

# USING QR CODE FOR INDOOR POSITIONING

---

A Thesis

Presented to the

Faculty of

California State University, Fullerton

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Electrical Engineering

---

By

Zheqi Li

Thesis Committee Approval:

David Cheng, Department of Electrical Engineering, Chair  
Jidong Huang, Department of Electrical Engineering  
Yun Zhu, Department of Electrical Engineering

Summer, 2017

ProQuest Number: 10268000

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10268000

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code  
Microform Edition © ProQuest LLC.

ProQuest LLC.  
789 East Eisenhower Parkway  
P.O. Box 1346  
Ann Arbor, MI 48106 – 1346

## ABSTRACT

This paper explored the feasibility and accuracy of using the Kinect sensor to assist robots or unmanned vehicles to navigate in indoor environments using the QR codes attached on the wall. During the experiment, the high definition RGB camera of the Kinect was firstly used to capture the images of the QR codes attached on the wall and the depth sensor was used to measure the distance between the Kinect camera and the QR codes. After that, we used the Open Source Computer Vision (OpenCV) library to scan the image taken by the camera to find out the QR codes within the image and highlight them on the screen. When the scanning was done, the ZBar open-source barcode reading library was applied to decode the QR codes to obtain the coordinate information. At last, we used the raw data to calculate the exact position of the Kinect camera within the coordinate system by solving the GPS equations using the Newton-Raphson method. Therefore the algorithm was tested on the Kinect Sensor. In this paper, the results from testing the Kinect sensor to locate the Kinect camera in indoor environment were given.

## TABLE OF CONTENTS

ABSTRACT.....	ii
LIST OF TABLES.....	v
LIST OF FIGURES .....	vi
Chapter	
1. INTRODUCTION .....	1
Outline .....	1
1.1 Introduction to the Experiment.....	1
1.2 Introduction to the Kinect.....	2
1.2.1 Kinect for Xbox 360 (V1).....	2
1.2.2 Kinect for Xbox One (V2).....	3
1.3 Introduction to the QR Code.....	5
1.3.1 Types of QR Code .....	6
1.3.2 QR Code Structure.....	9
2. THE METHODOLOGY AND ALGORITHM.....	11
Outline .....	11
2.1 The Global Positioning System .....	11
2.2 The Newton-Raphson Method.....	13
3. THE DEVELOPMENT ENVIRONMENT SETUP.....	15
Outline .....	15
3.1 Introduction to the Development Software and Libraries .....	15
3.1.1 Introduction to the Development Software.....	15
3.1.2 Introduction to the Libraries .....	16
3.1.3 Kinect for Windows Software Development Kit (SDK).....	17
3.2 The Working Environment Setup.....	18
3.2.1 Visual Studio 2015.....	18
3.2.2 OpenCV .....	18
3.2.3 ZBar .....	19

4. THE EXPERIMENTAL RESULTS.....	20
Outline .....	20
4.1 Initiating the Color Sensor.....	20
4.2 Initiating the Depth Sensor .....	23
4.3 Image Processing Using OpenCV .....	26
4.4 Decoding the QR Codes Using ZBar Library.....	36
4.5 Coordinate Calculation .....	40
4.6 Errors Analysis .....	46
4.6.1 First Static Experiment –Errors of the Depth Sensor Measurement.....	46
4.6.2 Second Static Experiment –Errors of Calculating the Position of the Kinect Sensor .....	48
5. CONCLUSION AND FUTURE STUDY .....	51
REFERENCES .....	53

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Summary and Comparison of the Kinect v1 and Kinect v2 .....	5
2. Intersection of Spheres.....	13
3. Indoor Distance Measured by the Depth Sensor.....	26
4. Mean Error and Standard Deviation of the Experiments.....	47
5. Mean Error and Standard Deviation of the XYZ-Coordinate.....	50

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. The Kinect v1 .....	3
2. Kinect v2 sensor components .....	4
3. QR code model 1 .....	6
4. QR code model 2 .....	7
5. Micro QR code.....	7
6. IQR code .....	8
7. SQRC .....	8
8. Frame QR code .....	9
9. QR code structure .....	10
10. 0The GPS system.....	11
11. Intersection of the spheres .....	12
12. Microsoft visual studio 2015 community edition .....	16
13. Kinect for Windows SDK 2.0.....	17
14. Image taken by the color sensor.....	23
15. Image flipped using OpenCV .....	27
16. Image grayscale .....	28
17. The image processed using canny algorithm .....	30
18. Contour hierarchy .....	31

19.	The contour hierarchy of a position detection pattern .....	32
20.	Four QR codes in paper A .....	36
21.	Four QR codes in paper B.....	39
22.	The decoded messages .....	39
23.	QR codes in position A.....	40
24.	Coordinates of the QR codes in Position A .....	41
25.	QR codes in position B .....	45
26.	The result of the position .....	46
27.	The Kinect depth sensor collected the depth data in a distance of 804 mm .....	48
28.	QR codes attached on the wall.....	49
29.	The mean and standard deviation of the errors .....	50



## CHAPTER 1

### INTRODUCTION

#### Outline

1.1 Introduction to the Indoor Navigation. This chapter mainly talks about the approach explored for indoor navigation and a new approach for indoor robot navigation will be discussed.

1.2 Introduction to the Kinect. This chapter mainly introduces the hardware of the Kinect sensor.

1.3 Introduction to the QR Code. This chapter introduces types and structure of QR code.

#### 1.1 Introduction to the Indoor Navigation

Indoor navigation is of great importance to unmanned vehicles and robots. Since the GPS reception is normally non-existent inside buildings, a variety of approaches of positioning, including Wi-Fi, Bluetooth, sonar laser scanners and vision localization, have been explored. Wi-Fi based localization has been found to be the most popular and suitable one. It uses signal strength measurements from Wi-Fi access points for localization [5]. However, the Wi-Fi base localization system of interaction with Wi-Fi is complicated and it requires extra infrastructure to support. So there is still no standard for the indoor navigation system.

In order to solve the indoor navigation problem, we are exploring an approach using Kinect sensor to read the QR codes attached on the wall to navigate and position in indoor environment. QR code has been widely used for manufacture, business, etc. It is cheap and easy to generate. Using QR codes to assist indoor navigation is efficient, easy-to-use and low-cost.

In this paper, we've explored the feasibility and accuracy of the Kinect sensor in indoor robot navigation. Variable software and libraries was used to develop the application and the result was demonstrated at the end.

## 1.2 Introduction to Kinect

Kinect is a motion sensing input device by Microsoft for the Xbox video game console. It is a device that includes a high definition RGB camera, a depth sensor and a multi-array microphone.

### 1.2.1 Kinect for Xbox 360 (V1)

The first generation of the Kinect was first distributed in 2010 for the Xbox 360 video game console and the Windows PC version was released later in 2012 [16], as shown in the Figure 1 [23].

Kinect v1 came out with a color camera with a resolution of 640 x 480, which can resolution up to 1280 x 1024 at a lower frame rate and other color formats such as UYVY [16]. More importantly, the Kinect v1 can capture images by its IR camera before it has been converted into a depth map as 640 x 480 video, or 1280 x 1024 at a lower frame rate [16].

The depth sensor of the Kinect v1 consists of an infrared laser projector combined with a monochrome CMOS sensor, which captures video data in 3D under any ambient

light conditions [35]. The sensing range of the depth sensor is adjustable, up to 4.5 meters, by the software based on the environment [34]. The depth horizontal field of view of the Kinect v1 is 57 degrees and the depth vertical field of view is 43 degrees. With the Kinect for Windows Software Development Kit (SDK), the Kinect v1 can track 2 full skeletons in view and define up to 20 skeleton joints.

The multi-array microphone of the Kinect v1 consists of four microphones in a line. It is capable of capturing the sound of the environment, recording audio and detect the location and direction of the source in front of the device.



*Figure 1.* The Kinect v1.

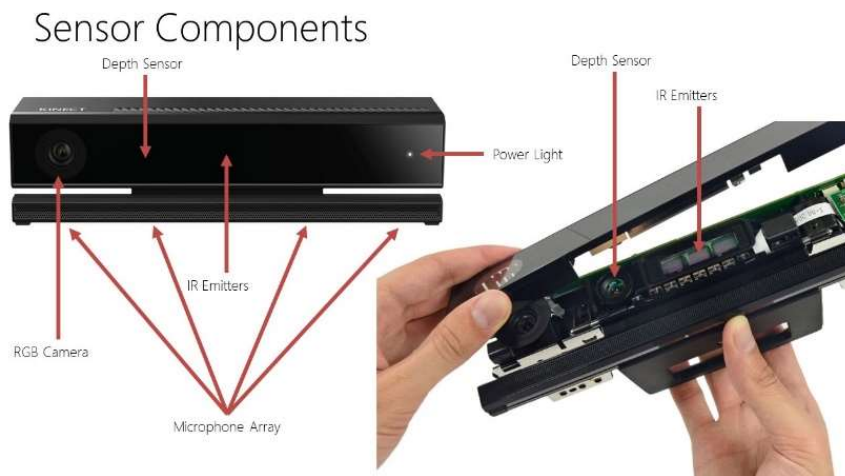
### 1.2.2 Kinect for Xbox One (V2)

The second generation of the Kinect was released on 2013 for the Xbox One video game console [16]. Compared to the first version of the Kinect, the updated one has a noticeable improvement. The Kinect v2 can process 2 gigabits of data per second. So the Kinect v2 uses a USB 3.0 controller for its data streaming.

The color camera of the Kinect v2 has a higher resolution of 1920 x 1080 and the video captured in 1080p can be displayed on screen in the same resolution. The video communication has been improved due to the fast data streaming.

The depth sensor of the Kinect v2 can be seen as a range or a 3D camera, which is capable of measuring the distance to the subject from 0.5 meters up to 8 meters. Instead of normal color pixels, the pixels returned by the Kinect v2 include the data of the distance to the point. For this reason, we can use this feature to ascertain the distance to certain points while taking images. The Kinect v2 also has a wider horizontal field of view of 70 degrees and vertical field of view of 60 degrees. Moreover, the Kinect v2, as shown in Figure 2 [20], is capable of tracking up to six bodies simultaneously and define up to 25 skeleton joints.

The multi-array microphone of the Kinect v2 consists of four microphones placed on the bottom of the device. It can capture the sound, record audio and detect the location and direction of the source from the environment as well.



*Figure 2.* Kinect v2 sensor components.

Since the Kinect v2 is an updated version, there are some improvements of the Kinect v2 compared to the Kinect v1. We can see that there is a summary and comparison of Kinect v1 and Kinect v2 as shown in table 1 [2]. This should be able to

give us a better idea about the main differences between these two devices. Taking into consideration of the compatibility of the Kinect to the applications and the accuracy of the experiment, we decided to use the Kinect v2 as our experimental apparatus.

Table 1. Summary and Comparison of the Kinect v1 and Kinect v2

Feature	Kinect for Windows v1	Kinect for Windows v2
Color Camera	640 x 480 @30fps	1920 x 1080 @30fps
Depth Camera	320 x 240	512 x 424
Max Depth Distance	4.5 M	8 M
Min Depth Distance	40 cm in near mode	50 cm
Depth Horizontal Field of View	57 degrees	70 degrees
Depth Vertical Field of View	43 degrees	60 degrees
Tilt Motor	Yes	No
Skeleton Joints Defined	20 joints	25 joints
Full Skeletons Tracked	2	6
USB Standard	2.0	3.0
Supported OS	Win 7, Win 8	Win 8, Win 8.1, Win 10

### 1.3 Introduction to QR Code

QR code (abbreviated from Quick Response Code) is the trademark for a type of matrix barcode as well as a machine-readable label that contains information [25]. It was firstly designed for the vehicle manufacture in 1994, later it was found very convenient and efficient for people to display certain text or email and track item information, etc. There are many QR code generators online for people to generate their own QR codes that contains certain information.

QR code can provide the following features [32]:

1. High capacity encoding of data;
2. Small printout size;
3. Dirt and damage resistant;
4. Readable from any direction;
5. Structured appending feature;
6. Error correction feature.

### 1.3.1 Types of QR Code

There are totally six types of QR codes currently used in the world. They are QR code model 1, QR code model 2, micro QR code, IQR code, SQRC and Frame QR.

Model 1 is the original QR code, as shown in Figure 3 [32]. It has 14 versions (73 x 73 modules) at most, which can contain up to 1,167 numerals [32].



*Figure 3.* QR code model 1.

Model 2 is an improved version of Model 1, as shown in Figure 4 [32]. It has the largest version up to 40 (177 x 177 modules), which can store up to 7,089 numerals [32]. Now when we talk about QR code, it usually refers to Model 2.



Figure 4. QR code model 2.

Micro QR code is a mini version of QR code, as shown in Figure 5 [32]. It has only one orientation detecting pattern. The largest version of this code is M4 (17 x 17 modules), which can contain up to 35 numerals [32].

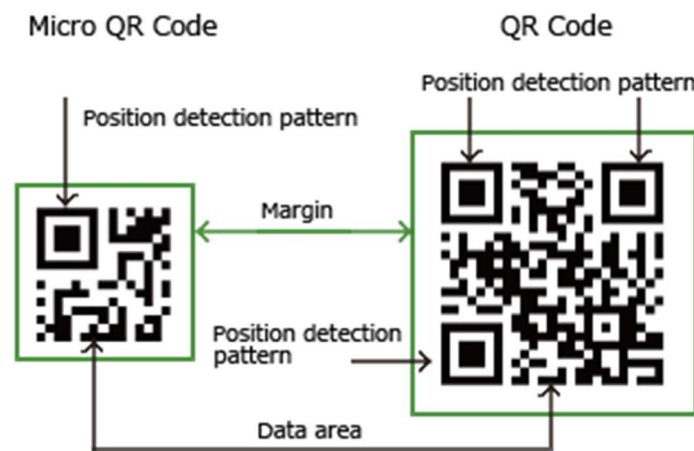


Figure 5. Micro QR code.

IQR code, as shown in Figure 6 [32], is a matrix-type 2D code allowing easy reading of its position and size [32]. This code can be either a rectangular code, turned-over code, black-and-white inversion code or dot pattern code, allowing a wide range of applications in various areas [5]. The maximum version can be up to 61 (422 x 422 modules), which can store about 40, 000 numerals [32].

iQR Code

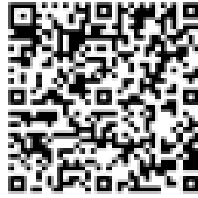
iQR Code  
(Rectangular type)

Figure 6. iQR code.

Security QR codes (SQRC) is a type of QR code equipped with reading restricting function [28], as shown in Figure 7 [3]. This code can be used to store some private information and public information at the same time since it has two parts of the code. The general scanner or mobile phone can only scan and decode the public part of the code, only the specific scanner, like a SQRC-ready scanner, can scan and decode both public part and private part of the code. The most importantly, the appearance and apparent properties remain the same as the normal QR code.



Figure 7. SQRC.

FrameQR, as shown in Figure 8 [30], is a QR code with a “canvas area” that can be flexibly used [12]. The canvas area located in the center of the QR code, where you can insert image, letters and so on and has no influence on the QR code decoding. The



canvas area is also adjustable. This kind of code usually is used on business card and product to make an impression or give out some information.



*Figure 8.* Frame QR code

### 1.3.2 QR Code Structure

We can see from the Figure 9 [26], a QR code model 2 consists of the following parts:

1. Position Detection Patterns: They are located at three corners of the QR codes (except Micro QR, it only has one), it can be used to be detected and read quickly.
2. Separators: Separate for Position Detection Patterns to improve efficiency of the position detection.
3. Timing Pattern: White and black modules are alternately arranged to determine the coordinate [33].
4. Alignment Patterns: It is used to improve the efficiency of detection when there is displacement of modules due to distortion [33].
5. Format Information: It contains the error correction rate and mask pattern of the code [33].
6. Data: This part contains the information of the QR code.
7. Error Correction: When a part of the QR code is missing or damaged, the error correction code can be applied to restore the data.

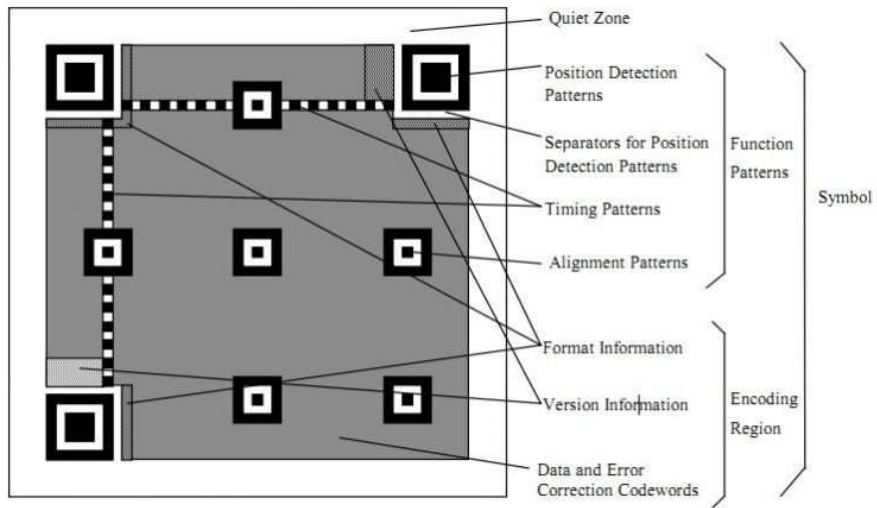


Figure 9. QR code structure.

## CHAPTER 2

### THE METHODOLOGY AND ALGORITHM

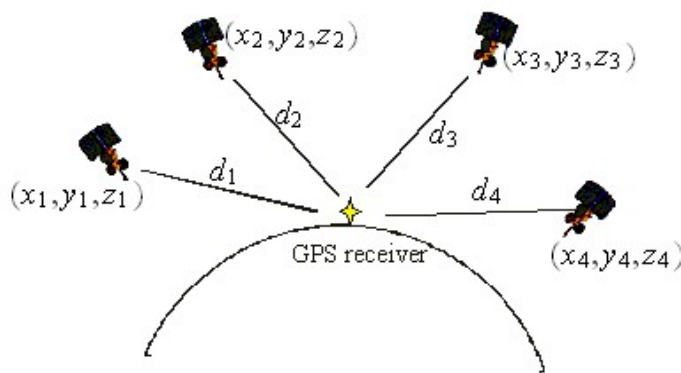
#### Outline

2.1 The Global Positioning System (GPS). This chapter introduces the Global Positioning System and the system positioning equations.

2.2 The Newton-Raphson Method. This chapter introduce the algorithm named the Newton-Raphson method to solve the system positioning equations.

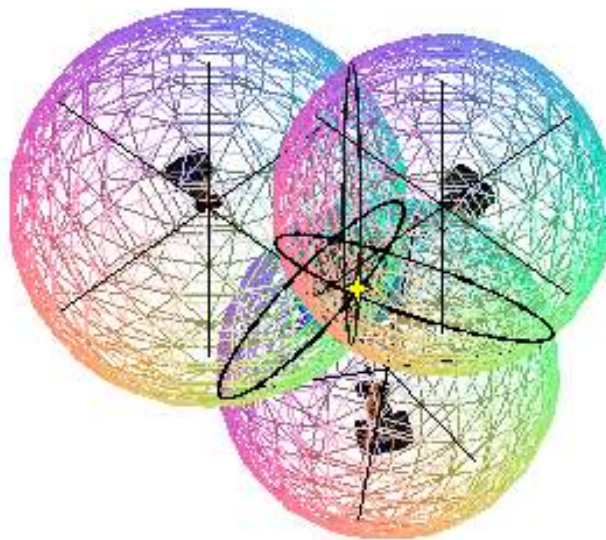
#### 2.1 The Global Positioning System

The Global Positioning System (GPS) is a worldwide radio – navigation system. It uses the satellites, which orbit 11,000 nautical miles above the Earth, as reference points to calculate positions, as shown in Figure 10 [31].



*Figure 10.* The GPS system.

Each satellite is equipped with an accurate clock to let it broadcast signals coupled with a precise time message and the ground unit receives the satellite signal, which travels at the speed of light [31]. According to the time difference in the satellite, the distance between the satellite and the GPS receiver can be easily calculated. If we have the distances of three satellites to the GPS receiver and know where these three satellites exactly are, we should be able to know the exact location of the GPS receiver, as shown in Figure 11 [31].



Three spheres

*Figure 11.* Intersection of the spheres.

Since the clock in the GPS receiver and the clock in the satellite are not synchronized, the time offset should also be considered. So to make the position to be more accurate, GPS usually uses at least 4 satellites to calculate a position. Here is an easy way to see the observations from four satellites can give the exact position. See table 2 [31].

Table 2. Intersection of Spheres

Intersection	Equivalency	Result
Two spheres	Circle	Circle
Three spheres	Circle $\cap$ Sphere	Two points
Four spheres	Two points $\cap$ Sphere	One point

We know that  $(x_i, y_i, z_i)$ ,  $i = 1, 2, 3, 4$  are the exact positions of the satellites. The system equations are [31]:

$$\begin{aligned}
 \sqrt{(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2} + ct_B &= d_1 \\
 \sqrt{(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2} + ct_B &= d_2 \\
 \sqrt{(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2} + ct_B &= d_3 \\
 \sqrt{(x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2} + ct_B &= d_4
 \end{aligned}$$

where  $c$  is the speed of light and  $t_B$  is the receiver clock offset time.

Similarly, assuming we have the distance between the Kinect sensor and QR codes measured by the Kinect depth sensor and the decoded position information of the QR codes, which are the  $(x_i, y_i, z_i)$ ,  $i = 1, 2, 3, 4$ , then we should be able to know the exact position of the Kinect sensor by solving these system equations.

When we get the raw data from the Kinect sensor, we need to use the Newton-Raphson method to solve the system equations to calculate the exact position of the Kinect sensor.

## 2.2 The Newton-Raphson Method

The system equation:

$$\sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} + ct_B = d_i$$

can be converted into:

$$f_i = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 - (d_i - T)^2 = 0$$

where  $T = ct_B$ .

So in order to solve the equation with the Newton-Raphson method, we make an initial guess  $x_0$ , calculate  $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$ . Then replace  $x_0$  in the expression  $x_0 - \frac{f(x_0)}{f'(x_0)}$  with  $x_1$  and compute  $x_2$  in the same manner [31].

We compute:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

and continue to do so until the value of  $f(x_n)$  is sufficiently close to zero.

## CHAPTER 3

### THE DEVELOPMENT ENVIRONMENT SETUP

#### Outline

3.1 Introduction to the Development Software and Libraries. This chapter mainly introduces the development software and libraries required for the application and the experiment.

3.2 The Working Environment Setup. This chapter shows how to set up the working environment for the application development.

#### 3.1 Introduction to the Development Software and Libraries

Since there are variant software that we can use to program the Kinect sensor, we need to choose one that is most suitable to us. So we introduce the software used for developing the applications for Kinect sensor:

##### 3.1.1 Introduction to the Development Software

Visual Studio is an integrated development environment (IDE) on Windows from Microsoft. It can be used to develop various applications and it supports 36 different programming languages, such as C/C++, C#, VB.NET, HTML/XHTML, JavaScript, etc. [21]. The user interface is highly customizable. Now a free version of Visual Studio called Community edition (see Figure 12) is provided by Microsoft, we can download it from the Microsoft official website.

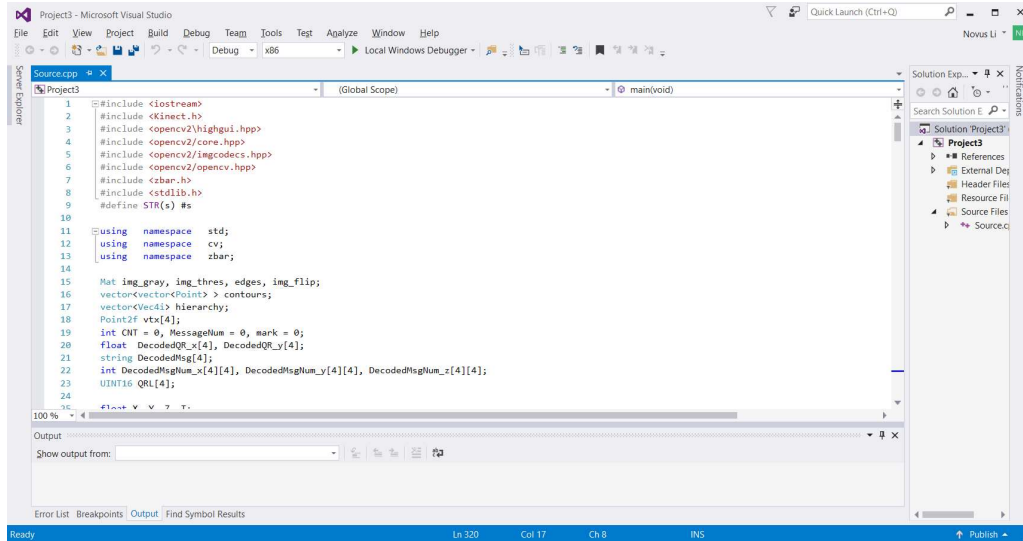


Figure 12. Microsoft visual studio 2015 community edition.

### 3.1.2 Introduction to the Libraries

There are some libraries we need to know for the image processing and QR code decoding. We introduce these two libraries below:

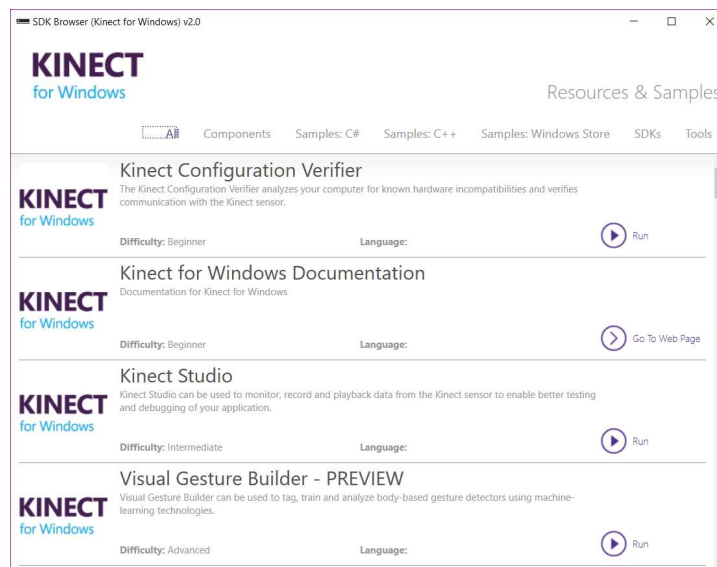
**3.1.2.1 OpenCV Library.** Open Source Computer Vision Library (OpenCV) is a library of programming functions that focus on real-time applications [24]. OpenCV is based on C++, there are also bindings in Python, Java and MATLAB [24]. This library is cross-platform and free for use. In order to find the QR codes within the images taken by the Kinect camera, the OpenCV is used to scan the contours within the image to find out the QR codes and highlight them with color lines so that we can see clearly if the camera has captured the QR codes.

**3.1.2.2 ZBar Library.** ZBar is an open-source C barcode reading library. It can be used to read any barcodes from various sources, such as image, video, etc. It supports real-time scanning of video streams and can recognize EAN-13, UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2 of 5 and QR code symbologies [6].



### 3.1.3 Kinect for Windows Software Development Kit (SDK)

Kinect for Windows SDK (see Figure 14) is a non – commercial Kinect SDK for Windows users by Microsoft in spring 2011 [16]. The SDK includes Windows compatible PC drivers for Kinect sensor and it supports C++, C# and Visual Basic in Microsoft Visual Studio 2010 [16]. The latest version of SDK is Kinect for Windows SDK 2.0 and it was released in 2014, which requires the Windows system to be Windows 8, 8.1, Windows Embedded 8 or Windows 10 [16]. This is a required software for Windows users to develop applications for the Kinect sensor. We need to make sure that we install this software on the PC correctly before we plug the Kinect device in it.



*Figure 13.* Kinect for Windows SDK 2.0.

In this experiment, we used Microsoft Visual Studio 2015 to develop the application for the Kinect. Not only because the Visual Studio 2015 supports various libraries, but also it works with the Kinect for Windows SDK 2.0 on PC very well and does not need to install extra drivers.

### 3.2 The Working Environment Setup

When we have decided the software for the Kinect application development, we can start configuring the working environment. Before we configure any software on PC or tablet, we need to install the Kinect for Windows SDK first. After the installment of the SDK is done, we can continue to configure the Visual Studio 2015.

#### 3.2.1 Visual Studio 2015

Here is the configuration of the Visual Studio 2015 community edition for the Kinect sensor:

1. Start a new project, build a new item, then right click the new project in Solution Explorer and go to “Properties”;
2. Go to “General” under “C/C++”, add “\$(KINECTSDK20\_DIR)\inc” in the “Additional Include Directories”;
3. Go to “General” under “Linker”, add “\$(KINECTSDK20\_DIR)\Lib\x86” in “Additional Library Directories”;
4. Go to “Input” under “Linker”, add “kinect20.lib” in “Additional Dependencies”;
5. Include the “Kinect.h” in the header files.

#### 3.2.2 OpenCV

Here is the configuration of the OpenCV library in the Visual Studio 2015:

1. Right click “This PC”, go to “Properties”, go to “Advanced system settings” on the left side, then go to “Environment Variables”;

2. Build a new variable named “OPENCV”, the variable value is the directory of the “build” file of the opencv, such as “X:\opencv\build” (“X” is the letter of the drive where you keep the OpenCV library) ;
3. Edit “Path” variable in “System variables” and add “;%OPENCV%\x86\vc12\bin” at the end;
4. Include the “opencv2\opencv.hpp” in the headers files.

### 3.2.3 ZBar

Here is the configuration of the ZBar library in the Visual Studio 2015:

1. Right click the project, go to the “Properties”;
2. Go to “VC++ Directories”, add the directory of the “include” file in the ZBar folder, such as “X:\ZBar\include”, in “Include Directories”;
3. Add the directory of the “lib” file in the ZBar folder in “Library Directories”;
4. Go to “General” under “C/C++”, then add the directory of the “include” file in the ZBar folder in “Additional Include Directories”;
5. Go to “General” under “Linker”, then add the “lib” file in the ZBar folder in “Additional Library Directories”;
6. Include the “zbar.h” in the header files.

## CHAPTER 4

### THE EXPERIMENTAL RESULTS

#### Outline

4.1 Initiating the Color Sensor. This chapter shows how to initiate the color sensor of the Kinect in Visual Studio 2015 with C++.

4.2 Initiating the Depth Sensor. The chapter shows how to initiate the depth sensor of the Kinect in Visual Studio 2015 with C++.

4.3 Image Processing Using OpenCV. This chapter shows how to use the OpenCV library to process the image taken by Kinect camera.

4.4 Decoding QR Codes Using ZBar Library. This chapter shows how to use ZBar library to decode QR codes and obtain the data for further process.

#### 4.1 Initiating the Color Sensor

In order to initiate the color sensor, we initiated the Kinect sensor first. Since we used C++ programming language, the codes to turn on the Kinect sensor are:

```
IKinectSensor *mySensor = nullptr;
GetDefaultKinectSensor (&mySensor);
mySensor -> Open ();
```

The code “IKinectSensor” represents the Kinect sensor device [14], the second line of the codes “GetDefaultKinectSensor” means getting the default Kinect sensor and the third line “mySensor -> Open ()” is to turn on the Kinect sensor.

We can use the sensor to access data from several sources, including: color, depth, etc. Functionally, each source from the sensor is controlled by the following three things [17]:

Source type: Inspect or configure a source and open a source reader;

Source reader type: Access to the source's frames using eventing or polling;

Frame type: Access the data from a particular frame from the source.

So to obtain the frame of the color sensor, we need to get the source first. To get the color frame source, the codes are (Beginner\_YH, 2016):

```
IColorFrameSource *mySource = nullptr;
```

```
mySensor -> get_ColorFrameSource (&mySource);
```

After the color frame source was obtained, we opened the source reader. The codes to open the source reader are (Beginner\_YH, 2016):

```
IColorFrameReader *myReader = nullptr;
```

```
mySource -> OpenReader (&myReader);
```

When the source reader was opened, we got the frame from the source reader. However, there are chances that we fail to obtain the frame from the source reader, so we need to use a member function "AcquireLatestFrame ()" to test if the frame has been successfully acquired. Noticed that Kinect color sensor uses YUY2 video format, if we want to store and process the image using OpenCV, we need to convert the format into RGBA. So we used the codes:

```
int height = 0, width = 0;
```

```
IFrameDescription *myDescription = nullptr;
```

```
mySource->get_FrameDescription(&myDescription);
```

```

myDescription->get_Height(&height);

myDescription->get_Width(&width);

Mat img(height,width,CV_8UC4);

IColorFrame *myFrame = nullptr;

while (1)

{

if (myReader -> AcquireLatestFrame (&myFrame) == S_OK)

{

UINT    size = 0;

myFrame -> CopyConvertedFrameDataToArray(width * height * 4,

(BYTE*)img.data, ColorImageFormat_Bgra);

imshow("TEST", img);

myFrame -> Release();

}

}

```

The code “mySource->get\_FrameDescription(&myDescription)” means that to get the information of the frame source, then code “myDescription->get\_Height(&height)” and “myDescription->get\_Width(&width)” are to obtain the height and width of the frame source so that we can store the data into the 4-channel matrix. When the experiment was done, we released all the source and frame by adding these codes at the end:

```

myReader -> Release();

myDescription -> Release();

```

```
mySource -> Release();
```

```
mySensor -> Close();
```

```
mySensor -> Release();
```

An experiment was conducted in Visual Studio 2015 to test the Kinect color sensor. Just as shown in Figure 14, the Kinect color sensor worked properly. However we noticed that the image taken by the Kinect color sensor is a mirrored image, it needs to be further processed in the later experiment.



*Figure 14.* Image taken by the color sensor.

#### 4.2 Initiating the Depth Sensor

The Kinect depth sensor initiation was similar to the color sensor initiation. The first thing we did was turning on the Kinect sensor using the codes:

```
IKinectSensor *mySensor = nullptr;
```

```
GetDefaultKinectSensor (&mySensor);
```

```
mySensor -> Open ();
```

In order to obtain the depth frame, the depth source needs to be obtained using the codes:

```
IDepthFrameSource *MySource = nullptr;

MySensor->get_DepthFrameSource (&mySource);
```

Then we need to open the depth source reader:

```
IDepthFrameReader *MyReader = nullptr;

MySource->OpenReader (&myReader);
```

After the source reader was turned on, we got the depth frame from the source reader. The width and height of the depth frame were also required for further calculation:

```
int Height = 0, Width = 0;

IFrameDescription *MyDescription = nullptr;

MySource->get_FrameDescription (&MyDescription);

MyDescription->get_Height (&height);

MyDescription->get_Width (&width);
```

In order to measure the depth distance from the subject in the depth image, we need to know the coordinate of exact point in the image and use the equation  $Q = x + y * 512$  to calculate the value Q, which represents the pixel number in the depth image. Because the depth sensor has a resolution of 512 x 424, the depth space of the depth sensor describes a 2D location on the depth image where x is the column and y is the row and  $x = 0, y = 0$  corresponds to the top left corner of the image.

We have conducted an experiment to test the accuracy and the range of the depth sensor of the Kinect sensor in indoor environment using the codes:



```

IDepthFrame *MyFrame = nullptr;

while (1)

if (MyReader->AcquireLatestFrame(&MyFrame) == S_OK)

{

UINT size = 0;

UINT16 *buffer = nullptr;

MyFrame->AccessUnderlyingBuffer (&size,&buffer);

cout << buffer[width * (height / 2) + width / 2] << endl;

MyFrame->Release ();

}

MyReader->Release ();

MySource->Release ();

mySensor->Close ();

mySensor->Release ();

```

We can see that indoor environment distance measured by the depth sensor (see Table 3). The distance measured by the depth sensor has a difference of about 1 centimeter. This could be caused by the lighting condition of the environment. Also, the objects with different color have impacts on the measurement accuracy due to the different reflectance for infrared.

Since the difference of the distance measured is usually about 1 centimeter, we will consider fixing this error when we process the data from the Kinect depth sensor by subtracting 1 and then the data returned by the depth sensor should be acceptable.

Table 3. Indoor Distance Measured by the Depth Sensor

Distance(cm) measured by laser distance meter	Distance (cm)measured by Kinect
0	0
25	0
50	51.3
75	76.4
100	101.2
200	201.2
300	301.4
400	400.9
500	501.1

#### 4.3 Image Processing Using OpenCV

The image taken by the Kinect color sensor shows that it is a mirrored image. So the image needs to be process by OpenCV library so that the QR code in the image can be read correctly. OpenCV library provides a function named “flip” to flip an image either around y-axis or x-axis:

```
void flip(const Mat&src, Mat&dst, int flipCode)
```

The parameters:

src – Input array;

dst – Output array of the same size and type as src;

flipCode – a flag to specify how to flip the array; 0 means flipping around the x-axis, positive means flipping around y-axis, and negative means flipping around both axes.

The color space of the Kinect color sensor describes a 2D location on the color image where  $x$  is the column and  $y$  is the row. So  $x = 0, y = 0$  corresponds to the top left corner of the image and  $x = 1919, y = 1079$  ( $\text{width} - 1, \text{height} - 1$ ) corresponds to the bottom right [9]. From Figure 15, we can see that the image taken by the color sensor was a mirrored image, we need to flip the image around the  $y$ -axis for further processing. The code to flip the image was:

```
flip (img, img_flipped, 1);
```

“img” in the function is the source image and the “img\_flipped” means the destination image. The “flipCode” number “1” means the image is flipped around  $y$ -axis (see Figure 15).



*Figure 15.* Image flipped using OpenCV

In order to find the region of interest within the image for further processing, we converted the image into grayscale. OpenCV library provides a function “cvtColor” to convert an image from one color space to another [22]:

```
void cvtColor (InputArray src, OutputArray dst, int code, int dstCn = 0)
```

The parameter:

src – Source image;

dst – Destination image;

code – The color space conversion code;

dstCn – the number of channels in the destination image; if the parameter is 0.

The number of the channels will be derived automatically from src and the code [22].

In this case that we wanted to convert the image from RGB format to gray, the conversion code is "CV\_RGB2GRAY". We can see as shown in Figure 16, the image has been converted to grayscale by using the codes:

```
cvtColor(img_flip, img_gray, COLOR_RGB2GRAY);
```



*Figure 16. Image grayscale*

Then we used the Canny algorithm to process the grayscale to extract useful structural information and reduce the amount of data to be processed. The function “Canny” can find edges in an image using the Canny algorithm [11]:

```
void Canny (InputArray image, OutputArray edges, double threshold1, double
threshold2, int apertureSize = 3, bool L2gradient = false)
```

The parameter:

image – Single-channel 8-bit input image;

edges – Output edge map, it has the same size and type as image;

threshold1 – First threshold for the hysteresis procedure;

threshold2 – Second threshold for the hysteresis procedure;

apertureSize – Aperture size for the Sobel() operator;

L2gradient – a flag, indicating whether a more accurate  $L_2$  norm =

$\sqrt{(dI/dx)^2 + (dI/dy)^2}$  should be used to calculate the image gradient magnitude, or whether the default  $L_1$  norm =  $|dI/dx| + |dI/dy|$  is enough [11].

All we need are the parameter “image”, “edges”, “threshold1” and “threshold2”, the rest of the parameters can use default values. So the function can be:

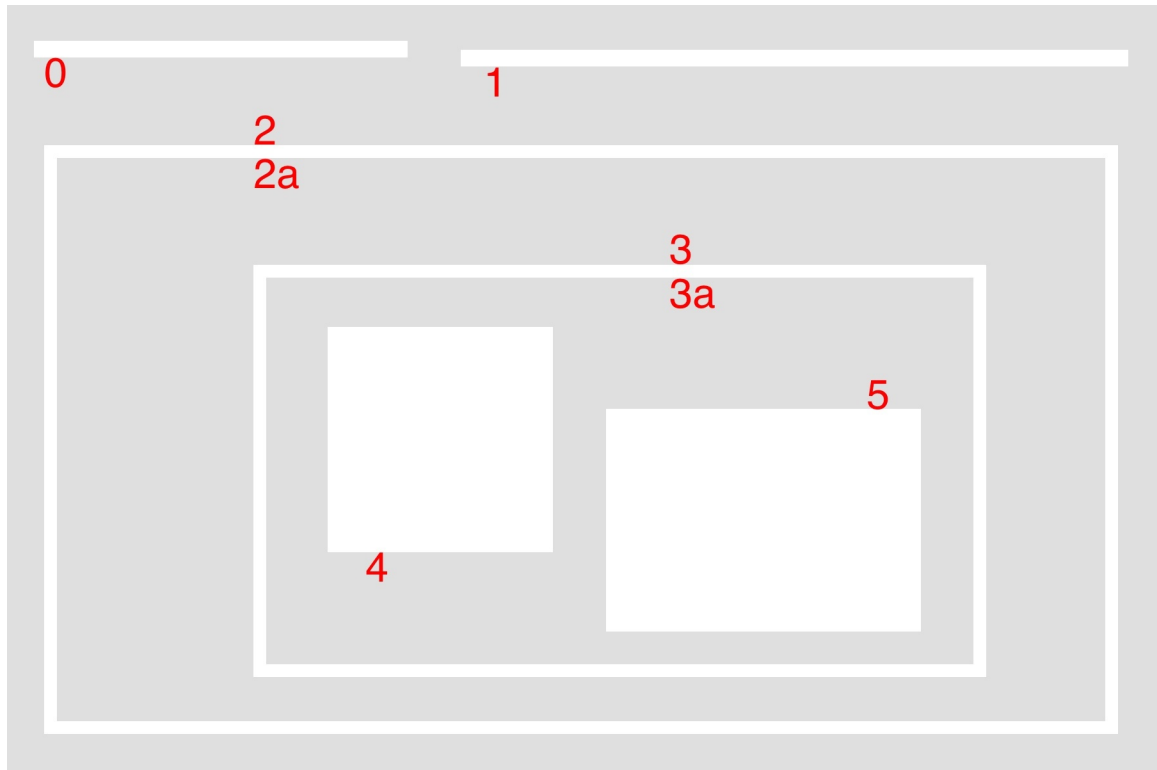
```
Canny(img_gray, edges, 100, 200);
```

The reason we choose 100 and 200 for “threshold1” and “threshold2” are from experiments (see Figure 17). These values differ depending on different lighting conditions. We realize that when we use 100 and 200 for the upper and lower limits respectively, the contours can be found quickly with accuracy.



*Figure 17.* The image processed using canny algorithm.

Before we dive into the contour hierarchy processing, we need to understand the hierarchy of contours first. We can see from Figure 18 [7], there are some shapes locating inside other shapes. In this case, we can call outer one as parent and inner one as child [8]. So that contours in an image has some relationship to each other. The relationship is called Hierarchy. We can state that contours 0, 1, 2 are external or outermost and they are in same hierarchy level, which is hierarchy-0. Contour 2a can be considered as a child of contour 2 and contour 2a can be considered in hierarchy-1. Contour 3 is also child of contour 2 and it is in hierarchy-2. We can consider that contour 3 is parent of contour 3a. Finally contours 4, 5 are the children of contour 3a and they come in the last hierarchy level [8].



*Figure 18.* Contour hierarchy.

After we understand the hierarchy of contours, we can see that the Figure 19 [19] shows the contour hierarchy of the position detection pattern of the QR code. The hierarchy array will differ depending on the different modes. Now that we have already converted the image to grayscale and processed the image using the Canny algorithm, all we need to do next is using the function “findContours” to retrieve all the contours that has more than 5 hierarchies and the function “drawContours” to highlight all of them and show them on the screen. So that we can see clearly that if the Kinect sensor has detected the QR codes in the image.

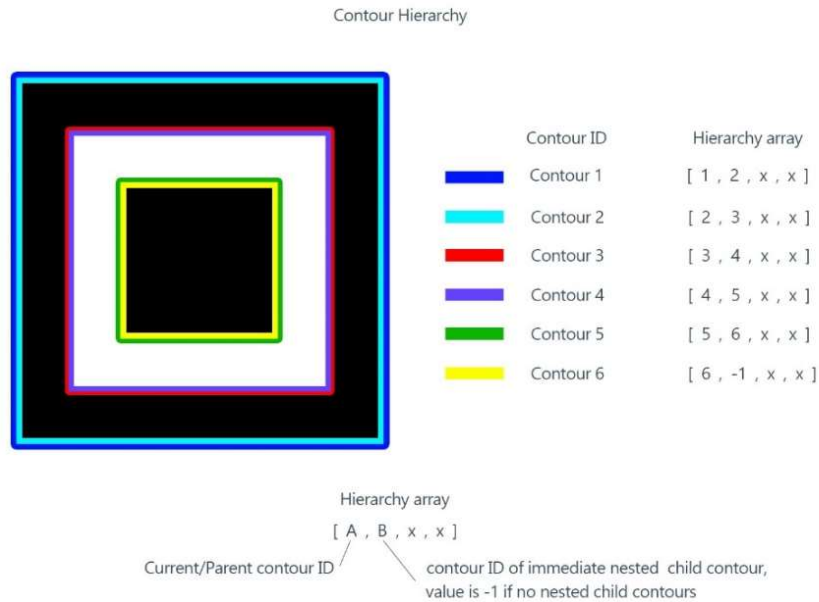


Figure 19. The contour hierarchy of a position detection pattern.

The function “findContours” from the OpenCV library can be used to find contours in a binary image. We can see the function [29]:

```
void findContours (InputOutputArray image, OutputArrayOfArrays contours,
OutputArray hierarchy, int mode, int method, Point offset = Point())
```

Parameters:

image – An 8-bit single-channel image;

contours – Detected contours;

hierarchy – Optional output vector which containing information about the image topology;

mode – Contour retrieval mode. There are four modes:

“CV\_RETR\_EXTERNAL” retrieves only the extreme outer contours,

“CV\_RETR\_LIST” retrieves all of the contours without establishing any hierarchical



relationships, “CV\_RETR\_CCOMP” retrieves all of the contours and organizes them in to a two-level hierarchy, “CV\_RETR\_TREE” retrieves all of the contours and reconstructs a full hierarchy of nested contours. We use “CV\_RETR\_TREE” in this experiment.

method – Contour approximation method, including

“CV\_CHAIN\_APPROX\_NONE”, “CV\_CHAIN\_APPROX\_SIMPLE” and

“CV\_CHAIN\_APPROX\_TC89\_L1, CV\_CHAIN\_APPROX\_TC89\_KCOS”.

“CV\_CHAIN\_APPROX\_SIMPLE” is used for the method, which means compresses horizontal, vertical, and diagonal segments and leaves only their end points [29].

So this function turned out to be:

```
findContours (edges, contours, hierarchy, RETR_TREE,
CHAIN_APPROX_SIMPLE).
```

The function “drawContours” is to draw contours outlines or filled contours [29]:

```
void drawContours (InputOutputArray image, InputArrayOfArrays contour, int
contourIdx, const Scalar& color, int thickness = 1, int line Type = 8, InputArray
hierarchy = noArray(), int maxLevel = INT_MAX, Point offset = Point())
```

Parameters:

image – Destination image;

contours – All the input contours.

contourIdx – Parameter indicating a contour to draw with;

line Type – Line connectivity;

hierarchy – Optional information about hierarchy;

maxLevel – Maximal level for drawn contours;

offset – Optional contour shift parameter;

contour – Pointer to the first contour;

externalColor – Color of external contours;

holeColor – Color of internal contours.

In this experiment, we drew the line with red color, which meant we've used "Scalar (0, 0, 255)" as our "color" parameter. In order to make the line more obvious, we made the "thickness" equal 2. So the programming codes were:

```
vector<vector<Point> > contours;

vector<Vec4i> hierarchy;

Point2f vtx[4];

findContours(edges, contours, hierarchy, RETR_TREE,
CHAIN_APPROX_SIMPLE);

for (int i = 0; i < contours.size(); i++)
{
    int k = i;
    int c = 0;
    while (hierarchy[k][2] != -1)
    {
        k = hierarchy[k][2];
        c = c + 1;
    }
    if (c >= 5)
    {
```

```

RotatedRect box = minAreaRect(contours[i]);

box.points(vtx);

drawContours(img_flip, contours, i, Scalar(0, 0, 255), 2, 8, hierarchy, 0);

}

}

```

The “`contour.size ()`” means the number of contours that has been found. Since we chose “`CV_RETR_TREE`” mode for the function “`findContour`”. It retrieved all the contours and create a full family hierarchy list. Every detected contour has a hierarchy array and the third element in the hierarchy represents the child contour. If there is no child contour, the third element of the hierarchy will be -1. So the programming code “`while (hierarchy[k] [2] != -1)`” was to examine the hierarchy number of every contour. Then every contour that has more than 5 hierarchy levels, we drew a box with red color lines around it. When three position detection patterns of a QR code has been highlighted by OpenCV, it meant that the QR code can be read by ZBar library.

We can see in the Figure 20, it shows that the position detection patterns of the four QR codes has been recognized and highlighted on screen by OpenCV.

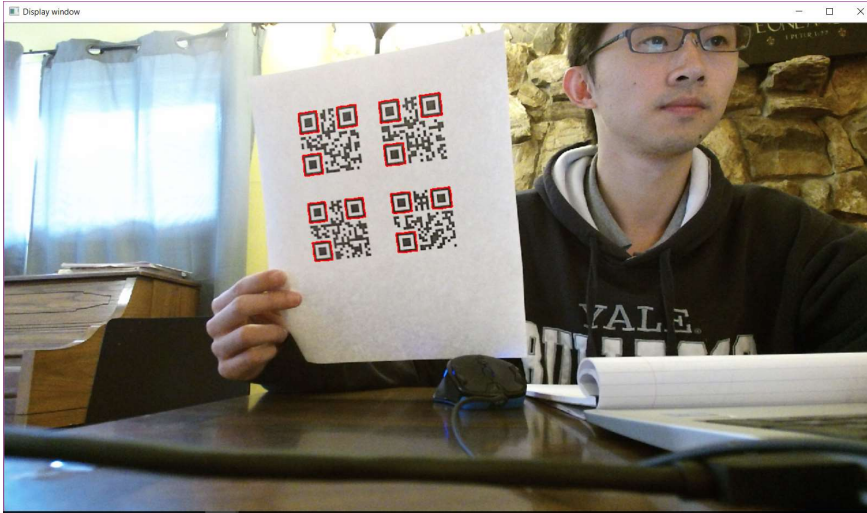


Figure 20. Four QR codes in paper A.

#### 4.4 Decoding QR Codes Using ZBar Library

When the position detection patterns of a QR code has been recognized, we used ZBar library to decode the QR code and extract the information from it. Using ZBar library to decode multiple QR codes can be quick and accurate. First, we created a scanner and configure the scanner:

```
ImageScanner scanner;
```

```
scanner.set_config(ZBAR_QRCODE, ZBAR_CFG_ENABLE, 1);
```

The parameter “ZBAR\_QRCODE” means the scanner will seek for the QR codes in the image, the “ZBAR\_CFG\_ENABLE” control whether specific symbologies will be recognized, the value 1 means for specific symbology.

```
int Width = img_gray.cols;
```

```
int Height = img_gray.rows;
```

```
Image image(Width, Height, "Y800", img_gray.data, Width * Height);
```

```
int n = scanner.scan(image);
```

Since the color has no effect on the QR code reading, we can just use the grayscale image processed by OpenCV. The first two lines of the codes is to obtain the width and height of the image. The third line is to store image data alone with associated format and size metadata. The fourth line is to scan the QR codes of the image. For extracting the results from the QR codes, we use:

```

    for (SymbolIterator symbol = image.symbol_begin(); symbol !=
image.symbol_end(); ++symbol)
    {
        DecodedMsg[N] = symbol->get_data();
        for (int j = 0; DecodedMsg[N][j] != '\0'; j++)
        {
            if (DecodedMsg[N][j] == '$')
            for (int i = 0; i < 4; i++)
            {
                DecodedMsgNum_x[N][i] = DecodedMsg[N][j + 2 + i] - '0';
                DecodedMsgNum_y[N][i] = DecodedMsg[N][j + 8 + i] - '0';
                DecodedMsgNum_z[N][i] = DecodedMsg[N][j + 14 + i] - '0';
            }
            break;
        }

        X_[N] = DecodedMsgNum_x[N][0] * 1000 + DecodedMsgNum_x[N][1] * 100 +
DecodedMsgNum_x[N][2] * 10 + DecodedMsgNum_x[N][3];

```

```

Y_[N] = DecodedMsgNum_y[N][0] * 1000 + DecodedMsgNum_y[N][1] * 100 +
DecodedMsgNum_y[N][2] * 10 + DecodedMsgNum_y[N][3];

Z_[N] = DecodedMsgNum_z[N][0] * 1000 + DecodedMsgNum_z[N][1] * 100 +
DecodedMsgNum_z[N][2] * 10 + DecodedMsgNum_z[N][3];

DecodedQR_x[N] = symbol->get_location_x(N);
DecodedQR_y[N] = symbol->get_location_y(N);

N++;

}

```

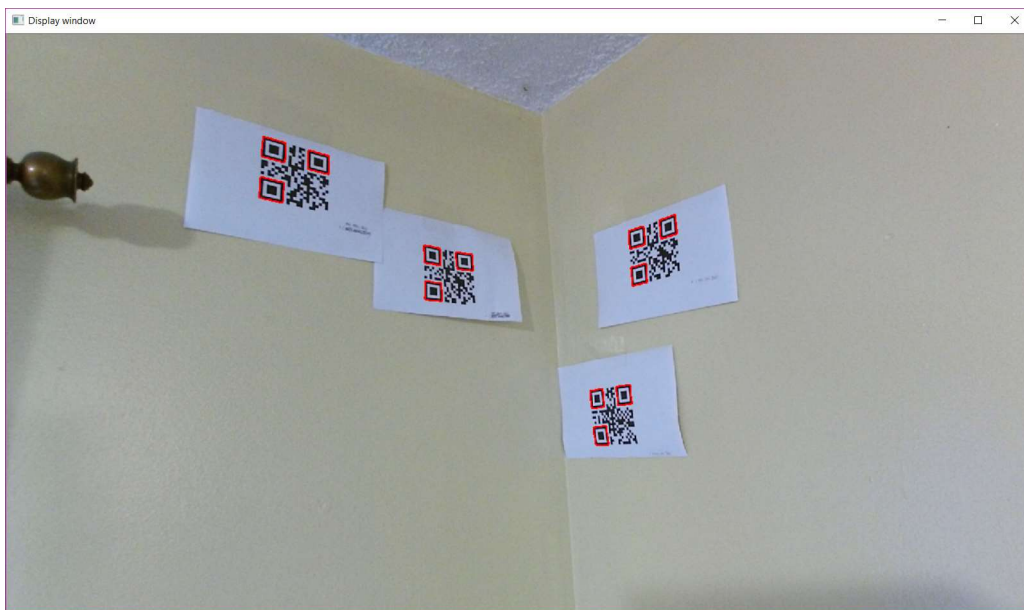
The “symbol->get\_data ()” is the decoded information of the QR code, we put it into the string “DecodedMsg [N]”. We should mention that the QR codes we generated includes the coordinate format such as “\$X1234\$Y1234\$Z1234”, so we extract the coordinate information within the QR code and put them into separate array respectively, such the QR code x-axis coordinate is stored in the array “X\_[N]”. “symbol->get\_location\_x(N)” and “symbol->get\_location\_y(N)” represent the x and y coordinate of the QR code respectively on the image and they are stored in the “DecodedQR\_x[N]” and “DecodedQR\_y[N]” respectively.

In order to test if the codes work properly, we generated four QR codes with different coordinate information and printed them in paper B, then used Kinect camera to read the QR codes. We can see from Figure 21 that the position detection patterns of four QR codes had been recognized and highlighted.



#### 4.5 Coordinate Calculation

Since the Kinect sensor has been working properly on image processing and QR code decoding, we used Newton-Raphson method to solve the GPS system equations to calculate the coordinate position of the Kinect sensor. We have conducted an experiment to test the result. We can see from Figure 23, there were four QR codes attached on the wall.



*Figure 23.* QR codes in position A.

After the QR codes had been decoded by the Kinect sensor, the Figure 24 shows that the first QR code is (500, 400, 500) and the distance to it is 575 mm, the second QR code is (300,800, 400) and the distance to it is 629 mm, the third QR code is (300, 600, 500) and the distance to it is 615 mm, the fourth QR code is (400, 400, 700) and the distance to it is 637 mm.



## The Newton-Raphson Method

$$f_i = (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 - (d_i - T)^2 = 0$$

Then we also assumed that:

$$F \begin{bmatrix} x \\ y \\ z \\ T \end{bmatrix} = \begin{bmatrix} f_1(x, y, z, T) \\ f_2(x, y, z, T) \\ f_3(x, y, z, T) \\ f_4(x, y, z, T) \end{bmatrix} = \begin{bmatrix} (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 - (d_1 - T)^2 = 0 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 - (d_2 - T)^2 = 0 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 - (d_3 - T)^2 = 0 \\ (x - x_4)^2 + (y - y_4)^2 + (z - z_4)^2 - (d_4 - T)^2 = 0 \end{bmatrix}$$

The derivative of  $F$  is the 4x4 Jacobian matrix given by:

$$\begin{aligned}
J(x, y, z, T) &= \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial z} & \frac{\partial f_1}{\partial T} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial z} & \frac{\partial f_2}{\partial T} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial z} & \frac{\partial f_3}{\partial T} \\ \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial y} & \frac{\partial f_4}{\partial z} & \frac{\partial f_4}{\partial T} \end{bmatrix} \\
&= \begin{bmatrix} 2(x - x_1) & 2(y - y_1) & 2(z - z_1) & 2(d_1 - T) \\ 2(x - x_2) & 2(y - y_2) & 2(z - z_2) & 2(d_2 - T) \\ 2(x - x_3) & 2(y - y_3) & 2(z - z_3) & 2(d_3 - T) \\ 2(x - x_4) & 2(y - y_4) & 2(z - z_4) & 2(d_4 - T) \end{bmatrix}
\end{aligned}$$

The function G is defined as (Leibovici, C., 2015):

$$G(x) = x - J(x)^{-1}F(x)$$

where  $J(x)^{-1}$  is the inverse matrix of  $J(x)$ .

From the Newton-Raphson method, we reiterated the function starting with an initial  $x_0$  and generating for  $n \geq 1$ :

$$x_n = G(x_{n-1}) = x_{n-1} - J(x_{n-1})^{-1}F(x_{n-1})$$

We can write as:

$$\begin{bmatrix} x_n \\ y_n \\ z_n \\ T_n \end{bmatrix} = \begin{bmatrix} x_{n-1} \\ y_{n-1} \\ z_{n-1} \\ T_{n-1} \end{bmatrix} - \left( J(x_{n-1}, y_{n-1}, z_{n-1}, T_{n-1}) \right)^{-1} F(x_{n-1}, y_{n-1}, z_{n-1}, T_{n-1})$$

The initial guess:

$$x_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ T_0 \end{bmatrix} = \begin{bmatrix} 500 \\ 500 \\ 500 \\ 500 \end{bmatrix}$$

So we use C++ programming language to achieve the goal, the programming codes were:

```
float X_[4], Y_[4], Z_[4], D_[4], R_[4];
```

```

float X0 = 500, Y0 = 500, Z0 = 500, T0 = 500;

float X_Recur[10], Y_Recur[10], Z_Recur[10], T_Recur[10];

void GPS(void)

{
    int flag = 0;

    for (int i = 0; i < 4; i++)

    {
        J[i][0] = 2 * (X0 - X_[i]);

        J[i][1] = 2 * (Y0 - Y_[i]);

        J[i][2] = 2 * (Z0 - Z_[i]);

        J[i][3] = 2 * (D_[i] - T0);

        F[i] = pow(X0 - X_[i], 2) + pow(Y0 - Y_[i], 2) + pow(Z0 - Z_[i], 2) - pow(D_[i] -
T0, 2);

    }

    for (int j = 0; j < 6; j++)

    {

        if (flag == 1)

            for (int i = 0; i < 4; i++)

            {

                J[i][0] = 2 * (X_Recur[j] - X_[i]);

                J[i][1] = 2 * (Y_Recur[j] - Y_[i]);

                J[i][2] = 2 * (Z_Recur[j] - Z_[i]);

                J[i][3] = 2 * (D_[i] - T_Recur[j]);

```

```

    F[i] = pow(X_Recur[j] - X_[i], 2) + pow(Y_Recur[j] - Y_[i], 2) +
pow(Z_Recur[j] - Z_[i], 2) - pow(D_[i] - T_Recur[j], 2);

    }

    Mat Jaco(4, 4, CV_32F, J);

    Mat InvJaco = Jaco.inv();

    X_Recur[j + 1] = X_Recur[j] - (InvJaco.at<float>(0, 0) * F[0] +
InvJaco.at<float>(0, 1) * F[1] + InvJaco.at<float>(0, 2) * F[2] + InvJaco.at<float>(0, 3) *
F[3]);

    Y_Recur[j + 1] = Y_Recur[j] - (InvJaco.at<float>(1, 0) * F[0] +
InvJaco.at<float>(1, 1) * F[1] + InvJaco.at<float>(1, 2) * F[2] + InvJaco.at<float>(1, 3) *
F[3]);

    Z_Recur[j + 1] = Z_Recur[j] - (InvJaco.at<float>(2, 0) * F[0] +
InvJaco.at<float>(2, 1) * F[1] + InvJaco.at<float>(2, 2) * F[2] + InvJaco.at<float>(2, 3) *
F[3]);

    T_Recur[j + 1] = T_Recur[j] - (InvJaco.at<float>(3, 0) * F[0] +
InvJaco.at<float>(3, 1) * F[1] + InvJaco.at<float>(3, 2) * F[2] + InvJaco.at<float>(3, 3) *
F[3]);

    flag = 1;

    }

    }

```

The “X\_Recur”, “Y\_Recur” and “Z\_Recur” in the codes are the x, y and z of the Kinect camera in the coordinate system, the “T\_Recur” is the time offset  $ct_B$ . We can see

from Figure 25, the QR codes attached on the wall has been captured by the Kinect camera.



*Figure 25.* QR codes in position B.

After the QR codes were decoded by the ZBar library, the decoded information was shown on the screen (see Figure 26). In this experiment, in order to make the result more accurate, we have reiterated the function for 6 times. The screen showed that the coordinate of the first QR code was (200, 400, 500) and the distance to the QR code was 882 mm, the second coordinate of the QR code was (600, 400, 200) and the distance to the QR code was 895 mm, the third coordinate of the QR code was (500, 400, 800) and the distance to the QR code was 830 mm, the fourth coordinate of the QR code was (0, 700, 500) and the distance to the QR code was 862 mm. The coordinate of the Kinect camera was (478.989, 770.76, 435.299) and the time offset is  $4.5 \times 10^{-6}$ s. From the experiment, we can see that the Newton-Raphson method works very well in Visual Studio 2015 to solve the GPS system equations.

#### 4.6 Error Analysis

In order to test the accuracy of the experiment, we conducted some experiments to calculate the errors and standard deviation of the Kinect depth sensor. From the following experiments, we collected 100 groups of the depth data, then compare with the actual distance measured by a laser distance meter and we obtained 100 groups of errors, after that we computed the mean and the standard deviation of the errors from the data.

The first experiment we conducted was keeping the Kinect depth sensor measuring the distance to a QR code and collecting the depth data for 100 groups, then compared with the accurate data measured by a laser distance meter to see the error changes. We assumed:

$$E = r_{\text{Kinect}} - r_{\text{true}}$$

Here,  $E$  represents error,  $r_{\text{Kinect}}$  is the distance between the Kinect sensor and the QR code and  $r_{\text{true}}$  is the actual distance measured by the laser distance meter.

We can see the results of Kinect depth sensor measuring the distance to a QR code for 100 groups of data in different location from Table 4. The errors and the standard deviation show that even the distance become bigger, the Kinect depth sensor can still collect the depth data stably.

Table 4. Mean Error and Standard Deviation of the Experiments

True Distance (mm)	804	1007	1201	1401	1600	1803	2003
Mean Measured Distance (mm)	809.074	1021.74	1219.84	1409.25	1556.03	1781.64	2018.06
Errors (mm)	5.074	14.74	18.84	8.25	-43.97	-21	15.06
Standard Deviation (mm)	1.33419	1.67557	2.82487	1.54034	1.4509	1.46245	2.45005

We can see the Figure 27 that when the depth data was collected, there is a constant fluctuation. The fluctuation of the measured data could be caused by the lighting condition of the indoor environment and the color of the wall. Because the Kinect depth sensor uses infrared light, which could be interfered by the dark or reflective surfaces. Generally, the errors and the standard deviation are acceptable.

```

F:\C++Programs\Project3\Debug\Project3.exe
#74 distance is: 805.862 mm.
#75 distance is: 808.132 mm.
#76 distance is: 809.267 mm.
#77 distance is: 808.132 mm.
#78 distance is: 809.267 mm.
#79 distance is: 808.132 mm.
#80 distance is: 806.997 mm.
#81 distance is: 810.403 mm.
#82 distance is: 811.538 mm.
#83 distance is: 809.267 mm.
#84 distance is: 809.267 mm.
#85 distance is: 808.132 mm.
#86 distance is: 810.403 mm.
#87 distance is: 808.132 mm.
#88 distance is: 808.132 mm.
#89 distance is: 808.132 mm.
#90 distance is: 809.267 mm.
#91 distance is: 806.997 mm.
#92 distance is: 810.403 mm.
#93 distance is: 808.132 mm.
#94 distance is: 809.267 mm.
#95 distance is: 809.267 mm.
#96 distance is: 809.267 mm.
#97 distance is: 806.997 mm.
#98 distance is: 810.403 mm.
#99 distance is: 809.267 mm.
#100 distance is: 810.403 mm.
The mean is: 809.074 mm
The standard deviation is: 1.33419 mm

```

*Figure 27.* The Kinect depth sensor collected the depth data in a distance of 804 mm.

#### 4.6.2 Second Static Experiment –Errors of Calculating the Position of the Kinect Sensor

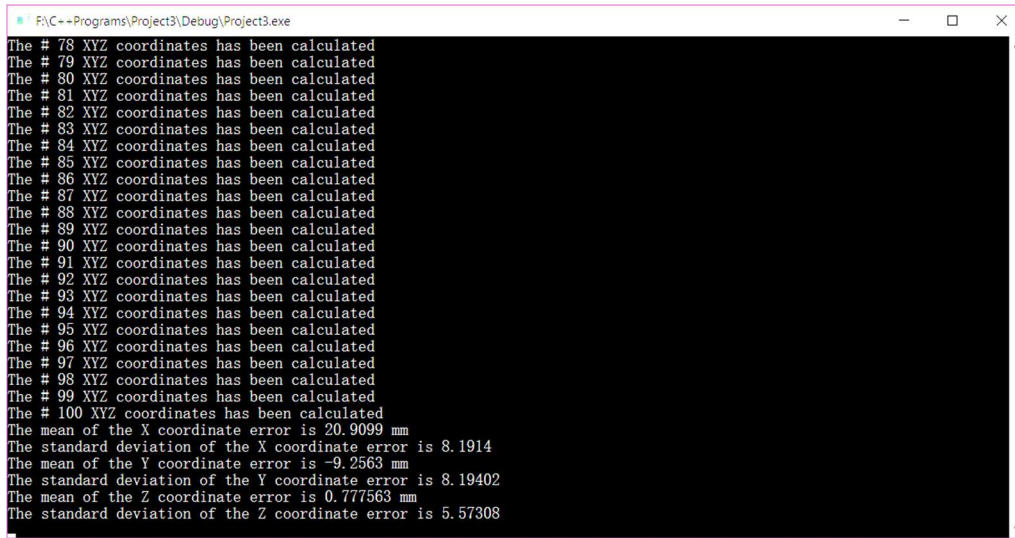
After we have tested the accuracy of the depth sensor, we conducted another experiment to see how the errors changes by measuring multiple QR codes at the same time to calculate the exact location of the Kinect camera (see Figure 28).





*Figure 28.* QR codes attached on the wall.

There are four new QR codes attached on the wall, we set up the Kinect sensor in a fixed location (905, 854, 809), this coordinate was measured by the laser distance meter. Then the Kinect sensor kept measuring the depth data of the QR codes and collected 100 groups of data and compared with the actual coordinate of the Kinect sensor, then calculated the mean and standard deviation of the errors of the xyz-coordinate, as shown in Figure 29.



```

F:\C++\Programs\Project3\Debug\Project3.exe
The # 78 XYZ coordinates has been calculated
The # 79 XYZ coordinates has been calculated
The # 80 XYZ coordinates has been calculated
The # 81 XYZ coordinates has been calculated
The # 82 XYZ coordinates has been calculated
The # 83 XYZ coordinates has been calculated
The # 84 XYZ coordinates has been calculated
The # 85 XYZ coordinates has been calculated
The # 86 XYZ coordinates has been calculated
The # 87 XYZ coordinates has been calculated
The # 88 XYZ coordinates has been calculated
The # 89 XYZ coordinates has been calculated
The # 90 XYZ coordinates has been calculated
The # 91 XYZ coordinates has been calculated
The # 92 XYZ coordinates has been calculated
The # 93 XYZ coordinates has been calculated
The # 94 XYZ coordinates has been calculated
The # 95 XYZ coordinates has been calculated
The # 96 XYZ coordinates has been calculated
The # 97 XYZ coordinates has been calculated
The # 98 XYZ coordinates has been calculated
The # 99 XYZ coordinates has been calculated
The # 100 XYZ coordinates has been calculated
The mean of the X coordinate error is 20.9099 mm
The standard deviation of the X coordinate error is 8.1914
The mean of the Y coordinate error is -9.2563 mm
The standard deviation of the Y coordinate error is 8.19402
The mean of the Z coordinate error is 0.777563 mm
The standard deviation of the Z coordinate error is 5.57308

```

Figure 29. The mean and standard deviation of the errors.

The data from Figure 29 is listed in Table 5, it shows the mean and the standard deviation of the errors from 100 groups of data collected. The mean of the x-coordinate is relatively large, it might be caused by the distortion of the QR code image taken by the Kinect color sensor since one of the QR codes had a certain angle towards the Kinect sensor.

Table 5. Mean Error and Standard Deviation of the XYZ-Coordinate

Coordinate	Mean Error (mm)	Standard deviation (mm)
X	20.9099	8.1914
Y	-9.2563	8.1940
Z	0.7776	5.5730

## CHAPTER 5

### CONCLUSION AND FUTURE STUDY

From the experiments we have conducted, the Kinect color sensor was used to capture high definition images, the OpenCV library was applied to process the image and recognize and highlight the QR codes. The depth sensor measured the distances between the Kinect sensor and QR codes. After the QR codes in the image were recognized, ZBar library decoded the QR codes and returned the decoded information for further process. After the data was obtained, we used the Newton-Raphson method to solve the system equations and calculate the exact position of the Kinect sensor.

However, we noticed that the RGB camera and the depth camera of the Kinect are in different locations. The RGB camera has higher definition, so the image taken by the RGB camera is wider than the depth camera. On the other hand, the depth sensor has a higher view than the RGB camera. That means there is an overlap part of view in the center of the two cameras. So in order to obtain the accurate data for further process, we need to calibrate or register these two cameras.

Secondly, during the experiments, we realized that the measuring angle had certain influence on the depth sensor. The closer to the central point of the depth sensor, the more accurate the data could be collected. This could be caused by the attenuation of the infrared transmission since the central point of the depth image has shorter distance to the depth sensor than the four corners.

For all to say, the Kinect v2 is a very potential and comprehensive motion sensor. With the RGB camera and the depth sensor, it also can be applied to scan the environment and draw a 3D color image of the environment. The depth sensor can also help robots for obstacles avoidance. The body tracking, facial recognition and gesture recognition applications will make robot capable of communicating with people easily. More applications can be developed for the Kinect sensor to be a more important part of robots or unmanned vehicles.

## REFERENCES

- [1] Amzoti. (2015, Mar 7). Solving a non-linear, multivariable system of equations. Retrieved from <https://math.stackexchange.com/questions/1179036/solving-a-non-linear-multivariable-system-of-equations>
- [2] Ashley, J. (2014, March 5). QUICK REFERENCE: KINECT 1 VS KINECT 2. Retrieved from <http://www.imaginativeuniversal.com/blog/2014/03/05/Quick-Reference-Kinect-1-vs-Kinect-2/>
- [3] Barcodes. (n. d.). In SATO. Retrieved June 1, 2017. Available from <http://www.satoasiapacific.com/singapore/products/barcode.aspx>
- [4] Beginner\_YH. (2016, February 8). Kinect For Windows V2 开发日志五：使用 OpenCV 显示彩色图像及红外图像. Retrieved from <http://www.cnblogs.com/xz816111/p/5184881.html>
- [5] Biswas, J., Veloso, M. (2010, July 15). WiFi localization and navigation for autonomous indoor mobile robots. IEEE. doi: 10.1109/ROBOT.2010.5509842
- [6] Brown, J. (2011). ZBar bar code reader. Available from <http://zbar.sourceforge.net/>
- [7] CallMeWhy. (2016, August 24). 学习笔记：使用 OpenCV 识别 QRCode. Retrieved from <https://blog.callmewhy.com/2016/04/23/opencv-find-qrcode-position/>
- [8] Contours Hierarchy. (2015, December 18). In OpenCV. Retrieved May 29, 2017. Available from [http://docs.opencv.org/3.1.0/d9/d8b/tutorial\\_py\\_contours\\_hierarchy.html](http://docs.opencv.org/3.1.0/d9/d8b/tutorial_py_contours_hierarchy.html)
- [9] Coordinate mapping. (n. d.). In Microsoft. Retrieved June 1, 2017. Available from <https://msdn.microsoft.com/en-us/library/dn785530.aspx>
- [10] C., Nathan. (2011, February 3). Kinect Depth vs. Actual Distance. Retrieved from <http://mathnathan.com/2011/02/depthvsdistance/>
- [11] Feature Detection. (n. d.). In OpenCV. Retrieved May 21, 2017. Available from [http://docs.opencv.org/2.4/modules/imgproc/doc/feature\\_detection.html?highlight=canny](http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny)

- [12] Frame QR. (n. d.). In DENSO WAVE. Retrieved May 28, 2017. Available from <http://www.qrcode.com/en/codes/frameqr.html>
- [13] Fry, B. & Reas, C. (2001). Processing. Retrieved from <https://processing.org/people/>
- [14] IKinectSensor Interface. (n. d.). In Microsoft. Retrieved June 1, 2017. Available from <https://msdn.microsoft.com/en-us/library/microsoft.kinect.kinect.ikinectsensors.aspx>
- [15] IQR Code. (n. d.). In DENSO WAVE. Retrieved May 28, 2017. Available from <http://www.qrcode.com/en/codes/iqr.html>
- [16] Kinect. (n. d.). In Wikipedia. Retrieved May 28, 2017, from <https://en.wikipedia.org/wiki/Kinect>
- [17] Kinect API Overview. (n. d.). In Microsoft. Retrieved June 1, 2017. Available from <https://msdn.microsoft.com/en-us/library/dn782033.aspx>
- [18] Leibovici, C. (2015, Mar 7). Solving a non-linear, multivariable system of equations. Retrieved from <https://math.stackexchange.com/questions/1179036/solving-a-non-linear-multivariable-system-of-equations>
- [19] Lojaya, L. (2014, Oct 25). OPENCV: QR CODE DETECTION AND EXTRACTION. Retrieved from <http://dsynflo.blogspot.in/2014/10/opencv-qr-code-detection-and-extraction.html>
- [20] Lower, B et al., (2014). Programming Kinect for Windows v2 Jump Start [PowerPoint slides]. Retrieved from <http://brightguo.com/k2-dev-video/>
- [21] Microsoft Visual Studio. (n. d.). In Wikipedia. Retrieved May 25, 2017, from [https://en.wikipedia.org/wiki/Microsoft\\_Visual\\_Studio](https://en.wikipedia.org/wiki/Microsoft_Visual_Studio)
- [22] Miscellaneous Image Transformations. (n. d.). In OpenCV. Retrieved May 21, 2017. Available from [http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous\\_transformations.html#cvTcvtColor](http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvTcvtColor)
- [23] Nelson, J. G. (2011, July 15). Hacking the Kinect. Retrieved from <http://nerdtrek.com/hacking-kinect/>
- [24] OpenCV. (n. d.). In Wikipedia. Retrieved May 20, 2017, from <https://en.wikipedia.org/wiki/OpenCV>
- [25] QR code. (n. d.). In Wikipedia. Retrieved May 28, 2017, from [https://en.wikipedia.org/wiki/QR\\_code](https://en.wikipedia.org/wiki/QR_code)

- [26] QR Code Introduction. (n. d.). In OnBarcode. Retrieved June 12, 2017. Available from [http://www.onbarcode.com/qr\\_code/](http://www.onbarcode.com/qr_code/)
- [27] Riyad A. EL-laithy, Jidong Huang & Michael Yeh. (2012). Study on the Use of Microsoft Kinect for Robotics Applications. Position Location and Navigation Symposium (PLANS), 2012 IEEE/ION. doi: 10.1109/PLANS.2012.6236985
- [28] SQRC. (n. d.). In DENSO WAVE. Retrieved May 28, 2017. Available from <http://www.qrcode.com/en/codes/sqrc.html>
- [29] Structural Analysis and Shape Descriptors. (n. d.). In OpenCV. Retrieved May 29, 2017. Available from [http://docs.opencv.org/2.4/modules/imgproc/doc/structural\\_analysis\\_and\\_shape\\_descriptors.html?highlight=findcontours#findcontours](http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours)
- [30] Taylor, L. (2017, Feb). What Are FrameQR Codes. Retrieved from <http://qrcode.meetheed.com/question37.php>
- [31] The GPS. (n. d.). Retrieved from <http://www.math.tamu.edu/~dallen/physics/gps/gps.htm>
- [32] Types of QR Code. (n. d.). In DENSO WAVE. Retrieved May 28, 2017. Available from <http://www.qrcode.com/en/codes/>
- [33] What is a QR code. (n. d.). In Keyence. Retrieved May 29, 2017. Available from [http://www.keyence.com/ss/products/auto\\_id/barcode\\_lecture/basic\\_2d/qr/](http://www.keyence.com/ss/products/auto_id/barcode_lecture/basic_2d/qr/)
- [34] Wilson, M. (2009, June 3). Testing Project Natal: We Touched the Intangible. Retrieved from <http://gizmodo.com/5277954/testing-project-natal-we-touched-the-intangible/>
- [35] 8bitjoystick. (2009, Jun 1). “Project Natal” 101. Retrieved from <http://blog.seattlepi.com/digitaljoystick/2009/06/01/e3-2009-microsoft-at-e3-several-metric-tons-of-press-releaseapalloza/>