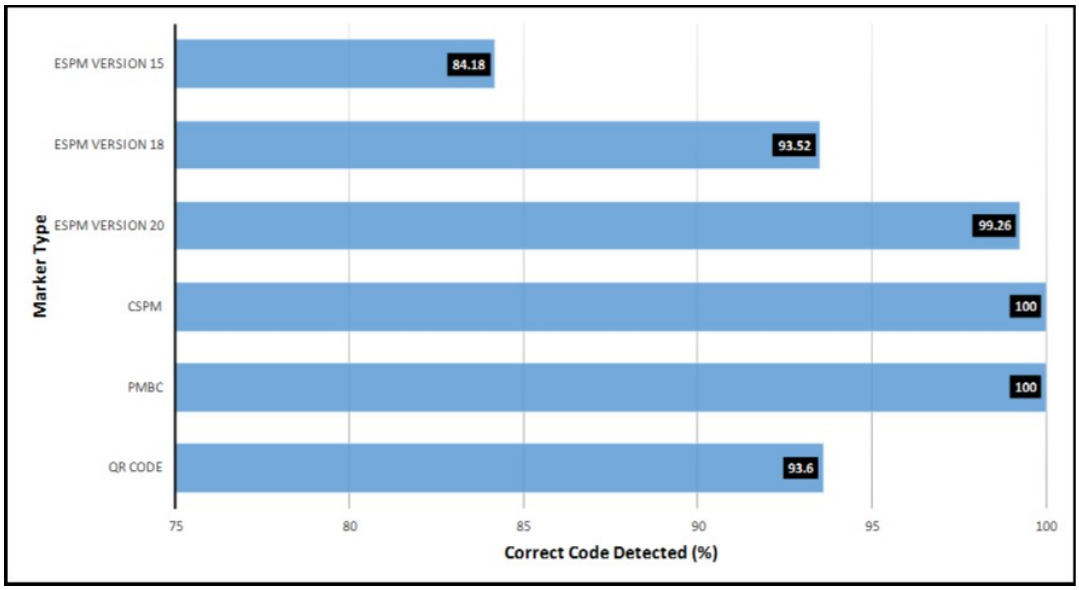


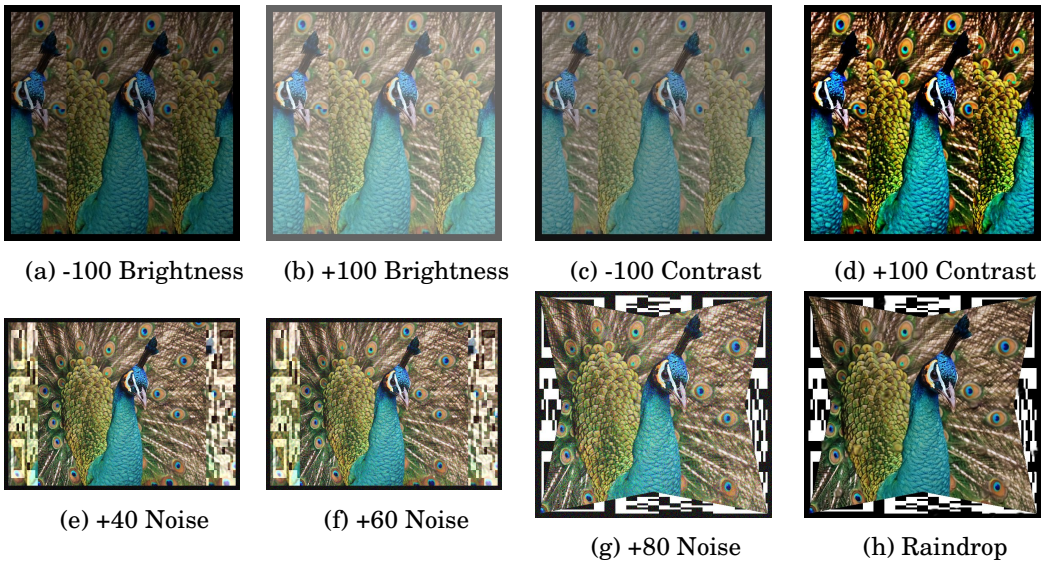
Figure 7.3: To measure the technical performance of each marker, we calculate the correct code detected frame percentage over 1000 frames captured. The average performance as shown is calculated after five trails.

Figure 7.3 shows the results. This experiment shows that our proposed marker presents an acceptable technical performance. The accuracy is declined while the code covered area is getting smaller. The reason for this is when the code covered area is getting smaller; more noise would occur during the marker resize step (ESPM version 20 gives the result of 99.260% whereas ESPM version 15 gives the result of 84.18% over five trails). However, the hidden information is still detectable given over 84% absolute in accuracy with the code covered area is up to 25% of the original marker. Again the new presentation area is large enough (up to 75% of the original

image) to provide useful information. We believe that it is a significant achievement to make our markers as an innovative approach for future AR usages. Another advantage of our proposed markers is that the differences between their performance and QR marker's result are not too significant (9.4%). However, the result shows that the ESPM version 20, CSPM and PMBC are giving higher accuracy (approximate 100%) compared to 93.6% of the QR marker's result.
Along with conducting the experiments in the perfect condition, we have also carried out three experiments to observe the ability to correctly decoded the hidden code after the following effects:

1. alternating the brightness and contrast of the tag.
2. scaling its original image.
3. adding noises and raindrops.

IrfanView tool[1] was used to alternate the sizes, apply effects, and add noises to the samples before checking the decoded results. Some IrfanView's output examples are shown in Figure 7.4.



(a) -100 Brightness (b) +100 Brightness (c) -100 Contrast (d) +100 Contrast

(e) +40 Noise (f) +60 Noise (g) +80 Noise (h) Raindrop

Figure 7.4: Four examples of PMBC markers (top row), CSPM and ESPM markers (bottom row) after various effects added by IrfanView.

7.1.1 Lighting conditions (brightness and contrast)

Different brightness (from -200 to +200) and contrast (from -100 to +100) levels were applied on top of the PMBC marker (top row of Figure 7.4) and also on CSPM and ESPM markers. The correctness result is recorded in Table 7.1. The QR code, PMBC, CSPM and ESPM version 20 can be easily recognized in most of lighting conditions. The ESPM version 18 can only detectable while the brightness is +100. The ESPM version 15 is completely failed under this exercise.

[1] http://www.irfanview.com/

Lighting conditions	QR code	PMBC	CSPM	ESPM v20	ESPM v18	ESPM v15
Brightness -200	0	100	0	0	0	0
Brightness -100	96.4	99.7	81.7	14.8	0	0
Brightness +100	99.9	99.7	100	99.5	95.9	0
Brightness +200	96.3	99.8	99.2	76.5	0	0
Contrast -100	99.9	100	100	99.7	0	0
Contrast -50	82.9	99.2	100	84.9	0	0
Contrast +50	99.8	99.7	100	45.8	0	0
Contrast +100	93.6	72.2	100	87.1	0	0

Table 7.1: Quality of hidden code detectability affected by changes in brightness and contrast (in %)

7.1.2 Different scaling

We also rescaled the test target markers to many different sizes (in inches): 11×11, 10×10, 9×9, 8×8, 7×7, 6×6. For each marker, we collected the hidden code detectability again, results are shown in Table 7.2.

Sizes	QR code	PMBC	CSPM	ESPM v20	ESPM v18	ESPM v15
6×6	100	99.5	0	0	0	0
7×7	100	100	0	0	0	0
8×8	100	72.5	3.4	0	0	0
9×9	100	100	8.4	0	0	0
10×10	100	100	42.5	0	0	0
11×11	100	99.92	87.48	98.2	92.5	83.4

Table 7.2: Quality of hidden code detectability affected by scaling of markers (in %)

7.1.3 Noise and raindrop effects

We have also applied some other IrfanView's built-in effects: random noises and raindrops on the markers (bottom row of Figure 7.4). The correctness of hidden code at each test is collected and showed in Table . From the results, the hidden code of QR code, PMBC and CSPM markers are still correctly detectable and remaining over 70% whereas the ESPM markers are completely failed.

Noises	QR code	PMBC	CSPM	ESPM v20	ESPM v18	ESPM v15
Noise +20	98	100	98.5	0	0	0
Noise +40	99.9	100	95.6	0	0	0
Noise +60	82.5	100	99.7	0	0	0
Noise +80	98.6	100	100	0	0	0
Raindrop	98.1	100	70.5	0	0	0

Table 7.3: Quality of hidden code detectability affected by noises and raindrops (in %)

7.2 Error Detection and Correction

Next, we evaluate the error correction of each proposed markers and quick response (QR) code. The failure level is started from 10% to 60% of the tested marker display. We then apply each failure level for each tested marker. For each marker, we follow these steps:

1. Step 1. Generate ten different failure markers based on the current failure level but given random failure location of the marker display.
2. Step 2. Scan each failure marker and identify its hidden code detected accuracy by repeating exercise steps in Section 7.1.
3. Step 3. Calculate the accuracy percentage of failure markers.
4. Step 4. Repeat above steps for each failure level (10% to 60%).

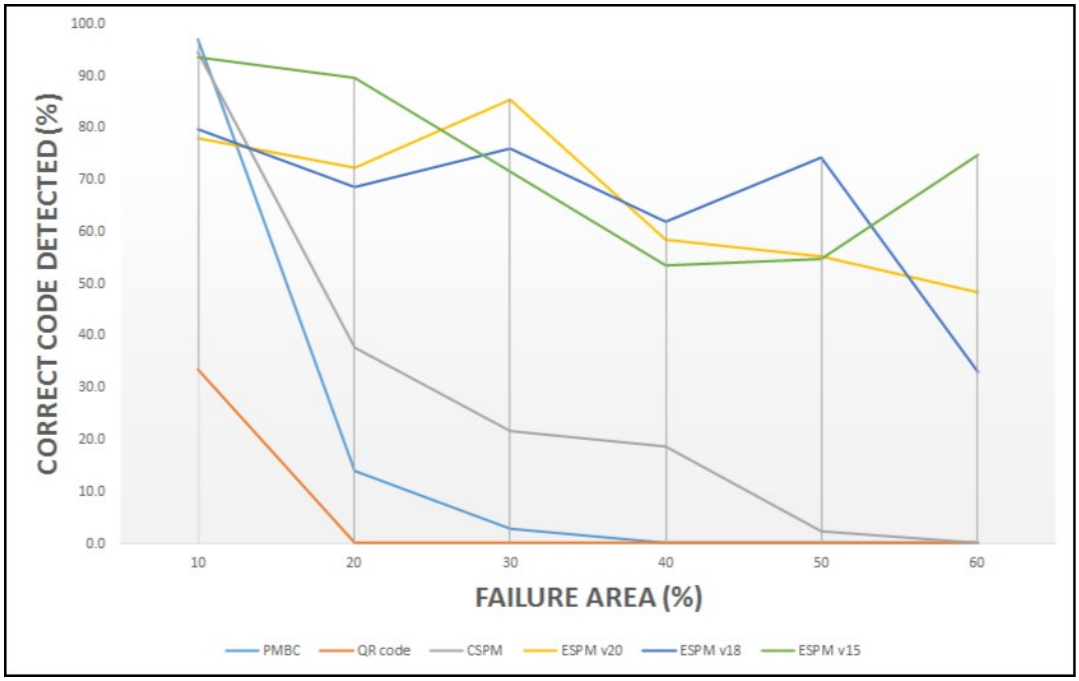
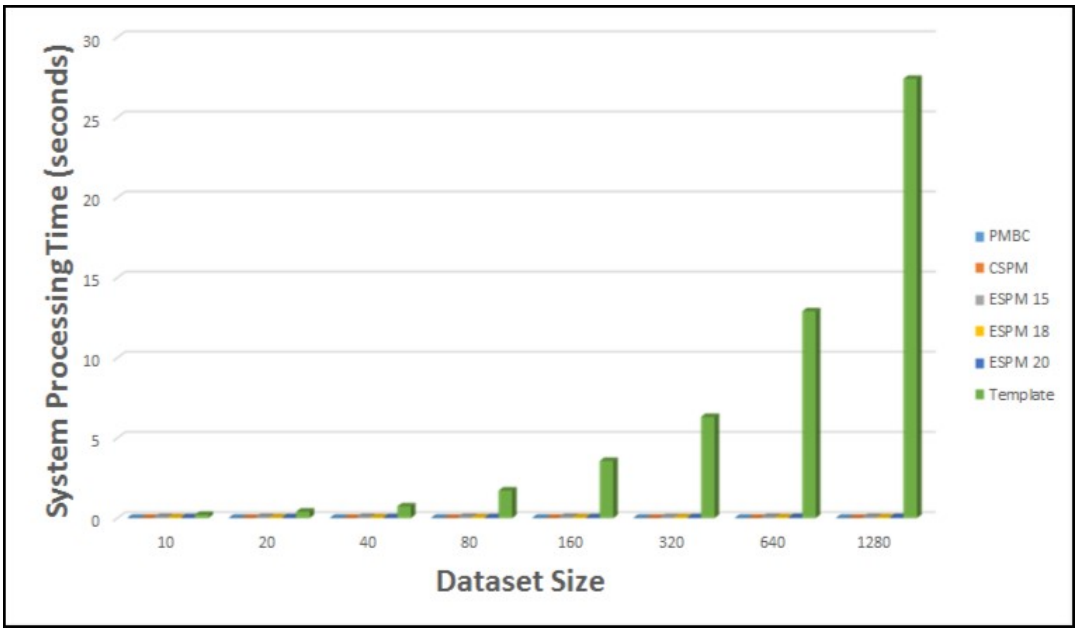


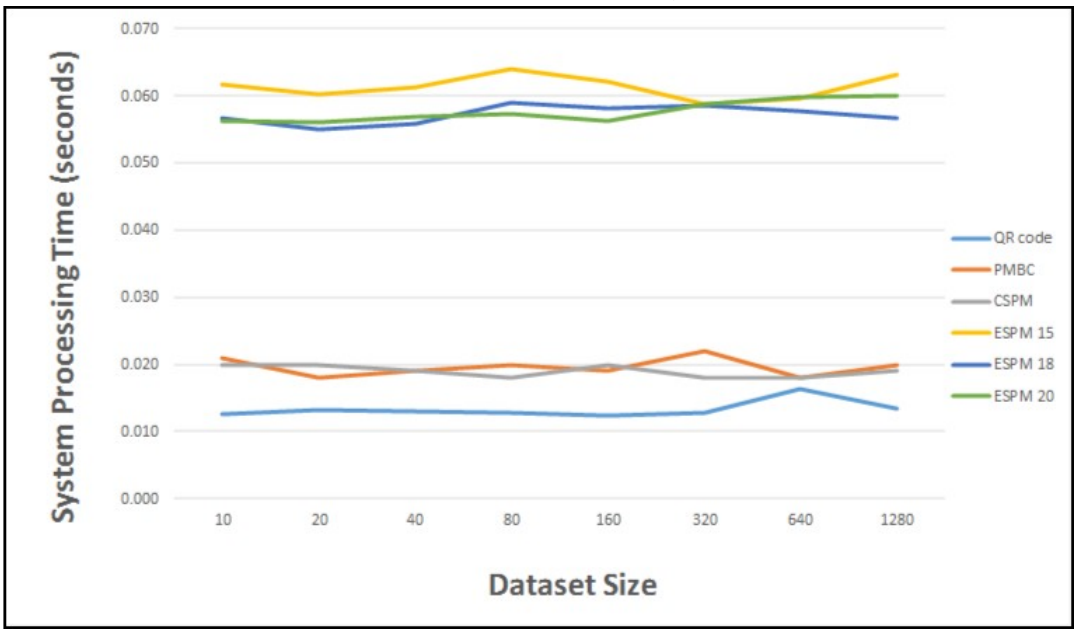
Figure 7.5: We present the error correction statistics for each of proposed markers and QR code under the same given conditions. The failure level used in this exercise is started from 10% to 60% of the marker's display area.

Figure 7.5 shows detailed results of error correction of our proposed marker against with QR code. All accuracies of the tested markers are declined while the failure level is incremented. We see that QR code fails to recover the error if the failure level is getting over 20% of the display. However, our proposed markers are still detectable given over 50% correctness for up to 50% of the failure level. Again this statistical result tells us that our proposed marker is more reliable in error correction capability as well as detectability.

7.3 System Processing Performance



(a) Proposed markers vs template marker system processing performance



(b) Proposed markers vs QR code system processing performance

Figure 7.6: The system performance of our proposed markers is greater than QR marker (Figure 7.6b) but much slower than the template marker (Figure 7.6a) which gives an exponential growth each time the data set size is doubled. The detailed result is shown in Table 7.4.

As stated above, template markers would consume much larger processing power and time while the size of data set gets increased each time. In this investigational exercises, we present

the system processing performance for each of proposed markers against with QR code marker and template marker. The size of test data set is started from the size of 10 and doubled each time until it reaches the size of 1280.

Figure 7.6 shows the system processing performances of proposed markers against template marker (Figure 7.6a) and QR code marker (Figure 7.6b). The ESPM is five times slower than QR code as it takes time to retrieve the internal image and reconstruct the QR code. The PMBC and CSPM are also giving a slower performance than QR code (0.007 seconds approximate). However, we could say that our proposed markers give equivalent performance to QR code as $< 0.1seconds$ is not a huge difference in practice. On the other hand, the proposed markers have a significant result compared to template marker which gives an exponential growth each time the data set size is double. Template marker would take over 27 seconds to process when the data set size is larger than 1200 whereas only 0.03 seconds on average for our proposed markers.

# Items	QR code	Template	PMBC	CSPM	ESPM v15	ESPM v18	ESPM v20
10	0.013	0.208	0.021	0.020	0.062	0.057	0.056
20	0.013	0.399	0.018	0.020	0.060	0.055	0.056
40	0.013	0.731	0.019	0.019	0.061	0.056	0.057
80	0.013	1.718	0.020	0.018	0.064	0.059	0.057
160	0.012	3.568	0.019	0.020	0.062	0.058	0.056
320	0.013	6.318	0.022	0.018	0.059	0.059	0.059
640	0.016	12.906	0.018	0.018	0.060	0.058	0.060
1280	0.013	27.412	0.020	0.019	0.063	0.057	0.060

Table 7.4: The proposed markers give the equivalent performance result with QR code and are much higher than template marker while the size of the data set is increased.

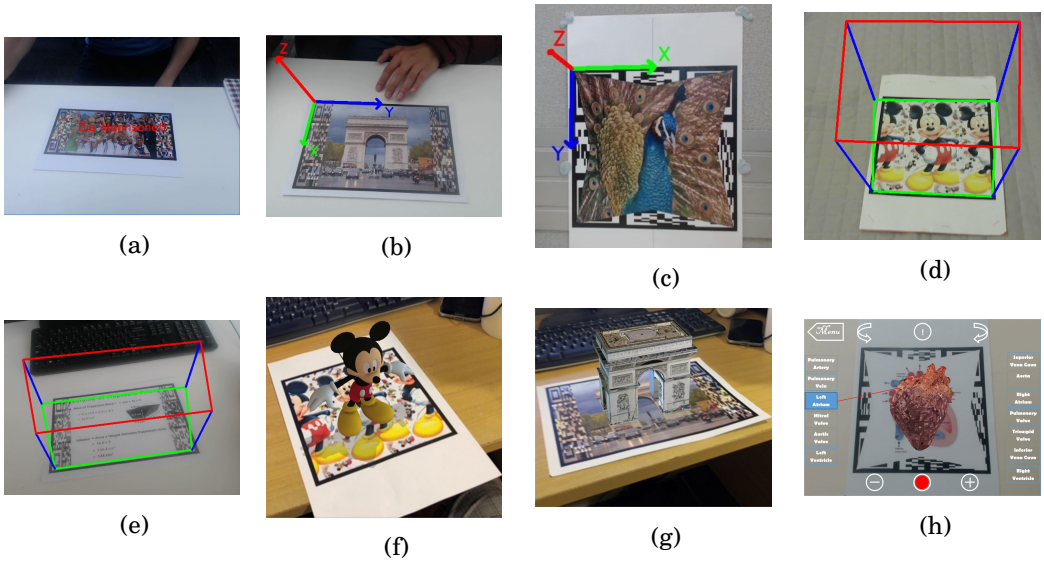
7.4 Augmented Reality Demos

The previous experiments show that our proposed marker has achieved the significant results against data marker and template marker. However, the ESPM is giving better error detection and correction ability compared to CSPM and PMBC; but ESPM is only working fine under the fine lighting condition and perfect scaling. Thus, the ESPM is suitable for education application where the detectability accuracy is the priority. In this section, we present the final result displayed on the screen of the computer as shown in Figure 7.7. The specific demo video links are presented in Table 7.5 and will be described more deeply in Appendix A. The AR applications were programmed with Python programming language [86] with the help of various open-source libraries such as: OpenCV [72] and OpenGL [87]. Our proposed markers can easily display the real-time pose estimation (Figure 7.7b, Figure 7.7c) showing 3D coordinate axis (X, Y, Z). They can also apply the same theory to render a simple shape of a cube in 3D (Figure 7.7d, Figure 7.7e). The experiment shows that the proposed markers are also capable of rendering the 3D object texture

Demo Video Name	URL
PMBC hidden code decrypted demo	https://www.youtube.com/watch?v=fOwu318KYOw
CSPM Hidden Code Decryption	https://www.youtube.com/watch?v=ciYqOKe8i1U
CSPM Pose Estimation	https://www.youtube.com/watch?v=DYCltnvkyYg
CSPM 3D object rendered	https://www.youtube.com/watch?v=jgAEj94cMAk

Table 7.5: Application YouTube demonstration links

(Figure 7.7f, Figure 7.7g). Figure 7.7h presents a mock-up AR application for medical training purpose while the students can view the virtual heart in 3D. What is more, this application helps students to enhance their imagination and offers the low-cost non-physical prototypes for training purposes.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 7.7: Our proposed markers could easily do the real time pose estimation (7.7b, 7.7c) and render 3D object on the top (7.7d, 7.7e, 7.7f, 7.7g). They also provide the great ability for users to interact with virtual world (7.7h) and could be a great opportunity for future commercial and different other purposes.

CHAPTER 8

CONCLUSIONS, LIMITATIONS AND FUTURE WORKS

*Parts of this chapter has been published in **paper 1** and **paper 2** listed in publication list*

After the introduction of several different new augmented reality (AR) markers and extensively investigating their characteristics, it clearly states that our proposed markers could benefit in both consumer-level and technical aspects. However, there are still few drawbacks, and limitations need to be concerned and improved in the future. In this chapter, we summarise the demonstrated results and contributions of this thesis and state out the constraints and discuss the future improvement research works.

8.1 Summary

This thesis objective aims to combine the technical advantages of both data markers and template markers to solve other drawbacks. Most today's similar applications are using either pictorial markers or data markers; each has its disadvantage. Firstly, pictorial markers have meaningful appearances, but they are computationally expensive and indecisively identified by computers. Secondly, data markers contain decodable numeric data, but they look uninteresting and uninformative. In another word, our proposed methods do not provide only the useful graphical content presentation, but also the capability to improve system performance and security issue. This has been done in the context of three AR markers: pictorial marker with hidden bar-code (PMBC), curtain styled pictorial marker (CSPM), and envelope styled pictorial marker (ESPM). Each marker could hold several different encoded information and also provides the self-error detection and correction capability.

The PMBC uses autostereogram theory presented in Section 3.1 to conceal a single one-dimensional bar-code in a graphical content. The PMBC is capable of storing up to 1,048,576 different hidden numbers, and the encoded regions occupy only 50% of the original image area. The Hamming codes [10] described in Chapter 3.2.2.2 was used in order to detect and correct any occurred errors during the decoding process. In the experimental results presented in Chapter 7, PMBC gives the detectability accuracy of over 90% on average in most of the test cases with the acceptable system performance.

The curtain style pictorial marker (CSPM) is the second proposed marker which could embed quick response (QR) code with a colour image. The QR code is divided into two different regions and hidden in CSPM left and right edges. The experimental results show that CSPM does not only give the equivalent system performance compared with QR code, but it also keeps the original image unchanged. The advantage of self- error correction gives CSPM higher chance to recover lost information in case of physical failure. Therefore, this design is believed to be a promising approach for use in applications where the traditional data marker would distract from the content presented. The envelope style pictorial marker (ESPM) is another type of introduced markers which could also embed QR code with a colour image but presents more extensive visual information presentation (85% of the original image) to CSPM. The ESPM also provides high error detection and correction accuracy (over 50% correctness for up to 50% of the failure level) compared to other proposed markers and the QR code.

Finally, this thesis has presented some software demos of the proposed markers by using Python programming language and open-source libraries. The demos show that all of our proposed markers be able to detect the hidden code and render the virtual object correctly. However, due to the time constraints, we only present the proposed prototype running under personal computer application with the normal high-standard cameras. Thus, these designs, in fact, can be improved to allow to experience on mobile devices such as smartphones and tablets which provide higher resolution cameras.

8.2 Contributions

The objectives of this thesis have been accomplished based on three research questions introduced in Chapter 1. Now we summarize all these together and see how the presented research contributions answer to the described research questions:

How to hide different bar-codes and QR code (2D) in template images?
Firstly, we described the concept of pictorial marker with hidden bar-code (PMBC) (see Chapter 5).

This proposed marker used the theory of autostereograms presented in Section 3.1 to conceal a multi-level bar-code optically. The binary codes are optically decorated on two edges of the marker; thus, most of the image texture depicted on the marker are reserved. The monotone and blank regions are automatically filled with the textures to maximize the decryption accuracy.

Secondly in order to increase the encode information storage capacity, we came up with the ideas of curtain style pictorial marker (CSPM) and envelope style pictorial marker (ESPM). These marker are capable to hide the QR code in a graphical content. CSPM design idea is similar to PMBC which is concealing the QR code information on two edges of the marker. The decryption process used image subtraction technique to determine the differences between the hidden code regions and the central image texture. ESPM is also used to hide the QR code information in the graphical content, but not on four edges of the marker instead of two edges. This technique is used in order to improve the presentation capability of the marker and maximize the self-error detection and correction.

Will the new proposed marker concepts have the equivalent technical performance to bar-code marker?

We believe that our proposed markers have the equivalent technical performance to bar-code marker (1D and 2D bar-code). In chapter 7, we have conducted several different experiments to identify the technical performance of each proposed markers against the bar-code under different conditions. Most of our proposed markers give over 90% of hidden code detectability accuracy in average which approximate equivalent to the bar-code marker. These results were presented in publication 1 and publication 2 (please see the publication list) and verified by many different reviewers at the international level. Thus, the experimental results presented in this thesis are accurate and reliable to prove that the proposed marker concepts have qualified for the technical performance level.

What are the future potential application domains?

In the experimental results, our proposed markers showed a relatively robust performance under various conditions and scaling; thus, they provide a promising AR approach to be used in many application such as trading card games, educations, and advertisements. We have also designed few application demos running on personal computer platform within this thesis. Education is one of the key domains we could aim to apply where the demand of combination of the traditional training and modern e-learning methods is very high (see publication 1 in the publication list). Tourism and off-line map guide are also considered where many people have found themselves lost in newly visiting places even if there is a map on their hands (see publication 7 in the publication list).

8.3 Limitation and Future Work

While our markers show a promising technical performance result, there are still a number of existing limitations that deem this marker not fully ready for the market:

- We were unable to perform the test on different camera models. The current camera has a resolution of 1920×1080, which is considered as normal high standard whereas 2k resolution or above is on demanded for camera models being used around the world.
- The time constraints and limited budget prevented a usability test from being set up. Even though we managed to receive positive feedback from other people, stating that these markers had a better appearance than the traditional ones. It is still unsure if our proposed marker can provide a more attractive AR marker for the general public.
- The the proposed prototypes are only designed to be running as the personal computer application instead of nowadays worldwide used mobile platform.
- The result of rendered virtual objects in application demos are not stable since the idea of AR is to render stable graphics which seems like real objects.
- The decode information does not look sharp enough and may be prone to errors due the techniques discussed in the thesis were built from fundamental image processing operations.

We believe that our proposed markers described in this thesis could overcome the current technical disadvantages of AR markers. In future, we will investigate this direction to build the AR applications for the mobile devices to enhance client's AR experiences. The high-resolution cameras of mobile devices are also giving us the capability to improve the hidden code detection sensitivity and expand the visual information presentation. Furthermore, we also need to consider and pay more attention to the most suitable image processing operations which could be used to improve the marker decryption results.

BIBLIOGRAPHY

[1] I. E. Sutherland, "A head-mounted three dimensional display," in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM, 1968, pp. 757–764.

[2] R. T. Azuma, "A survey of augmented reality," *Presence: Teleoperators and virtual environments*, vol. 6, no. 4, pp. 355–385, 1997.

[3] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *IEEE Computer Graphics and Applications*, vol. 21, no. 6, pp. 34–47, 2001.

[4] E. Mendez, D. Kalkofen, and D. Schmalstieg, "Interactive context-driven visualization tools for augmented reality," in *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2006, pp. 209–218.

[5] A. Tikanmäki and J. Röning, "Markers–toward general purpose information representation," in *IROS2011 Workshop on Knowledge Representation for Autonomous Robots*, 2011.

[6] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. John Wiley & Sons, 2009.

[7] J. P. Lewis, "Fast normalized cross-correlation," in *Vision Interface*, vol. 10, no. 1, 1995, pp. 120–123.

[8] M. Billinghurst, H. Kato, and I. Poupyrev, "The magicbook-moving seamlessly between reality and virtuality," *Computer Graphics and Applications, IEEE*, vol. 21, no. 3, pp. 6–8, 2001.

[9] E. Bertino, B. C. Ooi, R. Sacks-Davis, K.-L. Tan, J. Zobel, B. Shidlovsky, and D. Andronico, *Indexing techniques for advanced database systems*. Springer Science & Business Media, 2012, vol. 8.

[10] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[12] J. Carmigniani and B. Furht, "Augmented reality: an overview," in *Handbook of augmented reality*. Springer, 2011, pp. 3–46.

[13] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[14] M.-C. J. I. S. N. R. David Furióa, Santiago González-Gancedoa, "Evaluation of learning outcomes using an educational iphone game vs. traditional game," 2015. [Online]. Available: https://www.researchgate.net/profile/M_Carmen_Juan/publication/257171501/figure/fig1/AS:297326095683586@1447899473600/Fig-1-Representation-of-the-virtuality-continuum.png

[15] T. P. Caudell and D. W. Mizell, "Augmented reality: An application of heads-up display technology to manual manufacturing processes," in *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on*, vol. 2. IEEE, 1992, pp. 659–669.

[16] M. Bajura, H. Fuchs, and R. Ohbuchi, "Merging virtual objects with the real world: Seeing ultrasound imagery within the patient," in *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2. ACM, 1992, pp. 203–210.

[17] A. L. Janin, D. W. Mizell, and T. P. Caudell, "Calibration of head-mounted displays for augmented reality applications," in *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*. IEEE, 1993, pp. 246–255.

[18] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster, "A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment," in *Wearable Computers, 1997. Digest of Papers., First International Symposium on*. IEEE, 1997, pp. 74–81.

[19] M. S.-M. S. S. A. D. T. Heather Bellini, Wei Chen, "Profiles in innovation virtual and augmented reality," 2016. [Online]. Available: http://www.goldmansachs.com/our-thinking/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf

[20] A. Linden and J. Fenn, "Understanding gartner's hype cycles," *Strategic Analysis Report Nº R-20-1971. Gartner, Inc*, 2003.

[21] D. Zakariaie, "How google glass will deliver faster, better, cheaper insights," 2014. [Online]. Available: http://www.insightinnovation.org/iiex-presentations/David_Zakariaie_Glassic_How_wearables_will_deliver_cheaper_faster_better_insights.pdf

[22] R. Furlan, "The future of augmented reality: Hololens - microsoft's ar headset shines despite rough edges," *IEEE Spectrum*, vol. 53, no. 6, pp. 21–21, June 2016.

[23] Y. Heisler, "iphone 8 said to feature exciting next-gen augmented reality tech," Fox News, 2017. [Online]. Available: http://www.foxnews.com/tech/2017/02/15/iphone-8-said-to-feature-exciting-next-gen-augmented-reality-tech.html

[24] P. Caughill, "Here's why apple's custom gpu and a11 bionic chip are utterly revolutionary," 2017. [Online]. Available: https://futurism.com/heres-why-apples-custom-gpu-and-a11-bionic-chip-are-utterly-revolutionary/

[25] D. Van Krevelen and R. Poelman, "Augmented reality: Technologies, applications, and limitations," 2007.

[26] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell, "A survey of mobile phone sensing," *IEEE Communications magazine*, vol. 48, no. 9, 2010.

[27] S. K. Feiner, "Augmented reality: A new way of seeing," *Scientific American*, vol. 286, no. 4, pp. 48–55, 2002.

[28] M. P. Bartere and V. Nikam, "Digital composting using chroma keying."

[29] V. Vlahakis, M. Ioannidis, J. Karigiannis, M. Tsotros, M. Gounaris, D. Stricker, T. Gleue, P. Daehne, and L. Almeida, "Archeoguide: an augmented reality guide for archaeological sites," *IEEE Computer Graphics and Applications*, vol. 22, no. 5, pp. 52–60, 2002.

[30] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, M. Morris, and W. Piekarski, "Arquake: An outdoor/indoor augmented reality first person application," in *Wearable computers, the fourth international symposium on*. IEEE, 2000, pp. 139–146.

[31] M. Serino, K. Cordrey, L. McLaughlin, and R. L. Milanaik, "Pokémon go and augmented virtual reality games: a cautionary commentary for parents and pediatricians," *Current opinion in pediatrics*, vol. 28, no. 5, pp. 673–677, 2016.

[32] T. SimonCo, "Ar gaming apps," 2016. [Online]. Available: http://projetar.renous.fr/wp-content/uploads/2010/03/ARQuake.jpg

[33] H.-Y. Chang, H.-K. Wu, and Y.-S. Hsu, "Integrating a mobile augmented reality activity to contextualize student learning of a socioscientific issue," *British Journal of Educational Technology*, vol. 44, no. 3, pp. E95–E99, 2013.

[34] M. Billinghurst and A. Duenser, "Augmented reality in the classroom," *Computer*, vol. 45, no. 7, pp. 56–63, 2012.

[35] R. Malaka, K. Schneider, and U. Kretschmer, "Stage-based augmented edutainment," in *International Symposium on Smart Graphics*. Springer, 2004, pp. 54–65.

[36] U. Kretschmer, V. Coors, U. Spierling, D. Grasbon, K. Schneider, I. Rojas, and R. Malaka, "Meeting the spirit of history," in *Proceedings of the 2001 conference on Virtual reality, archeology, and cultural heritage*. ACM, 2001, pp. 141–152.

[37] A. Chandra and J. Skinner, "Technology growth and expenditure growth in health care," *Journal of Economic Literature*, vol. 50, no. 3, pp. 645–680, 2012.

[38] L. Engelen. [Online]. Available: https://www.aed4.eu/

[39] R. Ohbuchi, M. Bajura, and H. Fuchs, "Case study: Observing a volume rendered fetus within a pregnant patient," in *Visualization: Proceedings of the IEEE Conference on Visualization*, vol. 5. IEEE Computer Society Press, 1998.

[40] H. Fuchs, M. A. Livingston, R. Raskar, K. Keller, J. R. Crawford, P. Rademacher, S. H. Drake, A. A. Meyer *et al.*, "Augmented reality visualization for laparoscopic surgery," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 1998, pp. 934–943.

[41] C. Bichlmeier, F. Wimmer, S. M. Heining, and N. Navab, "Contextual anatomic mimesis hybrid in-situ visualization method for improving multi-sensory depth perception in medical augmented reality," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 2007, pp. 129–138.

[42] "Windshield tv screen to aid blind flying," vol. 102, no. 6, p. 101, 1954.

[43] F. H. Previc and W. R. Ercoline, *Spatial disorientation in aviation*. Aiaa, 2004, vol. 203.

[44] K. L. Hiatt, C. E. Rash, E. S. Harris, and W. H. Gilberry, "Apache aviator visual experiences with the ihadss helmet-mounted display in operation iraqi freedom," ARMY AEROMEDICAL RESEARCH LAB FORT RUCKER AL, Tech. Rep., 2004.

[45] M. Prigg. (2014) Google glass for war: The us military funded smart helmet that can beam information to soldiers on the battlefield. [Online]. Available: http://www.dailymail.co.uk/sciencetech/article-2640869/Google-glass-war-US-military-reveals-augmented-reality-soldiers.html

[46] H. Kato, M. Billinghurst, S. Weghorst, and T. Furness, "A mixed reality 3d conferencing application," *Human Interface Technology Laboratory*, 1999.

[47] (2017) Vuforia. [Online]. Available: https://www.vuforia.com/

[48] T. J. Soon, "Qr code," *Synthesis Journal*, vol. 2008, pp. 59–78, 2008.

[49] H. Kato and K. T. Tan, "Pervasive 2d barcodes for camera phone applications," *IEEE Pervasive Computing*, vol. 6, no. 4, 2007.

[50] R. Stevenson, "Laser marking matrix codes on pcbs," *Printed Circuit Design and Manufacture*, vol. 22, no. 12, p. 32, 2005.

[51] Y. P. Wang, "System for encoding and decoding data in machine readable graphic form," Sep. 7 1993, uS Patent 5,243,655.

[52] D. Brewster, *The Kaleidoscope: Its History, Theory and Construction: with Its Application to the Fine and Useful Arts*. J. Murray, 1858.

[53] C. W. Tyler and M. B. Clarke, "The autostereogram," in *SPIE Stereoscopic Displays and Applications*, vol. 1258, 1990, pp. 182–196.

[54] T. J. Baccei and R. Salitsky, "Random dot stereogram and method for making the same," Dec. 6 1994, uS Patent 5,371,627.

[55] G. Levine and G. W. Priester, *Hidden Treasures: 3-D Stereograms*. Sterling Publishing Company, Inc., 2008.

[56] I. J. Lee, J. Y. Min, H. Lee, S. Min, and S. Kim, "A scaling parameter optimization of watermarking using autostereograms as random dot images," in *Visual Information Engineering, 2003. VIE 2003. International Conference on*. IET, 2003, pp. 314–316.

[57] M. Laxer, J. Cohen, and L. Press, "An expanded guide to the keystone stereogram cards," *Journal of Behavioral Optometry*, vol. 2, no. 3, pp. 59–66, 1991.

[58] A. Katrusiak, "Crystallographic autostereograms," *Journal of Molecular Graphics and Modelling*, vol. 19, no. 3, pp. 363–367, 2001.

[59] R. Kimmel, "3d shape reconstruction from autostereograms and stereo," *Journal of Visual Communication and Image Representation*, vol. 13, no. 1-2, pp. 324–333, 2002.

[60] D. Scharstein and R. Szeliski, "Middlebury stereo vision page," *Online at http://www.middlebury.edu/stereo*, vol. 2, 2002.

[61] R. C. Gonzalez, R. E. Woods *et al.*, "Digital image processing," 1992.

[62] M. H. Chowdhury and W. D. Little, "Image thresholding techniques," in *Communications, Computers, and Signal Processing, 1995. Proceedings., IEEE Pacific Rim Conference on*. IEEE, 1995, pp. 585–589.

[63] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[64] D. Bradley and G. Roth, "Adaptive thresholding using the integral image," *Journal of Graphics Tools*, vol. 12, no. 2, pp. 13–21, 2007.

[65] J. S. Plank *et al.*, "A tutorial on reed-solomon coding for fault-tolerance in raid-like systems," *Softw., Pract. Exper.*, vol. 27, no. 9, pp. 995–1012, 1997.

[66] P. Sturm, "Pinhole camera model," in *Computer Vision*. Springer, 2014, pp. 610–613.

[67] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[68] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[69] S. Rajasekar, P. Philominathan, and V. Chinnathambi, "Research methodology," *arXiv preprint physics/0601009*, 2006.

[70] R. H. Von Alan, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[71] R. Bogdan and S. K. Biklen, "Qualitative research for education," 1992.

[72] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[73] M. Summerfield, "Programming in python 3," 2009.

[74] Q. N. Islam, *Mastering PyCharm*. Packt Publishing Ltd, 2015.

[75] L. OpenCV, "Computer vision with the opencv library," *GaryBradski & Adrian Kaebler-O'Reilly*, 2008.

[76] K. Konolige, "Small vision systems: Hardware and implementation," in *Proceedings of the International Symposium on Robotics Research*, vol. 8, Kanagawa, Japan, Oct 1997, pp. 203–212.

[77] S. Birchfield and C. Tomasi, "Depth discontinuities by pixel-to-pixel stereo," *International Journal of Computer Vision*, vol. 35, no. 3, pp. 269–293, 1999.

[78] G. Gimel'farb, "Stereo terrain reconstruction by dynamic programming," *Handbook of Computer Vision and Applications. Signal Processing and Pattern Recognition.*, vol. 2, pp. 505–530, 1999.

[79] R. Gong, "Belief Propagation Based Stereo Matching with Due Account of Visibility Conditions," Master's thesis, The University of Auckland, New Zealand, Computer Science department, 20011.

[80] V. Kolmogorov and R. Zabih, "Multi-camera Scene Reconstruction via Graph Cuts," in *Proceedings of the European Conference on Computer Vision*, vol. 2352, Copenhagen, Germany, 27 May - 2 Jun 2002, pp. 82–96.

[81] M. Nguyen, Y. H. Chan, P. Delmas, and G. Gimel'farb, "Symmetric dynamic programming stereo using block matching guidance," in *Image and Vision Computing New Zealand (IVCNZ), 2013 28th International Conference of*. IEEE, 2013, pp. 88–93.

[82] "Middlebury stereo vision webpage," 2001. [Online]. Available: http://vision.middlebury.edu/stereo/

[83] P. Kieseberg, M. Leithner, M. Mulazzani, L. Munroe, S. Schrittwieser, M. Sinha, and E. Weippl, "Qr code security," in *Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia*. ACM, 2010, pp. 430–435.

[84] J.-H. Kim, "Contour approximation method for representing a contour of an object," Jun. 30 1998, uS Patent 5,774,595.

[85] J. Brown, "Zbar bar code reader," 2007. [Online]. Available: http://zbar.sourceforge.net/

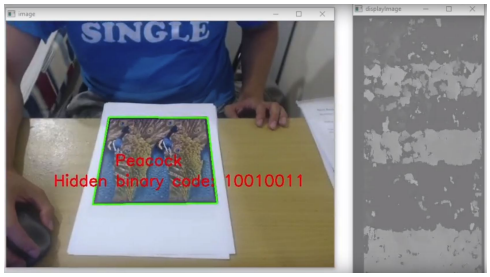
[86] G. Van Rossum *et al.*, "Python programming language." in *USENIX Annual Technical Conference*, vol. 41, 2007, p. 36.

[87] M. Woo, J. Neider, T. Davis, and D. Shreiner, *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

APPENDIX A: EXPERIMENT RESULTS VIDEO DEMOS

The experiment results described in this thesis involves the processing of video materials. The video results are not easy to express in either numerical or pictorial form. Hence the YouTube video clips links are described in the Section 7.4. This appendix section contains the videos descriptions found in the listed YouTube links above.

- **PMBC hidden code decrypted demo** (`https://www.youtube.com/watch?v=fOwu318KYOw`)**:**

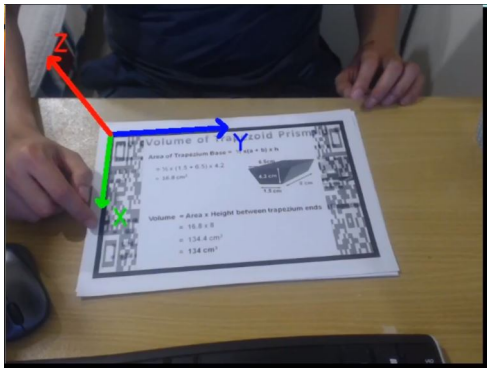


This video demonstrates hidden code decryption processing of pictorial marker with hidden barcode (PMBC) presented in Chapter 5. The video compares the decrypted codes of different marker textures and presents a sample of 3D rendered model which could be used as an example of future augmented reality usage.

- **CSPM Hidden Code Decryption** (`https://www.youtube.com/watch?v=ciYq0Ke8i1U`)**:**

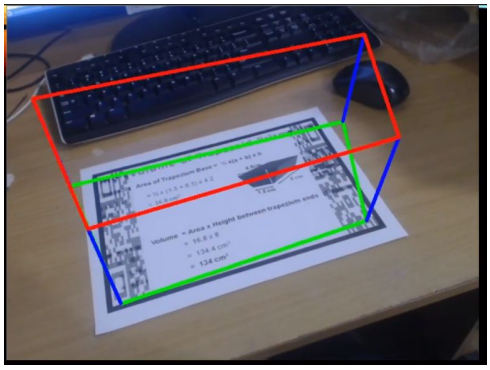
This video demonstrates hidden QR code decryption processing of curtain style pictorial marker (CSPM) presented in Chapter 6. The video compares the decrypted codes of different marker textures under different conditions and orientations.

- **CSPM Pose Estimation** (`https://www.youtube.com/watch?v=DYCltnvkyYg`)**:**



This video demonstrates the pose estimation of CSPM presented in Chapter 6. It also shows that the CSPM has the capability to render virtual information in the right orientation form.

- **CSPM 3D object rendered** (`https://www.youtube.com/watch?v=jgAEj94cMAk`)**:**



This video shows the sample of 3D object rendered on the top of CSPM marker.

APPENDIX B: PICTORIAL MARKER WITH HIDDEN BAR-CODE SOURCE CODE (WRITTEN IN PYTHON)

This appendix contains the source code written in Python to demonstrate in details about marker creation and hidden information decryption of pictorial marker with hidden bar-code

Pictorial Marker with Hidden Bar-code creation

Step 1: Create new marker with two mirrored regions from the imported image texture

```
1  import numpy as np
2  import cv2
3
4  mainImage = "micky.png"
5  outputImage = "mickyPMBC.jpg"
6
7  tempImage = cv2.imread(mainImage);
8  height, width, depth = tempImage.shape
9
10 textureImage = np.zeros((height,height,3), np.uint8)
11
12 wGap = (height - width)/2;
13
14 if wGap < 0:
15         wGap = 0
16         tempImage = cv2.resize(tempImage, (height, height))
17         height, width, depth = tempImage.shape
18
19 for h in xrange(0, height):
20         for w in xrange(0, width):
21                 textureImage[h, wGap + w] = tempImage[h,w]
22
```

```
23 for h in xrange(0, height):
24         for w in xrange(0, height/4):
25                 textureImage[h, w] = textureImage[h,height/2+w]
26                 textureImage[h, 3*height/4 + w] = textureImage[h,height/4+w]
27
28 textureImage = cv2.resize(textureImage, (512, 512))
29 height, width, depth = textureImage.shape
30 originalWidth = width
```

Step 2: Create marker stereogram to hide the encoded binary code

```
1  textureImage = createStereogram(textureImage)
2  stringCode = '10011011' #micky
3  stepLevel=16
4  numDisp=16
5  bSize=11
6
7  #divide the string by two
8  str1 = stringCode[:4]
9  str2 = stringCode[4:8]
10
11 #encode string 1
12 xStr1 = ''.join([str(bin(int(i, 2) & int(str1, 2)).count('1') % 2) for i in G])
13 #encode string 2
14 xStr2 = ''.join([str(bin(int(i, 2) & int(str2, 2)).count('1') % 2) for i in G])
15
16 finalEncodeString = '0'+xStr1+xStr2+'1'
17
18 def createStereogram(textureImage):
19         height, width, depth = textureImage.shape
20         smallHeight = height/stepLevel
21         for h in xrange(0, height):
22                 k = 0
23                 if finalEncodeString[h / smallHeight] == '1':
24                         k = 12
25
26                 for w in xrange(0, height/4 + k):
27                         if w < height/4:
28                                 textureImage[h, w] = textureImage[h,height/2 + w - k]
```

```
29                          textureImage[h, 3*height/4 + w - k] = textureImage[h,height/4
30              + w]
31          return textureImage
```

Step 3: Adding background to remove noise and reconstruct the marker stereogram

This is the optional step in order to improve the hidden code decryption accuracy.

```
1  def stereoMatch(textureImage):
2          grayTexture = cv2.cvtColor(textureImage, cv2.COLOR_BGR2GRAY)
3          height, width = grayTexture.shape
4          imgL = grayTexture[0:height, 0:height/2-1]
5          imgR = grayTexture[0:height, height/2:height-1]
6          stereo = cv2.StereoBM_create(numDisparities=numDisp, blockSize=bSize)
7          disparity = stereo.compute(imgL,imgR)
8          return disparity
9
10
11 originalImage = textureImage.copy()
12
13 disparity = stereoMatch(textureImage)
14
15 height, width = disparity.shape
16
17 blankIdentificationImg = np.zeros((height,width,1), np.uint8) #with 1 chanels
18
19 for h in xrange( bSize/2, height-bSize/2):
20         for w in xrange(numDisp+bSize/2, width-bSize/2):
21                 if disparity[h][w] < 0:
22                         blankIdentificationImg[h][w] = 255 #white
23
24 # noise removal
25 kernel = np.ones((3,3),np.uint8)
26 opening = cv2.morphologyEx(blankIdentificationImg,cv2.MORPH_OPEN,kernel,
27 iterations = 2)
28
29 # sure background area
30 sure_bg = cv2.dilate(opening,kernel,iterations=5)
31
```

```
32 unknown = sure_bg.copy()
33
34 sure_bg = cv2.blur(sure_bg,(35,35))
35
36 colourBG = cv2.imread(backgroundImage);
37 colourBG = cv2.resize(colourBG, (512, 512))
38
39 # draw unknow region on original image
40 height, width = disparity.shape
41 for h in xrange( 0, height):
42         for w in xrange(0, width):
43                 if sure_bg[h][w] > 0 :
44                         value = 1.0 * sure_bg[h][w] / 255.0
45                         originalImage[h][w] = value * colourBG[h][w] +
46             (1-value)*originalImage[h][w]
47                         originalImage[h][w+originalWidth/2] = value *
48             colourBG[h][w+originalWidth/2] + (1-value)*
49             originalImage[h][w+originalWidth/2]
50
51 textureImage = createStereogram(originalImage)
```

Step 4: Adding the border

```
border = 16
height, width, depth = textureImage.shape #Total pixel number: img.size
borderImage = img = np.zeros((height+2*border,width+2*border,3), np.uint8)
borderImage[border:height+border,border:width+border]=textureImage
cv2.imwrite(outputImage, borderImage)
```

Pictorial Marker with Hidden Bar-code hidden information decryption

Step 1: Retrieve the internal image from the captured frame

```
1 import numpy as np
2 import cv2
3
4 stepLevel=16
5 numDisp=16
```

```
6  bSize=11
7  stereo = cv2.StereoBM_create(numDisparities=numDisp, blockSize=bSize)
8  # the encoding matrix
9  G = ['1101', '1011', '1000', '0111', '0100', '0010', '0001']
10 # the parity-check matrix
11 H = ['1010101', '0110011', '0001111']
12 Ht = ['100', '010', '110', '001', '101', '011', '111']
13 # the decoding matrix
14 R = ['0010000', '0000100', '0000010', '0000001']
15
16 #Read video streams
17 cap = cv2.VideoCapture(0) # put file name here to read from a video file
18
19 while(True):
20         ret, image = cap.read()
21         gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
22         gray = cv2.bilateralFilter(gray, 11, 17, 17)
23
24     edged = cv2.Canny(gray, 30, 200)
25
26         # find contours in the edged image, keep only the largest
27         # ones, and initialize our screen contour
28
29         im2, cnts, hierarchy = cv2.findContours(edged.copy(), cv2.RETR_TREE,
30     cv2.CHAIN_APPROX_SIMPLE)
31         cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
32         screenCnt = None
33
34     # loop over our contours
35         for c in cnts:
36                 if cv2.contourArea(c)>10000:  # remove small areas like noise etc
37                         # approximate the contour
38                         peri = cv2.arcLength(c, True)
39                         approx = cv2.approxPolyDP(c, 0.04 * peri, True)
40
41                         # if our approximated contour has four points, then
42                         # we can assume that we have found our screen
43                         if len(approx) == 4:
```

```
44                                  screenCnt = approx
45                                  #break
46
47
48      if screenCnt is not None:
49
50                  # ##########################################
51                  pts = screenCnt.reshape(4, 2)
52
53                  rect = np.zeros((4, 2), dtype = "float32")
54
55                  # the top-left point has the smallest sum whereas the
56                  # bottom-right has the largest sum
57                  s = pts.sum(axis = 1)
58                  rect[0] = pts[np.argmin(s)]
59                  rect[2] = pts[np.argmax(s)]
60
61                  # compute the difference between the points -- the top-right
62                  # will have the minumum difference and the bottom-left will
63                  # have the maximum difference
64                  diff = np.diff(pts, axis = 1)
65                  rect[1] = pts[np.argmin(diff)]
66                  rect[3] = pts[np.argmax(diff)]
67
68                  # now that we have our rectangle of points, let's compute
69                  # the width of our new image
70                  (tl, tr, br, bl) = rect
71                  widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
72                  widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
73
74                  # ...and now for the height of our new image
75                  heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
76                  heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
77
78                  # take the maximum of the width and height values to reach
79                  # our final dimensions
80                  maxWidth = max(int(widthA), int(widthB))
81                  maxHeight = max(int(heightA), int(heightB))
```

```
82
83                  # construct our destination points which will be used to
84                  # map the screen to a top-down, "birds eye" view
85                  dst = np.array([
86                          [0, 0],
87                          [maxWidth - 1, 0],
88                          [maxWidth - 1, maxHeight - 1],
89                          [0, maxHeight - 1]], dtype = "float32")
90
91                  # calculate the perspective transform matrix and warp
92                  # the perspective to grab the screen
93                  M = cv2.getPerspectiveTransform(rect, dst)
94                  warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
95
96
97                  dst = cv2.resize(warp, (544, 544))
98                  roi = dst[16:512+16, 16:512+16]
99
100         disparity = stereoMatch(roi)
101
102         textFound = readDisparity(disparity)
103
104 cap.release()
```

Step 2: Matching the stereo images

```
1  #This method will be called in Step 1 code - line number 100
2  def stereoMatch(textureImage):
3          grayTexture = cv2.cvtColor(textureImage, cv2.COLOR_BGR2GRAY)
4          height, width = grayTexture.shape
5          grayTexture = cv2.medianBlur(grayTexture, 5)
6          imgL = grayTexture[0:height, 0:height/2-1]
7          imgR = grayTexture[0:height, height/2:height-1]
8          stereo = cv2.StereoBM_create(numDisparities=numDisp, blockSize=bSize)
9          disparity = stereo.compute(imgL,imgR)
10         return disparity
```

Step 3: Read the returned disparity value and output the decoded binary string

```
1  #This method will be called in Step 1 code - line number 101
2  def readDisparity(disapritymap):
3
4          height, width = disapritymap.shape
5          step = height/stepLevel
6
7          returnValue=''
8
9          for h in xrange(0, height, step):
10                 roi = disapritymap[h:h+step, numDisp+bSize/2:width-1]
11                 hist,bins = np.histogram(roi.ravel(),255,[1,200])
12                 for x in xrange(0, len(hist)):
13                         if hist[x] == hist.max():
14                                 modeValue = bins[x]
15                 if modeValue < 50:
16                         returnValue = returnValue + '0'
17                 else:
18                         returnValue = returnValue + '1'
19
20         firstBit = returnValue[0]
21         lastBit = returnValue[15]
22         front = returnValue[1:8]
23         back = returnValue[8:15]
24
25         if firstBit is '0' and lastBit is '1':
26                 return '' + correctError(front) + correctError(back)
27         else:
28                 return 'NaN'
29
30 def correctError(x):
31         z = ''.join([str(bin(int(j, 2) & int(x, 2)).count('1') % 2) for j in H])
32         if int(z, 2) > 0:
33                 e = int(Ht[int(z, 2) - 1], 2)
34         else:
35                 e = 0
36
37         # correct the error
38         if e > 0:
```

```
39                  x = list(x)
40                  x[e - 1] = str(1 - int(x[e - 1]))
41                  x = ''.join(x)
42
43          p = ''.join([str(bin(int(k, 2) & int(x, 2)).count('1') % 2) for k in R])
44          return p
```

APPENDIX C: CURTAIN STYLE PICTORIAL MARKER SOURCE CODE (WRITTEN IN PYTHON)

This appendix contains the source code written in Python to demonstrate in details about marker creation and hidden information decryption of curtain style pictorial marker

Curtain Style Pictorial Marker creation

Step 1: Make QR code

```
1  import numpy as np
2  import cv2
3  import qrcode
4
5  SeparateValue = 128
6  encodedURL = "Sample URL"
7
8  qr = qrcode.QRCode(
9      version=1,
10     error_correction=qrcode.constants.ERROR_CORRECT_H,
11     box_size=1,
12     border=0,
13 )
14
15 qr.add_data(encodedURL)
16     qr.make(fit=True)
17     img = qr.make_image()
18     qrArray = np.array(img)
19     qrWidth, qrHeight = qrArray.shape
20     QR_NUMBER = qrWidth
```

Step 2: Embed QR code in the imported image

```
1
```

```
2  def getBestDifference(changeImage):
3      lab = cv2.cvtColor(changeImage, cv2.COLOR_BGR2LAB)
4      l, a, b = cv2.split(lab)
5      if np.mean(l) < 128:
6          cl = np.where(l > 255 - SeparateValue, 255, l + SeparateValue)
7          limg = cv2.merge((cl, a, b))
8          return cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
9      else:
10         cl = np.where(l <= SeparateValue, 0, l - SeparateValue)
11         limg = cv2.merge((cl, a, b))
12         return cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)
13
14 imgTexture = cv2.imread("InputImage.jpg", cv2.IMREAD_COLOR)
15 height, width, depth = imgTexture.shape
16
17     # Resize the original image and add two blank spaces on the sides
18     imgTextureTemp = np.zeros((height, int(round(8*width/6.0)) , 3), np.uint8)
19     imgTextureTemp[0:height, int(round(width/6.0)): int(round(7*width/6.0))] =
20     imgTexture[0:height, 0: int(round(7*width/6.0))-int(round(width/6.0))]
21     imgTexture = imgTextureTemp.copy()
22     height, width, depth = imgTexture.shape
23
24     # blur the image
25     imgTexture = cv2.blur(imgTexture, (int(round(width/128.0)),
26     int(round(width/128.0))))
27
28     imgTextureL = imgTexture[0:height, 0:width / 2]
29     imgTextureR = imgTexture[0:height, width / 2:width]
30
31     # Put QR code to left and right side of the blur image
32     height, width, depth = imgTextureL.shape
33     widthSmallGap = 0.5 * width / QR_NUMBER
34     heightSmallGap = 1.0 * height / QR_NUMBER
35     for h in xrange(0, QR_NUMBER):
36         for w in xrange(0, QR_NUMBER):
37             y0 = int(round(heightSmallGap * h))
38             y1 = int(round(heightSmallGap * h + heightSmallGap))
39             x0 = int(round(widthSmallGap * w))
```

```
40              x1 = int(round(widthSmallGap * w + widthSmallGap))
41              if qrArray[h][w] == False:
42                  imgTextureL[y0:y1, x0:x1] = imgTextureL[y0:y1, x0 +
43                  width / 2:x1 + width / 2]
44              else:
45                  imgTextureL[y0:y1, x0:x1] = getBestDifference(imgTextureL[y0:y1, x0 +
46                  width / 2:x1 + width / 2])
47
48
49      height, width, depth = imgTextureR.shape
50      widthSmallGap = 0.5 * width / QR_NUMBER
51      heightSmallGap = 1.0 * height / QR_NUMBER
52      for h in xrange(0, QR_NUMBER):
53          for w in xrange(0, QR_NUMBER):
54              y0 = int(round(heightSmallGap * h))
55              y1 = int(round(heightSmallGap * h + heightSmallGap))
56              x0 = int(round(widthSmallGap * w))
57              x1 = int(round(widthSmallGap * w + widthSmallGap))
58              if qrArray[h][w] == False:
59                  imgTextureR[y0:y1, x0 + width / 2:x1 + width / 2] = imgTextureR[y0:y1,
60                  x0:x1]
61              else:
62                  imgTextureR[y0:y1, x0 + width / 2:x1 + width / 2] = getBestDifference(
63                  imgTextureR[y0:y1, x0:x1])
64
65      # Put QR code to left and right side of the original image
66      imgTexture = imgTextureTemp
67      height, width, depth = imgTexture.shape
68      widthL = int(round(0.25*percentage*width))
69
70      imgTexture[0:height, 0:widthL] = imgTextureL[0:height, 0:widthL]
71      imgTexture[0:height, width-widthL:width] = imgTextureR[0:height, width/2-
72      widthL:width/2]
```

Step 3: Adding border

```
1  height, width, depth = imgTexture.shape #Total pixel number: img.size
2  border = (max(height, width))/50
3  imgTextureBorder = img = np.zeros((height+2*border,width+2*border,3), np.uint8)
```

```
4 imgTextureBorder[border:height+border,border:width+border]=imgTexture
5 cv2.imwrite("OutputImage.jpg", imgTextureBorder)
```

Curtain Style Pictorial Marker hidden information decryption

Step 1: Retrieve the internal image from the captured frame

```
1  import numpy as np
2  import cv2
3  import zbar
4  import math
5  scanner = zbar.ImageScanner()
6  scanner.parse_config('enable')
7  SeparateValue = 128
8
9  # Read video streams
10 cap = cv2.VideoCapture(0)
11 rect = np.zeros((4, 2), dtype="float32")
12
13 while (True):
14         ret, image = cap.read()
15         gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
16         edged = cv2.Canny(gray, 30, 200)
17
18         # find contours in the edged image, keep only the largest
19         # ones, and initialize our screen contour
20         im2, cnts, hierarchy = cv2.findContours(edged.copy(), cv2.RETR_TREE,
21         cv2.CHAIN_APPROX_SIMPLE)
22         cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:10]
23         screenCnt = None
24         # loop over our contours
25         for c in cnts:
26             if 10000 < cv2.contourArea(c) < 80000:  # remove small areas like noise etc
27                 # approximate the contour
28                 peri = cv2.arcLength(c, True)
29                 approx = cv2.approxPolyDP(c, 0.04 * peri, True)
30                 # if our approximated contour has four points, then
31                 # we can assume that we have found our screen
32                 if len(approx) == 4:
```

```
33                  screenCnt = approx
34                  break
35          if screenCnt is not None:
36              # ########################################
37              pts = screenCnt.reshape(4, 2)
38              # the top-left point has the smallest sum whereas the
39              # bottom-right has the largest sum
40              s = pts.sum(axis=1)
41              rect[0] = pts[np.argmin(s)]
42              rect[2] = pts[np.argmax(s)]
43              # compute the difference between the points -- the top-right
44              # will have the minumum difference and the bottom-left will
45              # have the maximum difference
46              diff = np.diff(pts, axis=1)
47              rect[1] = pts[np.argmin(diff)]
48              rect[3] = pts[np.argmax(diff)]
49
50              # now that we have our rectangle of points, let's compute
51              # the width of our new image
52              (tl, tr, br, bl) = rect
53              widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
54              widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
55
56              # ...and now for the height of our new image
57              heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
58              heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
59
60              # take the maximum of the width and height values to reach
61              # our final dimensions
62              maxWidth = max(int(widthA), int(widthB))
63              maxHeight = max(int(heightA), int(heightB))
64
65              # construct our destination points which will be used to
66              # map the screen to a top-down, "birds eye" view
67              dst = np.array([
68                  [0, 0],
69                  [maxWidth - 1, 0],
70                  [maxWidth - 1, maxHeight - 1],
```

```
71                  [0, maxHeight - 1]], dtype="float32")
72
73              # calculate the perspective transform matrix and warp
74              # the perspective to grab the screen
75              M = cv2.getPerspectiveTransform(rect, dst)
76              warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
77
78              dst = cv2.resize(warp, (600, 600))
79              roi = dst[12:576 + 12, 12:576 + 12]
80
81              DecodeMarker(roi, 0.52)
82
83      cv2.waitKey(1)
```

Step 2: Decode the internal image to retrieve the QR code

```
1  #This method is called in Step 1 line 81
2  def DecodeMarker(ARTag, percentage=1.0):
3
4      height, width, depth = ARTag.shape
5      ARTag[0:height, width/8+1:7*width/8-1] = cv2.blur(ARTag[0:height,
6      width/8+1:7*width/8-1], (int(round(width/128.0)), int(round(width/128.0))))
7      ARTag = cv2.resize(ARTag, (1024, 256), interpolation=cv2.INTER_AREA)
8
9      ARTag = cv2.cvtColor(ARTag, cv2.COLOR_BGR2GRAY)
10
11     imgTextureLL = ARTag[0:256, 0:256]
12     imgTextureLR = ARTag[0:256, 256:512]
13     imgTextureRL = ARTag[0:256, 512:768]
14     imgTextureRR = ARTag[0:256, 768:1024]
15
16     thresh1 = cv2.absdiff(imgTextureLL, imgTextureLR)[0:256, 0:128]
17     thresh1[0:128, 0:128] = cv2.equalizeHist(thresh1[0:128, 0:128])
18     thresh1[128:256, 0:128] = cv2.equalizeHist(thresh1[128:256, 0:128])
19     thresh2 = cv2.absdiff(imgTextureRL, imgTextureRR)[0:256, 128:256]
20     thresh2[0:128, 0:128] = cv2.equalizeHist(thresh2[0:128, 0:128])
21     thresh2[128:256, 0:128] = cv2.equalizeHist(thresh2[128:256, 0:128])
22
23     lowerVal = 127
```

```
24      if percentage < 0.75:
25          thresh3 = np.zeros((256, 256), np.uint8)
26          thresh3[0:256, 0:128] = thresh1
27          thresh3[0:256, 128:256] = thresh2
28          ret, thresh3 = cv2.threshold(thresh3, lowerVal, 256, cv2.THRESH_BINARY)
29
30      else:
31          ret, thresh3 = cv2.threshold(thresh1 + thresh2, lowerVal, 255,
32          cv2.THRESH_BINARY)
33
34
35      kernel = np.ones((3, 3), np.uint8)
36      thresh3 = cv2.morphologyEx(thresh3, cv2.MORPH_CLOSE, kernel)
37
38      image = cv2.cvtColor(thresh3, cv2.COLOR_GRAY2BGR)
39      height, width, _  = image.shape
40      imgray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
41      raw = str(imgray.data)
42
43      imageZbar = zbar.Image(width, height,'Y800', raw)
44      scanner.scan(imageZbar)
45
46      textFound = "NAH"
47      for symbol in imageZbar:
48          textFound = symbol.data
49
50      print(textFound)
```

APPENDIX D: ENVELOPE STYLE PICTORIAL MARKER SOURCE CODE (WRITTEN IN PYTHON)

This appendix contains the source code written in Python to demonstrate in details about marker creation and hidden information decryption of envelope style pictorial marker

Envelope Style Pictorial Marker creation

Step 1: Make QR code

```
1  import numpy as np
2  import cv2
3  import qrcode
4
5  encodedURL = "Huy␣Le"
6
7  SeparateValue = 128
8  ratio = 20
9  borderRatio = 0.001
10
11 resizeQRWidth = 0
12 resizeQRHeight = 0
13 qr = qrcode.QRCode(
14     version=1,
15     error_correction=qrcode.constants.ERROR_CORRECT_H,
16     box_size=1,
17     border=0,
18 )
19
20 # Create QR code
21     qr.add_data(encodedURL)
22     qr.make(fit=True)
23     img = qr.make_image()
```

```
24      qrArray = np.array(img)
25      qrWidth, qrHeight = qrArray.shape
26      QR_NUMBER = qrWidth
```

Step 2: Divide QR code into four different sections

```
1  imgTexture = cv2.imread("InputImage.jpg", cv2.IMREAD_COLOR)
2  # Resize QR code to size of image
3      imgQR = np.zeros((height, width, 3), np.uint8)
4      height, width, depth = imgQR.shape
5
6      widthSmallGap = 1.0*width / QR_NUMBER
7      heightSmallGap = 1.0*height / QR_NUMBER
8      for h in xrange(0, QR_NUMBER):
9          for w in xrange(0, QR_NUMBER):
10             y0 = int(round(heightSmallGap * h))
11             y1 = int(round(heightSmallGap * h + heightSmallGap))
12             x0 = int(round(widthSmallGap * w))
13             x1 = int(round(widthSmallGap * w + widthSmallGap))
14             if qrArray[h][w] == False:
15                 imgQR[y0:y1, x0:x1] = 0
16             else:
17                 imgQR[y0:y1, x0:x1] = 255
18
19     topQR = GetTopTriangle(imgQR, "top")
20
21     leftQR = GetTopTriangle(imgQR, "left")
22
23     rightQR = GetTopTriangle(imgQR, "right")
24
25     bottomQR = GetTopTriangle(imgQR, "bottom")
26
27
28 def GetTopTriangle(imgQR, position):
29     height, width, depth = imgQR.shape
30     mask = np.zeros(imgQR.shape, dtype=np.uint8)
31
32     # Default for top triangle
33     roi_corners = np.array([[(0,0),(width/2,height/2), (width,0)]], dtype=np.int32)
```

```
34
35      if position == "left":
36        roi_corners = np.array([[(0,0),(width/2,height/2), (0,height)]], dtype=np.int32)
37      elif position == "right":
38        roi_corners = np.array([[(width,0),(width/2,height/2), (width,height)]],
39        dtype=np.int32)
40      elif position == "bottom":
41        roi_corners = np.array([[(0,height),(width/2,height/2), (width,height)]],
42        dtype=np.int32)
43
44      # fill the ROI so it doesn't get wiped out when the mask is applied
45      channel_count = imgQR.shape[2]  # i.e. 3 or 4 depending on your image
46      ignore_mask_color = (255,)*channel_count
47
48      cv2.fillPoly(mask, roi_corners, ignore_mask_color)
49
50      # apply the mask
51      masked_image = cv2.bitwise_and(imgQR, mask)
52      topQR = masked_image
53      return topQR
```

Step 3: Embed QR code in the divided sections

```
1  def Merge(imageTexture, qrMask, position):
2      lab = cv2.cvtColor(imageTexture, cv2.COLOR_BGR2LAB)
3      l, a, b = cv2.split(lab)
4      height, width, depth = imageTexture.shape
5
6
7      # Default for top triangle
8      qrMask = ResizeQRMask(qrMask, position)
9
10     roi_corners = np.array([[(0,0),(resizeQRWidth/2,resizeQRHeight/2),
11     (resizeQRWidth,0)]], dtype=np.int32)
12     if position == "left":
13       roi_corners = np.array([[(0,0),(resizeQRWidth/2,height/2), (0,height)]],
14       dtype=np.int32)
15     elif position == "right":
16         roi_corners = np.array([[(width,0),(width - (resizeQRWidth/2),height/2),
```

```
17          (width,height)]], dtype=np.int32)
18      elif position == "bottom":
19        roi_corners = np.array([[(0,height),(width/2,height - (resizeQRHeight/2)),
20        (width,height)]], dtype=np.int32)
21
22      # fill the ROI so it doesn't get wiped out when the mask is applied
23      channel_count = qrMask.shape[2]  # i.e. 3 or 4 depending on your image
24      ignore_mask_color = (0,)*channel_count
25
26      cv2.fillPoly(imageTexture, roi_corners, ignore_mask_color)
27
28      # apply the mask
29      masked_image = cv2.bitwise_or(qrMask, imageTexture)
30
31
32      return masked_image
33
34  def ResizeQRMask(qrMask, position):
35      global resizeQRWidth, resizeQRHeight
36
37      height, width, depth = qrMask.shape
38
39      background = np.zeros((height, width, 3), np.uint8)
40
41      if (position == "top") or (position == "bottom"):
42          returnMask = cv2.resize(qrMask,(width,(height * ratio) / 100))
43      else:
44        returnMask = cv2.resize(qrMask,((width * ratio) / 100,height))
45
46      resizeQRHeight, resizeQRWidth, depth = returnMask.shape
47
48      return CropImage(background, returnMask, position)
49
50  def CropImage(background, returnMask, position):
51      height, width, depth = background.shape
52
53      if (position == "top") or (position == "left"):
54          for h in xrange(0, resizeQRHeight):
```

```
55                 for w in xrange(0, resizeQRWidth):
56                     background [h,w] = returnMask[h,w]
57
58         elif (position == "bottom"):
59             for h in xrange(0, resizeQRHeight):
60                 for w in xrange(0, resizeQRWidth):
61                     background[(height - resizeQRHeight + h),w] = returnMask[h,w]
62
63         elif (position == "right"):
64             for h in xrange(0, resizeQRHeight):
65                 for w in xrange(0, resizeQRWidth):
66                     background[(height - resizeQRHeight + h),(width - resizeQRWidth + w)] =
67                     returnMask[h,w]
68         return background
69
70
71 mask = Merge(imgTexture,topQR,"top")
72 mask = Merge(mask,leftQR,"left")
73 mask = Merge(mask,rightQR,"right")
74 mask = Merge(mask,bottomQR,"bottom")
```

Step 4: Adding border

```
1 height, width, depth = mask.shape #Total pixel number: img.size
2 border = (max(height, width))/50
3 borderImage = img = np.zeros((height+2*border,width+2*border,3), np.uint8)
4 borderImage[border:height+border,border:width+border]=mask
5
6 cv2.imwrite('CSPM-10.jpg', borderImage)
7
8 cv2.imshow("borderImage", borderImage)
```

Envelope Style Pictorial Marker hidden information decryption

Step 1: Retrieve the internal image from the captured frame

```
1 import numpy as np
2 import cv2
3 import zbar
4
```

```
5  scanner = zbar.ImageScanner()
6  scanner.parse_config('enable')
7
8  lowVal = 85
9  ratio = 20
10
11 # Read video streams
12 cap = cv2.VideoCapture(0)
13
14 while (True):
15     ret, image = cap.read()
16
17     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
18     gray = cv2.bilateralFilter(gray, 11, 17, 17)
19
20     edged = cv2.Canny(gray, 30, 200)
21
22     # find contours in the edged image, keep only the largest
23     # ones, and initialize our screen contour
24
25     im2, cnts, hierarchy = cv2.findContours(edged.copy(), cv2.RETR_TREE,
26     cv2.CHAIN_APPROX_SIMPLE)
27
28     cnts = sorted(cnts, key=cv2.contourArea, reverse=True)[:10]
29     screenCnt = None
30
31     # loop over our contours
32     for c in cnts:
33         if cv2.contourArea(c) > 8000:  # remove small areas like noise etc
34             # approximate the contour
35             peri = cv2.arcLength(c, True)
36             approx = cv2.approxPolyDP(c, 0.04 * peri, True)
37
38             # if our approximated contour has four points, then
39             # we can assume that we have found our screen
40             if len(approx) == 4:
41                 screenCnt = approx
42             # break
```

```
43
44      if screenCnt is not None:
45          # ##########################################
46          pts = screenCnt.reshape(4, 2)
47
48          rect = np.zeros((4, 2), dtype="float32")
49          # the top-left point has the smallest sum whereas the
50          # bottom-right has the largest sum
51          s = pts.sum(axis=1)
52          rect[0] = pts[np.argmin(s)]
53          rect[2] = pts[np.argmax(s)]
54
55          # compute the difference between the points -- the top-right
56          # will have the minumum difference and the bottom-left will
57          # have the maximum difference
58          diff = np.diff(pts, axis=1)
59          rect[1] = pts[np.argmin(diff)]
60          rect[3] = pts[np.argmax(diff)]
61
62          # now that we have our rectangle of points, let's compute
63          # the width of our new image
64          (tl, tr, br, bl) = rect
65          widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
66          widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
67
68          # ...and now for the height of our new image
69          heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
70          heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
71
72          # take the maximum of the width and height values to reach
73          # our final dimensions
74          maxWidth = max(int(widthA), int(widthB))
75          maxHeight = max(int(heightA), int(heightB))
76
77          # construct our destination points which will be used to
78          # map the screen to a top-down, "birds eye" view
79          dst = np.array([
80              [0, 0],
```

```
81              [maxWidth - 1, 0],
82              [maxWidth - 1, maxHeight - 1],
83              [0, maxHeight - 1]], dtype="float32")
84
85          # calculate the perspective transform matrix and warp
86          # the perspective to grab the screen
87          M = cv2.getPerspectiveTransform(rect, dst)
88          warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
89
90          dst = cv2.resize(warp, (544, 544))
91          roi = dst[12:520 + 12, 12:520 + 12]
92
93          merge = processMaskImage(roi)
94
95          height, width, depth = merge.shape
96          imgray = cv2.cvtColor(merge, cv2.COLOR_BGR2GRAY)
97          raw = str(imgray.data)
98
99          imageZbar = zbar.Image(width, height,'Y800', raw)
100         scanner.scan(imageZbar)
101
102         textFound = ""
103         for symbol in imageZbar:
104             textFound = symbol.data
105             print (textFound)
106
107 cap.release()
```

Step 2: Decode the internal image to retrieve the QR code

```
1  #This method is called in Step 1 line 93
2  def processMaskImage(qrImage):
3
4      height, width, depth = qrImage.shape
5
6      topQR = GetTopTriangle(qrImage, "top")
7      topQR = cv2.resize(topQR,(width,height * 100 / ratio))
8      topQR = topQR[0:width, 0:width]
9
```

```
10      bottomQR = GetTopTriangle(qrImage, "bottom")
11      bottomQR = cv2.resize(bottomQR,(width,height * 100 / ratio))
12      heightQR, widthQR, _ = bottomQR.shape
13      bottomQR = bottomQR[(heightQR - height):heightQR, 0:width]
14
15
16      leftQR = GetTopTriangle(qrImage, "left")
17      leftQR = cv2.resize(leftQR,(width * 100 / ratio,height))
18      leftQR = leftQR[0:width, 0:width]
19
20
21      rightQR = GetTopTriangle(qrImage, "right")
22      rightQR = cv2.resize(rightQR,(width * 100 / ratio,height))
23      heightQR, widthQR, _ = rightQR.shape
24      rightQR = rightQR[0:width, (widthQR - width):widthQR]
25
26      merge = Merge(topQR,leftQR,rightQR,bottomQR,height,width)
27      merge = CorrectMask(merge)
28
29      return merge
30
31  def GetTopTriangle(imgQR, position):
32      height, width, depth = imgQR.shape
33      mask = np.zeros(imgQR.shape, dtype=np.uint8)
34
35      # Default for top triangle
36      roi_corners = np.array([[(0,0),(width/2,(height/2) * ratio / 100), (width,0)]],
37      dtype=np.int32)
38
39      if position == "left":
40        roi_corners = np.array([[(0,0),((width/2)* ratio / 100,height/2), (0,height)]],
41        dtype=np.int32)
42      elif position == "right":
43        roi_corners = np.array([[(width,0),( width - ((width/2)* ratio / 100),height/2),
44        (width,height)]], dtype=np.int32)
45      elif position == "bottom":
46        roi_corners = np.array([[(0,height),(width/2,height - ((height/2) * ratio /
47        100)), (width,height)]], dtype=np.int32)
```

```
48
49      channel_count = imgQR.shape[2]  # i.e. 3 or 4 depending on your image
50      ignore_mask_color = (255,)*channel_count
51
52      cv2.fillPoly(mask, roi_corners, ignore_mask_color)
53
54      # apply the mask
55      masked_image = cv2.bitwise_and(mask, imgQR)
56
57      return masked_image
58
59  def Merge(top,left,right,bottom,height,width):
60      merge = np.zeros((height, width, 3), np.uint8)
61
62      merge = cv2.bitwise_xor(merge, top)
63      merge = cv2.bitwise_xor(merge, left)
64      merge = cv2.bitwise_xor(merge, right)
65      merge = cv2.bitwise_xor(merge, bottom)
66      return merge
67
68  def CorrectMask(image):
69      lower_black = np.array([lowVal,lowVal,lowVal], dtype = "uint16")
70      upper_black = np.array([255,255,255], dtype = "uint16")
71      image = cv2.inRange(image, lower_black, upper_black)
72
73      image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
74      return image
```

APPENDIX E: RENDER VIRTUAL INFORMATION SOURCE CODE (WRITTEN IN PYTHON)

Render Simple 3D Cube with OpenCV

```
size = image.shape
        focal_length = size[1]
        center = (size[1]/2, size[0]/2)
        mtx = np.array(
                                [[focal_length, 0, center[0]],
                                [0, focal_length, center[1]],
                                [0, 0, 1]], dtype = "double"
                                )

        dist = np.zeros((4,1)) # Assuming no lens distortion


        objp = np.zeros((2 * 2, 3), np.float32)
        objp[:, :2] = np.mgrid[0:2, 0:2].T.reshape(-1, 2)


        distance = math.sqrt((rect[1][0] - rect[0][0])**2 )

        # Construct 3d points for the cube
        axis = np.float32([
            [rect[3][0], rect[3][1],0], #bottom left (b)
            [rect[0][0], rect[0][1],0], #top left (b)
            [rect[1][0], rect[1][1],0], #top right (b)
            [rect[2][0], rect[2][1],0], #bottom right (b)
            [rect[3][0], rect[3][1],distance],
            [rect[0][0], rect[0][1],distance],
            [rect[1][0], rect[1][1],distance],
```

```
28              [rect[2][0], rect[2][1],distance] ])
29
30
31          objp = np.array([
32                          [rect[3][0], rect[3][1],0], #bottom left (b)
33                          [rect[2][0], rect[2][1],0], #bottom right (b)
34                          [rect[1][0], rect[1][1],0], #top right (b)
35                          [rect[0][0], rect[0][1],0] #top left (b)
36                           ], dtype="float32")
37
38                           (success, rvecs, tvecs) =
39                           cv2.solvePnP(objp, rect, mtx,
40                           dist, flags=cv2.SOLVEPNP_ITERATIVE)
41
42          imgpts, _ = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
43          imgpts = np.int32(imgpts).reshape(-1,2)
44
45          #draw green floor
46          image = cv2.drawContours(image, [imgpts[:4]],-int(distance),(0,255,0),3)
47          #draw blue lines
48          for i,j in zip(range(4),range(4,8)):
49              image = cv2.line(image, tuple(imgpts[i]), tuple(imgpts[j]),(255),3)
50          #draw red top
51          image = cv2.drawContours(image, [imgpts[4:]],-int(distance),(0,0,255),3)
52      cv2.imshow("image", image)
```