Department of Computer Science and Engineering

The Chinese University of Hong Kong

2006 – 2007 Final Year Project Final Report

LYU 0604

Virtual Dart – an Augmented Reality Game on Mobile Device

Supervisor

Professor Michael R. Lyu

Prepared By:
Siu Ho Tung
19 April 2007

Abstract

In this report, we are going to describe the motivation, background information and problem encountered by our group when participating in the final year project. The objective of this project is to make use of camera reside in a mobile phone as well as the existing Motion Tracking Engine to implement a mobile game called “Virtual Dart”.

In the following sections, we would first introduce the idea of our final year project. Following is the introduction of Symbian OS (Nokia based S60 2nd & 3rd Edition), one of the popular Operating System used in modern Smart Phone. After that, we would present our program interface as well as our design and implementation concept. We would also like to discuss some algorithms which explored throughout the whole progress of our Final Year Project.

This report would include some experiment results which demonstrate our evaluation towards the available algorithms. In this report, we will use game and application interchangeably.

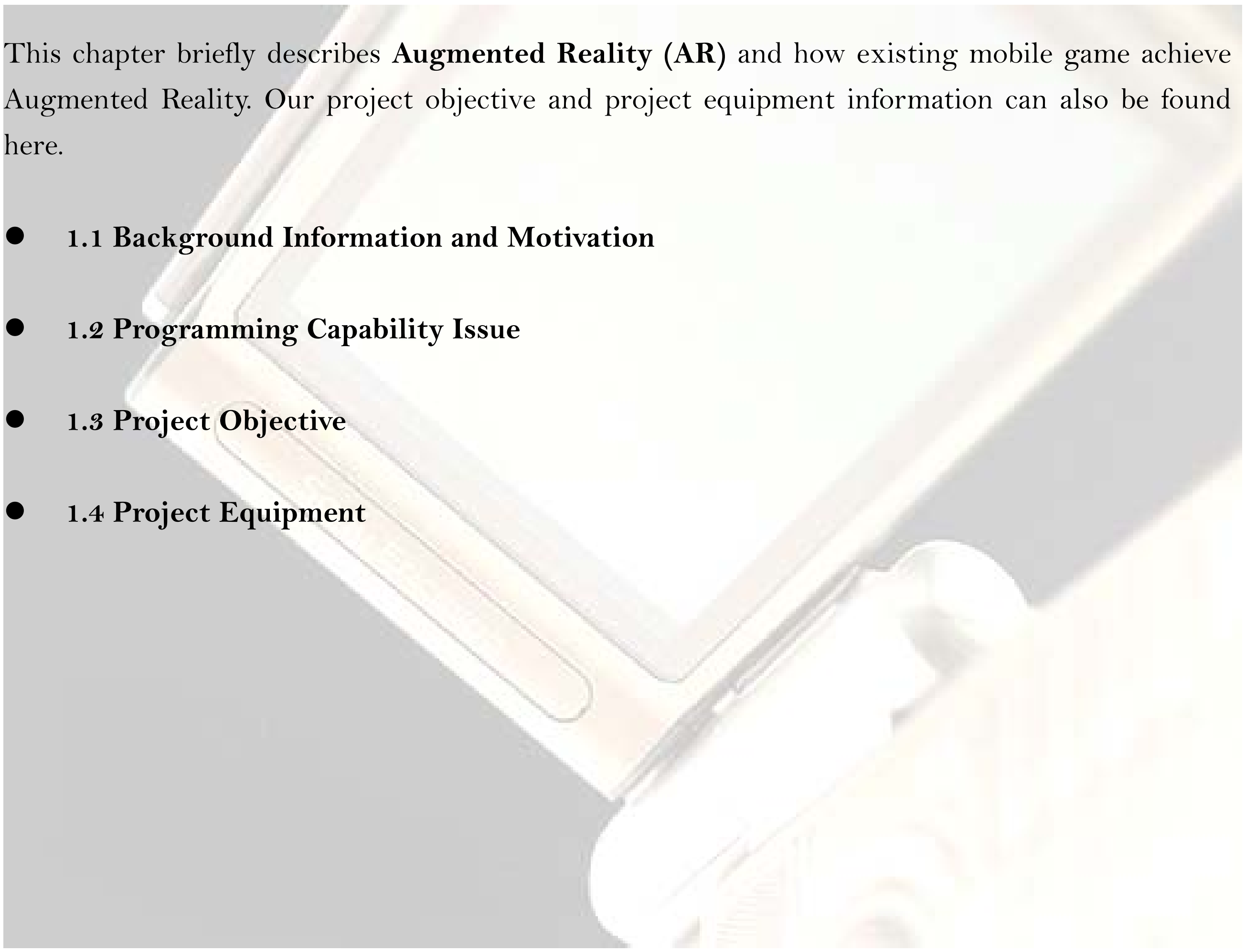
Any information appeared in semester 1 but may not be applicable in semester 2 would be marked as “Review” for just report completion purpose.

Abstract ... - 2 -
Chapter 1 ... - 5 -
Introduction ... - 5 -
1.1 Background Information and Motivation ... - 6 -
1.2 Programming Capability Issue ... - 8 -
1.3 Project Objective ... - 9 -
1.4 Project Equipment ... - 9 -
Chapter 2 ... - 10 -
Symbian Operating System Overview ... - 10 -
2.1 Basic Architecture of Symbian Operating System ... - 11 -
2.2 Development Environment ... - 13 -
2.3 Limitation of Symbian Phone ... - 15 -
2.4 Features Different in Symbian OS Series 60 2nd and 3rd Edition ... - 16 -
2.5 Porting Existing Program to Series 60 3rd Edition ... - 19 -
2.6 Why Symbian ... - 20 -
2.7 Conclusion ... - 20 -
Chapter 3 ... - 21 -
mVOTE Engine ... - 21 -
3.1 What is mVOTE? ... - 22 -
3.2 Block Matching Algorithm ... - 22 -
Chapter 4 ... - 25 -
Program Design and Implementation ... - 25 -
4.1 User Interface ... - 26 -
4.2 Program Assumption ... - 31 -
4.3 Projectile Motion Calculation ... - 33 -
4.4 Program Design ... - 36 -
4.5 Programming Trick ... - 39 -
4.6 Conceptual Program Prototype ... - 46 -
4.7 Feature Recognition in Semester 1 VS LED Recognition in Semester 2? ... - 50 -
Chapter 5 ... - 51 -
Object (LED) Recognition ... - 51 -
5.1 Introduction ... - 52 -
5.2 Harris Corner Detector in Semester 1 (Review) ... - 52 -
5.3 Fast Corner Detector in Semester 1 (Review) ... - 56 -
5.4 Select Feature from Corner List in Semester 1 (Review) ... - 60 -
5.5 Object Recognition Methods ... - 62 -
5.6 Feature Filter ... - 64 -
5.7 LED Recognition ... - 67 -

5.8 Motion Tracking Improvement .... - 70 -
Chapter 6 .... - 72 -
Project Progress, Difficulties and Future Work .... - 72 -
6.1 Project Progress .... - 73 -
6.2 Difficulties we face in the project .... - 74 -
6.3 Contribution of Work .... - 78 -
Chapter 7 .... - 80 -
Acknowledgement .... - 80 -
Chapter 8 .... - 81 -
References .... - 81 -
Appendix I .... - 83 -
Feature Selection (REVIEW) .... - 83 -
AI.2 Pictures under Normal Lighting Condition .... - 85 -
AI.3 Pictures under Insufficient Light Condition .... - 92 -
AI.4 Analysis .... - 96 -
Appendix II .... - 101 -
Parameter Adjustment for Fast Corner Algorithm (REVIEW) .... - 101 -
AII.1 Experiment Result .... - 102 -
AII.2 Analysis .... - 108 -
Appendix III .... - 109 -
LED Recognition Result and Discussion .... - 109 -
AIII.1 Experimental Result of Feature Filter .... - 110 -
AIII.2 Analysis on the Feature Filters .... - 118 -
AIII.3 Experimental Result of LED Recognition .... - 119 -
AIII.4 Analysis on LED Recognition .... - 124 -
Appendix IV .... - 125 -
Projectile Motion .... - 125 -
AIV.1 Experiment Result and Analysis .... - 126 -
Appendix V .... - 132 -
Color Model .... - 132 -
AV.1 Introduction .... - 133 -
AV.2 RGB Color Model .... - 133 -
AV.3 HSV Color Model .... - 134 -
AV.4 Transformation between RGB and HSV .... - 134 -

Chapter 1

Introduction



This chapter briefly describes **Augmented Reality (AR)** and how existing mobile game achieve Augmented Reality. Our project objective and project equipment information can also be found here.

- **1.1 Background Information and Motivation**
- **1.2 Programming Capability Issue**
- **1.3 Project Objective**
- **1.4 Project Equipment**

1.1 Background Information and Motivation

Mobile phones with built-in digital cameras and music players have become very popular and common nowadays because of their portability and handiness. Because of its popularity, there are so many mobile games evolved both written in J2ME as well as proprietary Development Platform. Some mobile games are similar to the typical or traditional games which can be found in handheld gaming device, for instance, NDS, Game Boy, PSP, etc (Fig 1.1). While some other games employed the use of augmented reality in order to make the game more exciting, realistic and interesting (Fig 1.2).

Fig 1.1

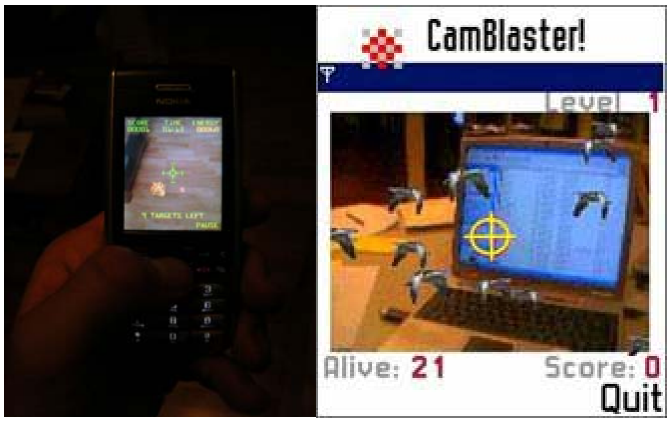


Fig 1.2

To achieve augmented reality in mobile game, the most popular and easily observable method would be the use of cameras resides in the mobile phone plus the use of computer generated graphics for dynamic environmental interaction. An example would be “Agent V” from Nokia 3230 mobile phone (Fig 1.3).

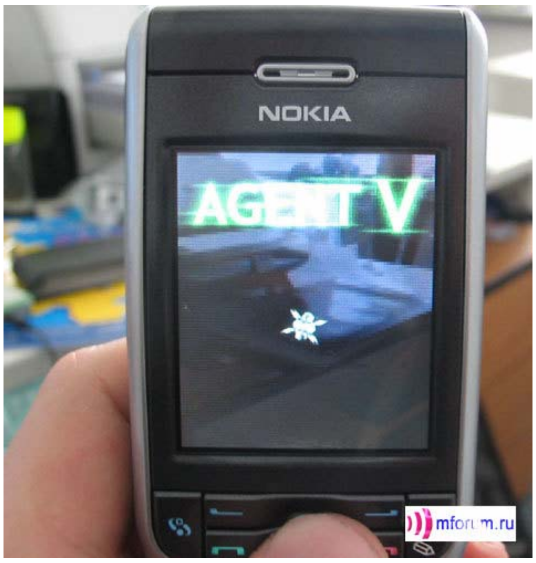


Fig 1.3

Another new and recent idea for achieving augmented reality in game would be using some motion or vibration sensors for movement in a game. Nokia 5500 "GrooveLab" demonstrated one of the uses of motion sensors for augmented reality. (Fig 1.4)

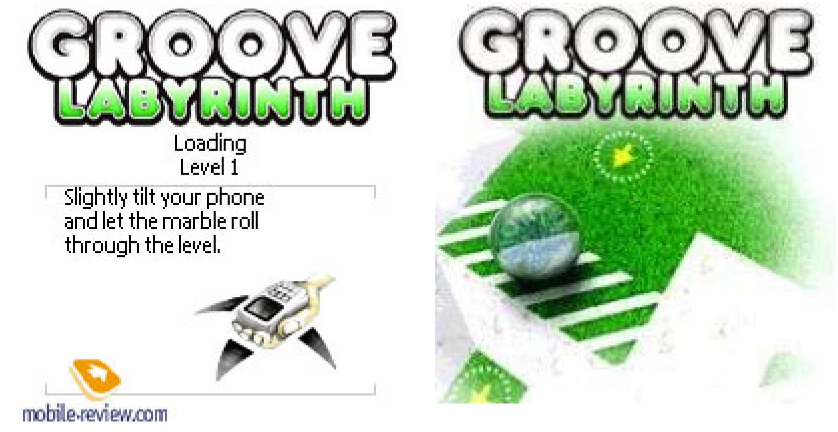

GROOVE LABYRINTH
Loading
Level 1
Slightly tilt your phone and let the marble roll through the level.
mobile-review.com
GROOVE LABYRINTH

Fig 1.4

Since our FYP focuses on the use of phone camera for augmented reality, there comes a question - Is it possible to add some more features to the existing games which make good use of phone camera as a mean for augmented reality? This is the motivation of our FYP project.

As the computation power as well as the image and video capture quality improve, real time video capture for motion tracking is no longer impossible. Many existing games utilized motion tracking as an additional and innovation for user input (mainly for direction movement). However, the existing games process no memory function to remember the associate of the external environment and the internal computer generated graphics. Our main objective of this Final Year Project is to demonstrate how games can process memory to “remember” its external environment for interaction base on existing motion tracking technique.

1.2 Programming Capability Issue

Few years age, users may feel panic to develop programs for mobile phones because they lacked supports from the vendors and at that moment, the processing power of a mobile phone was very limited. Now, things become better, there are panties of Development Platform for programmers to choose from, including, J2ME, Embedded C++ (For Windows Mobile Platform), Symbian C++ (For Symbian Platform), etc. Although programmers could write mobile phone program base on J2ME, it does not provide phone-specific API to access the camera. On the other hand, Symbian OS makes programming on camera-phone possible. Mobile phones which use Symbian as the Operating System (or so called Smart Phone) allow programmers to access most of the functions provided by the phones, including camera and image manipulation functions.

As Symbian OS is supported by a large number of vendors (e.g. Nokia, Sony Ericsson, Panasonic, Samsung, Siemens, etc) and provides an open platform for developers to work on, it is not difficult to imagine that Symbian OS would become one of the major Operating System for mobile phones in the foreseeable future. Our FYP project will base on Symbian OS as our target platform due to its programming capability.

1.3 Project Objective

The objective of our Final Year Project is not only to develop a game, but also to develop a way to demonstrate how a program can process memory to "remember" its external environment for interaction of Augmented Reality. To be more specific, we would like to show how a program can "remember" the Light Emitting Display (LED) in its external environment.

The game is a demonstration of our proposed methodology for Augmented Reality. In such a way, we do not limit ourselves to just implementing a game but have a higher level of abstraction. In addition, using such approach would not limit the possibility of our methodology to be applied to develop other mobile applications besides mobile games.

1.4 Project Equipment

Our project involves two Symbian mobile phones, Nokia N90 (Symbian OS v8.1a, Series 60 2nd Edition, Feature Pack 3) as well as Nokia N80 (Symbian OS v9.1, Series 60 3rd Edition). As the emulator provided by Nokia SDK does not simulate camera function, we would like to use the mobile phones as our testing and development platform directly for our project. The Nokia N90 was used as our prototype and algorithm testing platform, while Nokia N80 was the real platform for us to implement the desired program.

Chapter 2

Symbian Operating System Overview

This chapter would briefly describe the basic architecture of the Symbian Operating System. This chapter also outlines the differences between the Series 60 2nd Edition and 3rd Edition of the Symbian Operating System.

- **2.1 Basic Architecture of Symbian Operating System**
- **2.2 Development Environment**
- **2.3 Limitation of Symbian Phone**
- **2.4 Features Different in Symbian OS Series 60 2nd and 3rd Edition**
- **2.5 Porting Existing Program to Series 60 3rd Edition**
- **2.6 Why Symbian**
- **2.7 Conclusion**

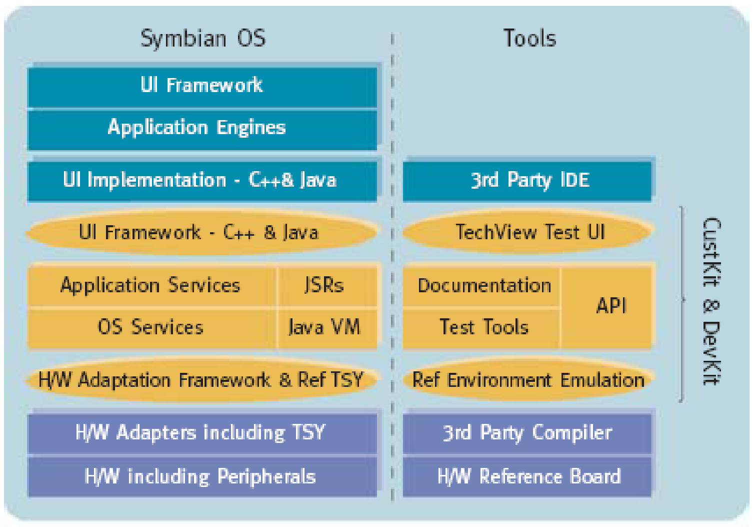
2.1 Basic Architecture of Symbian Operating System

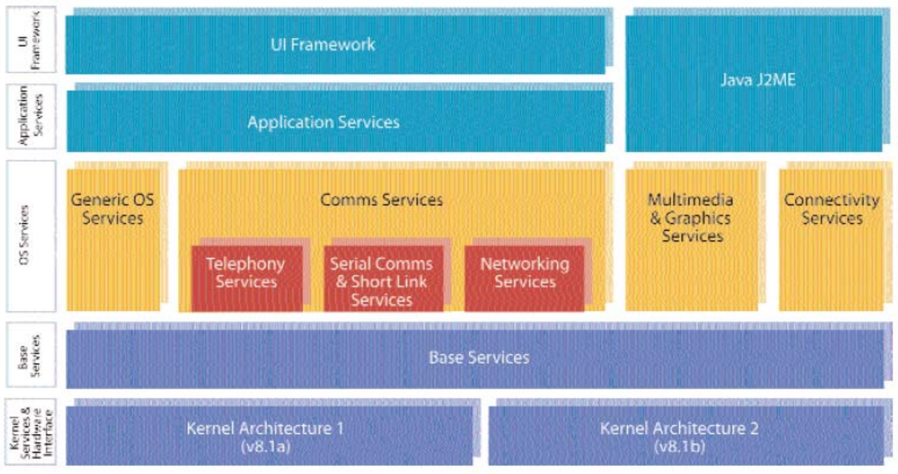
By the end of March 2005, shipments of Symbian OS phones exceeded an average of two million per month, and cumulative shipments since Symbian's formation reached 32 million phones. Also at that time, there were more than 4500 commercially available, third-party applications for Symbian OS phones. Year on year, phone shipments have been virtually doubling – and that trend appears likely to continue, or even increase, for the foreseeable future. (Adopted from Developing Software for Symbian OS – An Introduction to Creating Smartphone Applications in C++)

These figures suggest that Symbian OS is approaching maturity as the preferred Operating System for high and mid-range mobile phones, and that it offers an ideal platform to developers, on which they can create new and imaginative applications.

The architecture of Symbian OS v8.1 and v9.1 are described in the following diagrams respectively.

Symbian OS v9.1 System model





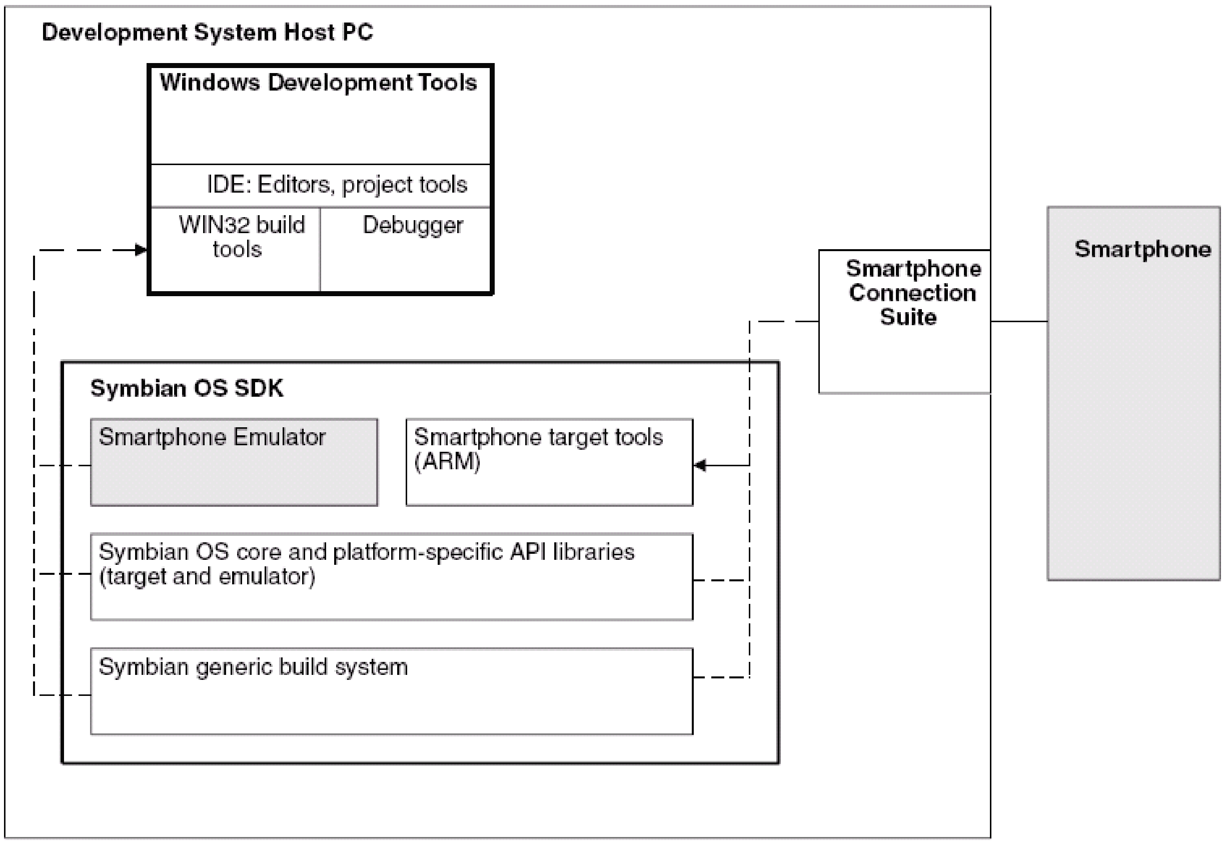
Symbian OS v8.1 System model

The main focus of this chapter is to illustrate how Symbian OS provides support on image processing in the phone and how the capability change across different Series 60 Platform.

Symbian use its own implementation of the C++ language called Symbian C++, optimized for handheld devices with limited memory. Programmers access the recourses, for instance, files, music players, camera, etc via the APIs provided by Nokia SDK.

2.2 Development Environment

Generally, our development environment is under Microsoft Visual Studio .Net 2003 with Nokia Symbian S60 SDK. The development tools can be formulated like this:

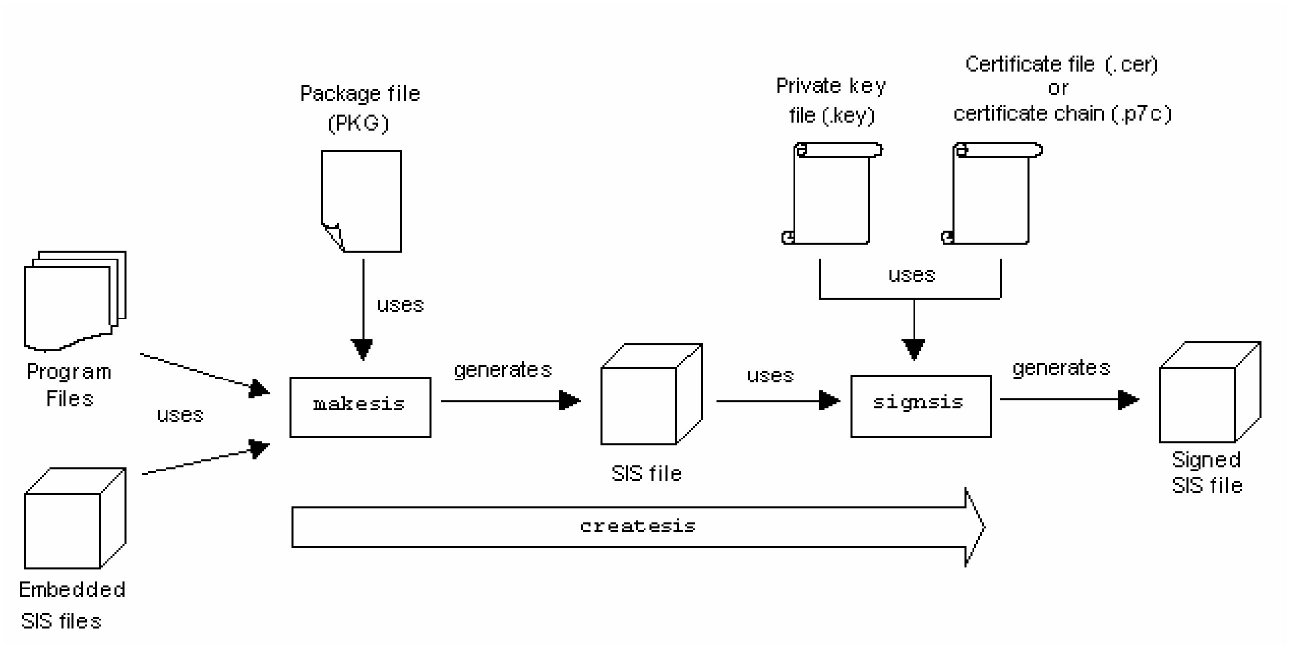

Development System Host PC
Windows Development Tools
IDE: Editors, project tools
WIN32 build tools
Debugger
Smartphone Connection Suite
Smartphone
Symbian OS SDK
Smartphone Emulator
Smartphone target tools (ARM)
Symbian OS core and platform-specific API libraries (target and emulator)
Symbian generic build system

There are emulators provided along with the Nokia SDK which is a Windows application that simulate a Smartphone entirely in software – complete with simulated buttons and display. This allows developers to run and debug Symbian OS software on the PC as opposed to running on a real device. However, there is a major drawback for the emulators. The fact is that emulators for Series 60 2nd Edition Feature Pack 3 as well as Series 60 3rd Edition do not provide camera simulation function. This explains why our project did not use much of the emulators. In other words, the emulators provide little assistant for us. We decided to develop program directly in the Symbian phone.

To install a program on a Symbian-based phone, it must be compiled into the Symbian OS installation (.sis) file format. Developers write package control (.pkg) files that define the files to be put in the SIS installation file. (Adopted from Symbian Developer Library)

From Symbian OS v9.1 (Operating System for Nokia N80), it requires that SIS files are authenticated when they are installed, so that malicious code cannot be installed to the phone. This means that the SIS file must be digitally signed. This action accounts for one of the reasons why Symbian OS v9.1 is not binary compatible with those previous versions.

The diagram below shows the key files and tools used in the process of creating a SIS file for Symbian OS v9.1.



The makesis tool uses the package file to create an unsigned SIS file. The signsis tool can then be used to sign the SIS file with a certificate to create a signed SIS file that can be installed. The createsis tool is a wrapper around these two tools, which allows the whole process to be done in one step. If the program is being signed by the developer, rather than being signed through Symbian Signed, then createsis can also create the certificate to use. (Adopted from Symbian Developer Library)

2.3 Limitation of Symbian Phone

Since we are programming on handheld devices which has limited resources (limited amount of memory and limited amount of CPU speed), these make programming on the Symbian phone a difficult task.

	Nokia N80 Specification	Nokia N90 Specification
Operating System	Symbian v9.1 (Nokia Series 60 3rd Edition)	Symbian v8.1a (Nokia Series 60 2nd Edition, Feature Pack 3)
CPU Speed	220MHz	220MHz
Memory	Internal Memory: 40MB External Memory: miniSD (up to 2GB)	Internal Memory: 31MB External Memory: RS-MMC (up to 1GB)
Display Size	352 x 416 pixels (256K Colors)	352 x 416 pixels (256K Colors)

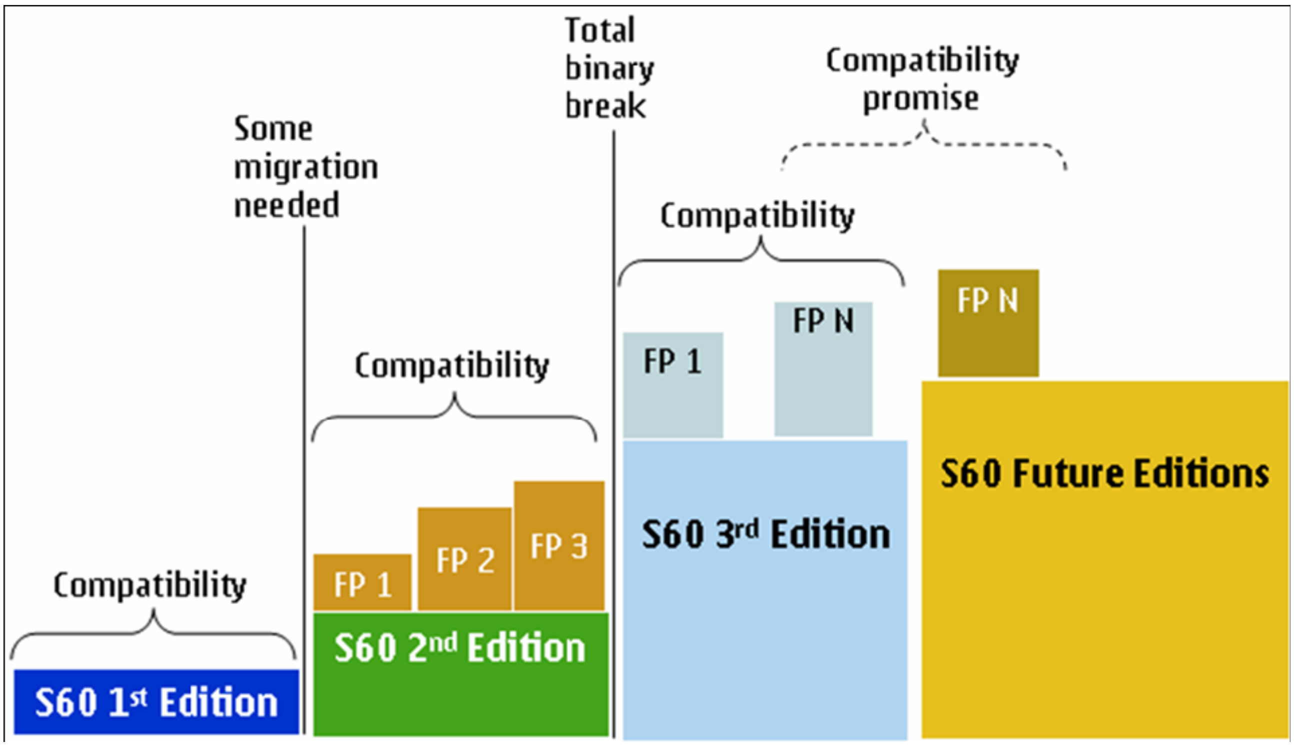
Since computation performance is an important factor in making a realistic augmented reality game. If the program takes too long time for the calculation of motion tracking, the frame rate will fall off. When ever possible, we would use the following operation to enhance the performance:

1. Bit shifting operations (<< and >>)
2. Integer add, subtract, and Boolean operations (&, | and ^)
3. Integer multiplication (*)

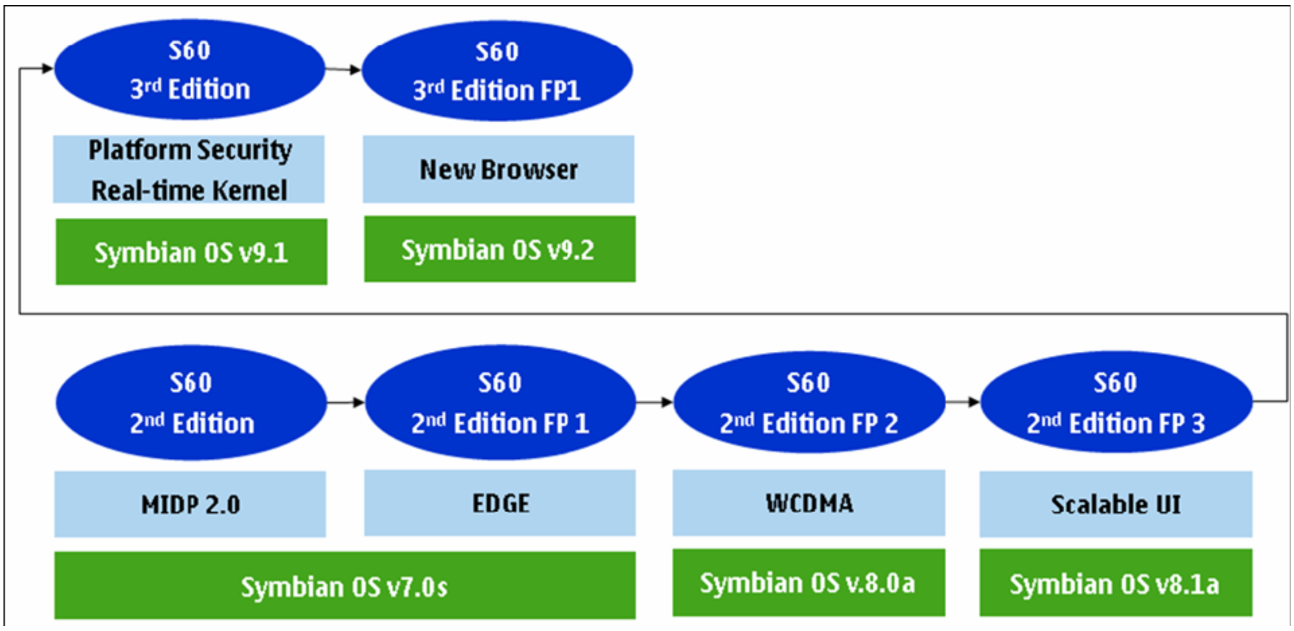
We would reduce the use of floating point computation as much as possible. It is because the current mobile phones do not equipped with a floating point arithmetic unit. As a result, floating point operation would slow the whole process down.

2.4 Features Different in Symbian OS Series 60 2nd and 3rd Edition

S60 3rd Edition is based on a new version of Symbian OS (v9.1). As motioned before, this new platform edition introduces a full binary break between S60 2nd and 3rd Editions, which means that applications need to be compiled using the new tools provided in the S60 3rd Edition in order to run on S60 devices based on the new platform edition. S60 3rd Edition also improves application security and confidentiality of user data by introducing platform security and different application capability levels. Below shows a diagram indicating the break point of the S60 2nd and 3rd Edition:



The following diagram shows the Nokia platform development:



Since Nokia Series 60 3rd Edition (Symbian OS v9.1) brought a large impact to the development of mobile application, here are some improvements made by S60 3rd Edition:

(Adopted from S60 Platform: Source and Binary Compatibility v1.6 by Nokia)

Most deprecated APIs will be removed
New S60 3rd Edition compilation tools cause a full binary break between S60 2nd and 3rd Editions. Because backward compatibility with S60 2nd Edition cannot be maintained anymore, most deprecated APIs will also be removed from S60 3rd Edition, while a number of new replacement APIs will be introduced. Most deprecated APIs will be removed from S60 3rd Edition.

Compatibility mode removed
In S60 3rd Edition the compatibility mode for legacy applications is not supported anymore. All applications are expected to be scalable (hard-coding to a specific screen resolution cannot be done anymore).

Platform security and new application architecture
The biggest change in Symbian OS v9.1 and in S60 3rd Edition is the platform security concept. Its main building blocks are Capabilities (set of privileges to applications), Data Caging (secure storage of data), Secure Interprocess Communication (IPC), and memory management. Platform security also requires a number of changes to the application architecture. The Server application concept is introduced to enable former embedded and embedder applications to run in different processes.
Data caging and introduction of the Server application require changes to Document Handler: Instead of file names, file handles are passed. Recognizers, notifiers, and converter plug-ins are implemented as ECOM plug-ins.
The installer has been completely rewritten to perform the additional checks (capabilities and certificates) that the platform security mandates. The installation file format has been changed from SIS to SISX (note, however, that the actual file extension is still `.sis`).

New compiler and tool chain - full binary break
S60 3rd Edition introduces new compilation tools (RVCT, GCC EABI), which causes a full binary break. Therefore all other compatibility issues caused by S60 3rd Edition are listed under Section SC breaks caused

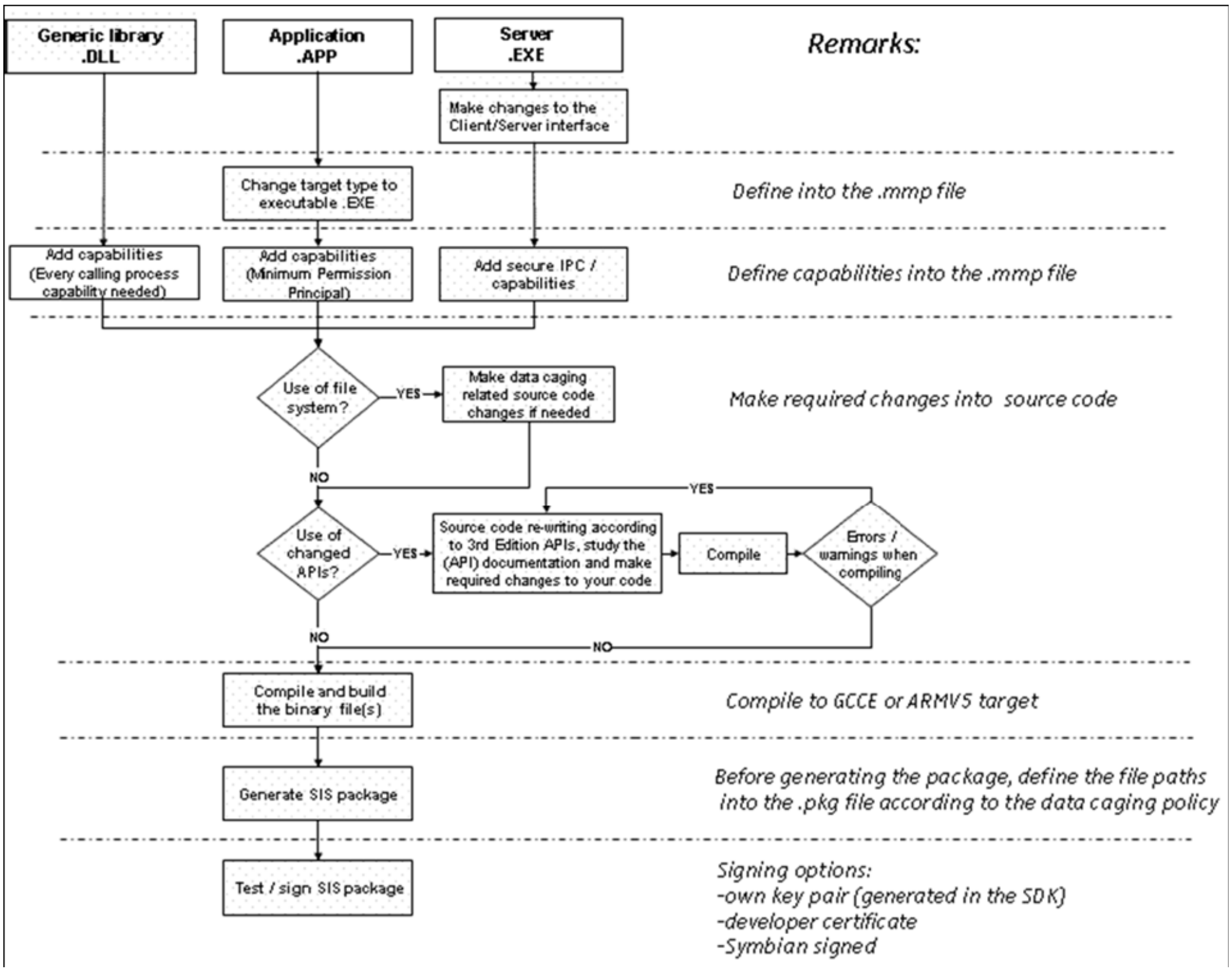
by S60 3rd Edition.

Nokia Camera API
The wrapper for the Nokia Camera API (Camera Server) is removed. The Symbian Onboard Camera API (ECam) replaces this deprecated API.

Real-time kernel (EKA2)
EKA2 is the only kernel version supported by Symbian from Symbian OS v9.1 onwards. The compatibility impacts of EKA2 are mainly focused on the need to rewrite device drivers, but otherwise a very limited amount of source code breaks.

2.5 Porting Existing Program to Series 60 3rd Edition

Since our target platform involve both Series 60 2nd Edition (Nokia N90 with Feature Pack 3) and Series 60 3rd Edition (Nokia N80). From time to time, we have to perform lots of code conversion between these two platforms for testing and debugging purpose. Here shows a high level description on how porting an existing program or library to S60 3rd Edition Platform can be achieved:


Generic library .DLL
Application .APP
Server .EXE
Remarks:
Make changes to the Client/Server interface
Change target type to executable .EXE
Define into the .mmp file
Add capabilities (Every calling process capability needed)
Add capabilities (Minimum Permission Principal)
Add secure IPC / capabilities
Define capabilities into the .mmp file
Use of file system?
YES
Make data caging related source code changes if needed
Make required changes into source code
NO
YES
Use of changed APIs?
YES
Source code re-writing according to 3rd Edition APIs, study the (API) documentation and make required changes to your code
Compile
Errors / warnings when compiling
NO
NO
Compile and build the binary file(s)
Compile to GCCE or ARMV5 target
Generate SIS package
Before generating the package, define the file paths into the .pkg file according to the data caging policy
Test / sign SIS package
Signing options:
-own key pair (generated in the SDK)
-developer certificate
-Symbian signed

2.6 Why Symbian

As mentioned previously, programmers can choose J2ME as their development platform besides Symbian. J2ME is cross-platform because there is a virtual machine to interpret the byte code of the program. However, J2ME does not provide any API for accessing the device camera, which makes our objective impossible to achieve. In addition to the API problem, J2ME does have one more drawback. Though it is cross platform, it uses Virtual Machine to interpret the code, and execution in interpretation mode is much slower than that execution of pre-compiled code.

This explains why our project favors the use of Symbian C++ instead of J2ME because of camera utilization as well as speed performance.

2.7 Conclusion

This chapter introduced the basic features of Symbian OS as well as the differences between 2nd and 3rd Edition Platform. This chapter also raises an importance consideration in our project design, that is, speed or computational time.

Chapter 3

mVOTE Engine

This chapter briefly describes functions of mVOTE engine which is created by our former FYP team. The mVOTE engine provides the function of motion tracking and feature selection.

- **4.1 What is mVOTE**
- **4.2 Block Matching Algorithm**

3.1 What is mVOTE?

The **M**obile **V**ideo **O**bject **T**racking **E**ngine (mVOTE^{TM}) is a SDK for developer to create mobile application using the mobile phone on-board camera as input device. By tracking the video object in the picture capture by the camera, mVOTE^{TM} can convert the corresponding movement of the camera into translational movement and degree of rotation. The mVOTE^{TM} enable the developer to create new digital entertainment experience for the mobile user.

(Adopted from mVOTE homepage, http://www.viewtech.org/html/mvote_tm_.html)

The Functions provided mVOTE engine:

1. Translational Motion Tracking
2. Rotational Motion Tracking
3. Feature Selection

3.2 Block Matching Algorithm

Block Matching Algorithm is the core of the mVOTE engine. The block matching algorithm is a kind of motion tracking algorithm, the basic idea of block matching algorithm is to divide the search window into blocks with equal size. For each block, the algorithm tries to find out which block in the search window would have a highest similarity to the feature block that we want to match.

How to measure the similarity between two blocks

The common way of measuring the similarity is to calculate the intensity different between two images. The two common ways to do calculation are:

1. Sum Absolute Difference (SAD):

   SAD is the summation of the absolute intensity difference between two N by N images X and Y. The equation is the following:

$$SAD(x,y)=\sum_{i=1}^{N}\sum_{j=1}^{N}|X(i,j)-Y(i,j)|$$

2. Sum Square Difference (SSD):

   SSD is the summation of the square of the intensity difference two N by N images X and Y. The equation is the following:

$$SSD(x,y)=\sum_{i=1}^{N}\sum_{j=1}^{N}\left[X(i,j)-Y(i,j)\right]^2$$

   Our ancestor chose SSD instead of SAD because it can enhance the performance of other part of the block matching algorithm.

How to find out the most similar block in search window

There are three major classes of method to find out the most similar block in search window.

1. The Exhaustive Search Algorithm (ESA)

   It is the brute force algorithm, it search all possible values and find the minimum value. It is the slowest one but it is one of the most accurate methods in finding the minimum value in the search window.

2. Fast Search Algorithm

   It is based on the assumption that the matching error will increase monotonically when it is moving away from the correct matching block. So it only tests on a subset of all possible value in the search window. It has a very critical disadvantage that it will be trapped by the local minimum not the global one. So we won't choose it as searching algorithm.

3. Fast Exhaustive Search Algorithm

   The Fast Exhaustive Search Algorithm is trying to reduce the number of value need to calculate by some simple test method. There are three methods proposed by our ascender:

   i. The Successive Elimination Algorithm (SEA)

      Apply the Minkowski inequality to eliminate the invalid block

   ii. PPNM (Progressive Partial Norm Matching)

      Apply the Minkowski inequality to eliminate the invalid block before calculate the Cost Function.

   iii. Partial Distortion Elimination (PDE)

      The idea of PDE Algorithm is shorten the time to calculating the SSD. If the Partial SSD (PSSD) is greater than the current minimum value of SSD, the remaining part of calculation of SSD on that block is useless and can be stopped. There is the definition of kth PSSD:

$$PSSD = \sum_{i=1}^{k} \sum_{j=1}^{N} \left[X(i, j) - Y(i, j)\right]^2 \quad \text{Where } k = 1, 2, 3, \ldots, N$$

There are other methods use in the MVOTE engine, such as Adaptive Search and Spiral Scan, but we haven't used them in other part of our project. So we don't mention them in here.

Chapter 4

Program Design and Implementation

This chapter would briefly describe the design and implementation of our program

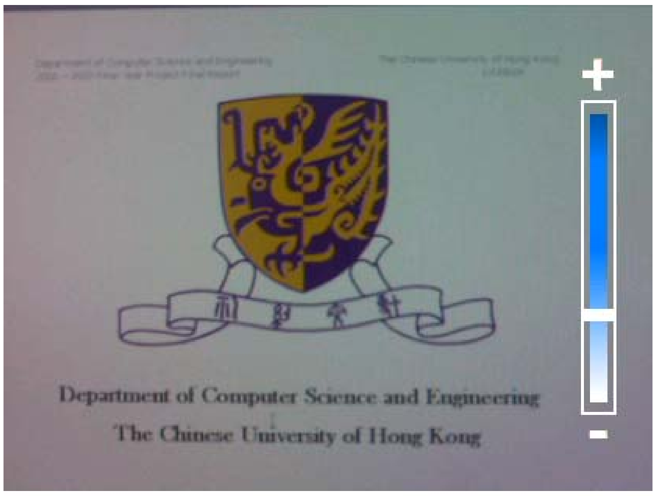
- **4.1 User Interface**
- **4.2 Program Assumption**
- **4.3 Projectile Motion Calculation**
- **4.4 Program Design**
- **4.5 Programming Trick**
- **4.6 Conceptual Program Prototype**
- **4.7 Feature Recognition in Semester 1 VS LED Recognition in Semester 2**

4.1 User Interface

This project was mainly tested and debugged on Symbian phone namely Nokia N80 and N90 mobile phones. We develop the game with the help of existing Motion Tracking (mVOTE) Engine (**with some modification**) which is developed by our ancestor (LYU0404).

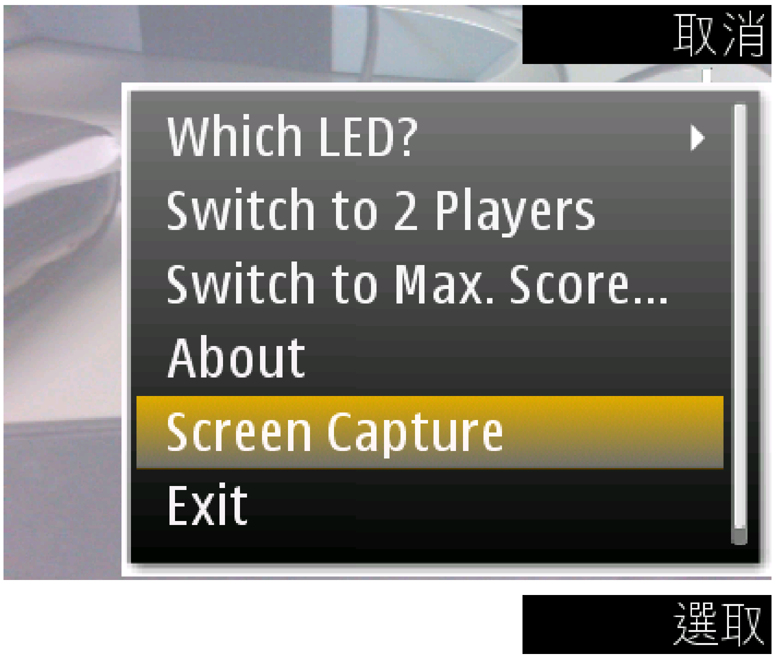
The program makes use of the suggested Symbian OS application framework comprising the Application, Application Document, Application UI and Application View (or Application Container) classes. In addition, we added a game engine, motion tracking engine, projectile calculation modules.

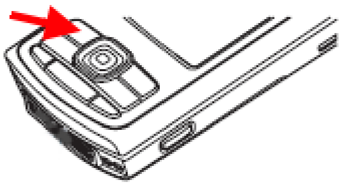
After starting the program up, user may see the below Graphical User Interface:


+
-
Department of Computer Science and Engineering
The Chinese University of Hong Kong

There is an indicating bar on the right hand side of the screen. The bar is used to show the digital zooming factor. Users may move up or down of the joystick in order to control the degree of digital zooming. If the marker is located on the bottom of the bar, then there would not be any digital zooming. The zoom function helps to convince the users to locate the LED.

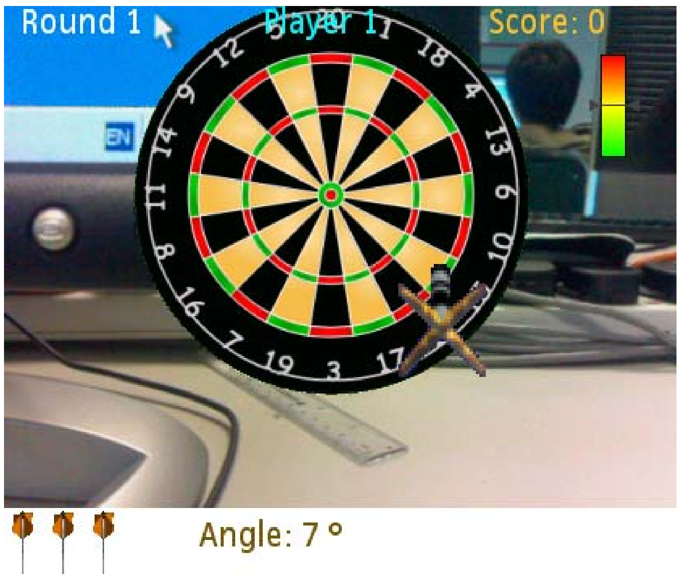
User may press the option button for option selection:



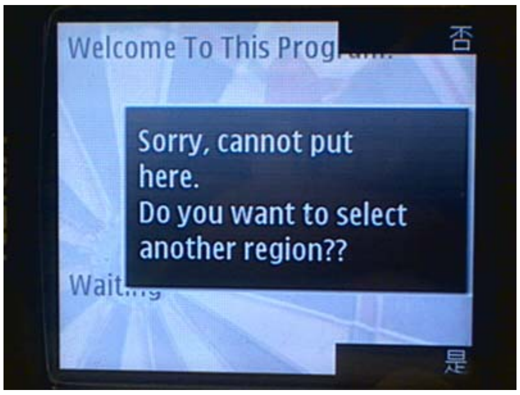


Users can choose to use either “Red” or “Green” LED by alter the setting under the “Which LED?” menu. In the default mode, the game is set as single player mode. Users may switch to 2 players mode by changing the setting provided in the main menu. The 2 players mode do not support blue tooth connections. This game also provides 2 score modes (Useful only when the game is in 2 players mode). One is “Min. Score Mode” and another one is “Max. Score Mode”. In “Min. Score Mode”, the users who score less would win the game. “Max. Score Mode” is defined similarly.

After adjusting the zooming factor, users may press the “left arrow” or “right arrow” key in the joystick to start playing the game. A successful recognition of the green LED would direct the user to screen similar to the photo on the right.



However, if the environment is too noisy or the program cannot locate the LED, then uers may observe the following screen.



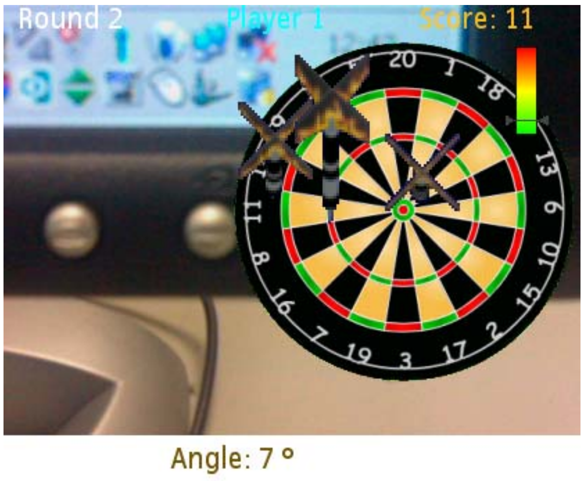
Sorry, cannot put here.
Do you want to select another region??

Each user has 2 round to play the game, and there would be 3 chances for users to throw the dart. The score information would be displayed on the upper right corner of the screen. The user may change to throwing angle by pressing “up arrow” and “down arrow” key in the joystick. The default value of the vertical angle would be 7^o. However, users are allowed to change any value between 0^o and 15^o.

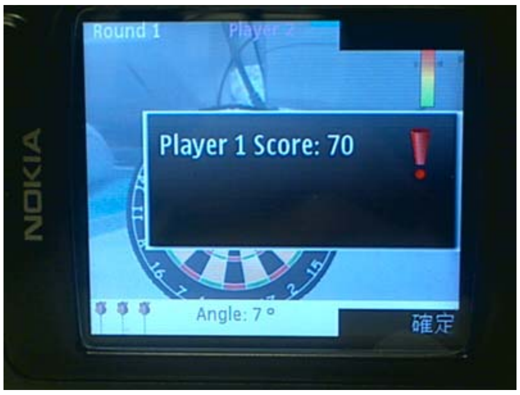
In addition, there is a gradient red to green indicating bar locate on the right hand side of the screen. This indicating bar acted as the energy level of the dart throwing. If the marker is located in the most upper region, the dart would be thrown at its maximum velocity. Similarly, if the marker is located at the bottom of the indicating bar, then the dart would be thrown at its minimum velocity. The marker would move back and fore with time.

Users can move the mobile phone (i.e. the camera) to locate the best position that suits them most.

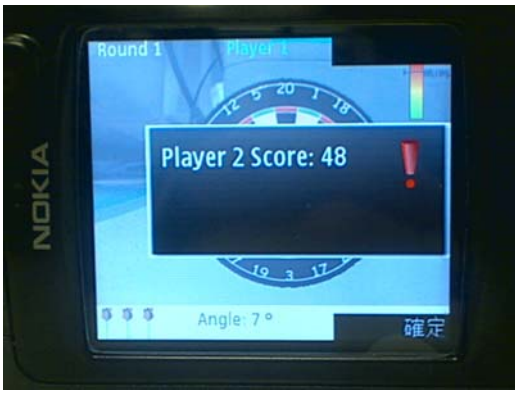
To throw a dart, simply press the “enter” key in the joystick of the phone. There would be a dart throwing animation shown on the screen.



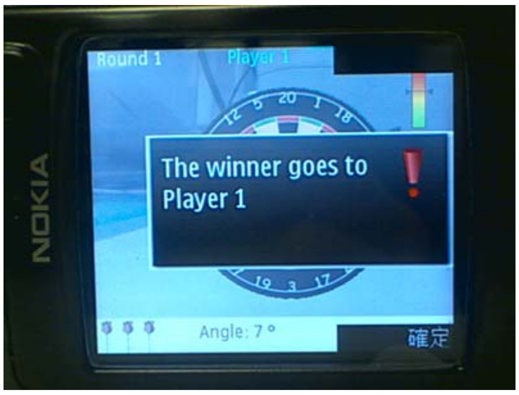
After the player threw all the darts, the score would be displayed to the users.


Player 1 Score: 70

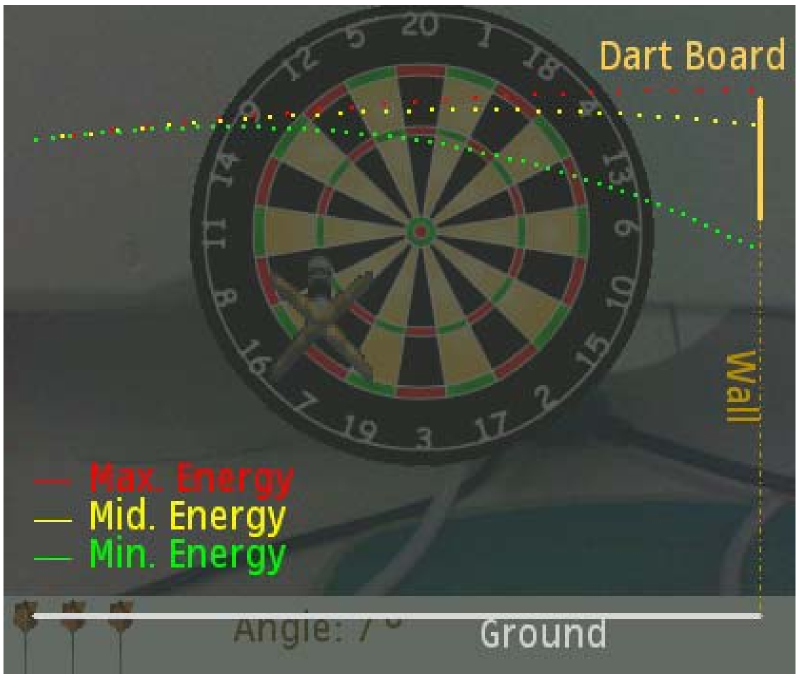
If the game is in “2 Players” mode, the turn would be go to player 2 after player 1 confirmed his/her score. Assume Player 2 also finished his/her turn. Another score dialog for player 2 would be displayed.


Player 2 Score: 48

Finally, the result would also be displayed. (Suppose the game is in “Max. Score Mode”

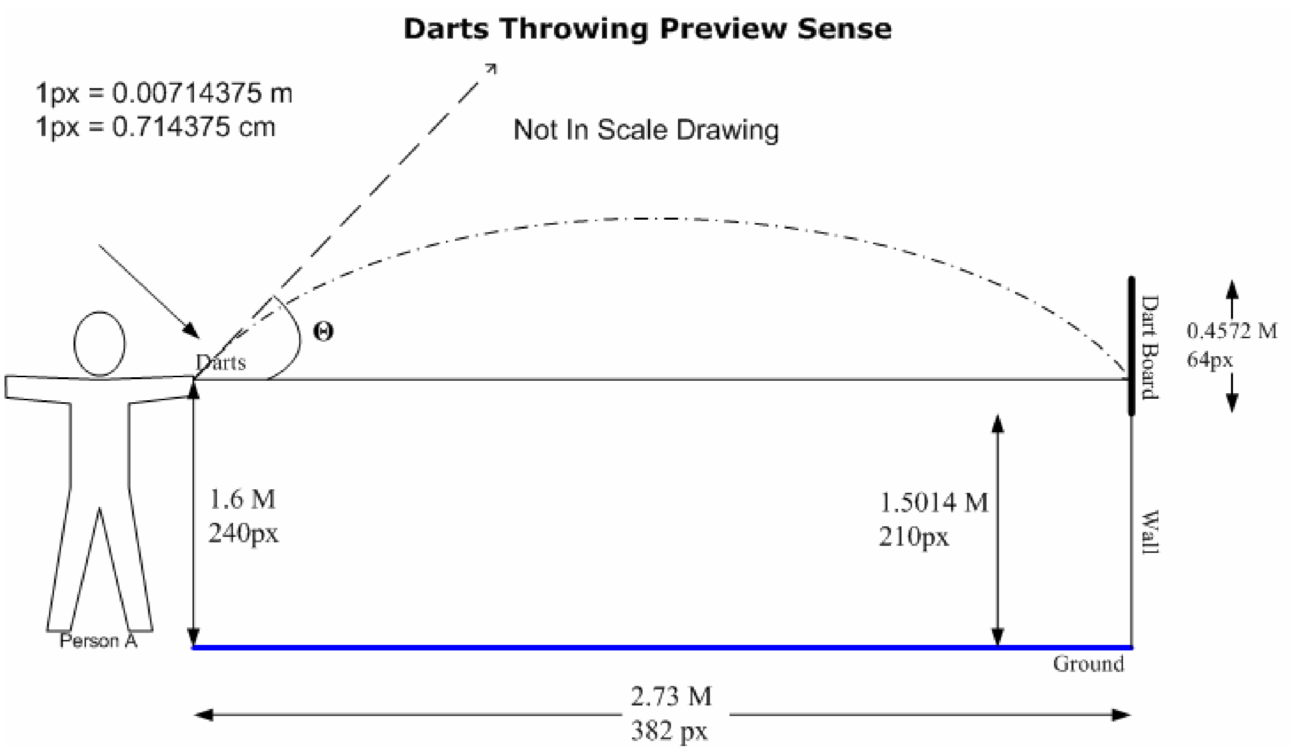

The winner goes to Player 1

Users can press “5” when playing the game (But not at the time when the dart throwing animation is in progress). The following screen would be displayed.



This screen shows the assumed dart throwing environment. Assume the dart board is located on the top right hand side of the screen, the projectile shows the hitting position at different energy level. (The assumptions can be found later in this chapter)

4.2 Program Assumption



Wikipedia stated the following:

"A regulation (Dart) board is 18 inches (45.72 cm) in diameter, and is divided into 20 sections. Each section is lined with thin metal wire. The numbers indicating the various scoring sections of the board are normally made of wire, especially on..."

Therefore, we chose our dart board to be 0.4572 Meter in diameter for the game. Official dart rules call for the center of the bullseye to be placed 5 feet 8 inches (173 cm) from the ground, and 7 feet 9 1/2 inches (237 cm) from the player. This is not the distance measured from the wall, but from the face of the dart board.

Clearly, according to these regulations, we are assuming our player is located at 2.73 meter in front of the dart board. Also, the dart board center is located above the ground by 1.73 meter. We further assume the distance between the dart which is held by the player's hand and the ground is about 1.6 meter.

The angle theta indicates the vertical angle between the horizontal line and the dart throwing direction.

However, how can we model the above data for our game? To answer the question, we use the following definition:

64 pixels = 45.72 cm ➔

64 pixels = 0.4572 m ➔

1 pixel = 0.0071435 m

With this definition, we can easily calculate the distance between the dart board center and the ground, distance between the player and the dart board. The calculated values were list on the above figure.

4.3 Projectile Motion Calculation

The core of our game makes use of projectile motion. We assumed there would be air resistance in our model. To begin with, let's take a look at simple projectile motion without any air resistance first.

Simple Projectile Motion with No Air Resistance

The basic problem is projectile motion, such as a cannon shell, neglecting air resistance. For planar motion with two dimensions, vertical and horizontal, let the horizontal be denoted by **x** and the vertical by **y**. If we first ignore air resistance, the equations of motion just follow Newton's second law. **(Eq. 1)**

$$\frac{d^2x(t)}{dt^2} = 0 \text{, and}$$

$$\frac{d^2y(t)}{dt^2} = -g$$

where g is the acceleration due to gravity. Rather than approximate the second derivative, which requires us to evaluate three consecutive points, let us resort to coupled first order equations. Since the first derivative of position is velocity and the first derivative of velocity is acceleration, we may write Equation 1 as following: **(Eq. 2)**

$$\frac{dx}{dt} = V_x \qquad \frac{dV_x}{dt} = 0$$

$$\frac{dy}{dt} = V_y \qquad \frac{dV_y}{dt} = -g$$

where v_x and v_y are the horizontal and vertical components of the velocity, respectively. The recursion relations for these four equations are as follows **(Eq. 3)**

$$x_{i+1} = x_i + V_{x,i}\Delta t \qquad V_{x,i+1} = V_{x,i}$$

$$y_{i+1} = y_i + V_{y,i}\Delta t \qquad V_{y,i+1} = V_{y,i} - g\Delta t$$

where we have approximated the first derivative with a divided difference and solved algebraically for the quantity after a time step Δt. Now given initial values for x, y, v_x, and v_y or equivalently the initial velocity and launch angle, we can use Equation 3 to calculate values of position and velocity at later times. By choosing Δt sufficiently small, we should be able to calculate the motion such that the truncated terms are of negligible size in comparison to first order terms.

Simple Projectile Motion with Air Resistance

In order to include the effects of air resistance, we introduce a "drag force", B_2, which is always opposed to the velocity vector. Adding this term to Equation 3, we obtain the equations of motion in component form **(Eq. 4)**

$$x_{i+1} = x_i + V_{x,i}\Delta t \qquad V_{x,i+1} = V_{x,i} - \left(\frac{B_2 \cdot V_i \cdot V_{x,i}}{m}\right)\Delta t$$

$$y_{i+1} = y_i + V_{y,i}\Delta t \qquad V_{y,i+1} = V_{y,i} - g\Delta t - \left(\frac{B_2 \cdot V_i \cdot V_{y,i}}{m}\right)\Delta t$$

where v_i is the magnitude of the velocity and not a component

$$V_i = \sqrt{(V_{x,i})^2 + (V_{y,i})^2}$$

The coefficient B_2 can be estimated by experimental means, such as a wind tunnel. However, we can also arrive at an approximation by making the simple estimate that the work done by the drag force should be proportional to the mass of air moved by the cross-sectional area of the projectile, which in turn must be proportional to the density of air, ρ. However, the calculation is rather complex and not suitable for our mobile phone platform. As a result, we simply set the value of B_2 as 0.003.

The initial velocities of **x** and **y** are:

$V_{x,0}$ = Initial Velocity * cos(θ) $V_{y,0}$ = Initial Velocity * sin(θ)

Where the initial velocity is determined by the energy indicating bar of the game, θ is the vertical angle of the dart which is shown in the figure in section 4.2.

In addition, we assume that the weight of a dart is about 20 gram. We further restrict the program to have $8ms^{-2}$ minimum initial velocity and $18ms^{-2}$ as the maximum initial velocity. Users are allowed to adjust the vertical dart angle from 0^o to 15^o degree in upward direction. For further details, please refer to Appendix IV.

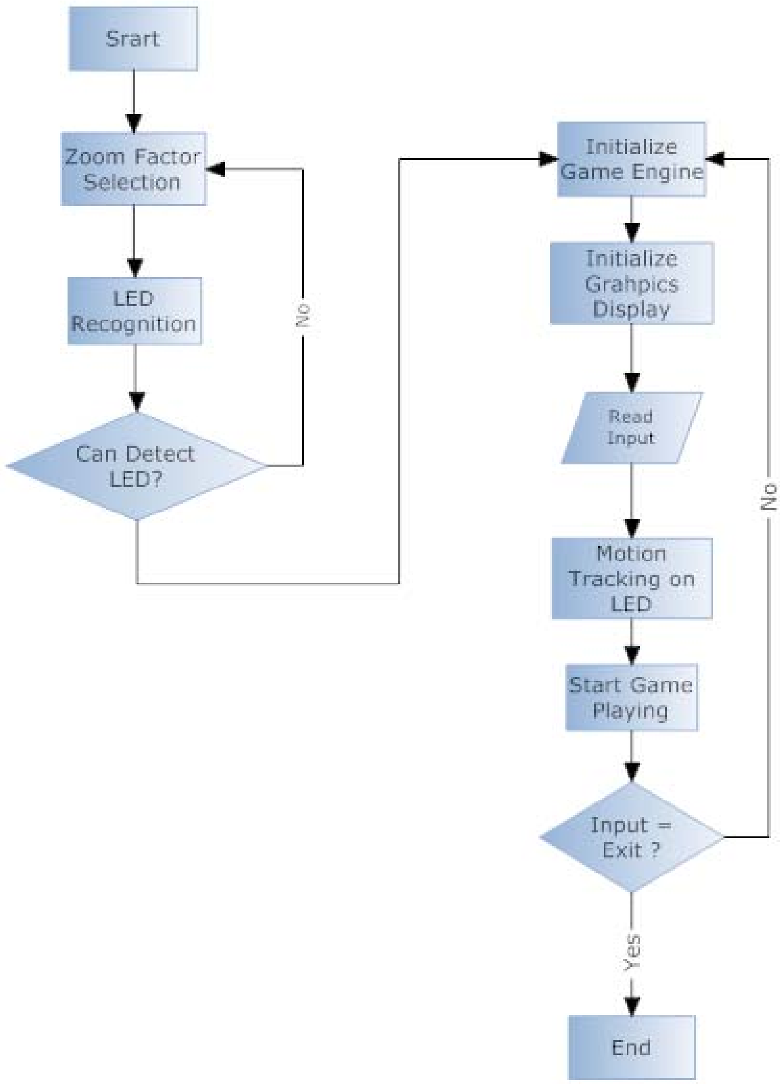
4.4 Program Design

The program consists of these core files:

File	Description
AR3rdapplication.h AR3rdapplication.cpp	An Application that creates a new blank document and defines the application UID.
AR3rddcoument.h AR3rddocument.cpp	A Document object that represents the data model and is used to construct the App Ui.
AR3rdappui.h AR3rdappui.cpp	This class defines the User Interface as well as ways to handle input key commands and commands from menu options.
AR3rdAppContainer.h AR3rdAppContainer.cpp	A container (or an Application View) object which displays data on the screen.
MVOTE.h MVOTE.cpp	Motion Tacking Engine
GameEng.h GameEng.cpp	The game engine. It handle the flow of the game.
Projectile.h Projectile.cpp	It handles the projectile calculation. It simulates the projectile for the real world
Score.h Score.cpp	It handles the calculation of the game score.
Animator.h Animator.cpp	It handles the animation of the dart throw.
AR3rd.rss	This describes the menus and string resources of the application.
AR3rd.mmp	It specifies the properties of a project in a platform and compiler independent way.

Program Flow

The following flow chart shows the flow of the game:



At the very beginning, users have to selection appropriate zoom factor before he or she can actually play the name. The zoom factor selection function is to help the users to locate the LED. The LED may be located far from the users. As a result, have a digital zoom function would convince them.

Assume the users focus on the appropriate LED, the users may then play the game by pressing corresponding key on the mobile phone.

After that, our program would perform an image processing to recognize any LED in the scene. If the program can detect any LED in the scene, then I would begin to execute codes for game playing part; otherwise, the program would alert the users that LED was not found and ask the users to selection some other better environment.

Once the program can recognize the LED, the program would performance initialization on the game engine as well as graphics display modules. The program would also track on the location of the LED by using motion tracking provided by mVOTE engine. It accepts input from the users for exiting the game.

In fact, in the start game playing module, it also accepts input from users to determine when the dart should be fired to throw. I also handle the animation as well as communicate constantly to the game engine module. This module also provides a dart throw preview function.

The flow of the game is similar to the description appeared in section 4.1

}

4.5 Programming Trick

Since our target platform is Symbian Operating System (to be more specific, our target is Nokia N80), we have to optimize our source code a bit in order to have a more efficient program. Here, we would like to demonstrate some examples for allowing efficient execution.

In our projectile motional calculating module, we have to make use of sin and cos functions. The sin and cos functions are needed because of the calculation of initial x and velocity respectively. However, these functions may involve quite a number of internal computations which may slow our program down. As a result, we decided to write our own sin and cos functions, which hard code the values. Since the possible allowed vertical angle is between 0 and 15. Therefore, we may implement the function in the following way.

```
float   Sin(int aAngle)
{
    switch(aAngle)
    {
        case(0):       return 0.0;                break;
        case(1):       return 0.01745240644;   break;
        case(2):       return 0.03489949670;   break;
        case(3):       return 0.05233595624;   break;
        ...
        case(12):      return 0.20791169082;   break;
        case(13):      return 0.22495105434;   break;
        case(14):      return 0.24192189560;   break;
        case(15):      return 0.25881904510;   break;
        default:        return 0.0;              break;
    }
}
```

The cos function can be implemented in a similar way.

```
float   Cos(int aAngle)
{
    switch(aAngle)
    {
        case(0):        return 1.0;             break;
        case(1):        return 0.99984769516;   break;
        case(2):        return 0.99939082702;   break;
        case(3):        return 0.99862953475;   break;
        ...
        case(12):       return 0.97814760073;   break;
        case(13):       return 0.97437006479;   break;
        case(14):       return 0.97029572628;   break;
        case(15):       return 0.96592582629;   break;
        default:        return 1.0;             break;

    }
}
```

By implementing the above functions, the program simply knows the sin or cos value of a given angle (in degree) without the need of calculation. Thus, the computational power is saved.

On the other hand, in the process of computing the projectile motion, the program needs to perform square root function from time to time. And square root function is much more complex and expensive than the sin and cos functions. Clearly, we have to find some other ways to approximate its value. Luckily, we found a very magic function – reciprocal square root.

The function can be used to compute the reciprocal of the square root, i.e. $x^{\frac{-1}{2}}$, was written by Greg Walsh, and implemented into the game Quake III by Gary Tarolli. As an approximation, it produced a relative error of less than 4%.

The first thing that pops out is the use of the number **0x5f3759df** in this function, which calculates the inverse square root of a float and why does this function actually work?

```
float Q_rsqrt( float number )
{
  long i;
  float x2, y;
  const float threehalfs = 1.5F;
  x2 = number * 0.5F;
  y  = number;
  i  = * ( long * ) &y;  // evil floating point bit level hacking
  i  = 0x5f3759df - ( i >> 1 ); // Would it work?
  y  = * ( float * ) &i;
  y  = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
  // y  = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed
  #ifndef Q3_VM
  #ifdef __linux__
    assert( !isnan(y) );
  #endif
  #endif
  return y;
}
```

According to an article in Code Maestro, not only does this function work, on some CPU Carmack's Q_rsqrt runs up to 4 times faster than (float)(1.0/sqrt(x), eventhough sqrt() is usually implemented using the FSQRT assembley instruction. So, how can we obtain the function that we want most – square root?

```
float SquareRootFloat(float number) {
    long i;
    float x, y;
    const float f = 1.5F;
    x = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
     y  = y * ( f - ( x * y * y ) );  //1st iteration
    y  = y * ( f - ( x * y * y ) );  //2nd iteration
     return number * y;
}
```

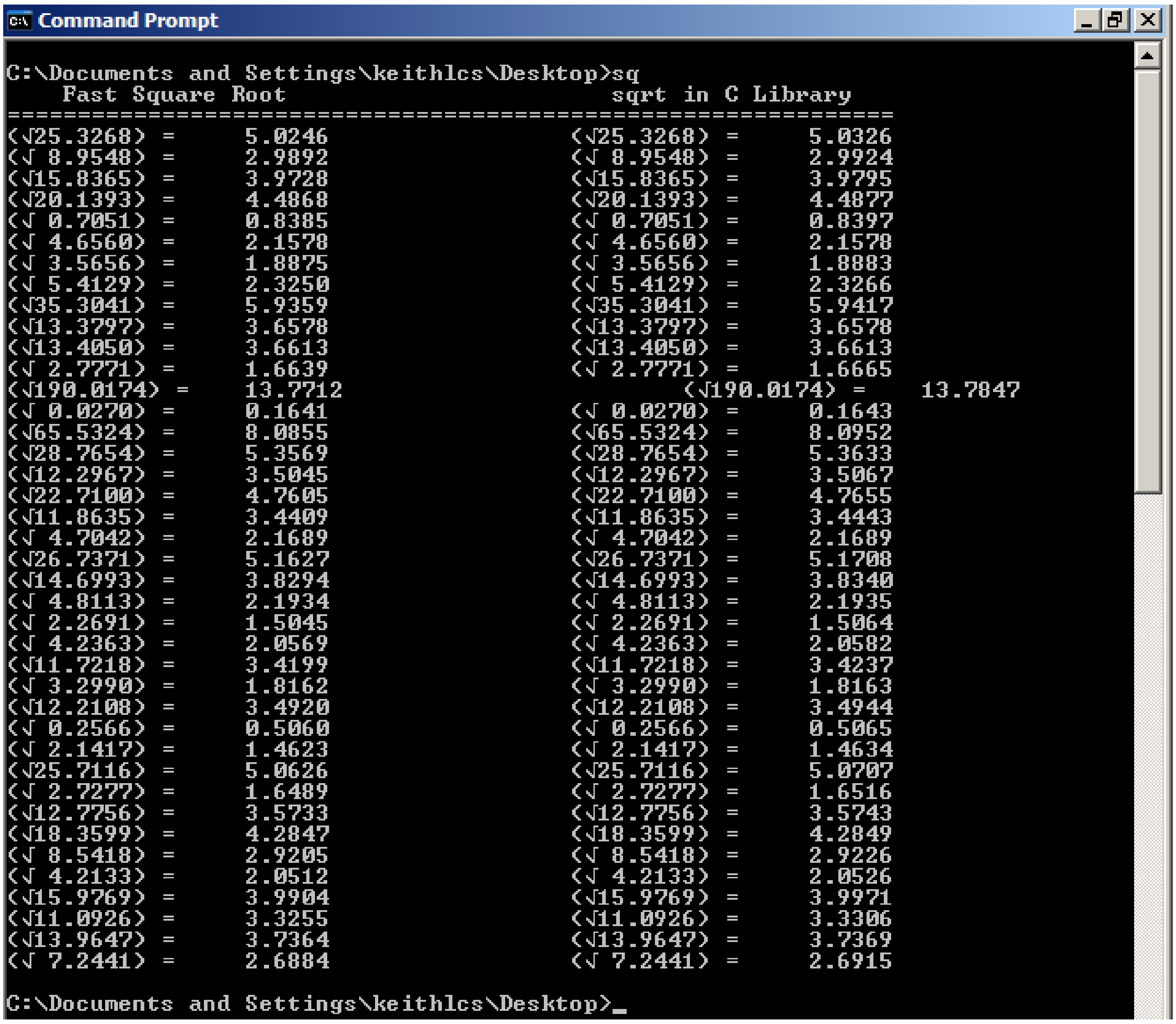
The only difference between the reciprocal of the square root and square root function is in the return value – instead of returning `y`, `return number*y` as the square root.

The code above implemented the well known Newton Approximation of roots. As in most other iterative approximation calculations, The Newton approximation is supposed to be ran in iterations. Each iteration enhances the accuracy until enough iterations have been made for reaching the desired accuracy. The general idea behind Newton's approximation is that whenever we have a guess y for the value of the square root of a number x, we can perform a simple manipulation to get a better guess (one closer to the actual square root) by averaging y with x/y. The really interesting aspect of this function is the magic constant 0x5f3759df, used to calculate the initial guess, in :

```
i  = 0x5f3759df - ( i >> 1 );
```

Hence, dividing the input by 2 and subtracting it from the magic constant. This constant works almost perfectly - Only one iteration is required for a low relative error of 10^{-3}.

We did perform a simple experiment to see the accuracy of the new square root function. We wrote a simple C program to perform the experiment. The experiment included the calculation the square root of randomly generated for 40 iterations. The left hand side of the result was by the new suggested square root function, the result shown on the right hand side was by the standard C library.

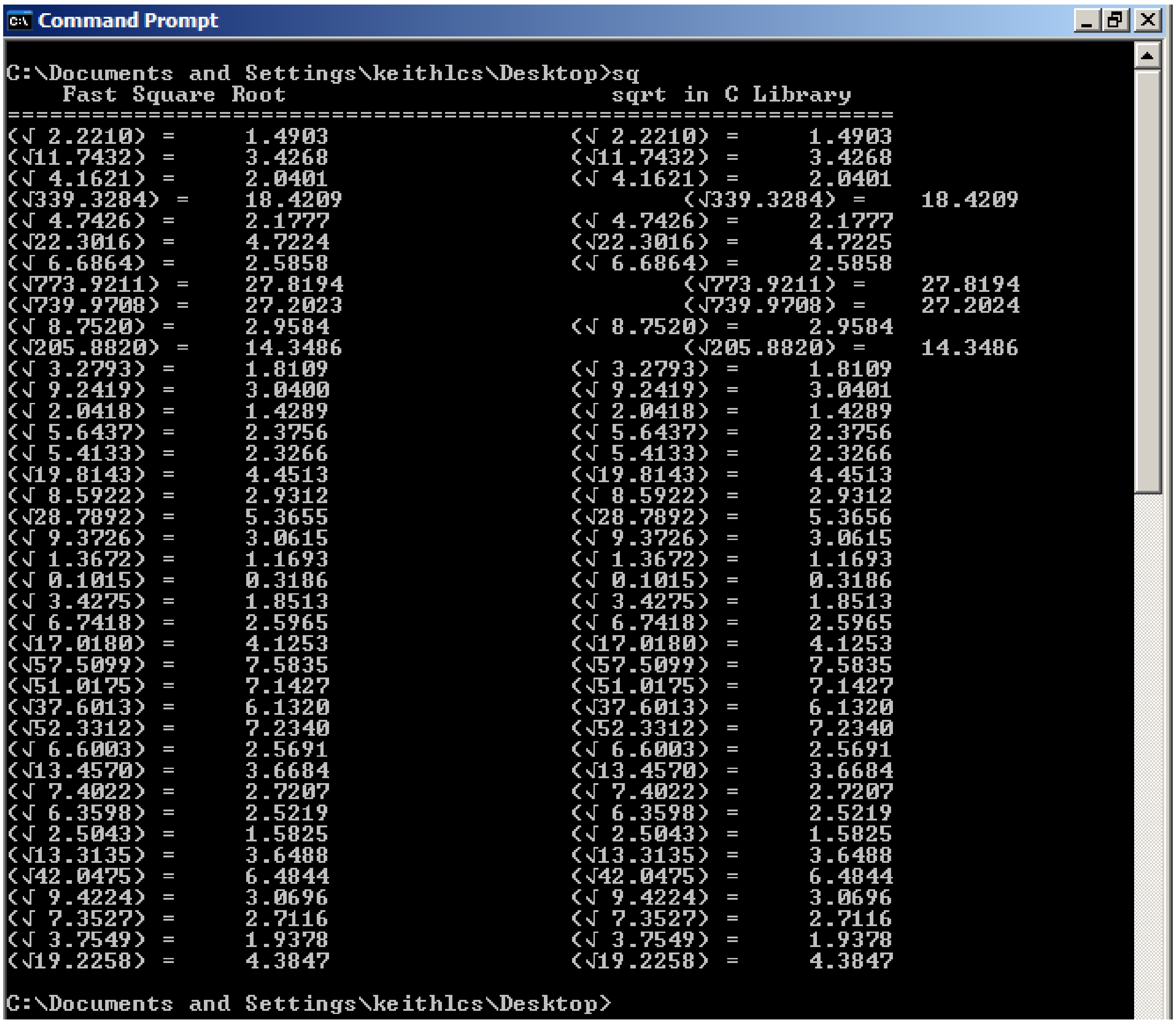


Command Prompt

```
C:\Documents and Settings\keithlcs\Desktop>sq
    Fast Square Root                       sqrt in C Library
================================================================
(√25.3268) =     5.0246                (√25.3268) =     5.0326
(√ 8.9548) =     2.9892                (√ 8.9548) =     2.9924
(√15.8365) =     3.9728                (√15.8365) =     3.9795
(√20.1393) =     4.4868                (√20.1393) =     4.4877
(√ 0.7051) =     0.8385                (√ 0.7051) =     0.8397
(√ 4.6560) =     2.1578                (√ 4.6560) =     2.1578
(√ 3.5656) =     1.8875                (√ 3.5656) =     1.8883
(√ 5.4129) =     2.3250                (√ 5.4129) =     2.3266
(√35.3041) =     5.9359                (√35.3041) =     5.9417
(√13.3797) =     3.6578                (√13.3797) =     3.6578
(√13.4050) =     3.6613                (√13.4050) =     3.6613
(√ 2.7771) =     1.6639                (√ 2.7771) =     1.6665
(√190.0174) =    13.7712                       (√190.0174) =    13.7847
(√ 0.0270) =     0.1641                (√ 0.0270) =     0.1643
(√65.5324) =     8.0855                (√65.5324) =     8.0952
(√28.7654) =     5.3569                (√28.7654) =     5.3633
(√12.2967) =     3.5045                (√12.2967) =     3.5067
(√22.7100) =     4.7605                (√22.7100) =     4.7655
(√11.8635) =     3.4409                (√11.8635) =     3.4443
(√ 4.7042) =     2.1689                (√ 4.7042) =     2.1689
(√26.7371) =     5.1627                (√26.7371) =     5.1708
(√14.6993) =     3.8294                (√14.6993) =     3.8340
(√ 4.8113) =     2.1934                (√ 4.8113) =     2.1935
(√ 2.2691) =     1.5045                (√ 2.2691) =     1.5064
(√ 4.2363) =     2.0569                (√ 4.2363) =     2.0582
(√11.7218) =     3.4199                (√11.7218) =     3.4237
(√ 3.2990) =     1.8162                (√ 3.2990) =     1.8163
(√12.2108) =     3.4920                (√12.2108) =     3.4944
(√ 0.2566) =     0.5060                (√ 0.2566) =     0.5065
(√ 2.1417) =     1.4623                (√ 2.1417) =     1.4634
(√25.7116) =     5.0626                (√25.7116) =     5.0707
(√ 2.7277) =     1.6489                (√ 2.7277) =     1.6516
(√12.7756) =     3.5733                (√12.7756) =     3.5743
(√18.3599) =     4.2847                (√18.3599) =     4.2849
(√ 8.5418) =     2.9205                (√ 8.5418) =     2.9226
(√ 4.2133) =     2.0512                (√ 4.2133) =     2.0526
(√15.9769) =     3.9904                (√15.9769) =     3.9971
(√11.0926) =     3.3255                (√11.0926) =     3.3306
(√13.9647) =     3.7364                (√13.9647) =     3.7369
(√ 7.2441) =     2.6884                (√ 7.2441) =     2.6915

C:\Documents and Settings\keithlcs\Desktop>_
```

From the above diagram, the fast square root was performed only once Newton method. We can see that the result is fairly accurate.

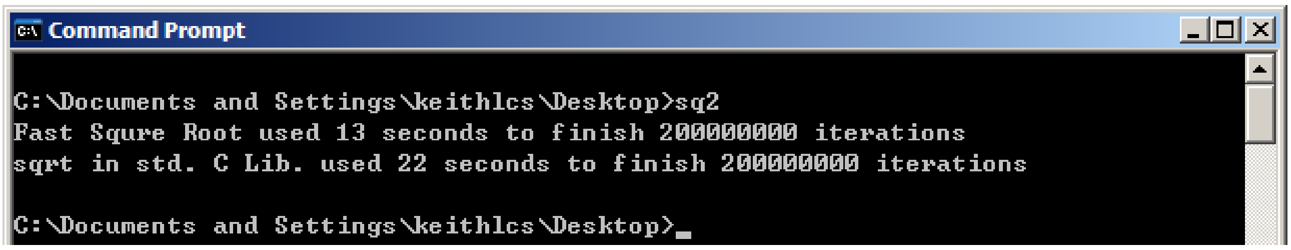


```
C:\Documents and Settings\keithlcs\Desktop>sq
    Fast Square Root                        sqrt in C Library
================================================================
(√ 2.2210) =     1.4903                  (√ 2.2210) =     1.4903
(√11.7432) =     3.4268                  (√11.7432) =     3.4268
(√ 4.1621) =     2.0401                  (√ 4.1621) =     2.0401
(√339.3284) =    18.4209                         (√339.3284) =    18.4209
(√ 4.7426) =     2.1777                  (√ 4.7426) =     2.1777
(√22.3016) =     4.7224                  (√22.3016) =     4.7225
(√ 6.6864) =     2.5858                  (√ 6.6864) =     2.5858
(√773.9211) =    27.8194                         (√773.9211) =    27.8194
(√739.9708) =    27.2023                         (√739.9708) =    27.2024
(√ 8.7520) =     2.9584                  (√ 8.7520) =     2.9584
(√205.8820) =    14.3486                         (√205.8820) =    14.3486
(√ 3.2793) =     1.8109                  (√ 3.2793) =     1.8109
(√ 9.2419) =     3.0400                  (√ 9.2419) =     3.0401
(√ 2.0418) =     1.4289                  (√ 2.0418) =     1.4289
(√ 5.6437) =     2.3756                  (√ 5.6437) =     2.3756
(√ 5.4133) =     2.3266                  (√ 5.4133) =     2.3266
(√19.8143) =     4.4513                  (√19.8143) =     4.4513
(√ 8.5922) =     2.9312                  (√ 8.5922) =     2.9312
(√28.7892) =     5.3655                  (√28.7892) =     5.3656
(√ 9.3726) =     3.0615                  (√ 9.3726) =     3.0615
(√ 1.3672) =     1.1693                  (√ 1.3672) =     1.1693
(√ 0.1015) =     0.3186                  (√ 0.1015) =     0.3186
(√ 3.4275) =     1.8513                  (√ 3.4275) =     1.8513
(√ 6.7418) =     2.5965                  (√ 6.7418) =     2.5965
(√17.0180) =     4.1253                  (√17.0180) =     4.1253
(√57.5099) =     7.5835                  (√57.5099) =     7.5835
(√51.0175) =     7.1427                  (√51.0175) =     7.1427
(√37.6013) =     6.1320                  (√37.6013) =     6.1320
(√52.3312) =     7.2340                  (√52.3312) =     7.2340
(√ 6.6003) =     2.5691                  (√ 6.6003) =     2.5691
(√13.4570) =     3.6684                  (√13.4570) =     3.6684
(√ 7.4022) =     2.7207                  (√ 7.4022) =     2.7207
(√ 6.3598) =     2.5219                  (√ 6.3598) =     2.5219
(√ 2.5043) =     1.5825                  (√ 2.5043) =     1.5825
(√13.3135) =     3.6488                  (√13.3135) =     3.6488
(√42.0475) =     6.4844                  (√42.0475) =     6.4844
(√ 9.4224) =     3.0696                  (√ 9.4224) =     3.0696
(√ 7.3527) =     2.7116                  (√ 7.3527) =     2.7116
(√ 3.7549) =     1.9378                  (√ 3.7549) =     1.9378
(√19.2258) =     4.3847                  (√19.2258) =     4.3847

C:\Documents and Settings\keithlcs\Desktop>
```

From the above diagram, the fast square root was performed twice the Newton method. We can see that the result is much better than the previous one.

In addition, we wrote another program to test their performance. We found out that the performance of the fast square root function is much better than the sqrt function found in standard C library.



```
C:\Documents and Settings\keithlcs\Desktop>sq2
Fast Squre Root used 13 seconds to finish 200000000 iterations
sqrt in std. C Lib. used 22 seconds to finish 200000000 iterations

C:\Documents and Settings\keithlcs\Desktop>_
```

In short, we used single Newton iteration to implement our fast square root function. It is because the accuracy is acceptable for our game and the performance is excellent.

In fact, we also performed some other optimization to our program code. For instance, instead of writing **`count * 2 + 1`** for positive number, we implemented in a better way, **`(count << 1) | 0x1`.** Furthermore, we tried to avoid division operation as much as possible. We tried to replace division operations with multiplication operations. For example, in section 4.2, the recurring equation of x velocity is $V_{x,i+1} = V_{x,i} - \left(\frac{B_2 \cdot V_i \cdot V_{x,i}}{m}\right)\Delta t$. It is observed that this equation requires the use of division. In order to optimize this statement, we can declare a constant, m', and stores the value of 1/m at compile time. The equation would become $V_{x,i+1} = V_{x,i} - \left(B_2 \cdot V_i \cdot V_{x,i} \cdot m'\right)\Delta t$. We can observe that the equation of $V_{x,i+1}$ is very similar to $V_{y,i+1}$, we can further reduce the number of multiplication operations in the following way.

$$Common = \left(B_2 \cdot V_i \cdot m'\right)\Delta t$$

$$V_{x,i+1} = V_{x,i} - \left(Common \cdot V_{x,i}\right)$$

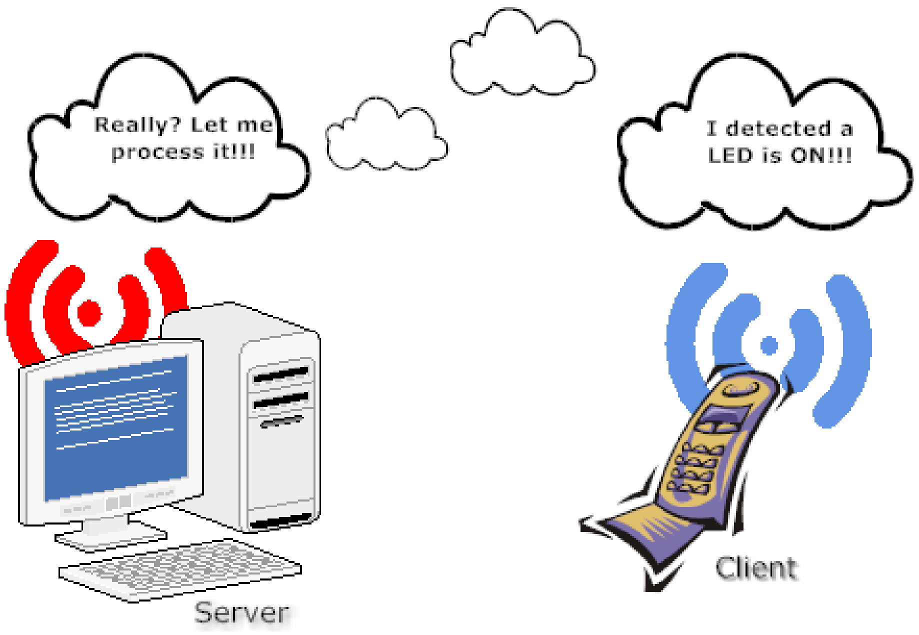
$$V_{y,i+1} = V_{y,i} - g\Delta t - \left(Common \cdot V_{y,i}\right)$$

In general, we would like to use shift bits and logical operations whenever possible. Reduce the number of division operations is also one of our aims. We have to find ways in order to use multiplication operations to replace the slow division operations. Sometimes, the functions were declared as inline in order to avoid too much overhead caused by function call.

4.6 Conceptual Program Prototype

Besides building the Virtual Reality mobile game, we also developed another mobile phone application (we call it as Conceptual Prototype) to demonstrate our objectives and concepts. Here, we would like to mention once again our objectives. The objective of our Final Year Project is not only to develop a game, but also to develop a way to demonstrate how a program can process memory to “remember” its external environment for interaction of Augmented Reality.

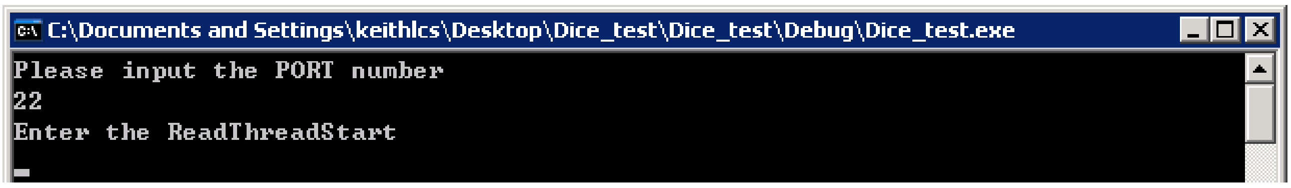
We used Nokia 6600 Mobile Phone to implement our conceptual prototype. Though this mobile phone is out dated and does not equipped with relatively powerful processor, it is enough to demonstrate our concept. In this prototype program, we would like to illustrate the use of our proposed LED recognition technique. (For the technical details, please read chapter 5)



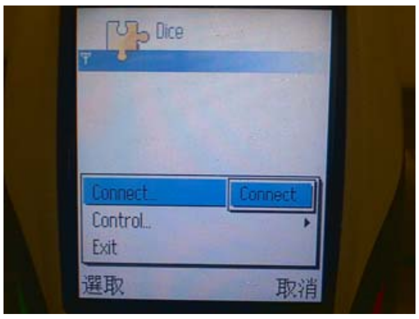
In our setting, there would be a computer with Bluetooth device acts as a server, and a mobile phone (Nokia 6600) acts as a client. The server is responsible for receiving signals and commands from the client. The client is responsible for detecting any LED in its environment with the use of build in camera. The client sends signals to the server whenever it detected there is/are LED(s). Though this settings and concepts are simple, we can make use of this concept in a number of areas.

For example, this concept can be extended to transport monitoring system. To day, there are many cameras installed at different major roads in Hong Kong for monitoring the traffic condition in nearly real time. The system can retrieve image data from the camera and then recognize the brightness of the environment. According to the data, the system can determine if the environment is bright enough and thus control the lighting conditions to facilitate a better driving condition.

In our setting, the client has to first establish a Bluetooth connection with the server before the detection can begin. In order to allow incoming connection to the server, the server has to enable a serial port for client's incoming connection. The diagram below shows the method to enable incoming connection in the server side.

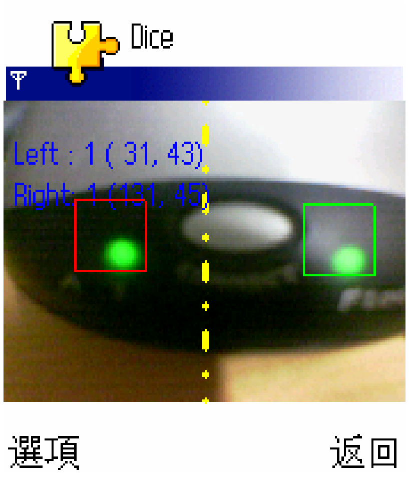

```
C:\Documents and Settings\keithlcs\Desktop\Dice_test\Dice_test\Debug\Dice_test.exe
Please input the PORT number
22
Enter the ReadThreadStart
_
```

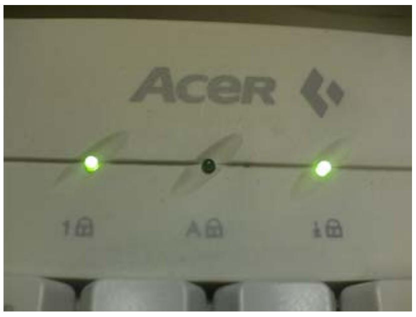
After that, the mobile phone client can connect to the server via Bluetooth. The figure on the right hand side demonstrates the method to establish a Bluetooth connection.



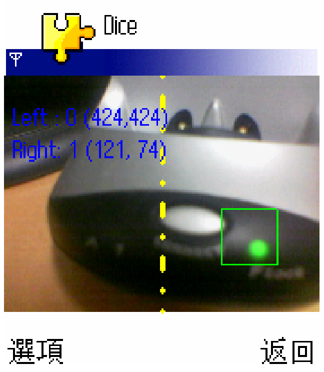
After that we may start the detection program in the client side, similar to the way shown on the right hand side.

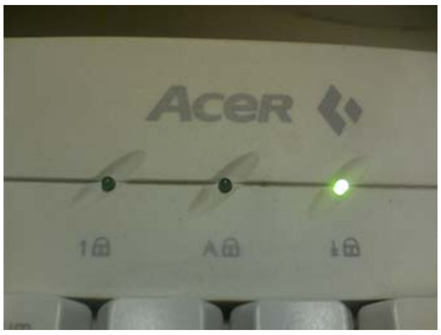
In our conceptual program model, we divided the detection into two parts. The program would only detect at most 2 green LEDs. One should be located on the left hand side and another one should be located on the right hand side of the camera. If the program can detect both LEDs, it would draw boxes to select the LEDs to indicate that the program can detect the present of LEDs in the environment. The program would then send signals to the server. In this conceptual program, the server would emulate the received LEDs pattern and reflect the result on to the LEDs in the keyboard. For example, if the client detected there are two LEDs as shown in the above diagram. The server would control the LEDs of the keyboard and emulate the received result which is shown on the left hand side diagram.

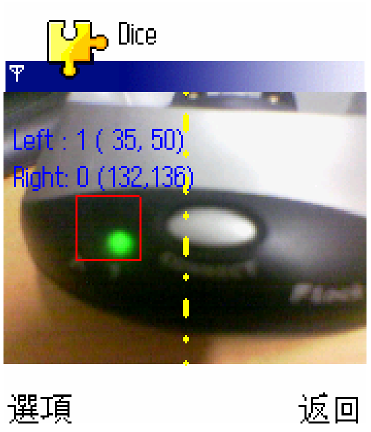




Similarly, if the client can only detect one LED, the server would simulate the result.







The previous two sets of figures show the result of the detection of LED done by mobile client and the simulated result done in the server side.

4.7 Feature Recognition in Semester 1 VS LED Recognition in Semester 2?

In semester 1, we did not use any LED recognition technique, we used feature recognition instead. In the feature recognition technique, we divided the search window into blocks. We use the saved features as the feature block and use whole screen as Search Window to find the most matching block in the screen.

Feature Recognition used the Fast Exhaustive Search Algorithm as searching method and SSD as the Cost Function mentioned in the previous semester to calculate the similarity between reference block and candidate block. The algorithm was speed up by using SEA to remove invalid candidate blocks, and then use PPNM to do second filter on candidate blocks and finally use PDE to remove invalid candidates block during the calculation of cost function.

Although the new feature algorithm in semester 1 applied many techniques to increase the speed of algorithm, but it was still very slow because it searched the whole screen to find the most matching features. (Please read Appendix I and II to review the experimental results for the feature recognition part done in semester one)

Due to the fact that the feature recognition was very slow and performance inefficient, we talked to the lab technician. He suggested us to work on recognition of some simpler things such as LED or Bar Code. Finally, we decided to work on recognition of LED because we thought that LED can be found more easily at home or office area. The technical details would be discussed in Chapter 5.

Chapter 5

Object (LED) Recognition

In this chapter, we describe how we can recognize the LEDs. The recognition part is a very important function in our program; it would affect the efficiency and performance of our program.

- **5.1 Introduction**
- **5.2 Harris Corner Detector in Semester 1 (Review)**
- **5.3 Fast Corner Detector in Semester 1 (Review)**
- **5.4 Select Feature from Corner List in Semester 1 (Review)**
- **5.5 Object Recognition Methods**
- **5.6 Feature Filter**
- **5.7 LED Recognition**
- **5.8 Motion Tracking Improvement**

We were using random features in the semester 1 for feature recognition, but we use specific feature – LED in semester 2 to solve the performance problems. Information in Section 5.1 to 5.4 is the review of the works we had done in the first semester. Please go to Section 5.5 onward for the works we have done in this semester 2.

5.1 Introduction

The feature is very important in the blocking matching part of our program because a "Good" feature can increase the efficiency of block matching algorithm as it can reduce the invalid candidate in early state of computation. (Please refer to Appendix I for the meaning of a feature)

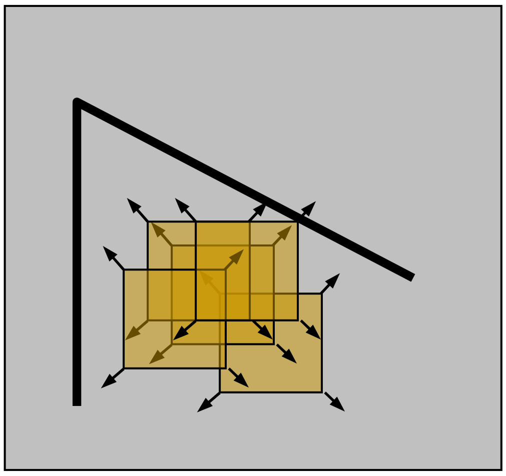
There are two conditions of a "Good" Feature:

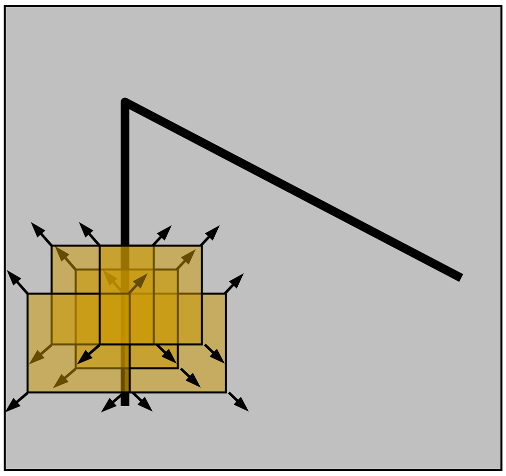
1. It should be descriptive enough to identify the feature from the environment, i.e. the "Good" feature should not be too small. It can increase the accuracy of the block matching algorithm.
2. It should have a large internal intensity different, i.e. the "Good" feature should contain a corner. It can record the correct direction of the movement, so it can increase the block matching accuracy.

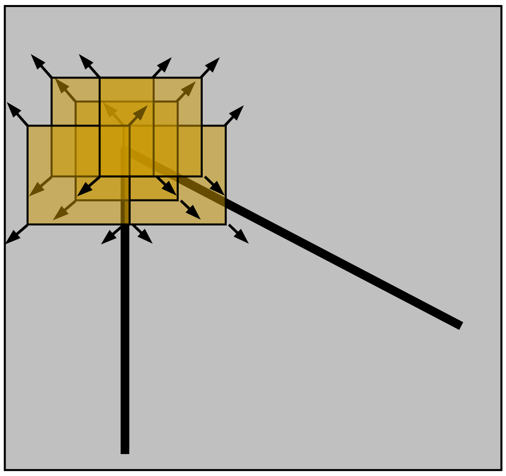
As we mentioned above, corner is a requirement of a good feature, a corner detector to detect the possible corners from the picture is very useful in the Feature Selection Algorithm.

5.2 Harris Corner Detector in Semester 1 (Review)

The basic idea of the Harris Corner Detector is calculate the intensity change in shifting all directions. In the "flat" region, there is no great change of intensity in all the direction. In the "edge" region, there is no considerable change of intensity across the edge direction. At the "corner" region, there is a significant change of intensity in all direction movement.


No significant change of intensity in flat region


No sharp change of intensity across the edge direction only


Intensity change significantly in all direction

The change of intensity of a shift [u, v] can be calculated by the following equation:

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

Where:

E(u, v): Total intensity change over the window

W(x, y): window function, it can be discrete (e.g. 1 inside the window, 0 outside the window) or continuous (e.g. a Gaussian distribution).

I(x, y): the Intensity of image at position (x, y).

If the shift [u, v] is small, it has a bilinear approximation:

$$E(x, y) = [u, v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

Where:

M is a 2×2 matrix computed from image derivatives

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

By evaluating the eigenvalues λ1, λ2 of M, we can identify the region is flat, edge or corner. In the flat region, both the λ1 and λ2 are small and E remains constant in all the direction. In the corner region, one of λ1 or λ2 is large, the other one is small and E has a large when it is not crossing the edge direction. In the corner region, both the λ1 and λ2 are large and E has very significant increase in all the directions.

The calculation of eigenvalues is computationally expansive because it involves calculation of square root. The following equation is used to calculate the eigenvalues of 2×2 matrix.

$$\lambda_{1,2} = \frac{1}{2}\left[(a_{11} + a_{22}) \pm \sqrt{4a_{12}a_{21} + (a_{11} - a_{22})^2}\right]$$

So Harris suggests a Corner Response Function R to calculate it:

$$R = \det M - k(traceM)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$traceM = \lambda_1 + \lambda_2$$

Where:

k is an empirical constant, typical value of k is range from 0.04 to 0.15.

R only depends on the eigenvalues of M and it is less computationally expansive than calculate the eigenvalues, so it can be use for detecting the corner. For a “flat” region, the absolute value of R is small. For “edge” region, R is negative and it has a large magnitude. For “corner” region, R is a very large value.

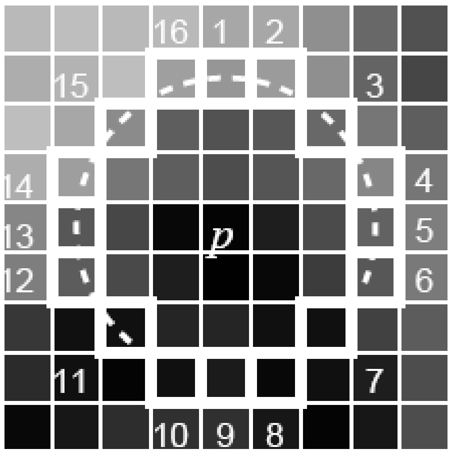
Although calculation of R is less expensive than calculation of eigenvalues, it is still very expensive for our platform, Symbian OS, which has no floating point unit.

5.3 Fast Corner Detector in Semester 1 (Review)

This algorithm is suggested by Edward Rosten in his paper Machine learning for high-speed corner detection, May 2006. [3]

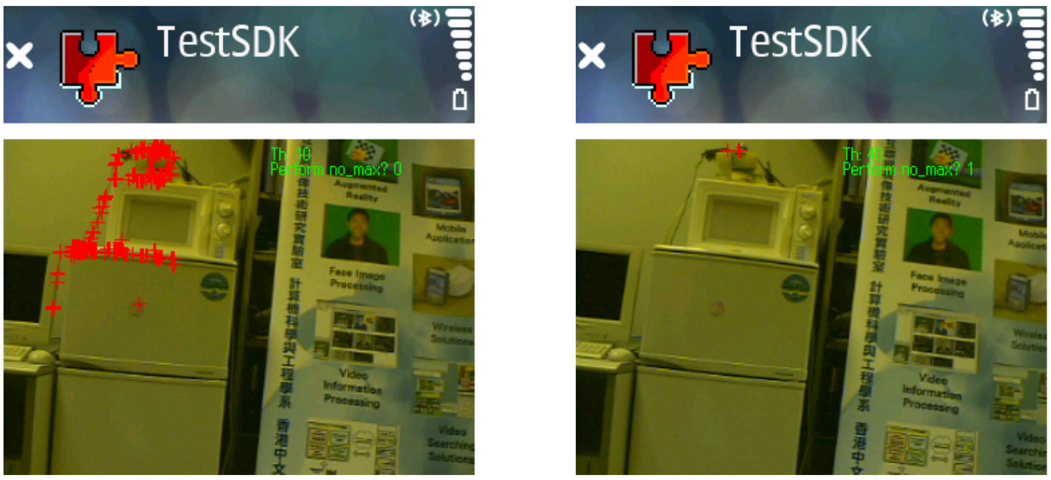
There is a class of corner detector which examines a small patch of image to see if it “looks” like a corner or not. In this class of corner detector, it doesn’t need a noise reduction step, such as Gaussian Filter, so it is less computation less than other corner detector. The FAST corner detector is inside this class of corner detector.

The main concept of FAST corner detector is considering the Bresenham circle of radius *r* around the candidate point which called nucleus. If the intensities of *n* continuous pixels on the circle are larger than nucleus by values *barrier* or smaller than nucleus by values *barrier*, the nucleus is a potential corner.



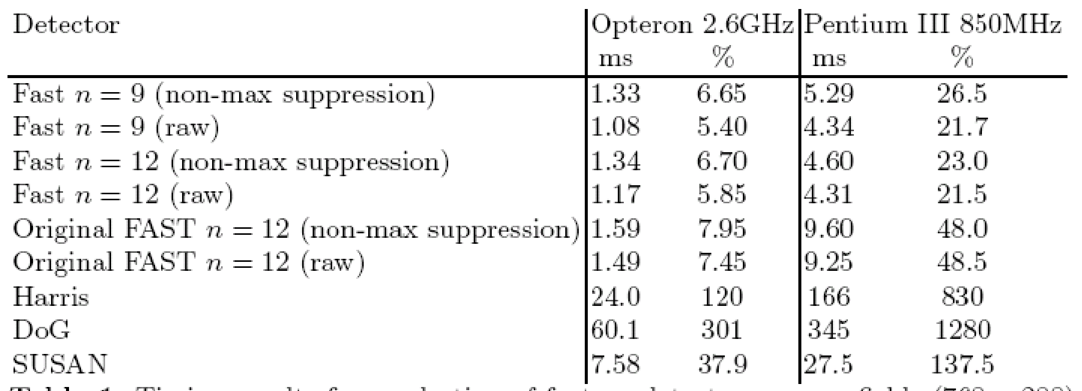
The Bresenham circle of radius 3 around the pixel p

The typical value for *r* is 3 that mean it will have 16 pixels on the circumference. The minimum value of n should be 9 to ensure that it will not detect the edge instead of corner. The value of barrier is directly related to the sensitivity of the corner detector. If the value of barrier is too small, the detector will be too sensitive, it may cause misclassification. On the other hand, if the value of barrier is too large, the detector may not be able to detect any corner. To get the suitable value of barrier, we had done an experiment of using different values of barrier on the same image on our Symbian testing platform, the full result of the experiment you can reference to Appendix II. After the experiment, we choose 25 as barrier value because it can detect a certain number of corners at many different environments.



Some of experimental results of the FAST corner detector on the Nokia N90

The FAST corner detector is very computation efficient, because it only does subtraction and comparison for the whole process detecting corner. The following experimental results are extracting from the Edward Rosten’s Paper:



Detector	Opteron 2.6GHz		Pentium III 850MHz	
	ms	%	ms	%
Fast $n = 9$ (non-max suppression)	1.33	6.65	5.29	26.5
Fast $n = 9$ (raw)	1.08	5.40	4.34	21.7
Fast $n = 12$ (non-max suppression)	1.34	6.70	4.60	23.0
Fast $n = 12$ (raw)	1.17	5.85	4.31	21.5
Original FAST $n = 12$ (non-max suppression)	1.59	7.95	9.60	48.0
Original FAST $n = 12$ (raw)	1.49	7.45	9.25	48.5
Harris	24.0	120	166	830
DoG	60.1	301	345	1280
SUSAN	7.58	37.9	27.5	137.5

Table 1. Timing results for a selection of feature detectors run on fields (768×288) of a PAL video sequence in milliseconds, and as a percentage of the processing budget per frame. Note that since PAL and NTSC, DV and 30Hz VGA (common for web-cams) have approximately the same pixel rate, the percentages are widely applicable. Approximately 500 features per field are detected.

The above result was done on PC. If the same experiment does on the Symbian platform, we believe that the different between the FAST corner detector and another corner detector will be much larger as Symbian doesn’t have a floating unit.

Non-maximal Suppression

Non Maximal Suppression is used as an intermediate step in mainly computer vision algorithm. Non-maximal Suppression means that finding out the local maxima of a pixel p around certain neighborhood. For the corner detectors, the non-maximal suppression means select all the corners with the local maximum of corner response function within a neighborhood.

FAST corner detector doesn’t define a corner response function, so we cannot apply the non-maximal suppression directly to filter the selected corners. Edward Rosten suggest a Score Function V, each selected corner should calculate V and use non-maximal suppression to remove corners which have corner with higher V within its neighborhood. There is the definition of Score Function V:

$$V = \max\left(\sum_{x \in S_{bright}} \left| I_x - I_p \right| - barrier \,,\ \sum_{x \in S_{dark}} \left| I_x - I_p \right| - barrier \right)$$

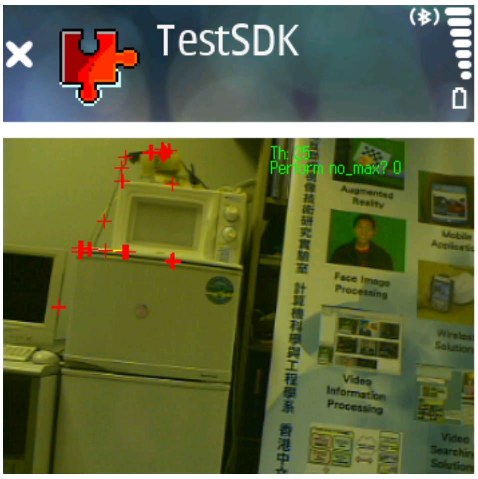
Where:

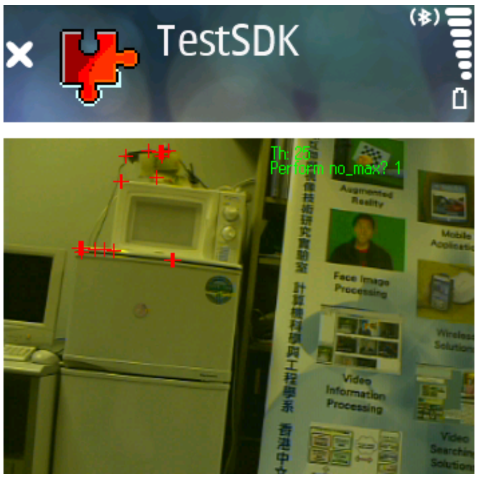
x are points of the Bresenham circle

$$S_{bright} = \{ x \mid I_x \geq I_p + barrier \}$$

$$S_{dark} = \{ x \mid I_x < I_p - barrier \}$$

We have done an experiment to show the different between using Non-maximal Suppression and not using Non-maximal Suppression. You can see the result from below (Detailed information can be found in Appendix II):





In our project, we define the size of neighborhood to be one, that mean it will check whether the adjacent points has a higher V or not.

Although the FAST corner detector is not robust under high noise environment, it is much faster than the other corner detectors. In the high noise environment, our motion tracking engine, mVOTE, is not accurate at all. There is no problem of us to use FAST corner detector as a part of our Feature Selection algorithm.

5.4 Select Feature from Corner List in Semester 1 (Review)

The corner detector will produce a list of detected corners, it may contain more than number of features that we want. So we need a corner selection step to choose number of features that we want from the corner list.

In our project, we need to keep track on three feature points during the application. We also set a constrain to these three points that they cannot be too close to each other because if the two points are too close, the features represent by each point may to overlapped, it will affect the accuracy of the motion tracking.

Our method is dividing the selection area into two equal parts vertically and run the corner detector on each part of the sub-areas. After running the corner detectors, we will get two lists of detected corners in each sub area and the corners are in the raster scanning order.

There is the pseudocode for select feature from corner list algorithm:

```
Feature1 = list1[first]
Feature2 = list2[first]

While ( list1.length > 1 And list2.length >1 )
Do
    fromOne = False
    If ( fromOne )
    Begin
        If ( list1.length > 1 )
        Begin
            Temp = list1[last]
        End
        fromOne = false
    End
    Else
    Begin
        If ( list2.length > 1 )
        Begin
            Temp = list2[last]
        End
        fromOne = true
    End
    If ( DistanceTest( temp, Feature1 ) AND DistanceTest( temp, Feature2) )
    Begin
        Feature3 = temp
        Break While Loop
    End
End
```

If we cannot find the third point for feature or either one of the list is empty at the beginning, we will reject the selection area and report to user that the program cannot find any feature.

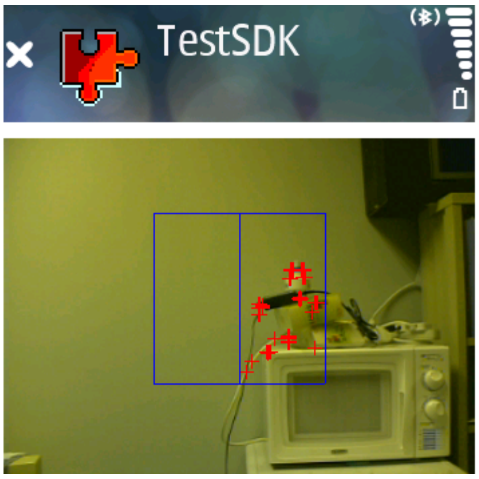


Fig a

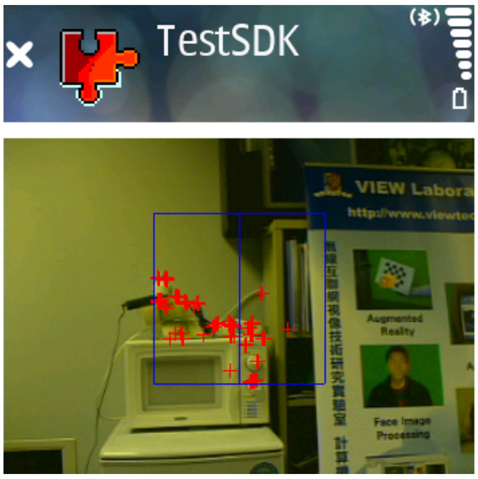


Fig b

Fig a. the selection area (marked by blue square) is rejected by our algorithm because left part of the area doesn't contain any corners.

Fig b. the selection area can pick the three points, two from the left half and one from the right part, for the features in motion tracking.

5.5 Object Recognition Methods

Due to the limited computational power and storage of mobile phone, it is very difficult to improve the current feature recognition algorithm. We had studied the Object Recognition methods to see it can improve the performance of the feature recognition or not.

There are many different kinds of Object Recognition method. There are three major steps in the Object Recognition:

1. Model Acquisition

   Object Recognition algorithm needs a model for object to be recognized. The model can be provided by the user or learned from the sample inputs images by using learning theory.

2. Retrieval

   There are three main tasks in this step:

   i. Localize the interested objects in the input image(s).
   ii. Identify or classify the interested objects.
   iii. Provide the relevant information (for example position, direction and size of the objects) of the interested objects.

3. Action

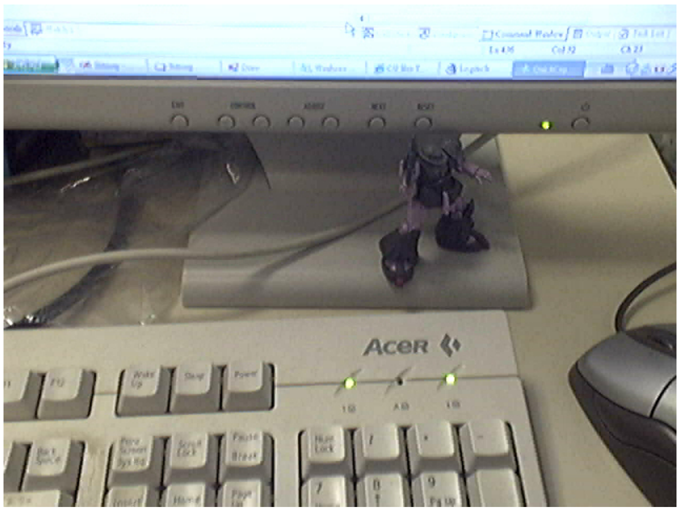
   From the information gather in second step, perform the relevant actions.

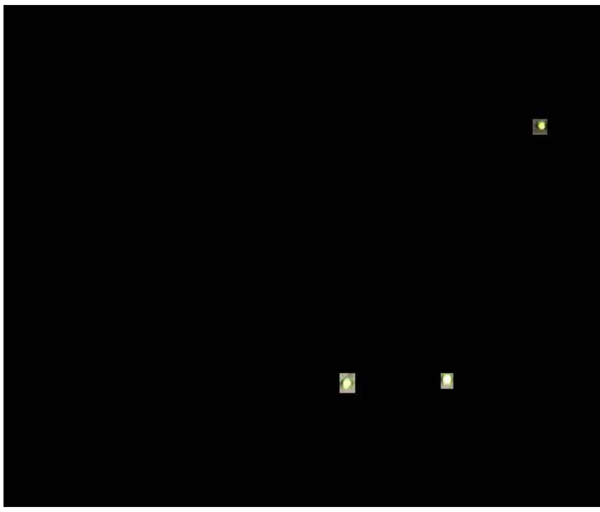
By using object recognition methods, you need to know what kind of object you want to recognize first. But the goal of our FYP is letting user to select an area, program will find out the possible features in the selected area and start playing the game. That mean it is using random feature instead of specific objects, so it cannot use the object recognition method directly. We decide to use object recognition to enhance the performance of the feature recognition. So, we need to decide use what object we need to recognize.

We have set a rule for selecting the objects for recognition, the object should be common in many different environments. So the user can play the game at anytime and anywhere. We decide to use green/red LEDs as the target objects of the object recognition because LEDs are common in the indoor environments, such as home, office and school. Also, LEDs can be easily separated from the background environment, so we can use this property to design specialize algorithm for it.

5.6 Feature Filter

The objective of the filter is separate the target objects – green/red LED from the surrounding environment, so that it can improve the performance and efficiency of the feature recognition algorithm. The principle of the feature filter is if the pixel cannot pass through filter condition, then it will change to black color. Otherwise, the pixel remains unchanged. We use green LED as target object to illustrate our idea.

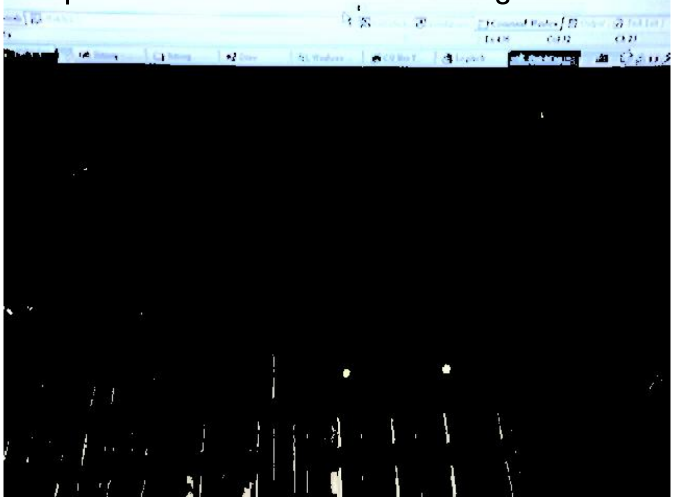




The left hand side picture is the input image of the feature filter and the right hand side is the output of the ideal feature filter.

Feature Filter by using HSV Color model

The target object is green LED, so the fist idea is can we use the brightness value of HSV model to separate the LED from the surrounding. The first filter considers only the brightness value of each pixel. The filter condition is brightness value greater than 0.9.



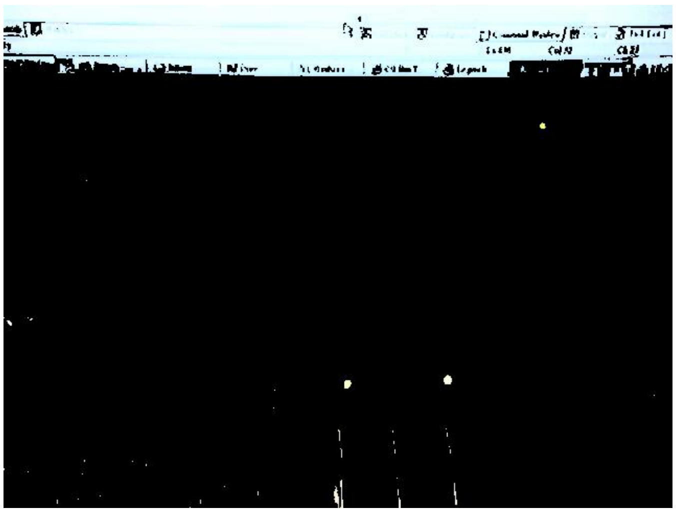
From the result, you can see that the bright region such as the LCD monitor, our target green LED and the reflect light on the keyboard are selected. We do not consider the color in the filtration. So, we can add Hue consideration in the filter. The following figure show the result of filter after adding Hue consideration, only pixel with Hue value within 60 to 100 can pass through the filter:

You can see that only one green LED on the keyboard is selected and there are some noises left in the monitor region.

We find out that HSV filter cannot completely separate the target from the surrounding environment. Also, image got from the camera on the mobile phone is in Bitmap format which is using RGB color model. The conversion from RGB to HSV invoke floating point operation, the conversion speed is very fast. This is another problem of the HSV filter.

Feature Filter by using RGB Color model

The target object is green LED, so we can set only the pixel with the large G value can pass through the filter. Here shows the result of filter using G value greater than 230.



You can see that many regions are selected, such as green LEDs, the white region of LCD Display.

We can add constraint to the other color channel, for example the blue value and red value need to smaller than certain value. We have done a series of experiment; we choose the G value need to greater than 230 and the B value need to smaller than 204. There is the result image:

You can see that all threes LEDs are separated and with only a little amount noise left. So we choose this filter as the filter we need of the program.

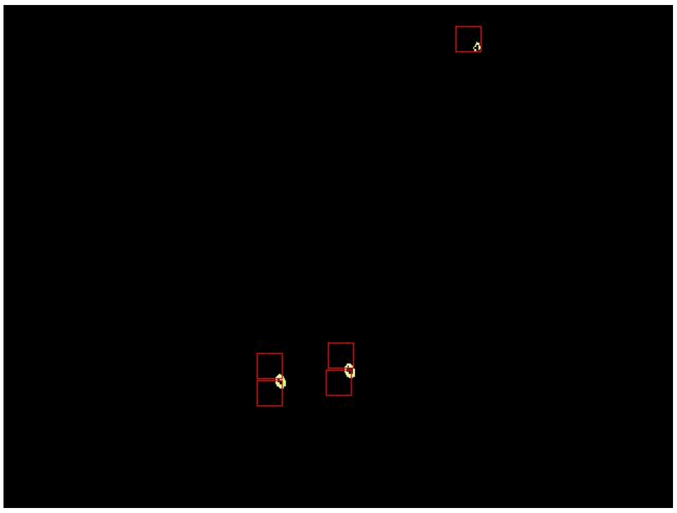
We can use similar technique to build feature filter for the red LED. We can set a lower bound for R and upper bound for G to build the feature filter.

5.7 LED Recognition

After passing the feature filter, we get the filtered image which most pixels are black and only the selected regions remain unchanged. The next step is telling the motion tracking algorithm where does it need to track. The motion tracking algorithm is using block matching method, so we only need to find out a block which contain the target object.

Only the non-black pixels are the area we want. So we can claim that if the block contains more than threshold of non-black pixels, then it is the block which contains the target object. We set threshold to be 25 for the following. The following is the pseudocode and the result of the feature selection algorithm:

```
Array of Block SelectedBlock = NULL
For each Block in the search range
    Count = 0
    For each pixel in the block
        IF needTest(pixel)
            IF pixel is not black
                Count = Count + 1
            ELSE
                Go to Next block
    IF Count > LightPixelThreshold
        SET needTest for all pixels in the Block be False
        Add Block to SelectedBlock
```

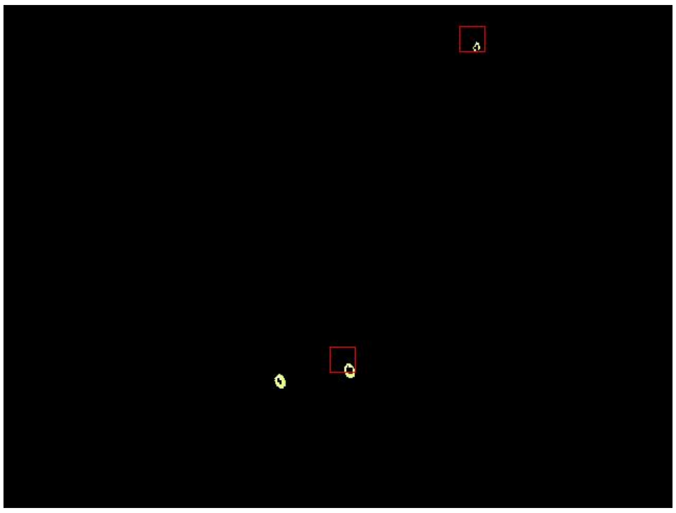


From the result you can see that the algorithm can only select the block with light region at the lower right corner. It is because we are using raster scan method to examine the block. It is not a good feature for motion tracking because it may lose easily during the motion tracking. It also selects more than one feature block from the same object. The speed of this algorithm is not fast because examine every pixel in the image.

To solve the shifting problem and increase the performance, we only examine the center region of the block. The center region is a concentric block with the half size of the original block. We also add a distance check mechanism to prevent two selected blocks is too close to each other. We only need two features for playing the game, so we can stop the algorithms after finding two features. Here is the pseudocode and the result of the improved feature selection algorithm:

```
FeatureCount = 0
SelectedBlock1, SelectedBlock2 = NULL
For each Block in the search range
    Count = 0
    For each pixel in the center region
```

```
        IF needTest (pixel)
            IF pixel is not black
                Count = Count + 1
            ELSE
                Go to Next center region
    IF Count > LightPixelThreshold
        IF FeatureCount = 1
            IF Distance (SelectedBlock1, Block) > DISTANCEBOUND
                SelectedBlock2= Block
                EXIT
        ELSE
            SET needTest for all pixels in the Block be False
            SelectedBlock1= Block
            FeatureCount = 1
```



You can see that the result of improved feature selection algorithm is much better. Also the speed of the recognition is faster.

Due to the concern of the performance of the motion tracking, we reduce the motion tracking points from two down to one. It is because reducing the number of tracking points can reduce the load of the motion tracking part of the program, so we can have more resource for the front end. We only need one feature now, so we can remove some part of the feature selection algorithm to reduce the time spend and memory usage of it. The pseudocode of the finalized version of feature selection algorithm:

```
SelectedBlock = NULL
For each Block in the search range
    Count = 0
    For each pixel in the center region
        IF pixel is not black
            Count = Count + 1
    IF Count > LightPixelThreshold
        SelectedBlock = Block
        EXIT
```

5.8 Motion Tracking Improvement

The motion tacking part is still using the function provided in the mVOTE engine. We find out that the region surrounding our target object, LEDs, are flat region. A flat region is a region that the pixels in the region which has almost the same color. The motion tracking algorithm cannot work well in flat region because the blocks inside the flat region are very similar, mVOTE cannot identify them. So we need to do a detector to detect the algorithm is tracking the target object or not.

We pass the tracking block and the eight surrounding blocks to the feature filter. After passing filter, we do the block matching. If the block sum of the matching block is zero, then we do the LED selection in the filtered region. The block with block sum is zero mean all the pixels in the block is black that imply motion tracking algorithm is lose tracking of the target object. So we need to relocate the feature for the motion tracking again. Why choose eight surrounding blocks? We find out that in many cases that the feature is within the eight surrounding blocks when it is lose tracking.

Chapter 6

Project Progress, Difficulties and Future Work

This chapter would briefly describes progress of our project, difficulties we faced during the project and what we are going to do in the future

- **6.1 Project Progress**
- **6.2 Difficulties we face in the project**
- **6.3 Future Work**

6.1 Project Progress

There is progress of out Final Year Project:

September 2006	1. Decide platform of FYP 2. Study the Symbian C++ Programming 3. Study the Algorithm use in mVOTE engine 4. Familiar with the development platform of Symbian
October 2006	1. Write simple Symbian program 2. Write the Symbian Testing platform 3. Study basic idea of Image Processing 4. Study some corners detector algorithm 5. Study the possibility of Z motion detection
November 2006	1. Implement the initial approach of feature recognition 2. Implement the FAST corner detector algorithm into Symbian Platform. 3. Implement our proposed approach of feature recognition 4. Prepare FYP presentation and demonstration 5. Write FYP report.
December 2006	1. Try to improve the feature recognition
January 2007	1. Study the possibility of hierarchical search method. 2. Study the MPEG encoding scheme.
February 2007	1. Implement Feature Filter. 2. Study projectile motion 3. Implement Game Engine
March 2007	1. Implement LED Recognition 2. Study Active Object of Symbian 3. Draw 3D of the dart 4. Study Bluetooth programming 5. Enhance the Motion Tracking
April 2007	1. Integration of Game engine and LED recognition 2. Fine tuning the program 3. Build the Conceptual Program Prototype 4. Write FYP Final Report 5. Present final presentation

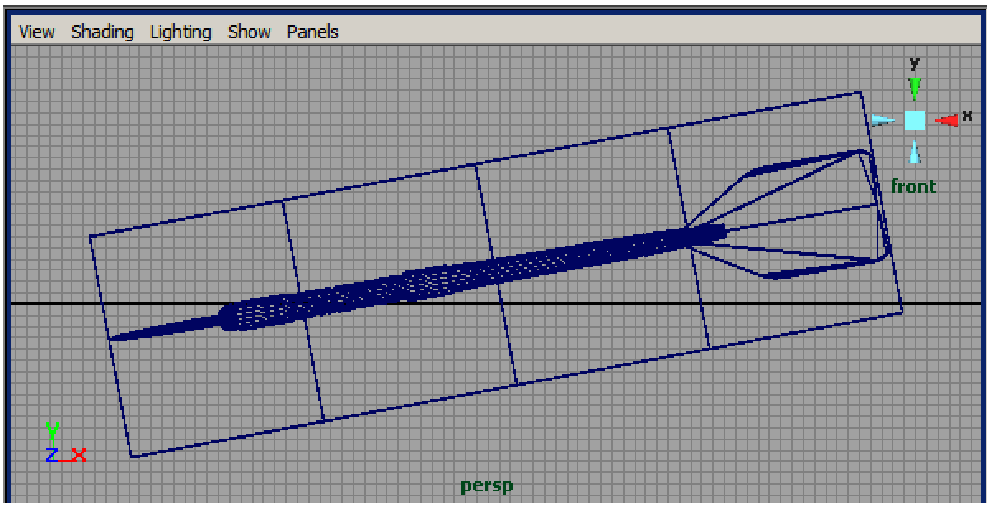
6.2 Difficulties we face in the project

To me, there are a number of difficulties when doing this project.

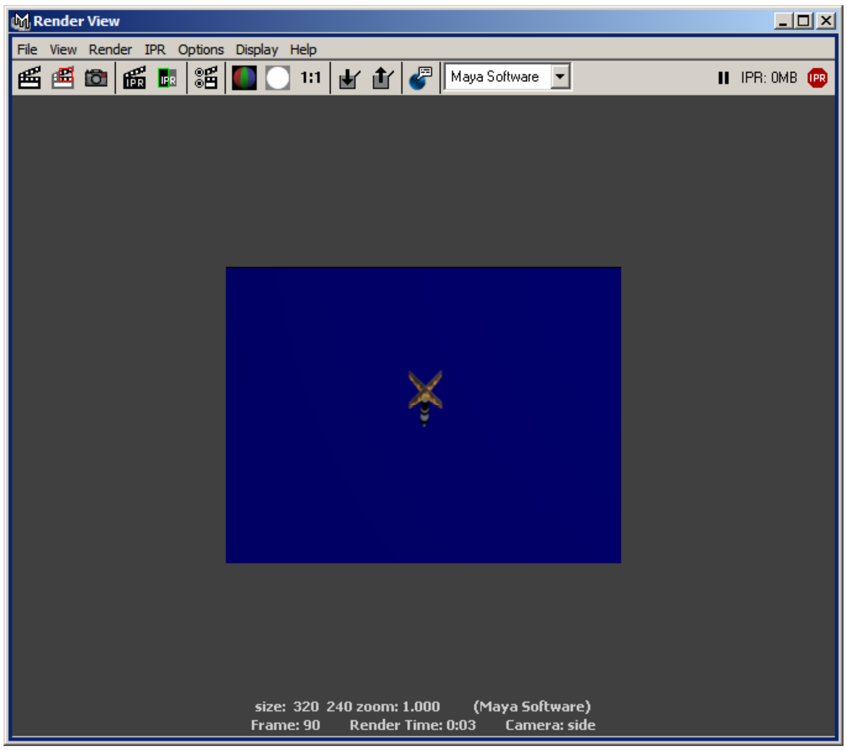
1. Before this final year project, we do not have background in image processing. It turned out that we had to put a lot of efforts in packing up these kinds of technique. In addition, we had to put concentration on the mVOTE Engine before we can actually modify it for motion tracking which suit our needs. As mVOTE Engine used quite a number of algorithms suggested in different papers. We had to read through the papers to enhance our understanding. We also found some other papers to improve the speed of recognition.

2. On the other hand we used Nokia N80 and N90 as our target platform. However, Nokia only provide the camera plug-in for Nokia 6600 which is S60 1st Edition (Symbian OS v7.0s). Therefore, we cannot use emulator for debugging and testing. We need to test our program on the target phone directly. As a result, every time, we had to transfer the executable code to the target phone and the following step was installation and testing. The whole process is thus much longer and inconvenient. In semester two, we built some C# program with the use of webcam to facilitate the development of the image filter for LED recognition as well as bitmap color removal process. After the algorithms are implemented and tuned in PC, we then transfer them to our target platform and perform further fine tuning and amendment.

3. Since we prepared our algorithm in PC computers using webcam. We found out that the image color captured by the webcam is different from that of mobile phone. We also found that among the provided mobile phones, the image color captured by these phone were largely different. We found out that the camera provided by Nokia N80 is not really in a good condition. Most of the time, the degree of color distortion is quite high and affect our LED recognition result. While Nokia N90 had a better image color quality and relive us from the above problems. As a result, we had to perform fine tuning on Nokia N80 from time to time to minimize the effect.
4. From this project, we also found out that developers have to put much more time in the graphics part for game development than in the core function of the game. For example, in our game, the dart throw would be displayed in term of animation. However, to build an animation, we have to collect a large amount of image data about the dart itself. It is because during different stages of dart throw, the dart would be in different angles of posture. To make the animation more realistic, we have to find ways to generate the dart images for each degree posture. In order to generate a set of such images, we decided to use Maya to build a dart 3D model for image generation.



And we had to render an image for every degree of rotation. The right hand side shows an example of rendering. The rendering process is time consuming and we had to manually alter the degree of rotation from frame to frame for rendering. From my point of view, this process is very time intensive and just for the user fanciness.



size: 320 240 zoom: 1.000 (Maya Software)
Frame: 90 Render Time: 0:03 Camera: side

5. Another difficulty is about creating “thread” in Symbian OS. During the animation, the program has to animate the dart throw as well as accepts information from our LED recognition module. The obverse solution would be to create “thread” in order to let the two “processes” run in parallel. However, there is not “thread” in Symbian OS. Instead, it uses so called “Active Object” and its concept is quite difficult to understand. Therefore, we had to spend a bit time in handling this issue.

6. Another difficulty is about the documentation support part. In building Symbian program, we had to consult the Nokia S60 documentation for reference. That document outlined the use of different functions, libraries, data types, etc. However, we have encountered that the required libraries (list on the document) were included in our project. However, the compiler cannot properly link the libraries to our program which caused compilation errors. It turned out that we search for the solution for over hours and finally discovered that we had to

include more libraries which were not documented in the Nokia S60 documentation. We also found a very interesting phenomenon that the Nokia S60 documentation included some classes and functions which were not yet implemented. However, the document itself did not mention this and we had to seek for help from some other forums.

7. Another difficulty is about debugging. We mentioned previously, we did not really have an emulator to debug our program. In other words, we cannot set any break points or watches to debug our program in IDE in real time. From time to time, we did have to write a lot of code to "print" the debug messages onto the screen for debugging. Though this method was time consuming, inefficient and troublesome, still that was the only way for our debug.

6.3 Contribution of Work

Semester 1

Before the start of the project, I have no idea about image processing and Symbian C++. So I and my partner had spent a lot of time to study the image processing, Symbian C++ and the working principle of the mVOTE engine.

After the background study, I change my focus to improve the functionality and the performance of the mVOTE engine. I had studied the many corner detector algorithms, such as Harris Corner, Difference of Gaussian (DoG), Laplacican of Gaussian (LoG), in order to enhance the feature selection of the mVOTE engine. After a discussion with my partner, we decide to use FAST corner detector to enhance the feature selection. I had also studied the possibility of adding Z-Motion detection to the mVOTE engine. After reading some papers, I find out it is not feasible to implement Z-Motion detection because many algorithms related to it involve a lot of floating point calculation.

At the middle of the semester, I and my partner want to build an algorithm to recognize a scene in the captured image. So we develop the feature recognition algorithm. The result of the feature recognition is poor. In many case, it uses about 30 seconds to re-cognize the feature defined before. The speed of algorithm is unsatisfactory, and it is the major focus of our works in semester 2.

Semester 2

At the start of Semester 2, I and my partner had read some paper about hierarchical search for block matching. After the experiment, we find out the performance is much worse than before. It is because the scaling step takes a lot of time. We had also tried other method to improve the accuracy and speed but the result is not very significant.

We talked to the lab technician about our difficulties. He suggested it may not be possible to have significant improvement before the end of semester because the problem is not easy. He also suggested us to do recognition on the more specific things such as bar code or LED. We decided to use LED as target object because it is common in many indoor environment.

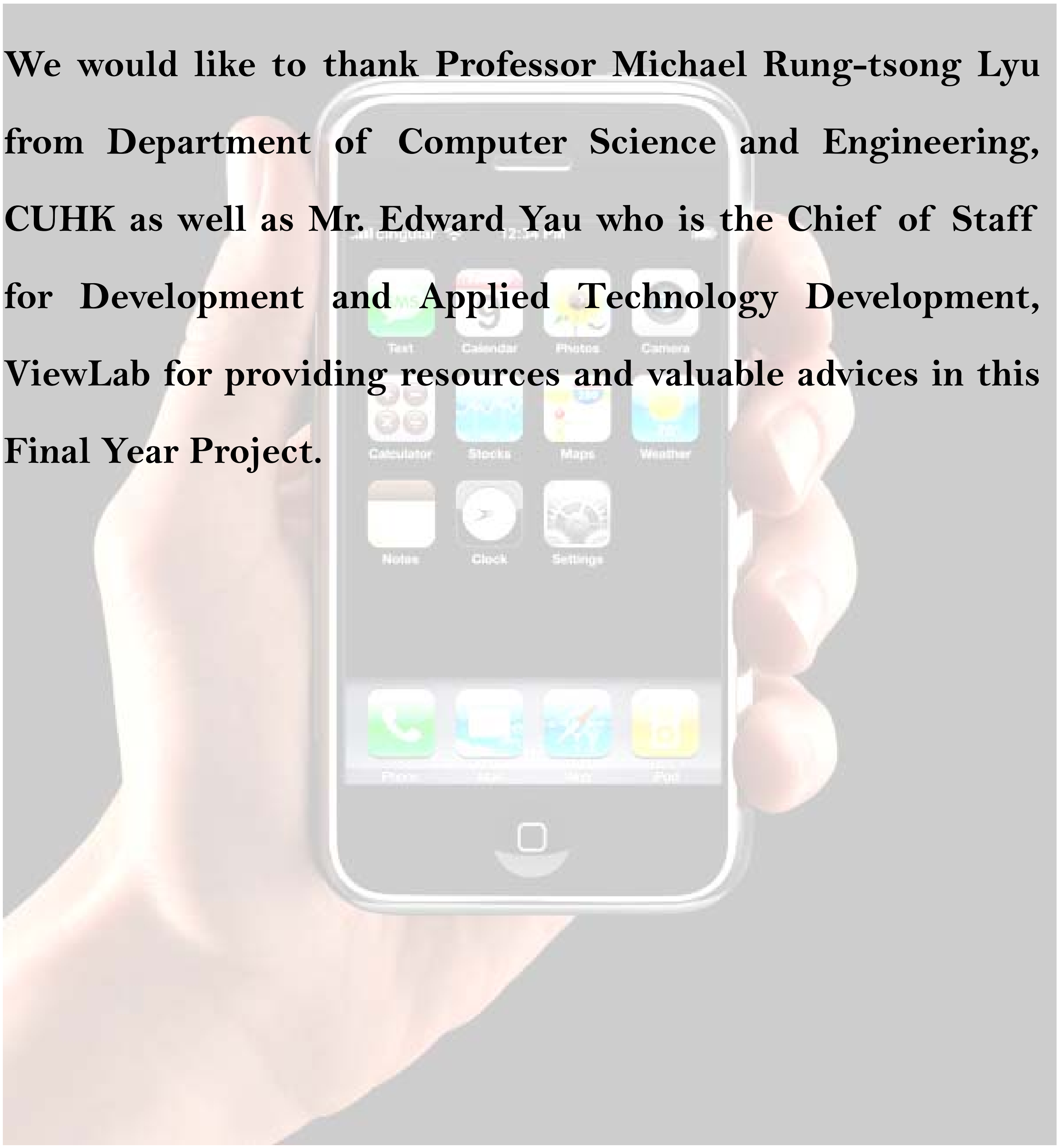
After this, we decided our work focus. My partner is more familiar with Symbian programming than me, so he worked on the game engine and I work for the LED recognition. First I developed the feature filter and LED recognition on Window platform by using Web Camera as input device. After completion of window version, I transferred the program to Nokia N90 for testing on Symbian platform. Finally, I transferred it to Nokia N80 and integrate with the game engine developed by my partner.

Conclusion

Through out this project, I have learnt a lot of techniques of developing application on Symbian platform. I also learn much knowledge about image processing. After the project, I know that game development is not easy as I thought before. It requires a lot of time and effort to develop the graphics of the game. Programming on Symbian platform is a very special experience for me because I do not have any experience on developing application on a platform with a lot of hardware limitation.

Chapter 7

Acknowledgement



We would like to thank Professor Michael Rung-tsong Lyu from Department of Computer Science and Engineering, CUHK as well as Mr. Edward Yau who is the Chief of Staff for Development and Applied Technology Development, ViewLab for providing resources and valuable advices in this Final Year Project.

Chapter 8

References

1. Harris, C., Stephens, M.: “A combined corner and edge detector.” In: Alvey Vision Conference. (1988) 147-151
2. Kuo-Liang Yeh, Wen-Hsien Fang, Mon-Chau Shie, Fei-pei Lai: “BITCEM: Adaptive Fast Block Motion Estimation via Binary transform Center of Mass Object Tracking” 康寧學報 (1993)
3. Deepak Turaga, Mohamed Alkanhal: “Search Algorithms for Block-Matching in Motion Estimation” 1998
4. Pedro F. Felzenszwalb, Daniel P. Huttenlocher: “Pictorial Structure for Object Recognition” International Journal of Computer Vision (2004)
5. David. G. Lowe: “Object Recognition from Local Scale-invariant features” Computer Vision (1999)
6. Edward Rosten and Tom Drummond: “Fusing points and lines for high performance tracking.” In: IEEE International Conference on Computer Vision (2005) 1508-1511
7. Edward Rosten and Tom Drummond: “Machine learning for high-speed corner detection” In European Conference on Computer Vision (2006) 430-443
8. R. Jin, Y. Qi, A. Hauptmann: "A Probabilistic Model for Camera Zoom Detection", Proceedings of ICPR 2002 Quebec, August 2002.
9. Po-Hung Chen, Hung-Ming Chen, Kuo-Liang Yeh, Mon-Chau Shie and Feipei Lai, “BITCEM: An Adaptive Block Motion Estimation Based on Center of Mass Object Tracking via Binary Transform,” 2001 IEEE Int'l Symp. on Intelligent Signal Processing and Communication Systems, Nashville, Tenn., USA, Nov. 2001
10. Noguchi, Y., Furukawa, J., Kiya, H.: “A fast full search block matching algorithm for MPEG-4 video”, Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference, 1999
11. J. Verestoy and D. Chetverikov, ‘Tracking Feature Points: A New Algorithm’, Proc. of 14th International Conf. on Pattern Recognition, pp. 1436-1438, Australia, 1998
12. Steve Babin,: “Developing Software for Symbian OS - An Introduction to Creating Smartphone Applications in C++”, Wiley 2005
13. Jo Stichbury, “Symbian OS Explained – Effective C++ Programming for Smartphones”, Wiley 2004
14. Meghna Singh, “Robust Tracking and Human Activity Recognitio”, 2004

15. David A. Forsyth, Jean Ponce: “Computer Vision: A Modern Approach” Prentice Hall, 2002
16. Forum Nokia: “S60 2nd Edition: Getting Started with C++ Application Development”, 2004
17. Forum Nokia: “S60 Platform: Porting from 2nd to 3rd Edition”, 2006
18. Forum Nokia: “S60 Platform: Scalable Screen-Drawing How-To”, 2006
19. Forum Nokia: “S60 Platform: Scalable UI Support”, 2006
20. Forum Nokia: “S60 Platform: Source and Binary Compatibility”, 2006
21. Forum Nokia: ”Image Conversion Library”, 2004
22. Jani Vaarala, Nokia: “OpenGL ES development on Serires 60 and Symbian”
23. http://www.viewtech.org/html/mvote_tm_.html
24. http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/index.html
25. http://www.phy.davidson.edu/StuHome/jocampbell/projectile/projectile.htm
26. http://physics.gmu.edu/~amin/phys251/Topics/NumAnalysis/Odes/projectileMotion.html
27. http://en.wikipedia.org/wiki/RGB_color_model
28. http://en.wikipedia.org/wiki/HSV_color_space

Appendix I

Feature Selection (REVIEW)

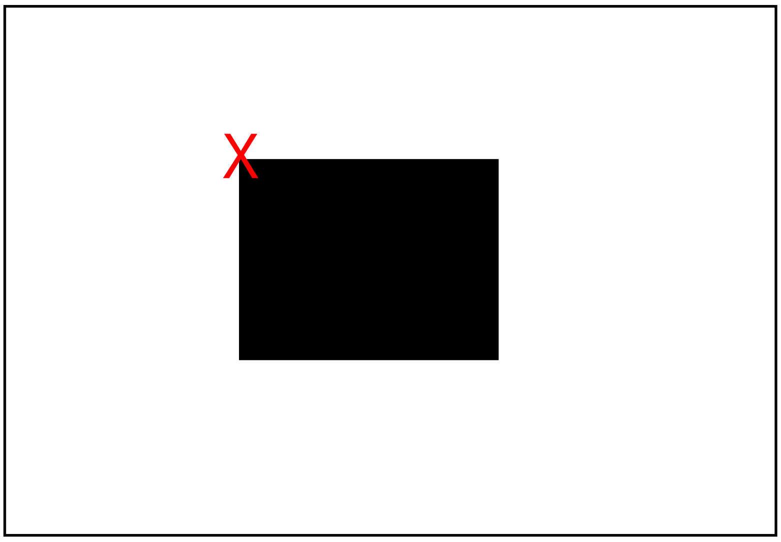
Appendix I describes the experiment conducted in semester 1 for testing the feature selection method used in the existing Motion Tracking Engine as well as the Fast Corner Detection Algorithm. This appendix also contains an analysis to determine which algorithm outperforms another one.

- **AI.1 What is a feature?**
- **AI.2 Pictures under Normal Lighting Condition**
- **AI.3 Pictures under Insufficient Light Condition**
- **AI.4 Analysis**

AI.1 What is a feature?

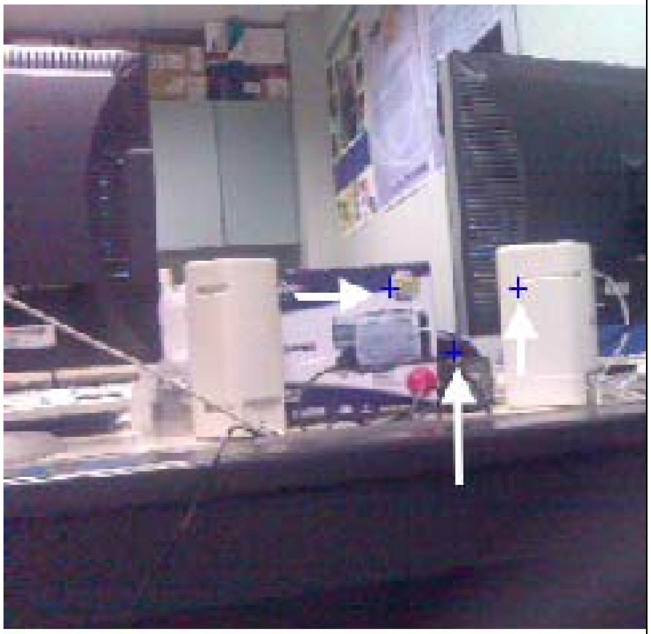
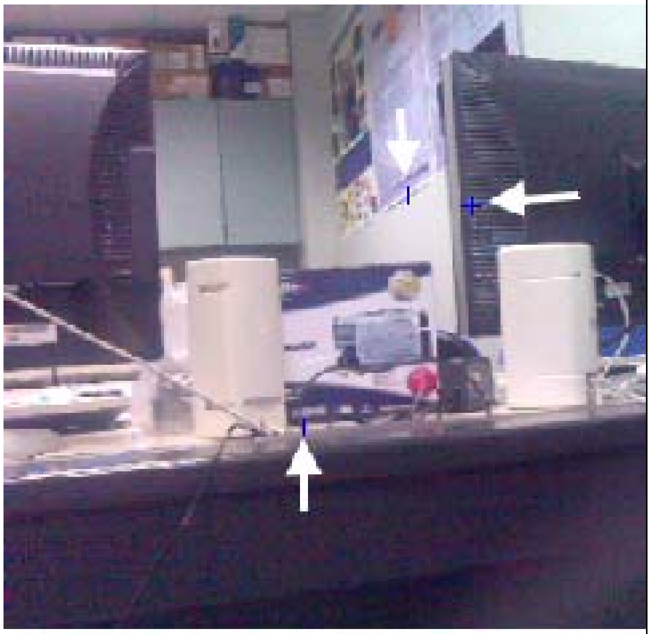
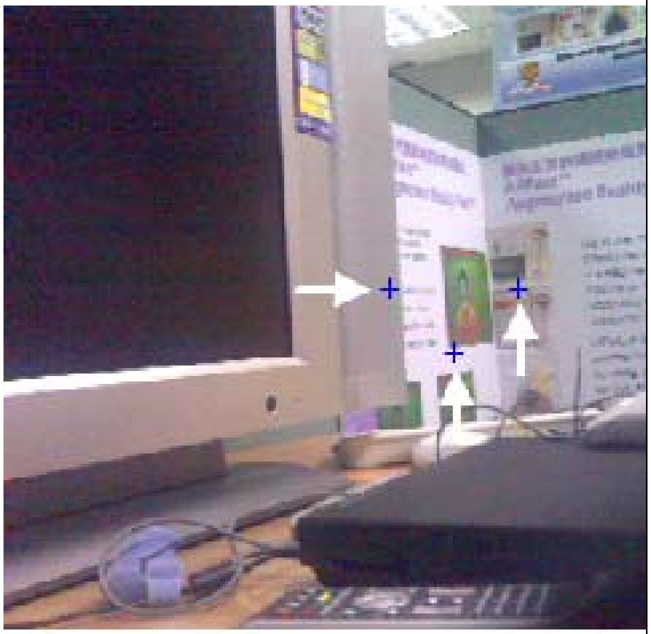
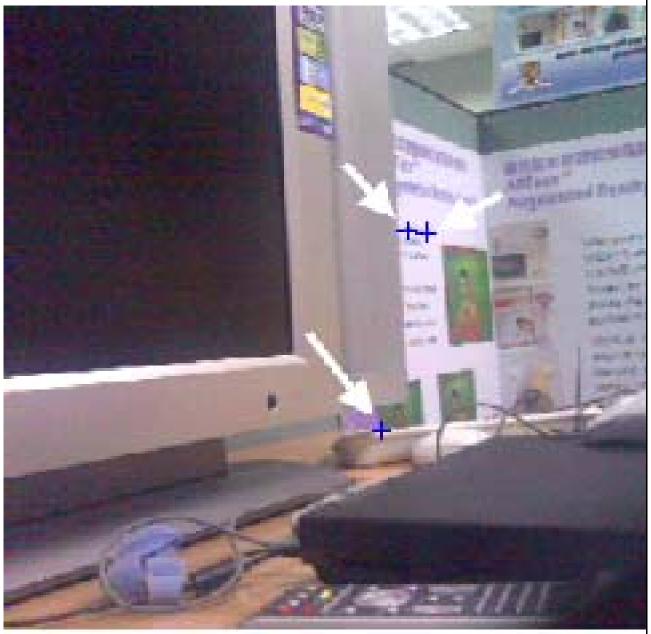
A foremost and fundamental question to be answered at this point would be, “What is a feature?” Features are sections of an image that are easily highlighted for the purpose of detection and tracking. Verestoy et al. [7] have defined features as local regions of interest. Features can be selected based on some measure of texture, edge sharpness, color and corners.

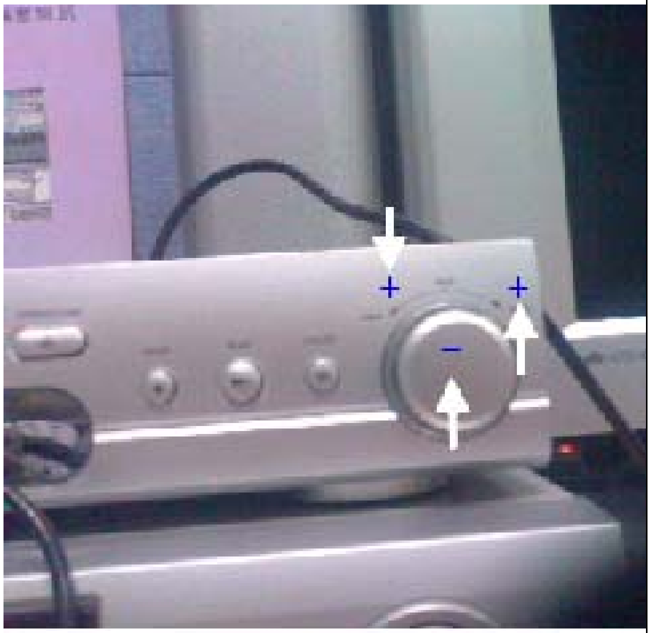
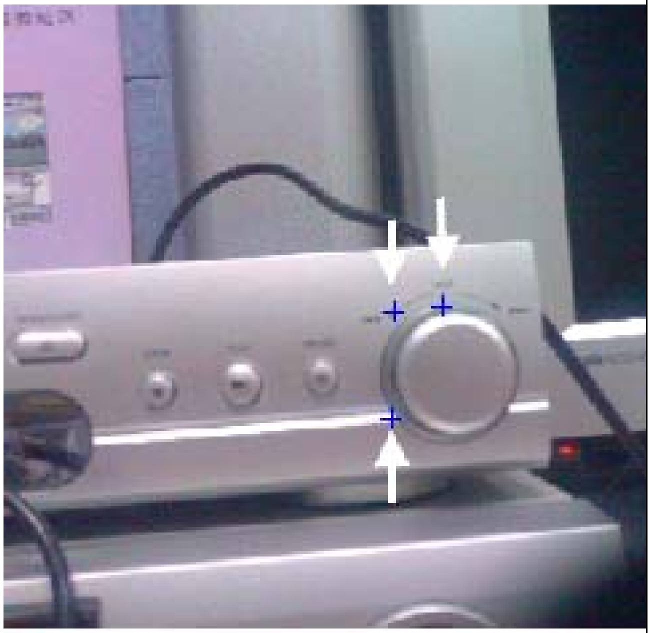
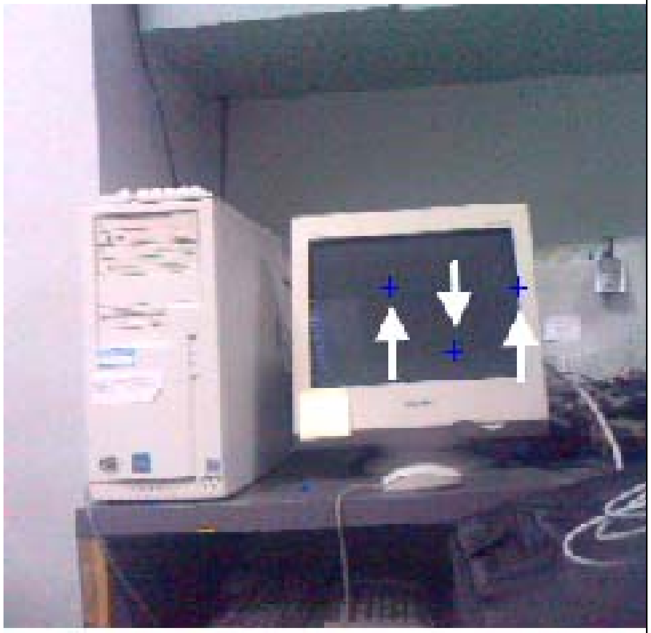
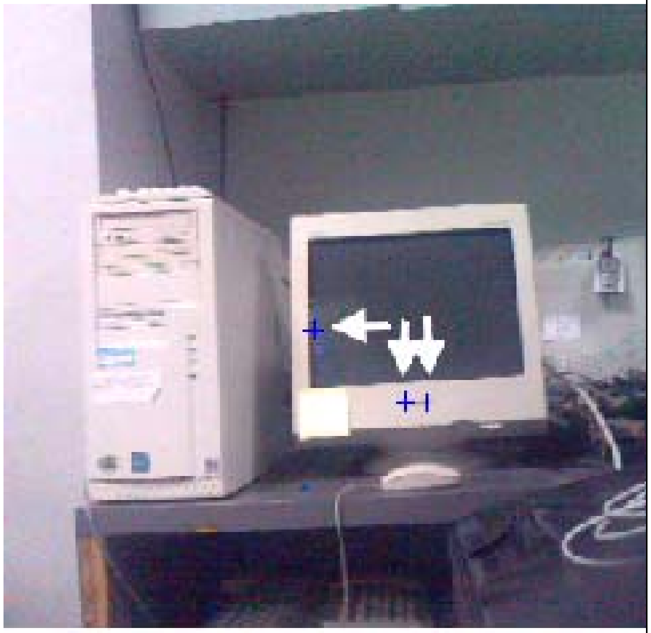
An example of a good feature would have a high contrast in relation to its immediate surroundings.

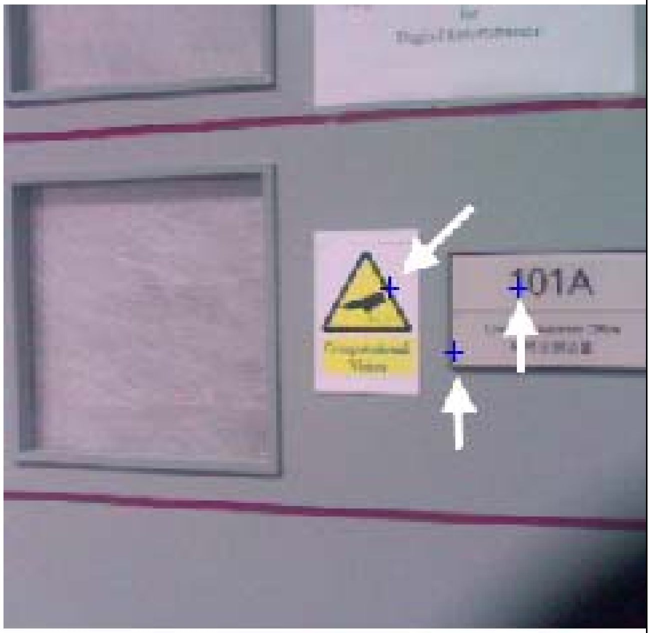
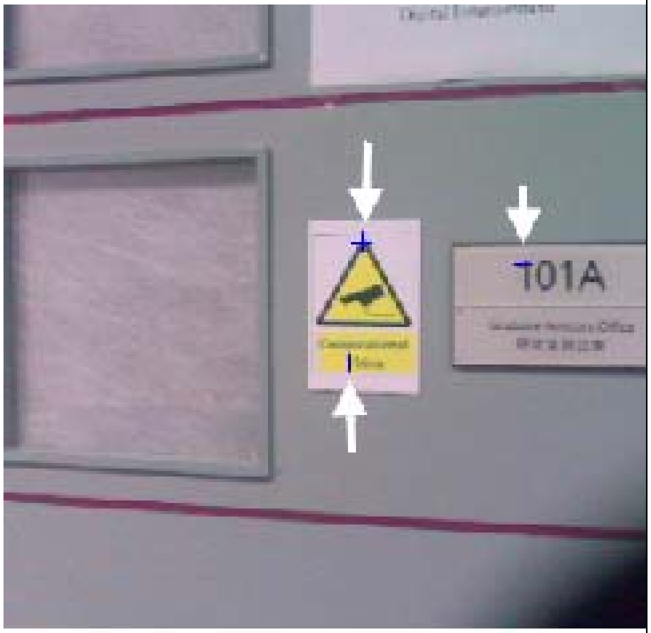
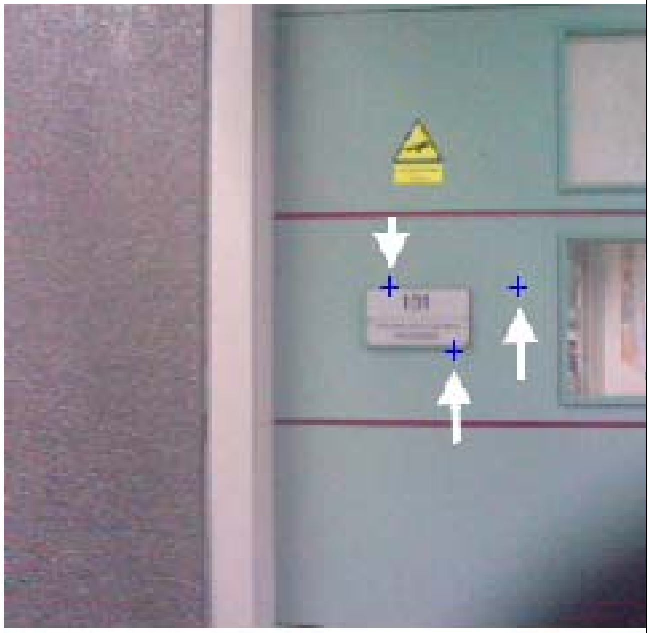
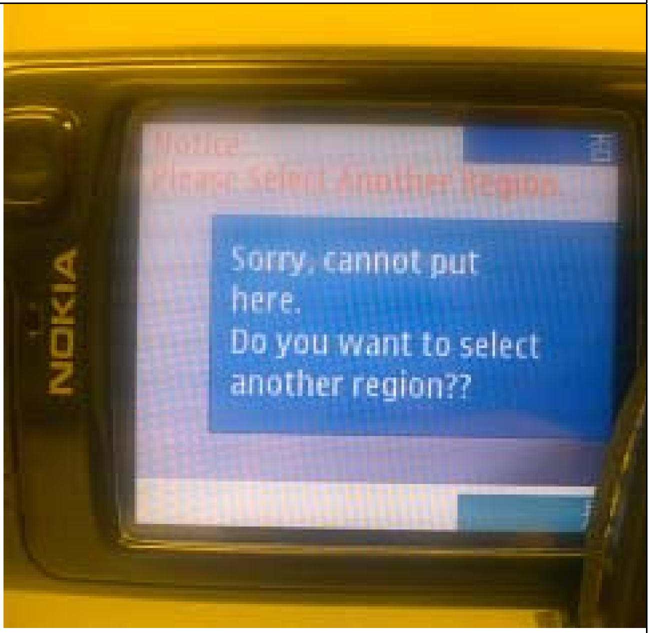


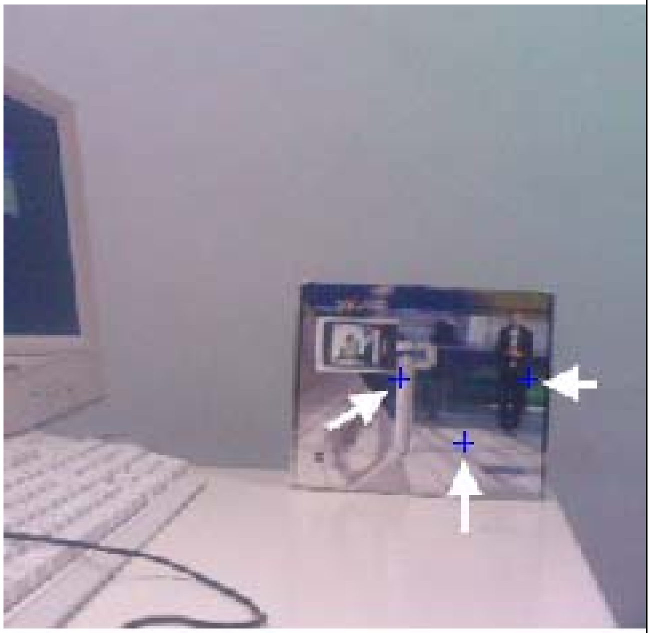
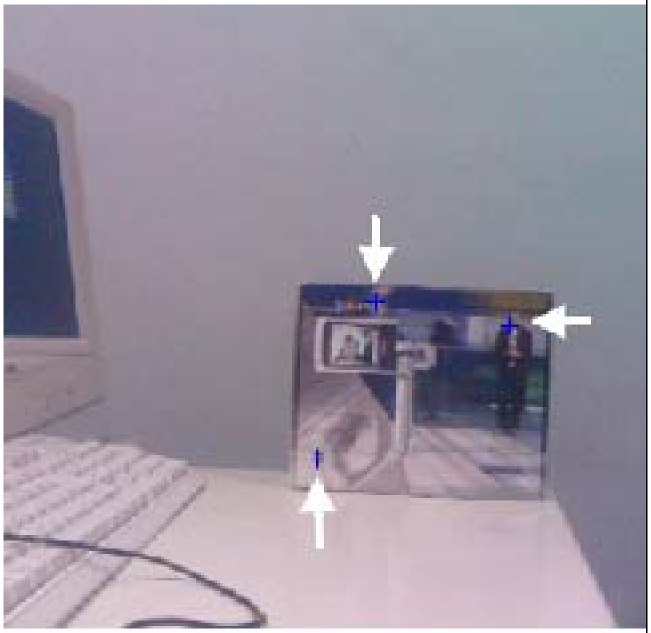
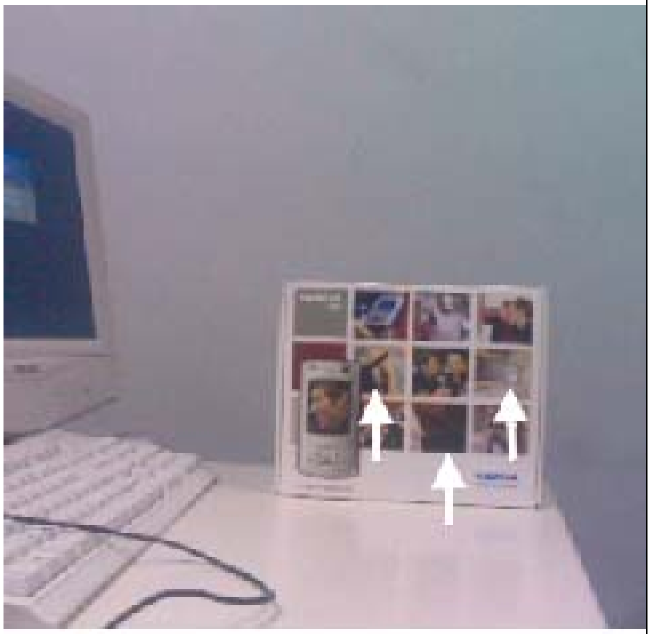
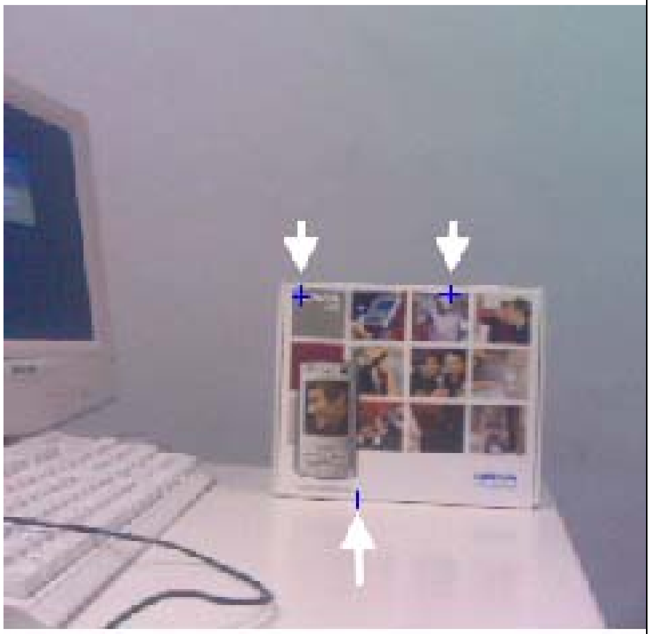
The above red marker shows the point which is a good feature because it has a high contrast to its immediate surroundings. In other words, it has a high intensity different.

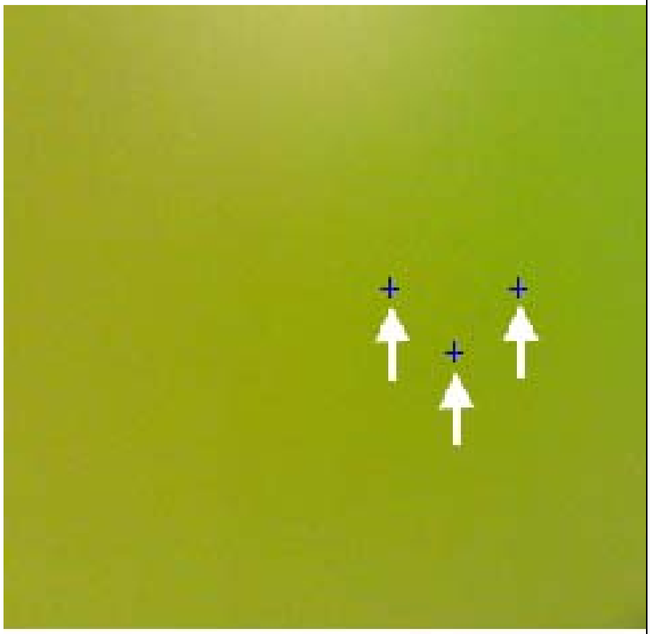
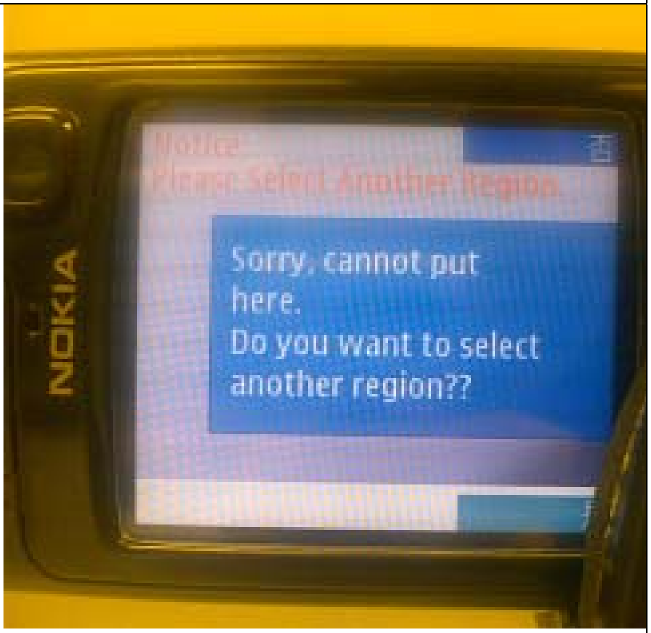
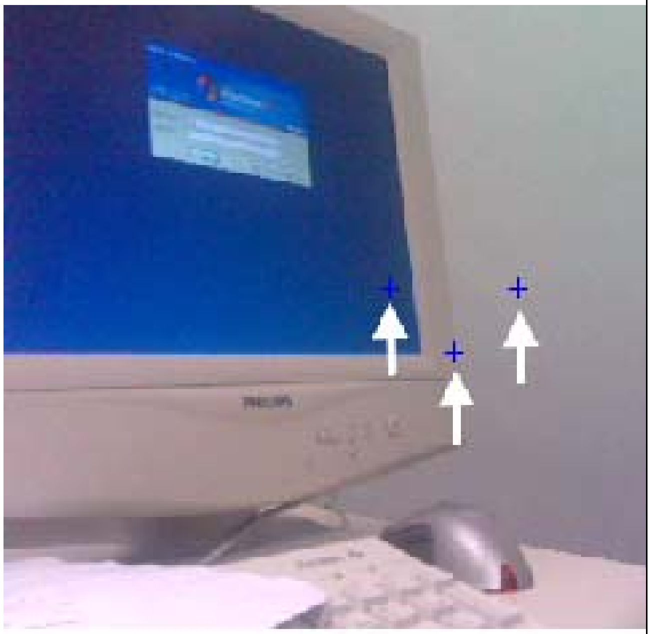
AI.2 Pictures under Normal Lighting Condition

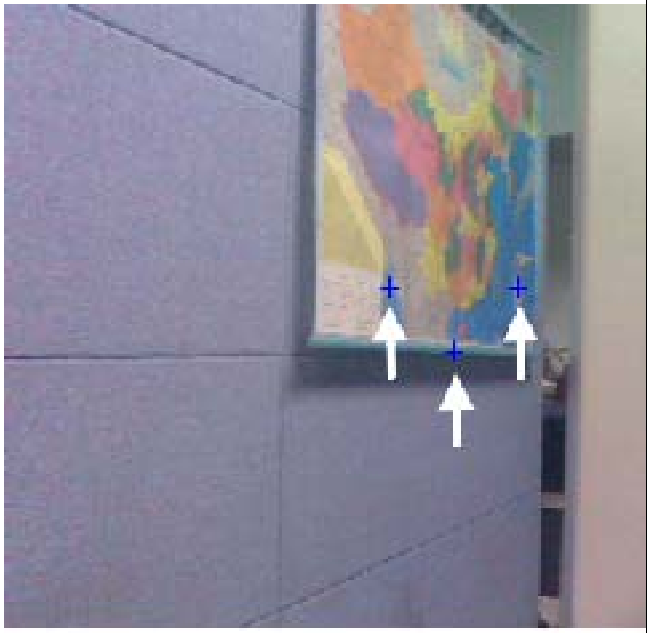
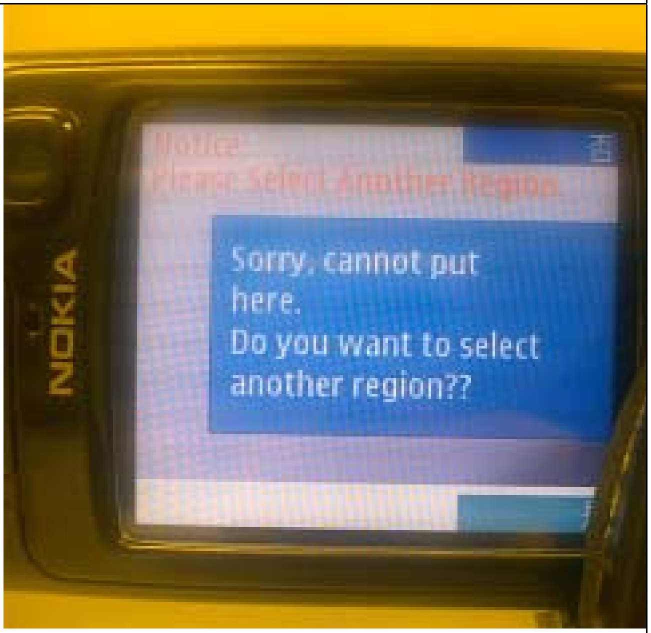
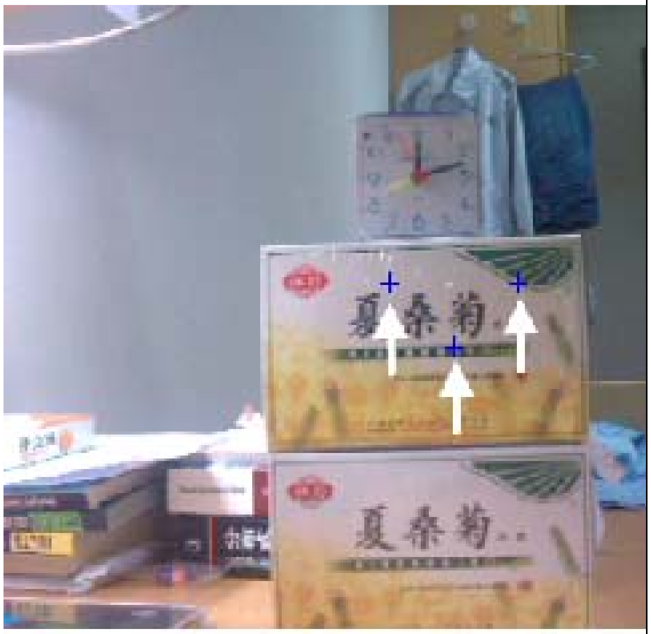
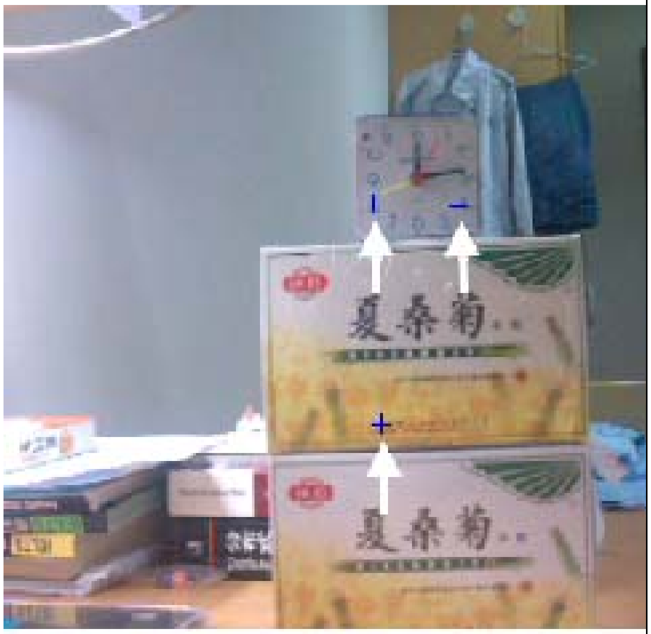
Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	
	

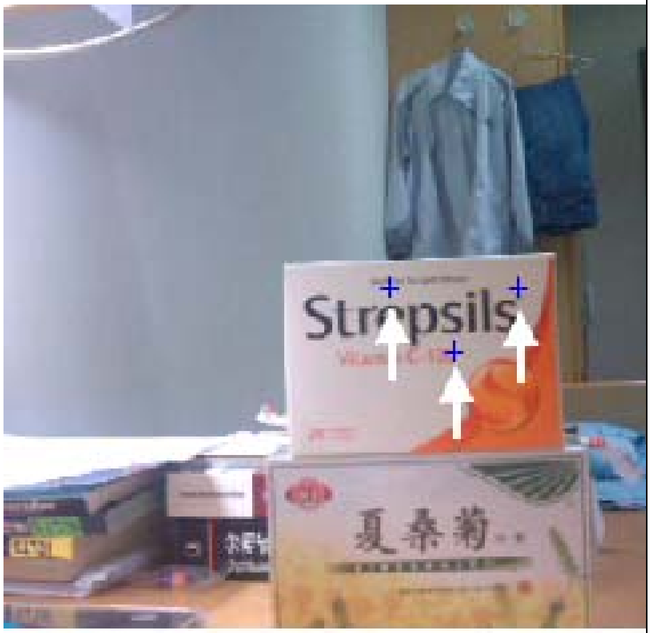
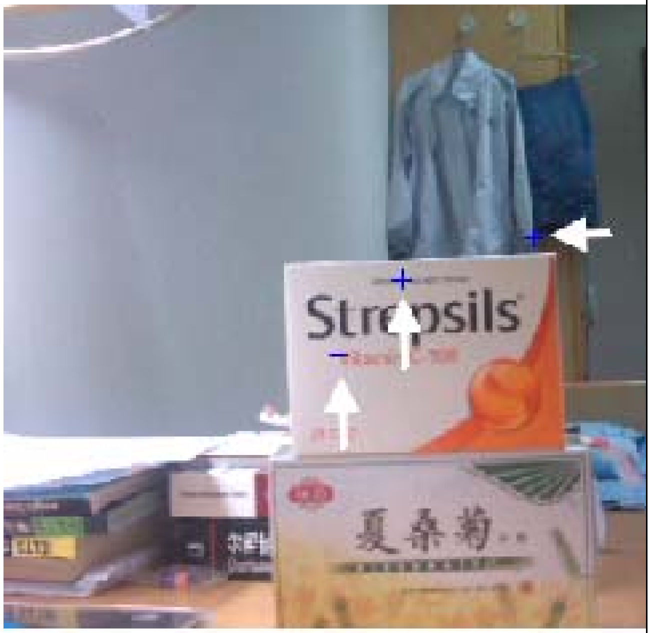
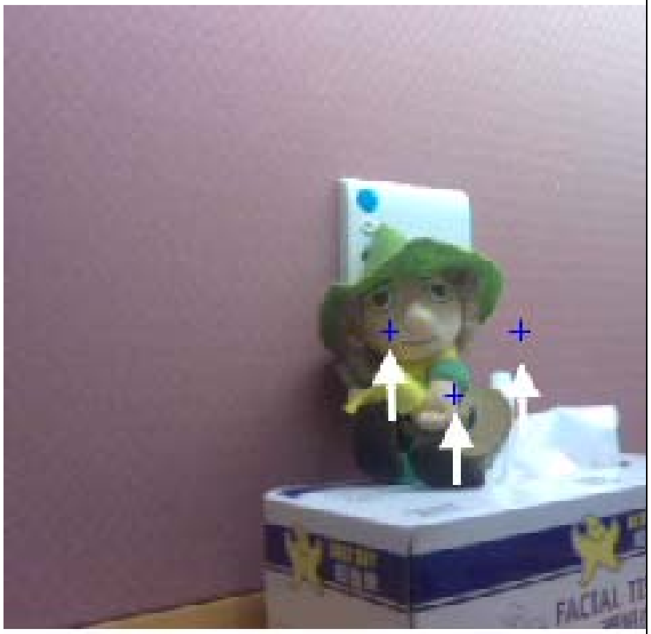
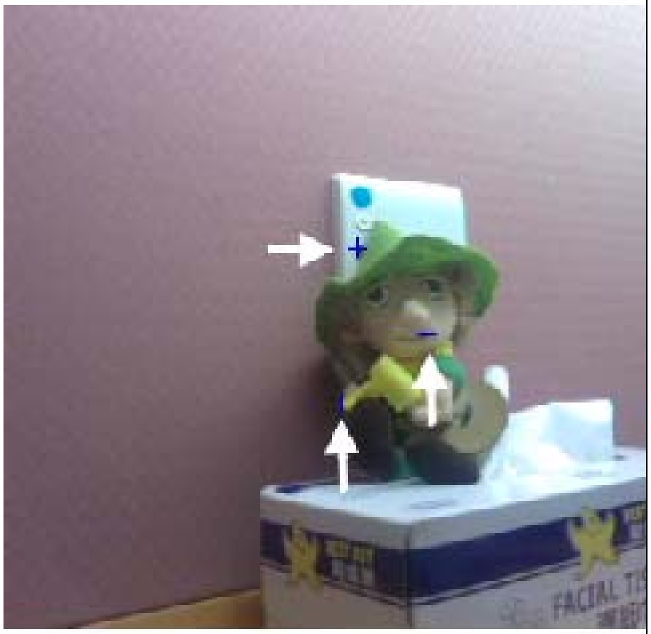
Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	
	

Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	
	 Notice Please Select Another Region Sorry, cannot put here. Do you want to select another region??

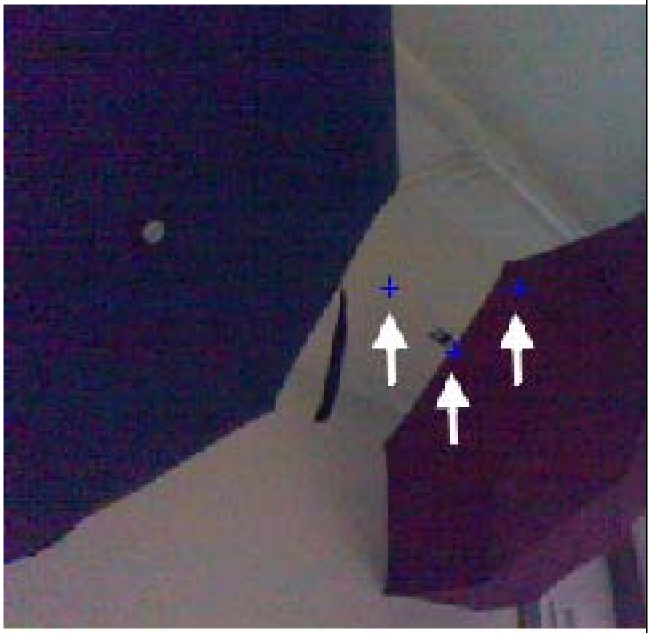
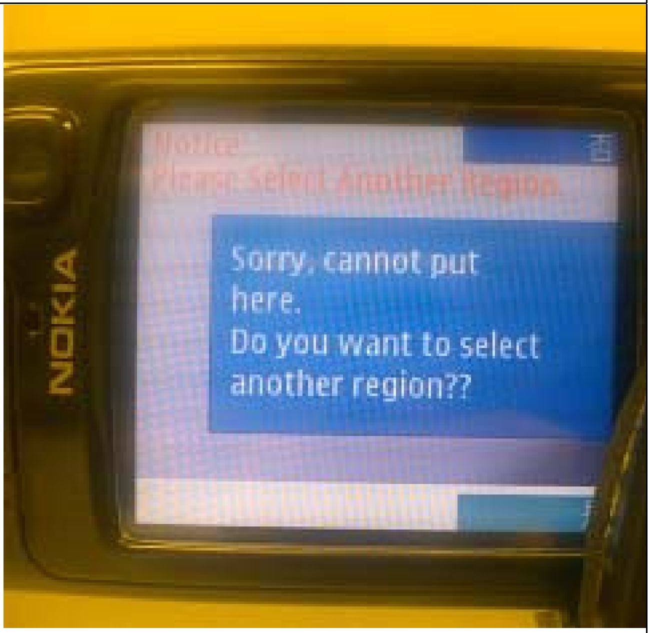
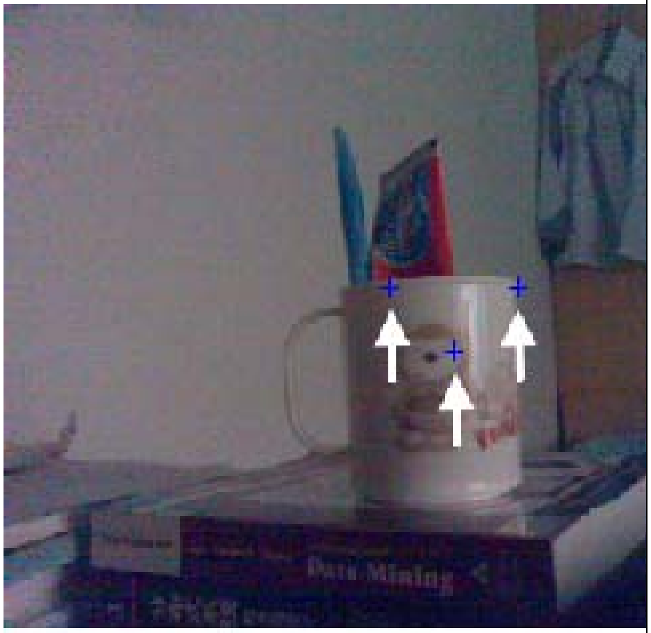
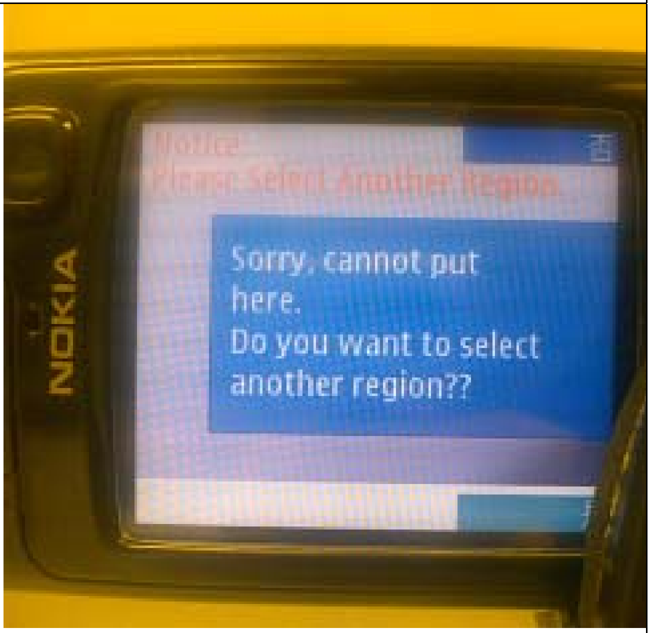
Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	
	

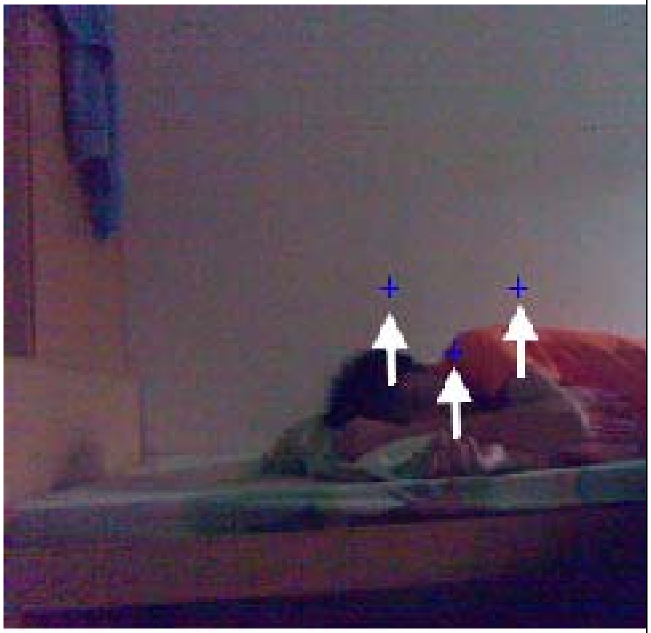
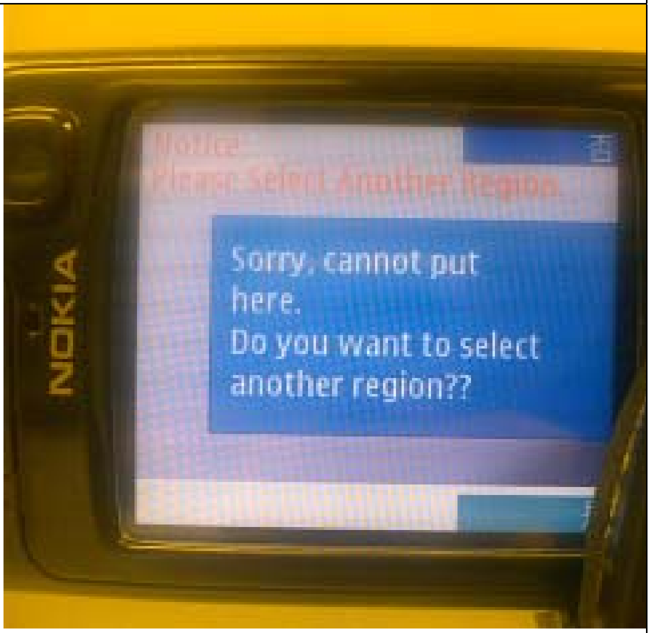
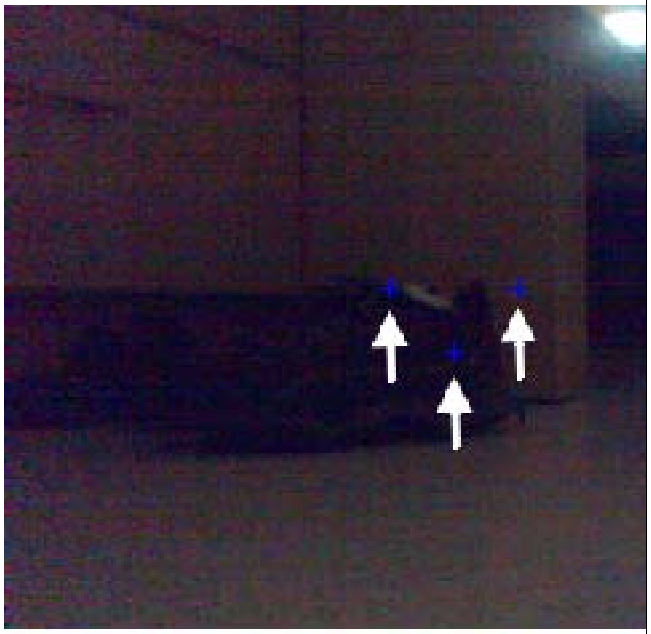
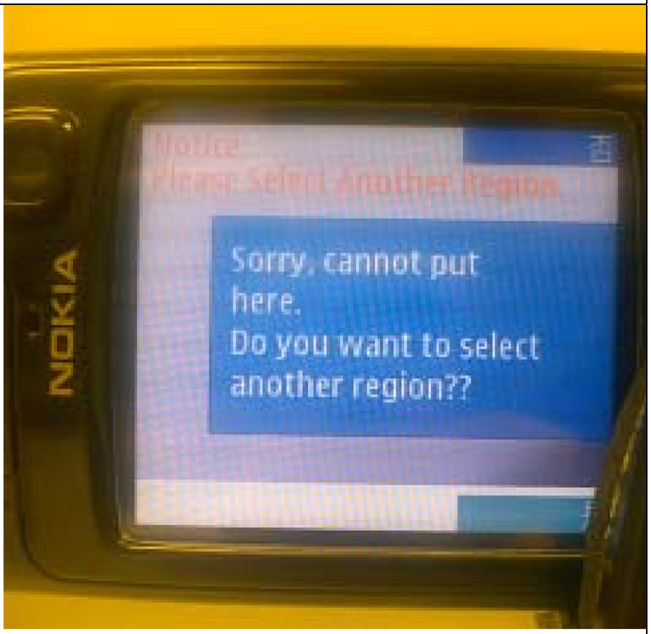
Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??
	Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??

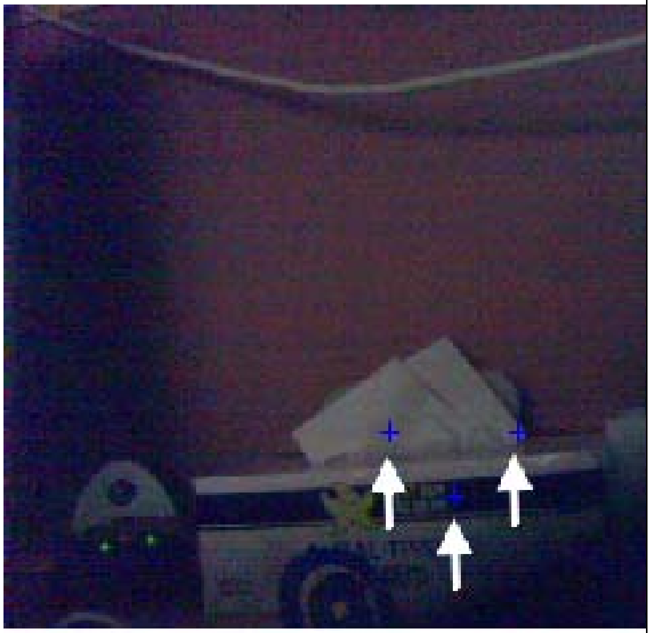
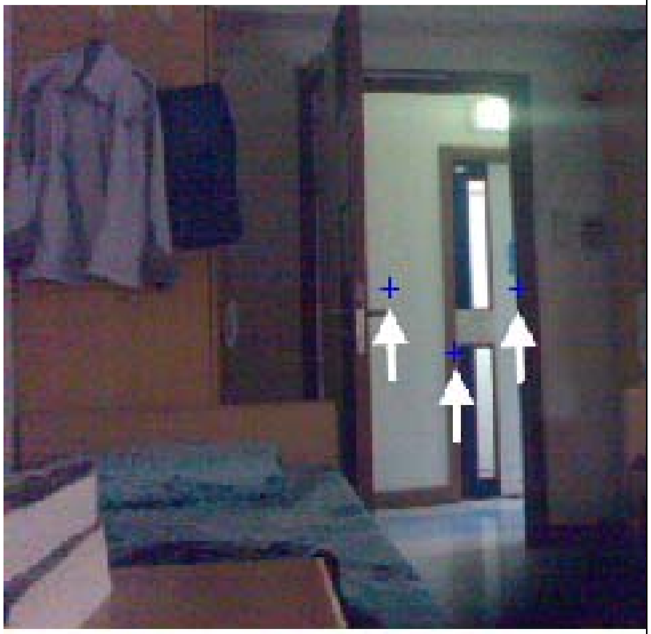
Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	 Notice Please Select Another Region Sorry, cannot put here. Do you want to select another region??
	

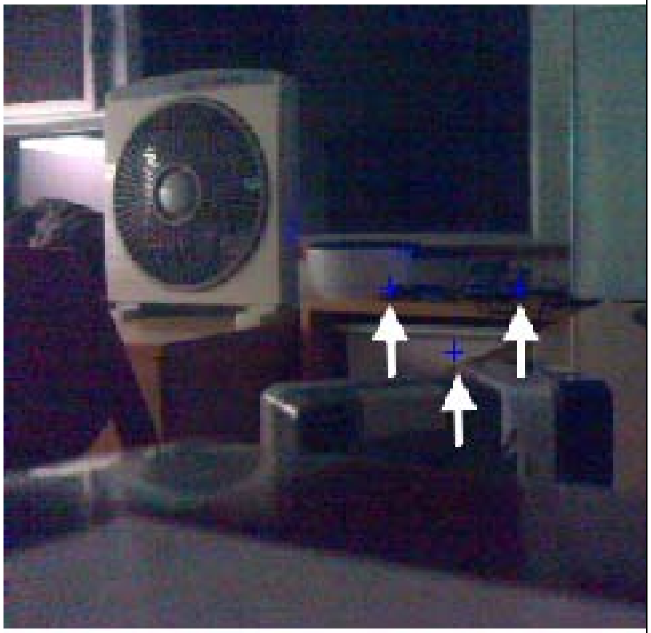
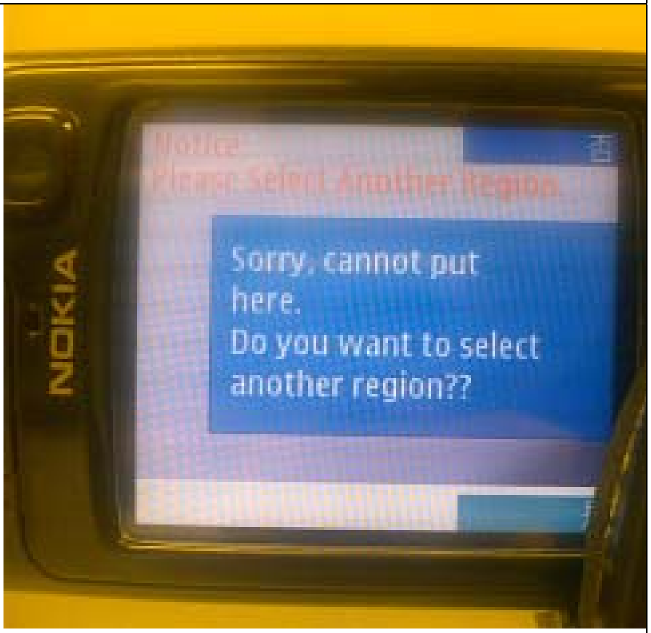
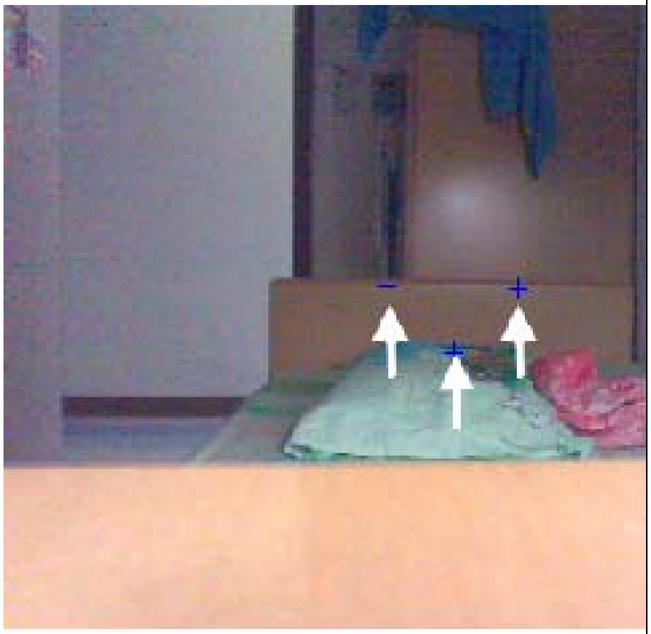
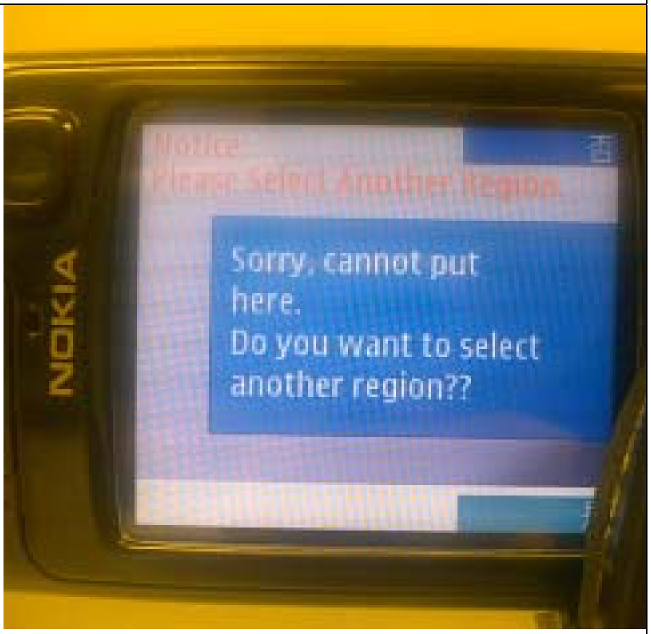
Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	
	

AI.3 Pictures under Insufficient Light Condition

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??

Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??

Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	Notice Please Select Another Region Sorry, cannot put here. Do you want to select another region??
	Notice Please Select Another Region Sorry, cannot put here. Do you want to select another region??

Feature Selection in MVOTE Engine	**Feature Selection in Fast Corner**
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??
	 Notice Please Select Another Region. Sorry, cannot put here. Do you want to select another region??

AI.4 Analysis

Why Feature Selection?

Using block matching algorithm for the motion tracking is a major component in our program. One may ask about which block should be chosen in the video frame for block matching. To answer this question, we must know very well our objective in order to acquire a better and more accurate result.

1. Features should be selected as descriptive as possible
2. The chosen block should facilitate the block matching algorithm and increase the accuracy of the algorithm
3. Feature extraction should be robust enough

In order to facilitate the objective, corner detection is used as feature selection in motion tacking. Formally, a corner can be defined as the intersection of two edges. A corner can also be defined as a point for which there are two dominant and different edge directions in a local neighborhood of the point.

An interest point is a point in an image which has a well-defined position and can be robustly detected. This means that an interest point can be a corner. There are many corner detection algorithms, for instance, Moravec Corner Detection Algorithm, Harris Corner Detection Algorithm, etc.

To strike a balance between having fast computational speed and reasonably accurate feature selection, former FYP team members (LYU0404) who developed the mVOTE Motion Tacking Engine Feature Selection function used Laplacian mask to calculate the intensity different between the current block with its neighbors for feature selection. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection originally.

- Gray level discontinuity → large output
- Flat background → zero output

The Laplacian L(x,y) of an image having pixel intensity values L(x,y) is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution mask that can approximate the second derivatives in the definition of the Laplacian. Three commonly used small masks are shown below:

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

The former FYP team would divide a frame into small rectangular blocks. Sum all the pixels value for each block, denoted as L_{xy}, and store it in a 2D array (Intensity of the block). After that, they calculate the variance of each block which represents the complexity of the block. Apply Laplacian Mask for the 2D array. Finally, they select the block which has the largest L_{xy} and large variance as feature block

For instance, if we apply the following Laplacian Mask to the given image, we would obtain the following result:

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacian Mask

14	25	68	60	66
16	67	20	16	95
4	29	8	21	99
68	62	66	127	113
120	33	37	121	67
2	65	61	109	60

Given Image

Output Image

	-227			

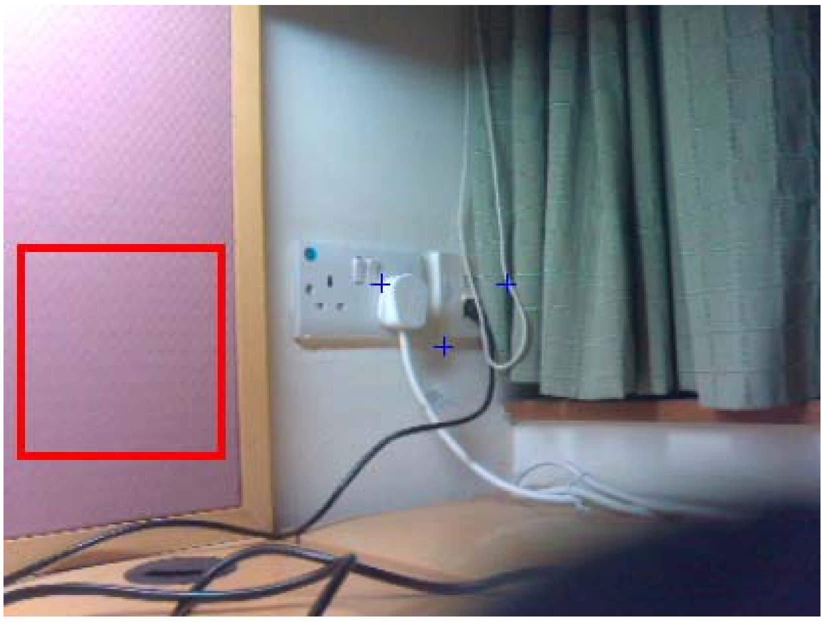
-1 x 68 + -1 x 62 + -1 x 66 + -1 x 120 + 8 x 33 + -1 x 37 + -1 x 2 + -1 x 65 + -1 x 61

= -227

Since the Fast Corner Algorithm is introduced in the previous chapter, this Appendix I would not go through it again.

What the result tells?

From the above experiment result, it is observed that under normal lighting condition, both algorithm works fine. With careful examination, we can see that the existing Motion Tracking Engine's Feature Selection does not work at optimum. Occasionally, this algorithm would select some flat region as a feature. By "flat region", we mean the region where the intensity level is similar or the same. The selected area in the following diagram shows the flat region.



In contrast, the Fast Corner algorithm works better than the existing algorithm. It is because in most of the case it can find a "corner" at which point, there exists a large intensity change. In some case, Fast Corner did not find any corner and thus the error message was displayed.

On the other hand, under the insufficient lighting condition, there are many noises in the photos. The existing algorithm finds some features but those so called "features" are not the good one. It is hard to imagine using those features to carry out motion tracking would produce a fair result. For Fast Corner algorithm, it finds nothing under insufficient lighting condition. It is good news for us, because under such environment, users can hardly play the game well due to the performance degradation of motion tracking as noises in the photo increase. Fast Corner said "No" to such unfavorable environment.

In our project, we only focus on the existing feature selection algorithm of Motion Tracking and the Fast Corner but not other corner detection algorithm, like Harris Corner. We ignore the use of Harris Corner for corner detection (for selecting feature) because this corner detection involved the use of computing determent of matrix which is a pretty high involvement for the Mobile Phones especially for those which equip no floating point unit.

In short, we prefer to use Fast Corner as our feature selection algorithm. The reasons are:

1. Such corner detection involved no intensive arithmetic computation.
2. It built a decision tree to determine if a certain point is likely to be a real corner.
3. It is easy to implement and make modification
4. It produces better result than the existing one
5. It rejects noisy photos

Appendix II

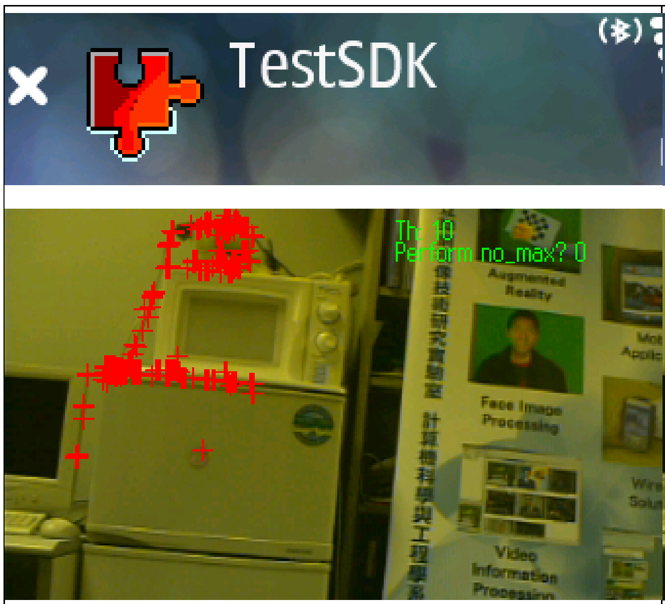
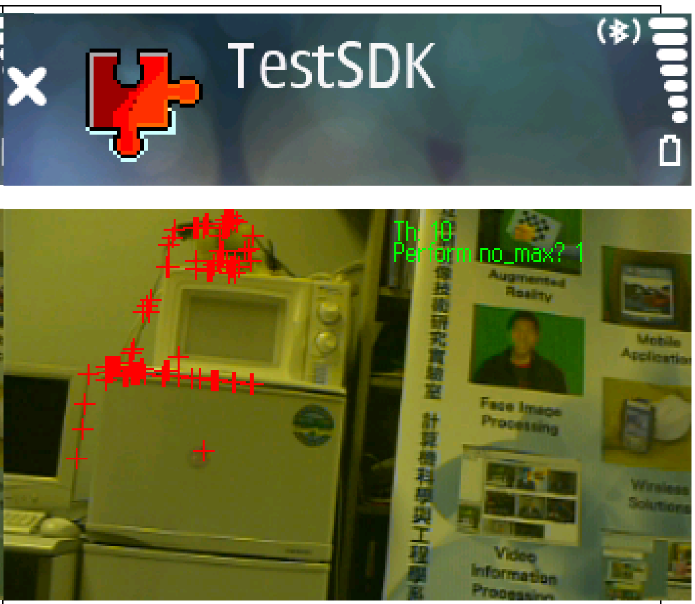
Parameter Adjustment for Fast Corner Algorithm (REVIEW)

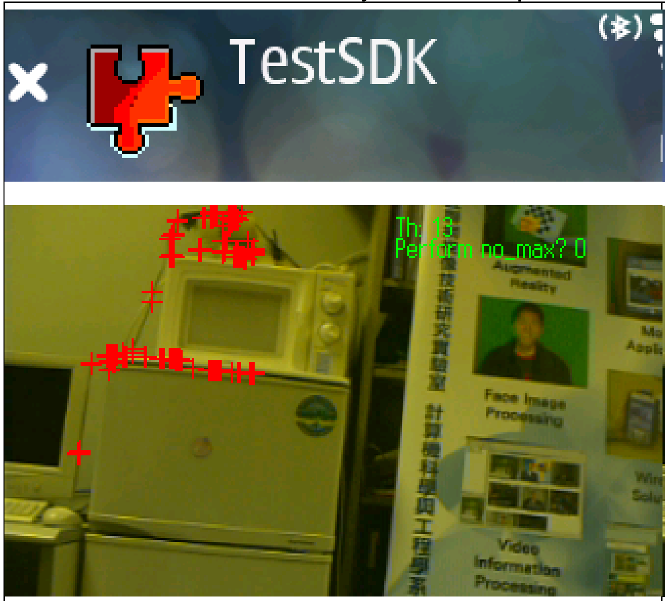
Appendix II describes the experiment conducted in semester 1 for testing how the corner detection algorithm would be affected by adjusting its parameter – barrier. This experiment also shows the different result obtained when running Fast Corner Algorithm with accessory function non-maximal suppression or without it. The use of this parameter and the non-maximal suppression is mentioned in the previous chapter.

- **AII.1 Experiment Result**
- **AII.2 Analysis**

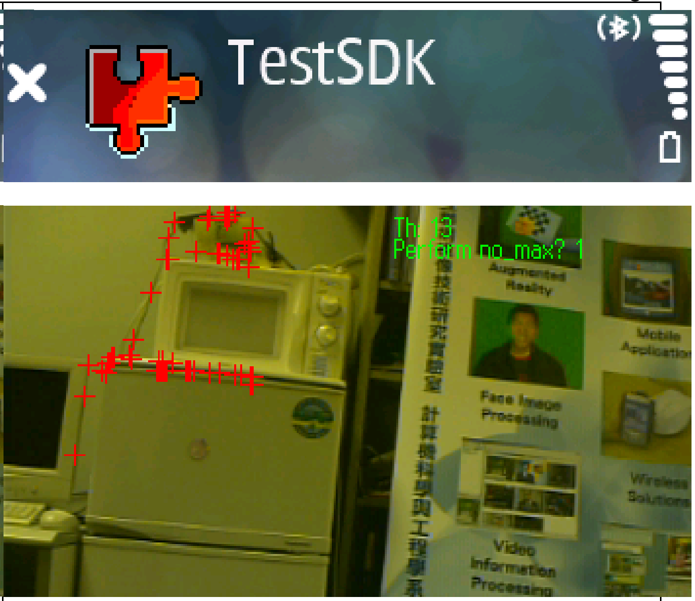
AII.1 Experiment Result

This section would show how different value of the parameter affects the number of corner found by the Fast Corner Detection Algorithm. The corner(s) would be marked as "+" in Red Color in the graphs shown below. The parameter value would be displayed in Green Color in form of "Th: z" where z is the parameter value used by Fast Corner Algorithm. The default parameter value is 20 in the original Fast Corner Algorithm.

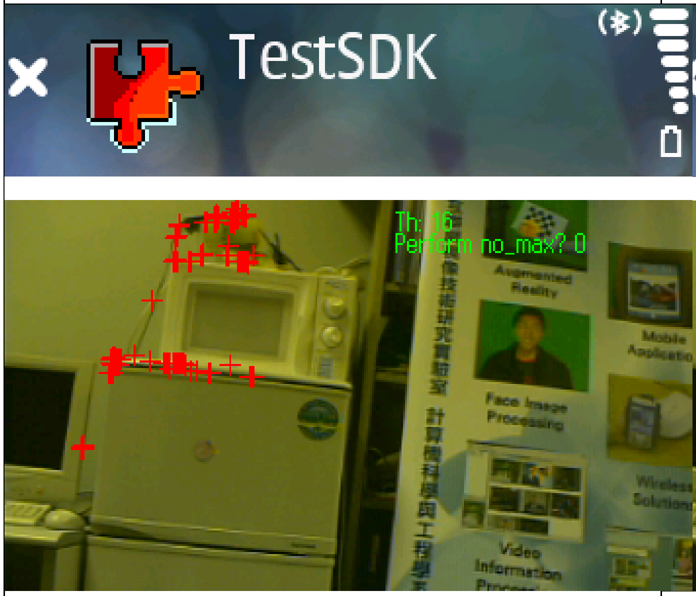
 Th: 10 Perform no_max? 0	 Th: 10 Perform no_max? 1



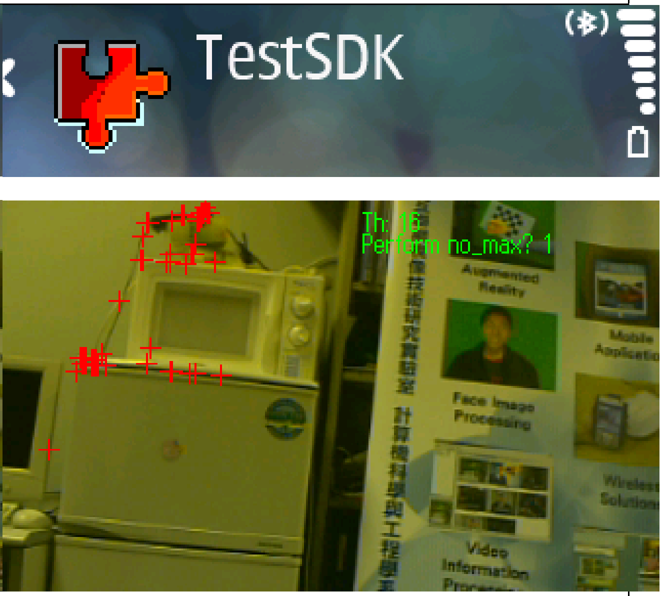
Th: 13
Perform no_max? 0



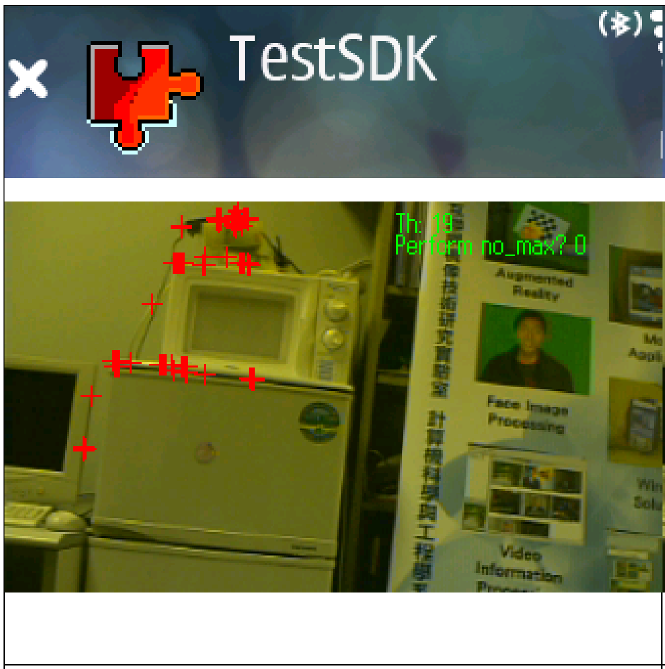
Th: 13
Perform no_max? 1

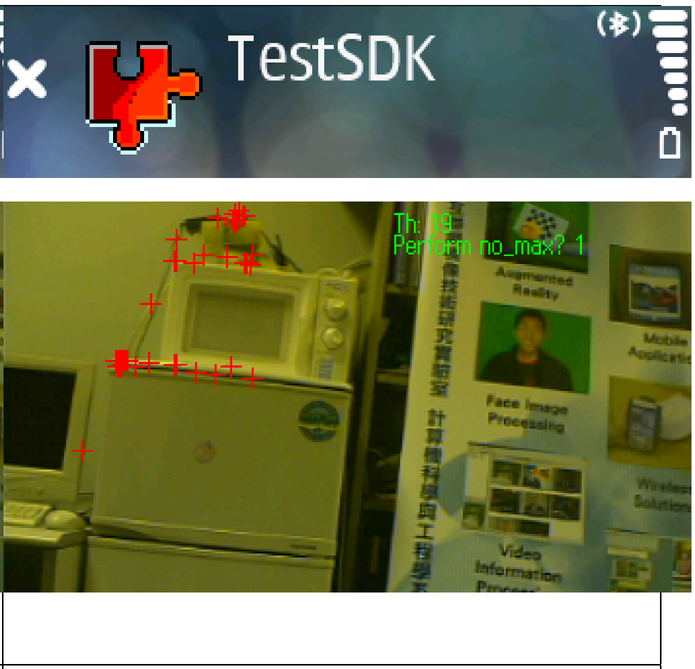


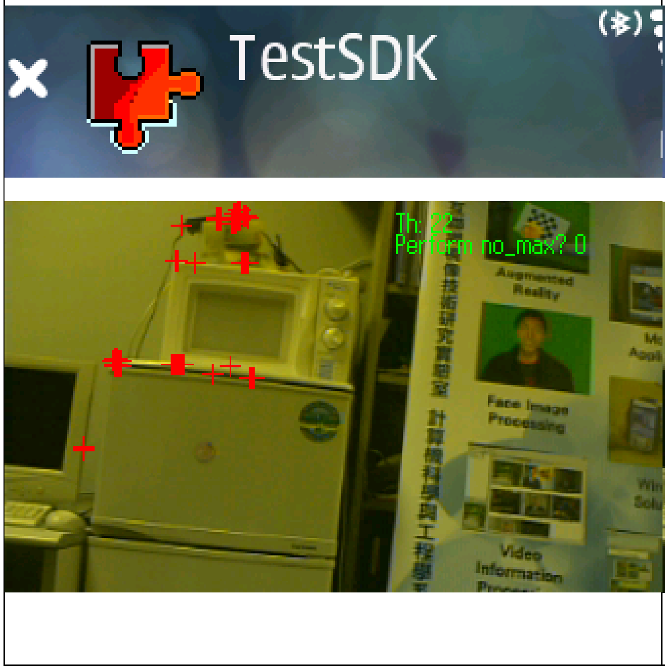
Th: 16
Perform no_max? 0

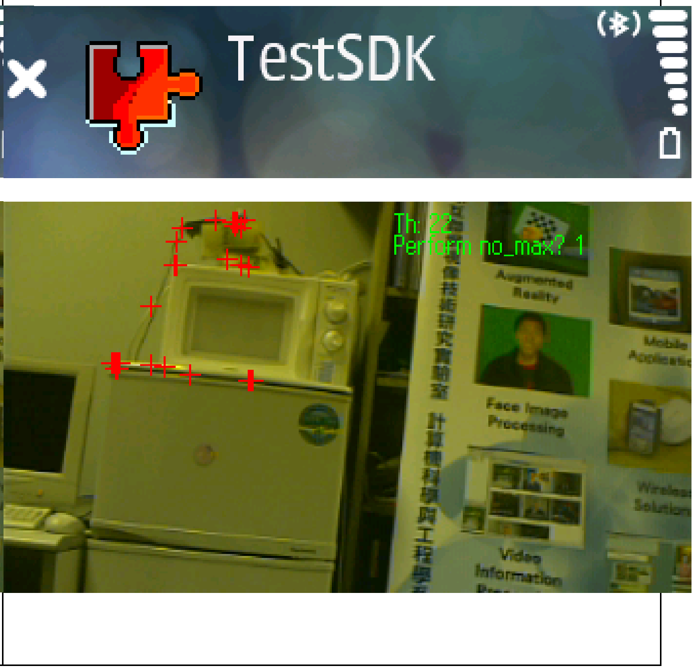


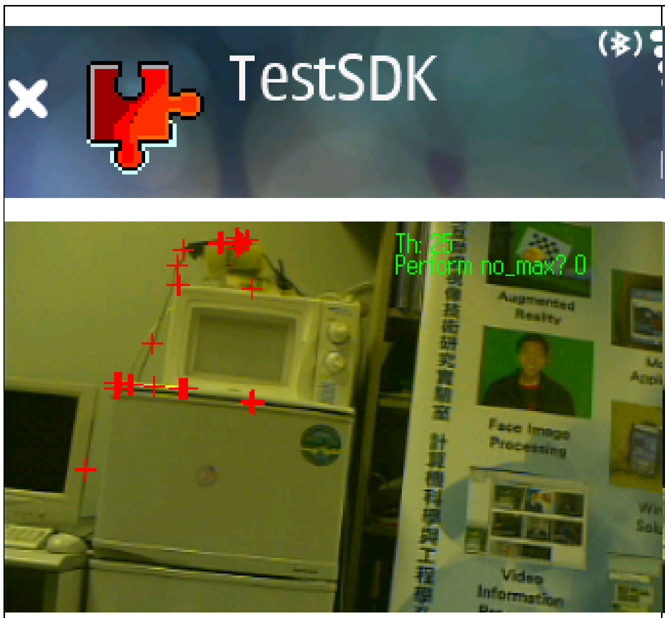
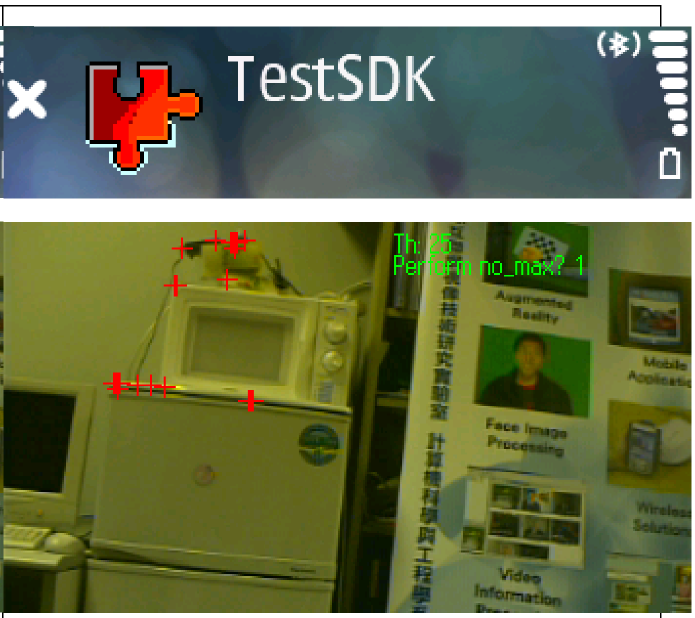
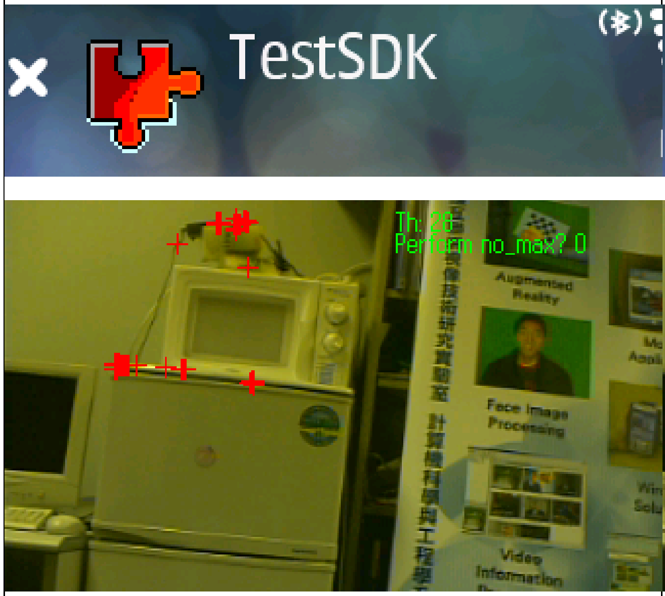
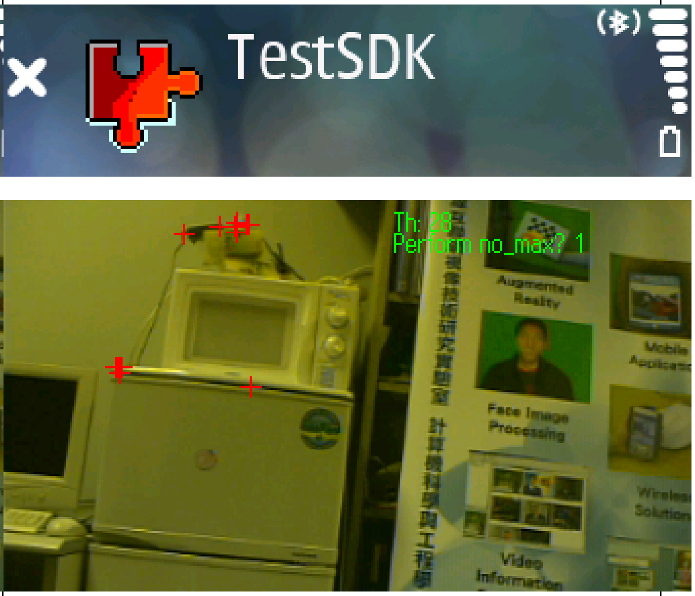
Th: 16
Perform no_max? 1

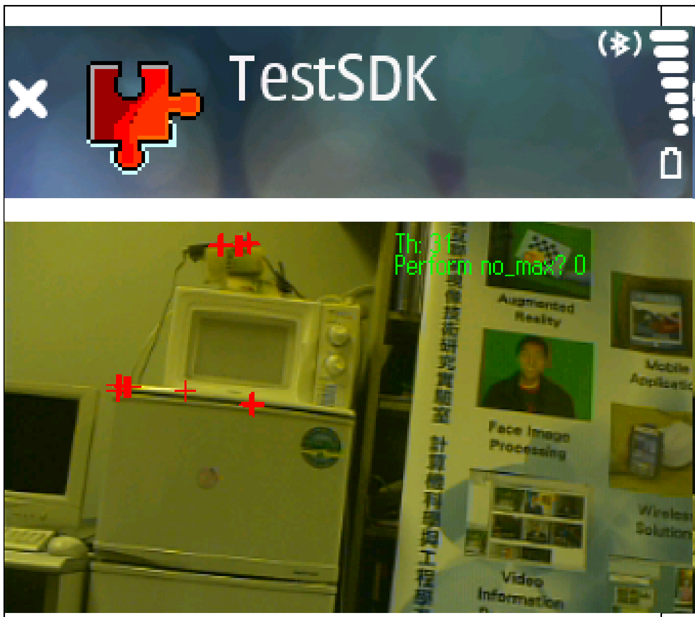
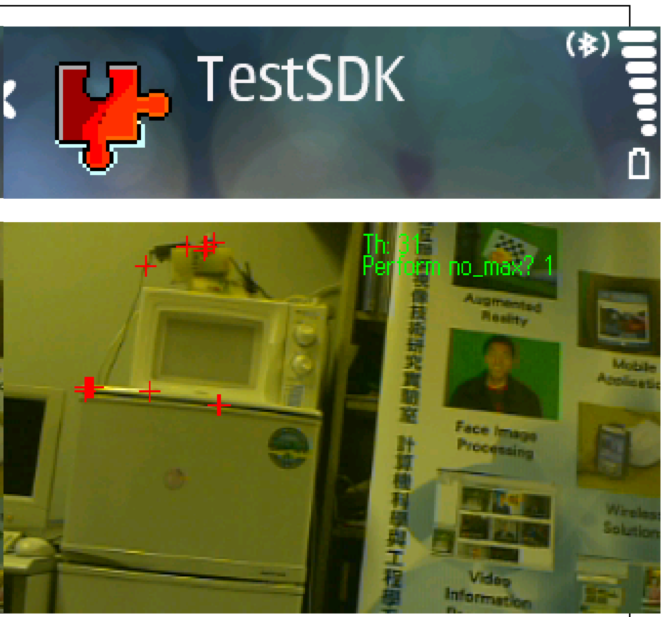
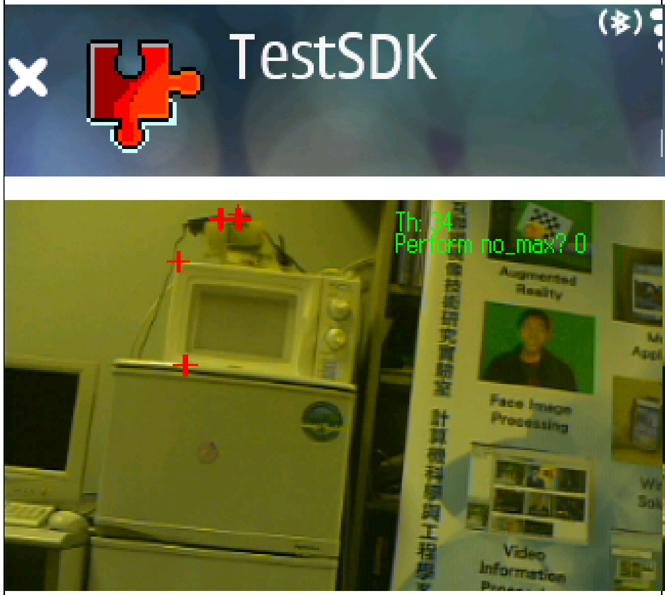
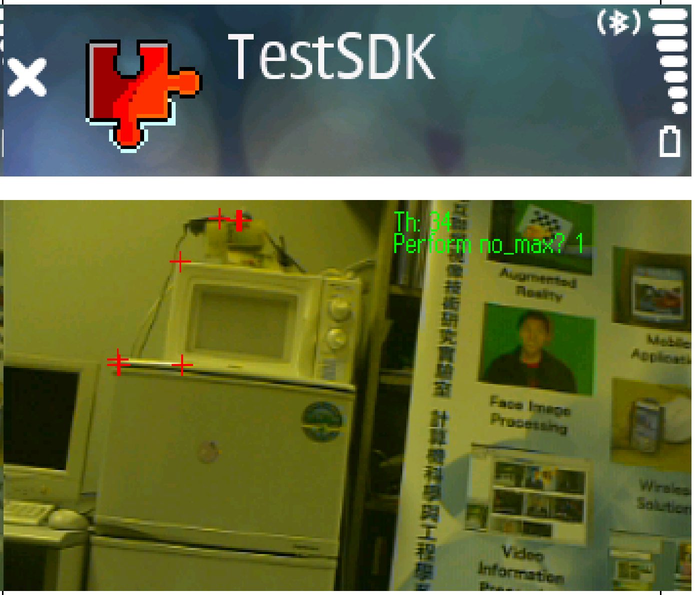

TestSDK
Th: 19
Perform no_max? 0

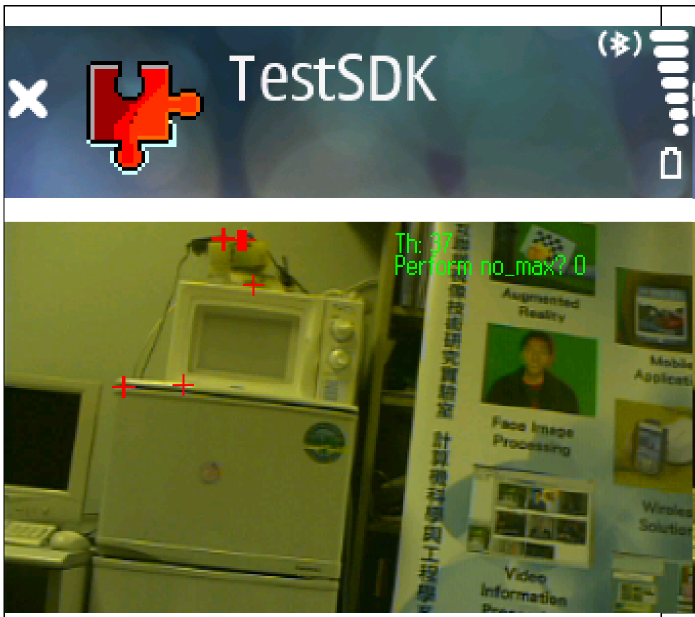
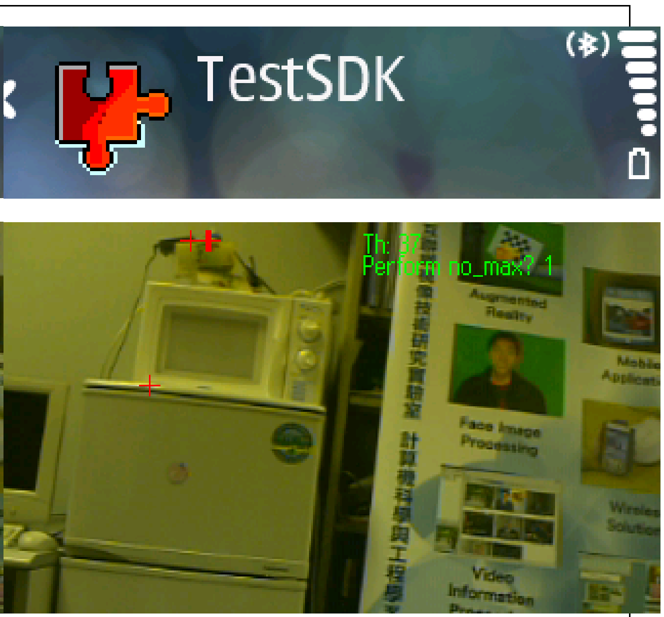
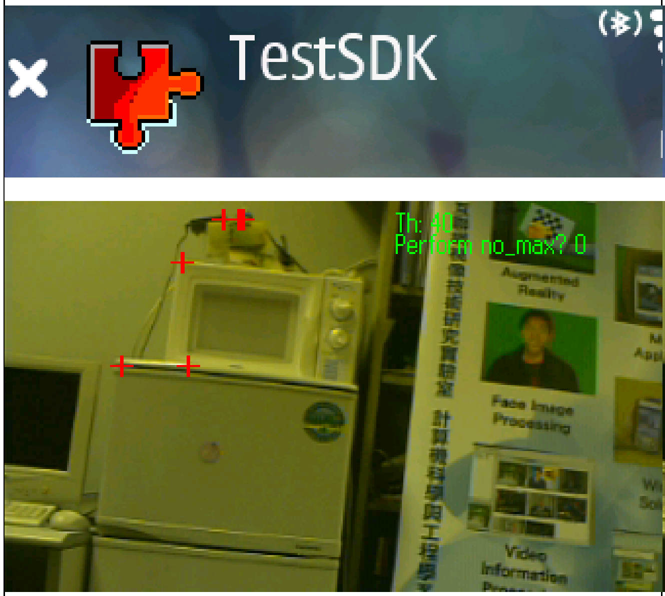
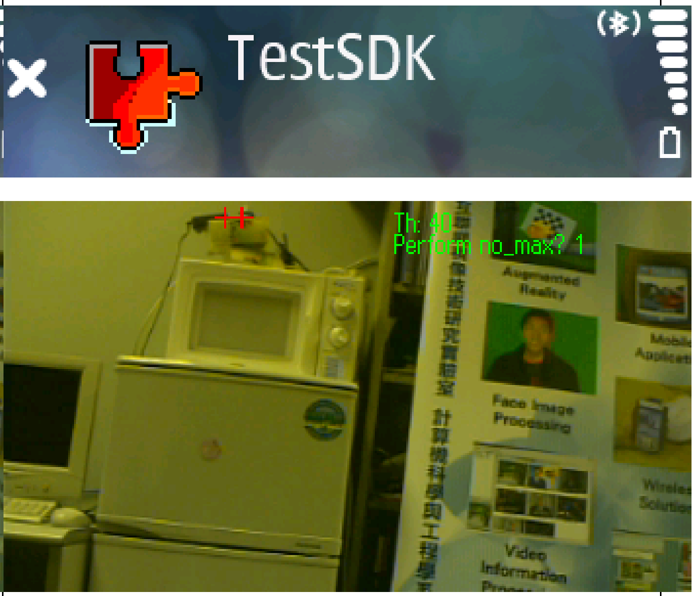

TestSDK
Th: 19
Perform no_max? 1


TestSDK
Th: 22
Perform no_max? 0


TestSDK
Th: 22
Perform no_max? 1

 Th: 25 Perform no_max? 0	 Th: 25 Perform no_max? 1
 Th: 28 Perform no_max? 0	 Th: 28 Perform no_max? 1

 Th: 31 Perform no_max? 0	 Th: 31 Perform no_max? 1
 Th: 34 Perform no_max? 0	 Th: 34 Perform no_max? 1

 TestSDK Th: 37 Perform no_max? 0	 TestSDK Th: 37 Perform no_max? 1
 TestSDK Th: 40 Perform no_max? 0	 TestSDK Th: 40 Perform no_max? 1

AII.2 Analysis

The above graphs show the different in term of corner selection versus value change in the parameter, barrier, in the algorithm. The results obtained on the left column are those using Fast Corner algorithm without the use of Non-maximal Suppression technique. While running Fast Corner algorithm with Non-maximal Suppression technique would produce results for the right column. In general, it is observed that the number of corner detected decrease with the increase of the parameter value. In other words, when the parameter value is high, the algorithm imposes stricter requirements for a certain pixel to become a corner. The quality of corner selection would be higher if we increase the parameter value. In addition, performing Non-maximal suppression would produce fewer corners than the one with no Non-maximal suppression; this has been explained in the previous chapter.

In some cases, as illustrated in the above graphs with parameter value 34 and 37, the number of corner detected using 37 as the parameter value is more than that obtained from value 34. To explain this, we believe it is due to our small mechanical vibration during the photo taking process and sometimes the lighting condition and noises also affect the performance but these conditions may not be obvious to human eyes.

If we set a higher value for the parameter, we may end up with having a higher probability that the users selected region where there is not enough features for game playing. If we set its value too low, we may obtain a set of poor quality corner which affect the accuracy when carrying out motion tracking. To strike a balance of it, we set the parameter value as 25. To make our program more robust, we decided to perform Non-maximal suppression.

Appendix III

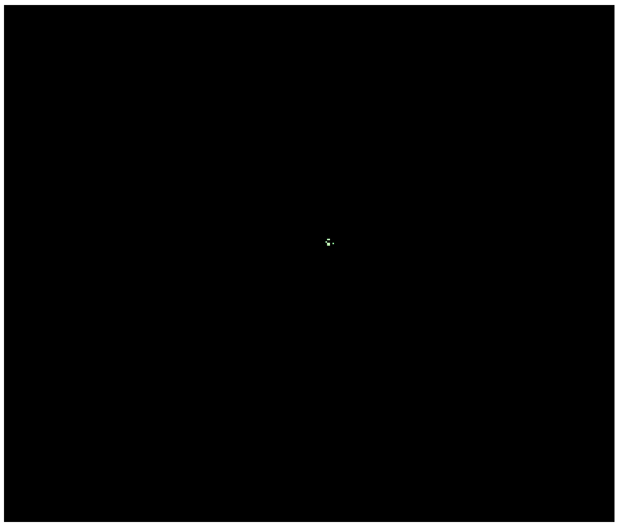
LED Recognition Result and Discussion

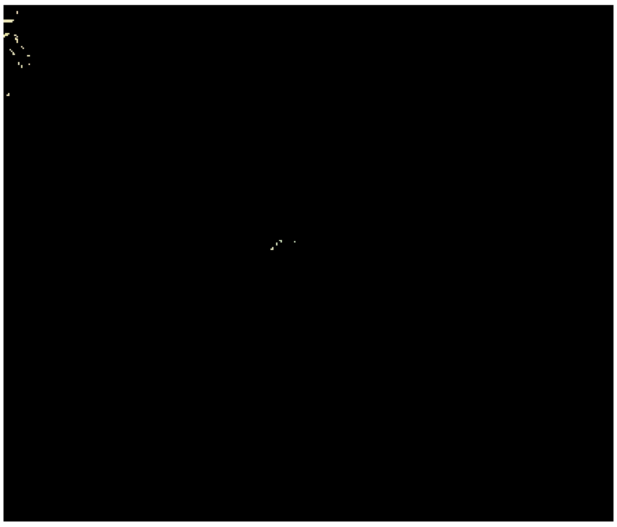
Appendix III describes experimental results of feature filter and LED recognition and a brief discussion of it.

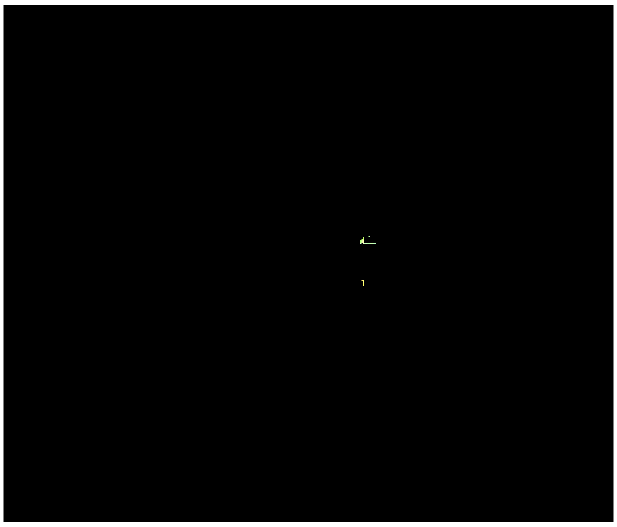
- **AIII.1 Experimental Result of Feature Filter**
- **AIII.2 Analysis on the Feature Filter**
- **AIII.3 Experimental Result of LED Recognition**
- **AIII.4 Analysis on LED Recognition**

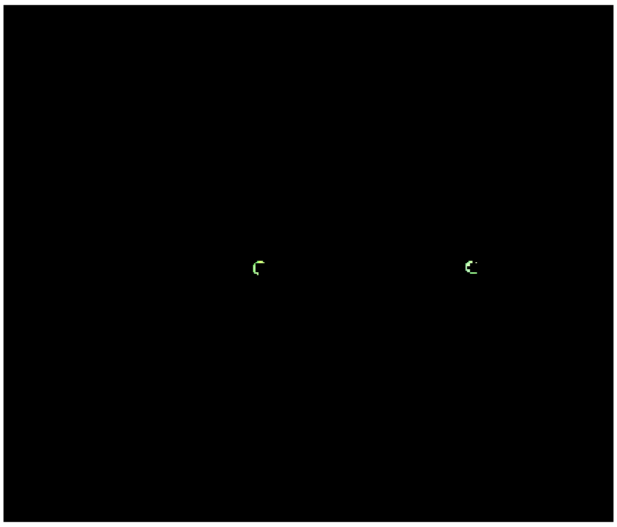
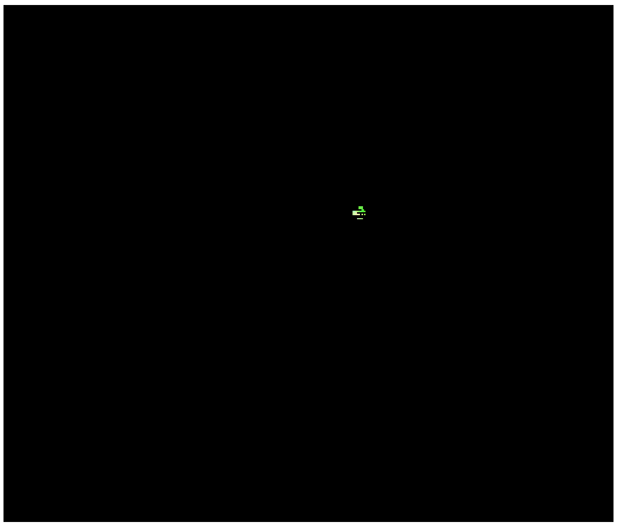
AIII.1 Experimental Result of Feature Filter

Experimental Result of Feature Filter for green LED

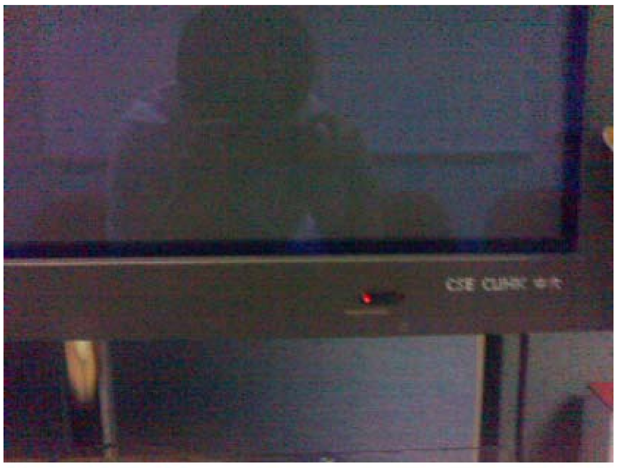
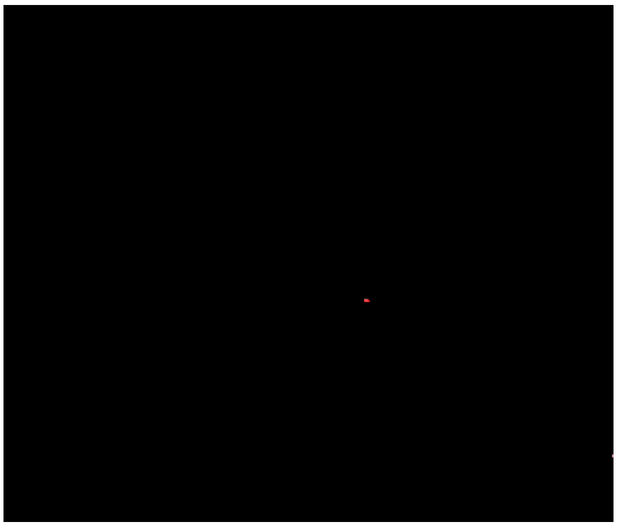
	

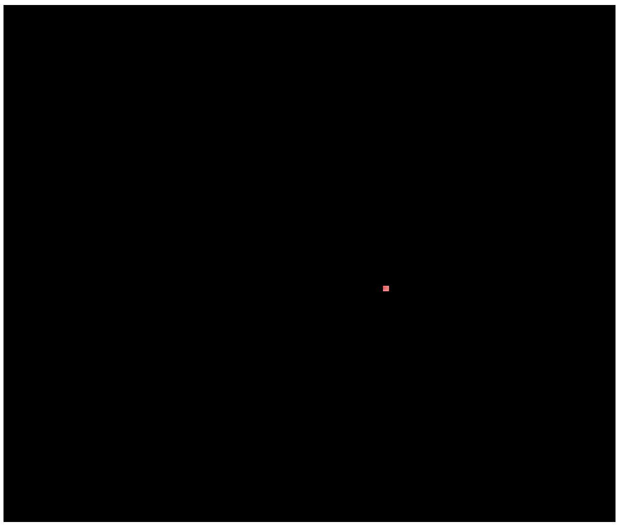
	

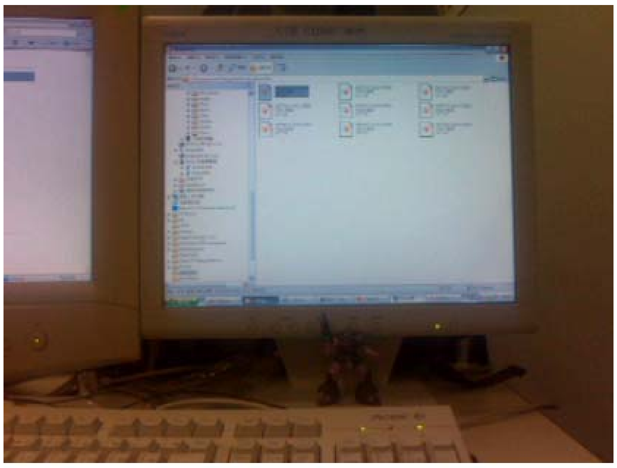
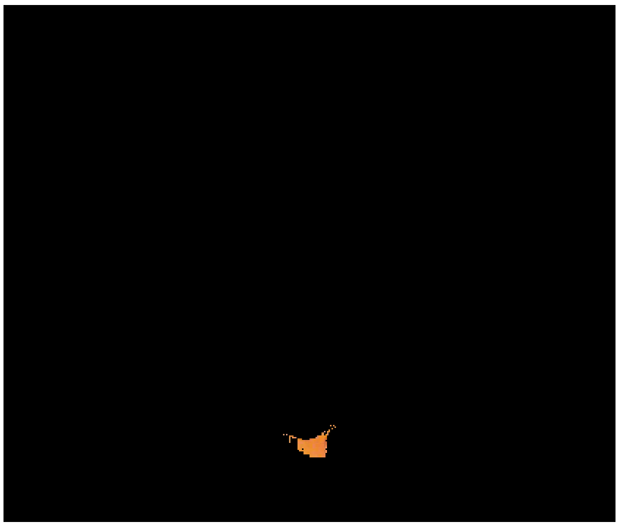
	

Experimental Result of Feature Filter for red LED

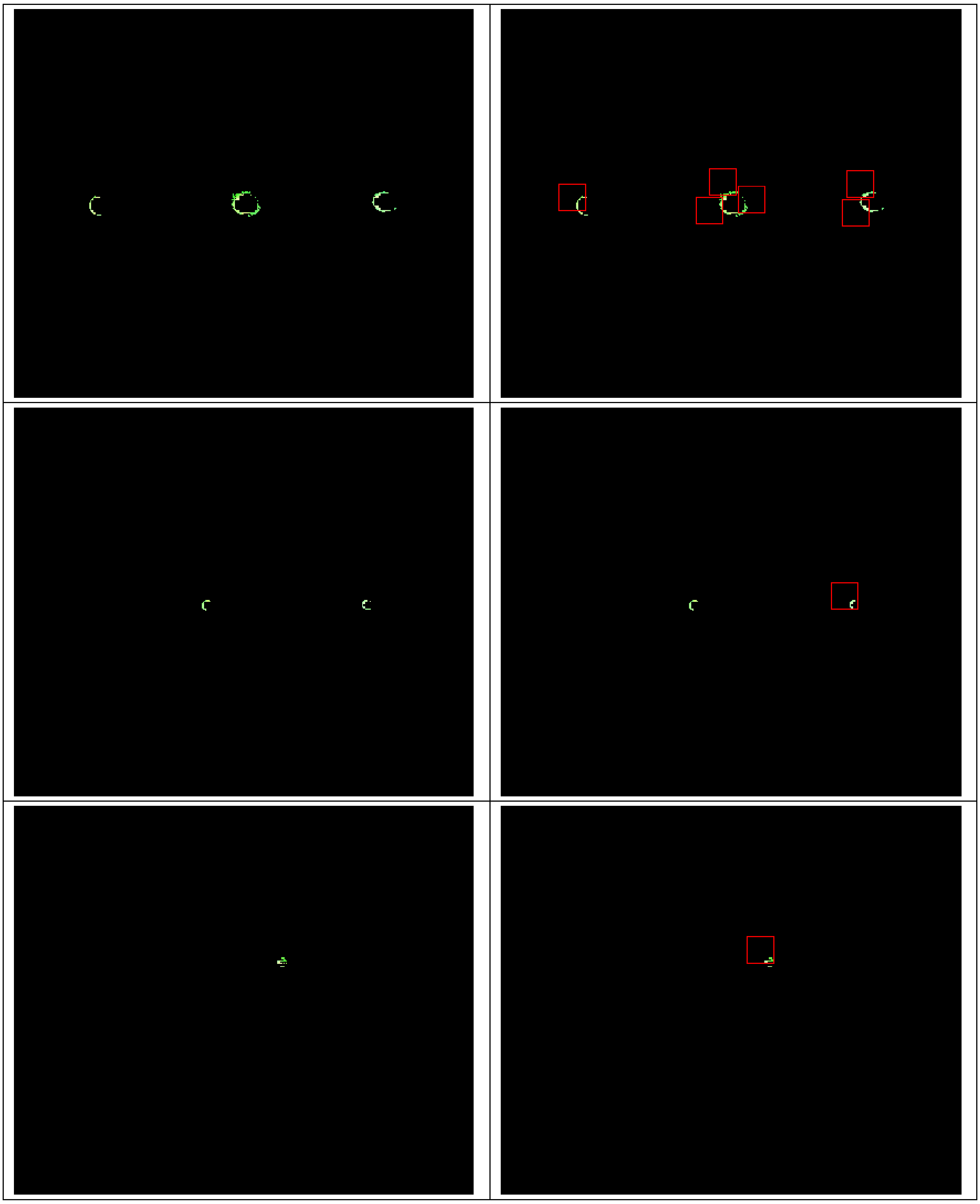
	
	

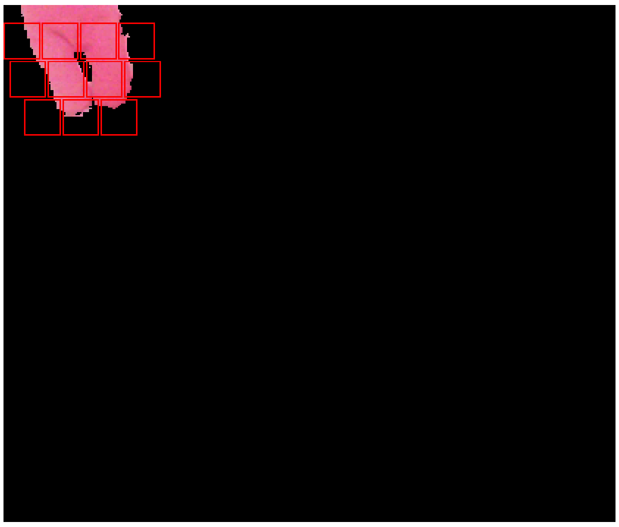
AIII.2 Analysis on the Feature Filters

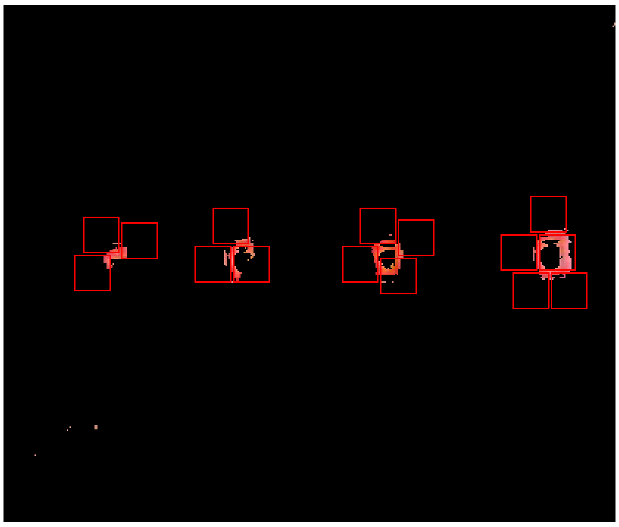
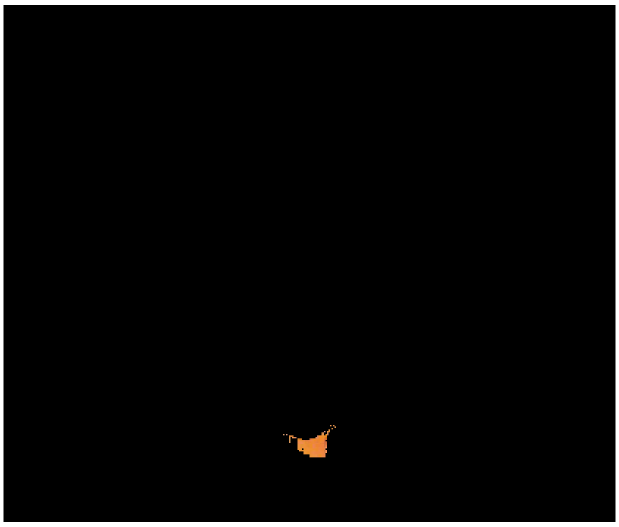
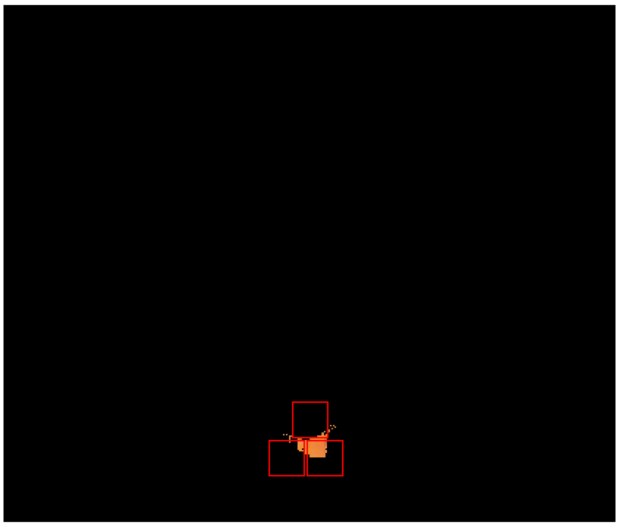
The left hand side image is the input of the feature filter and the right hand side is the output of the filter. From the result, you can see that the filter select the light regions (the light source and objects which can reflect light from the light source) of the image. In many cases, the feature filter can filter out the correct color but sometime it filter wrong colors, for example the yellow color can pass through the green feature filter. This problem can be solved by adding new constraint to the filter. For example, to solve yellow color can pass through the green feature filter; we can add an upper bound for the R value.

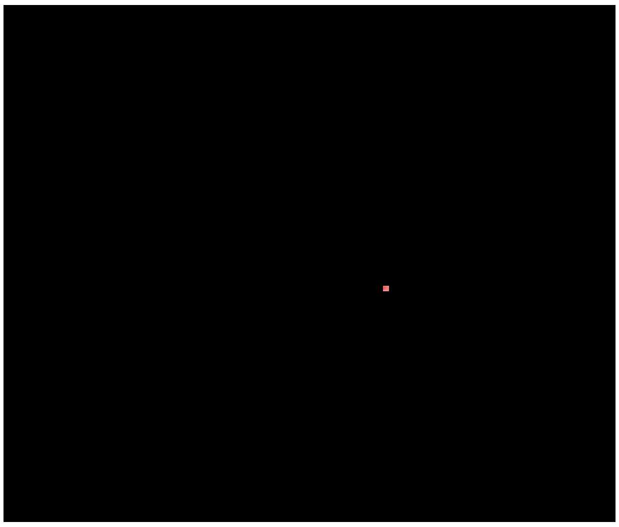
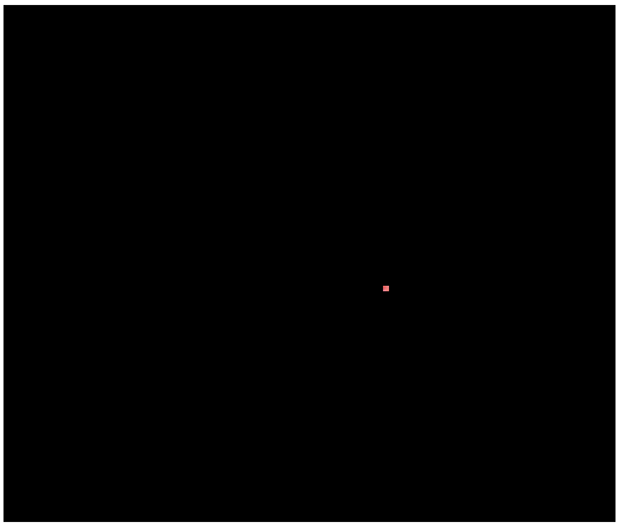
AIII.3 Experimental Result of LED Recognition

Experimental Result of First Version of LED Recognition

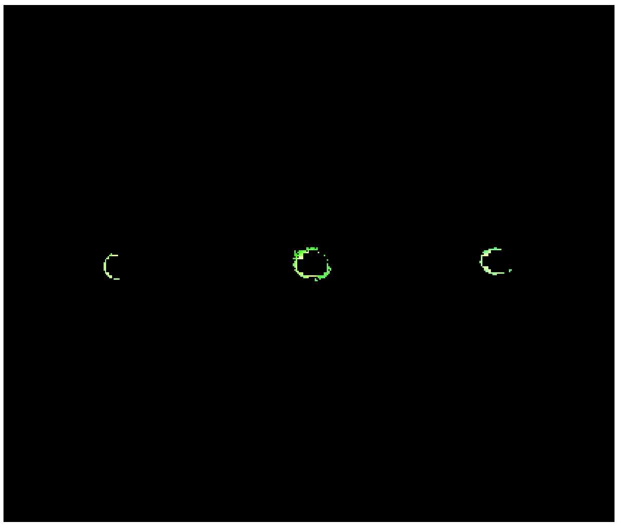
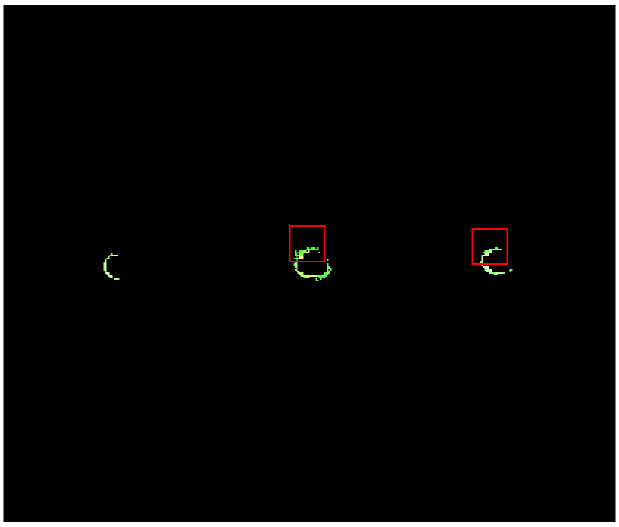
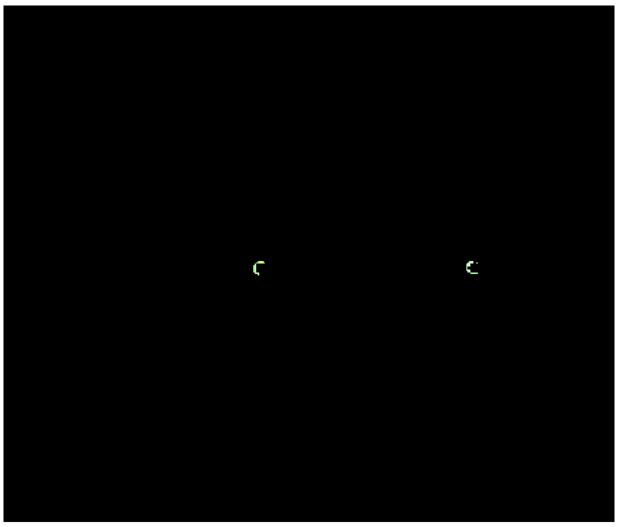
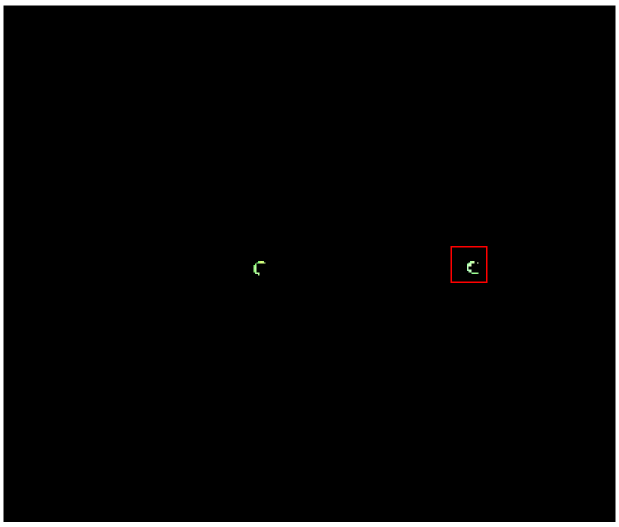
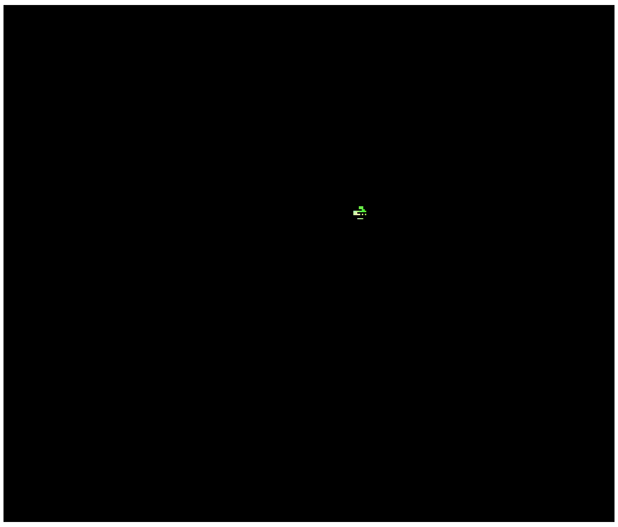
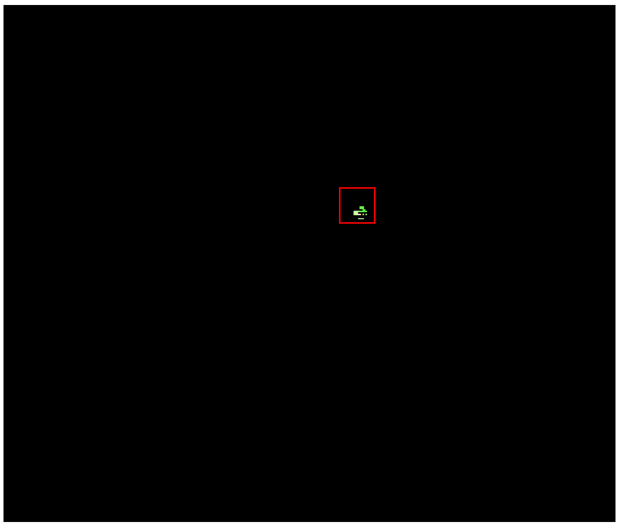


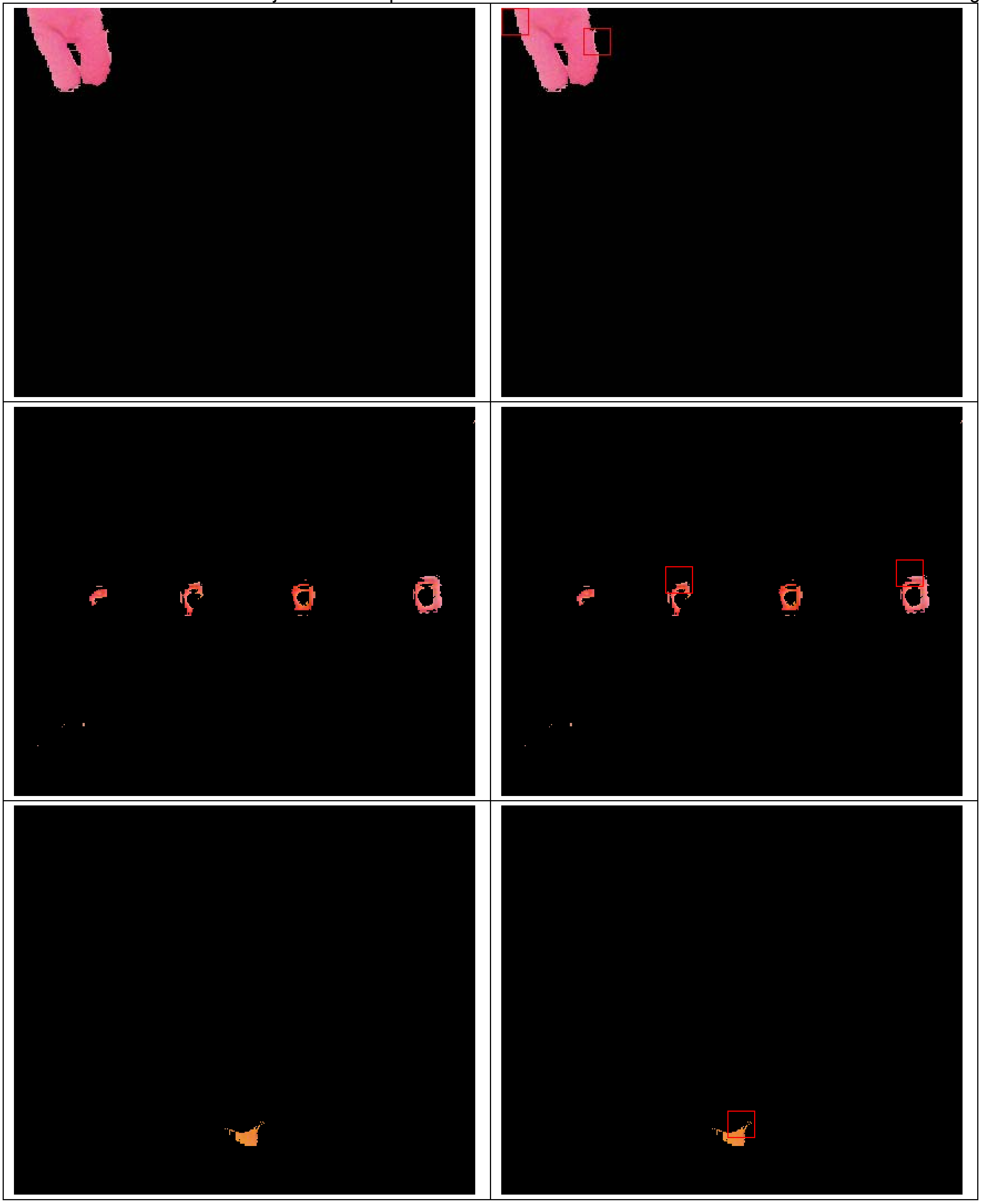


Experimental Result of Second Version of LED Recognition



The finalized version of LED recognition algorithm is a simplified version of second version of LED Recognition. So we do not provide the experimental result of it.

AIII.4 Analysis on LED Recognition

From results you can see that if the size of target object is similar to the block size, the result of LED recognition look like the best result of recognition. So, in order to get the best result of playing the game, the user should keep the size of target image similar to the block size. We have measured that if the distance between the camera and target object is about 8.5cm, the target image is similar size with the block.

Appendix IV

Projectile Motion

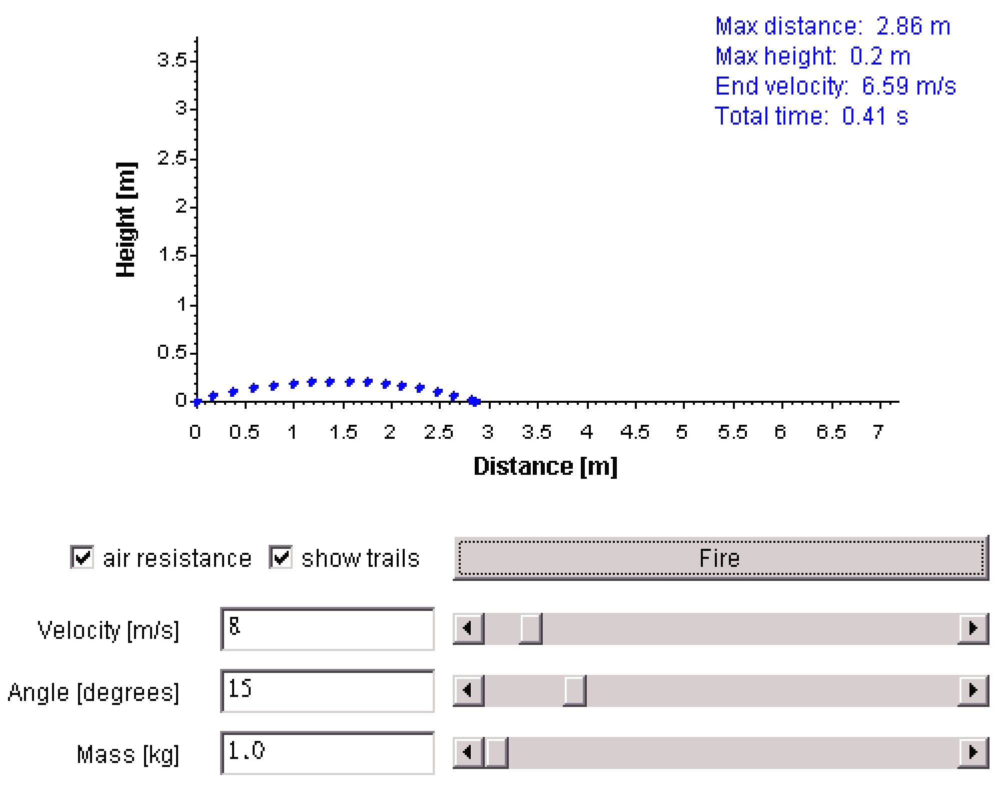
Appendix IV describes the experiment conducted in semester 2 for evaluating projectile motion data.

- **AIV.1 Experiment Result and Analysis**

AIV.1 Experiment Result and Analysis

In this experiment, we aimed to find out the appropriate velocity as well as appropriate vertical angle for users to play the game. We found a web page which offers a java applet to evaluate the project motion with parameters velocity, angle and mass.

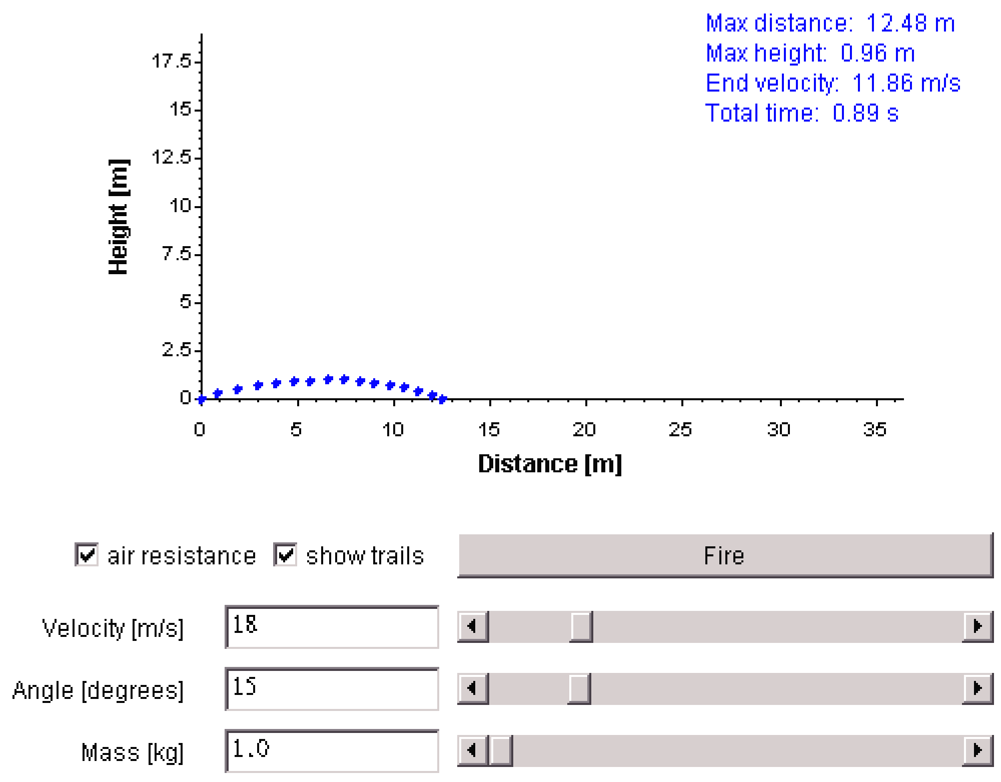
Though the applet offered by that we site could not allow us to alter the mass less than 1.0 KG, it helped us to visualize and estimate the project motion effect under differ parameters setting.



Max distance: 2.86 m
Max height: 0.2 m
End velocity: 6.59 m/s
Total time: 0.41 s

☑ air resistance ☑ show trails

Velocity [m/s] 8
Angle [degrees] 15
Mass [kg] 1.0



Max distance: 12.48 m
Max height: 0.96 m
End velocity: 11.86 m/s
Total time: 0.89 s

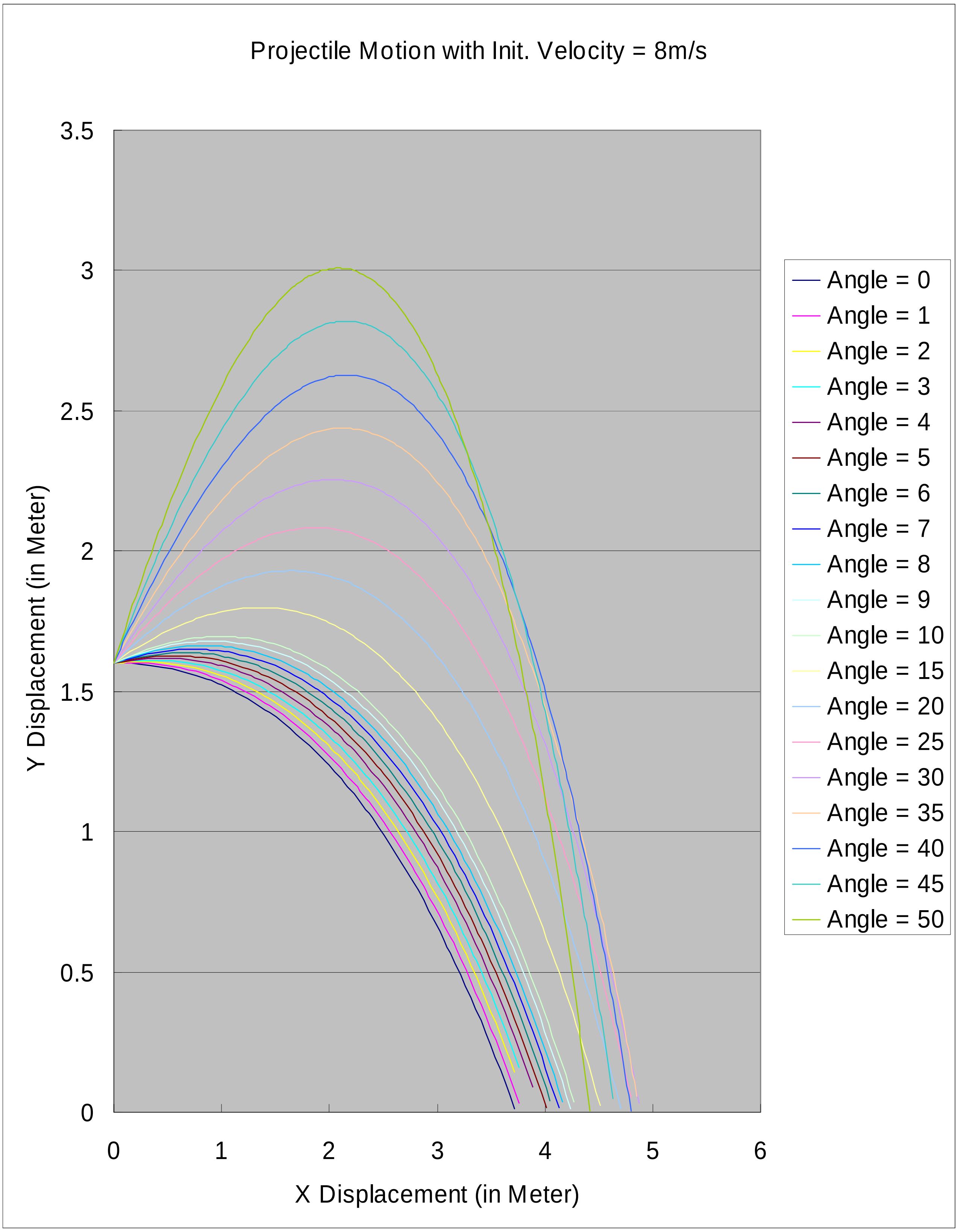
☑ air resistance ☑ show trails

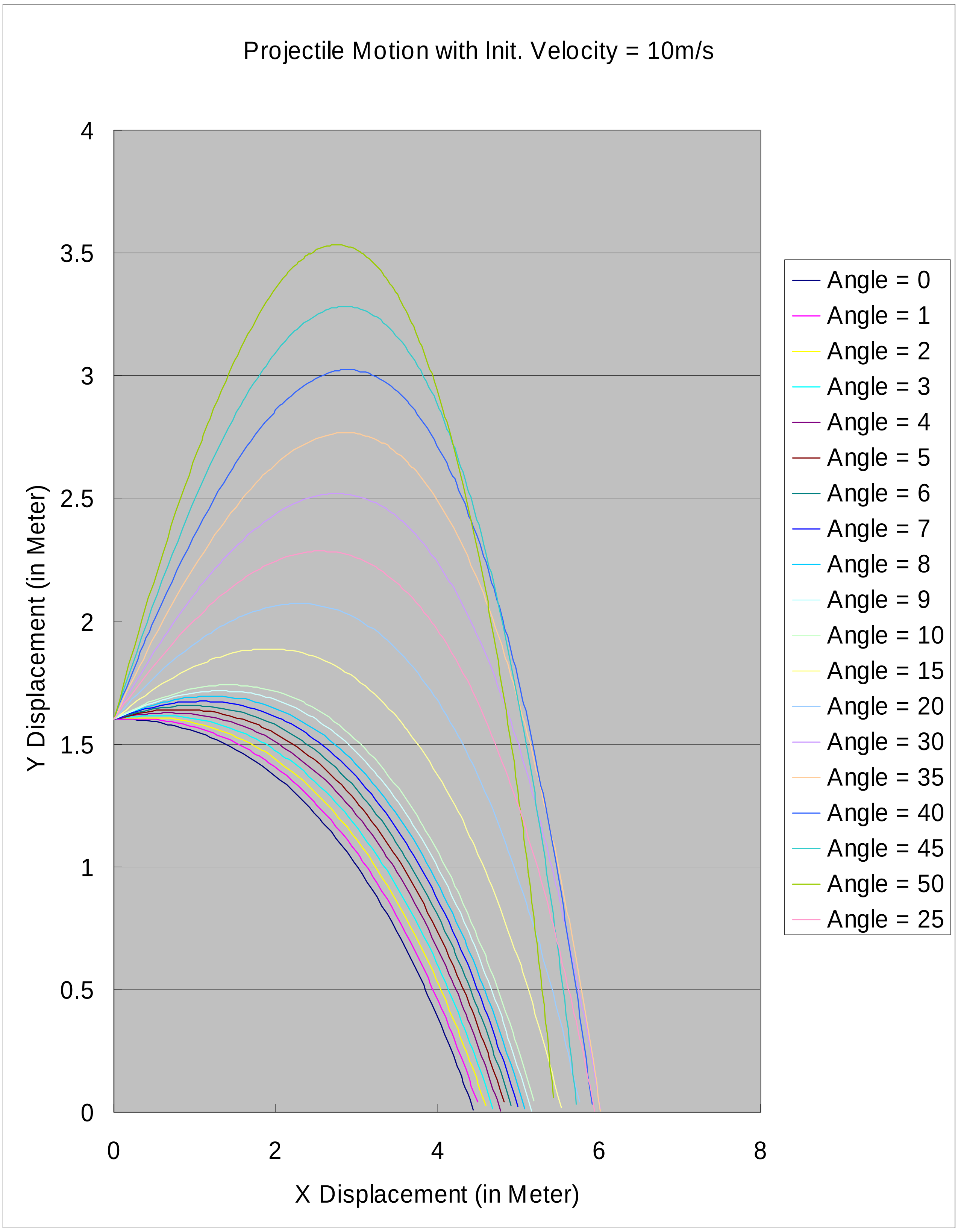
Velocity [m/s] 18

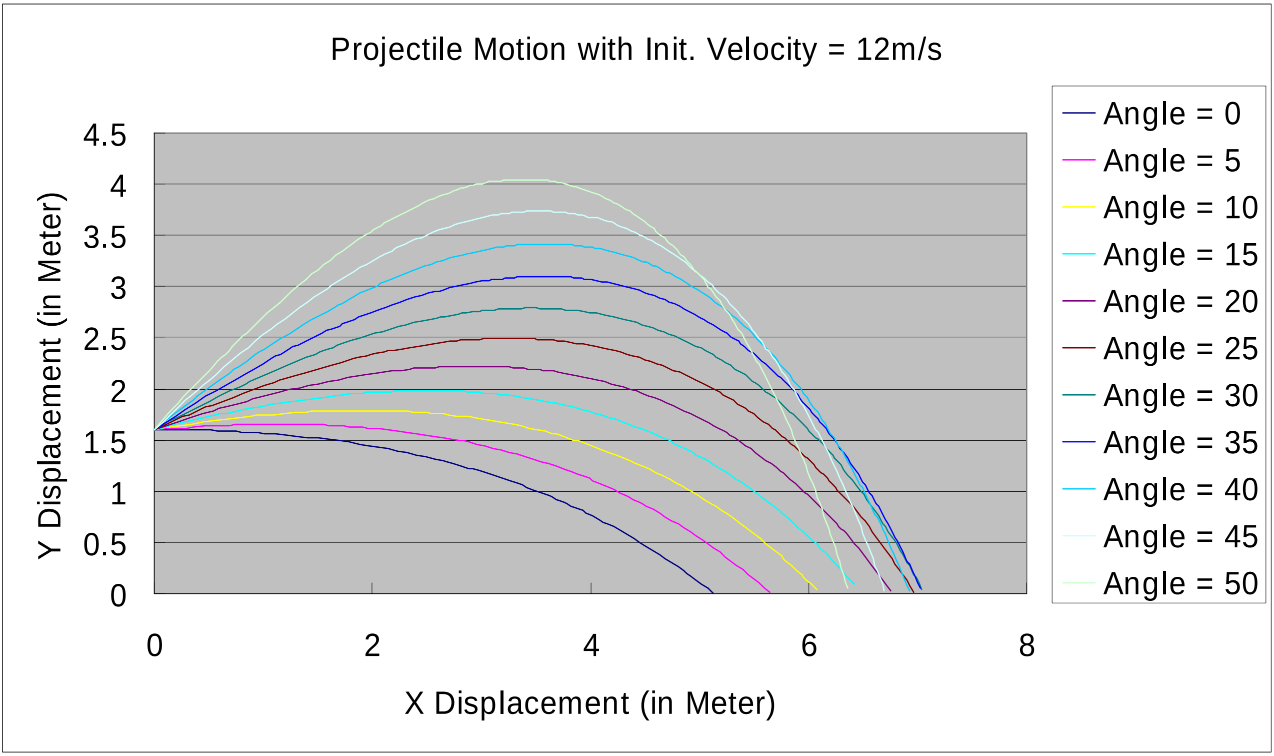
Angle [degrees] 15

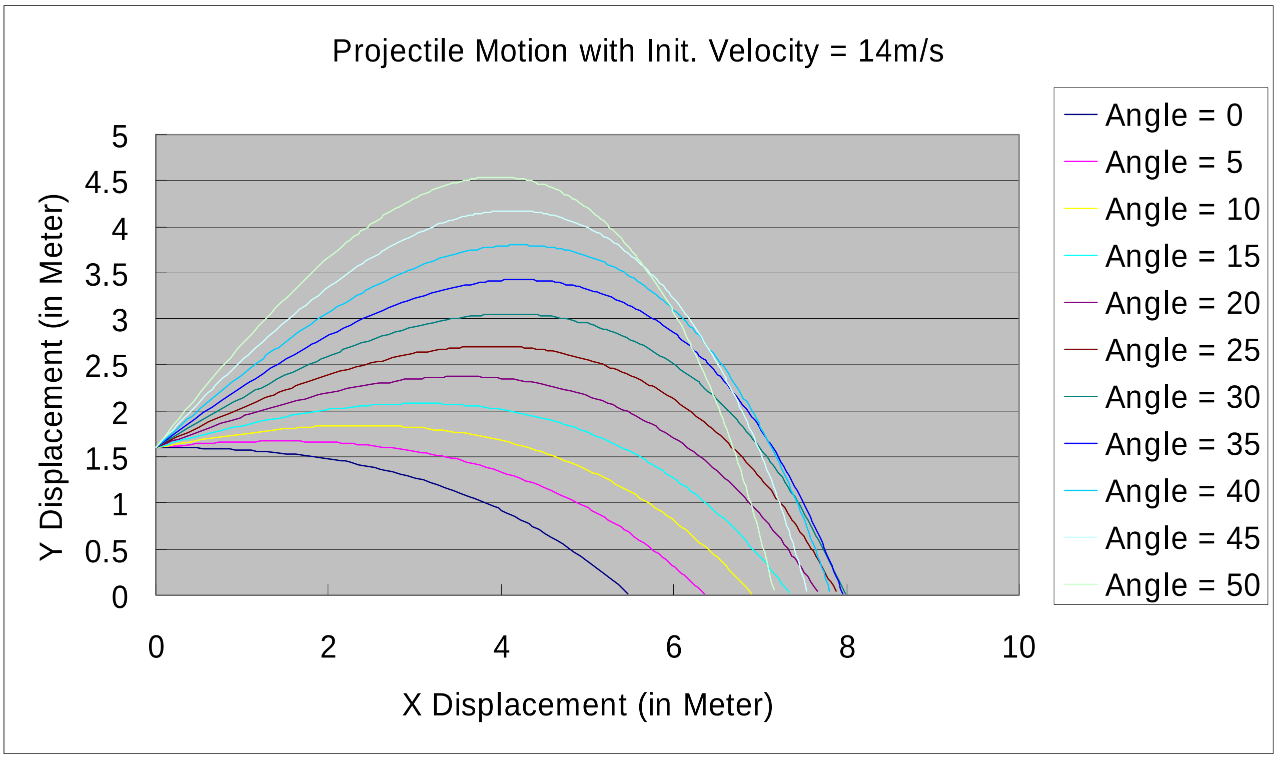
Mass [kg] 1.0

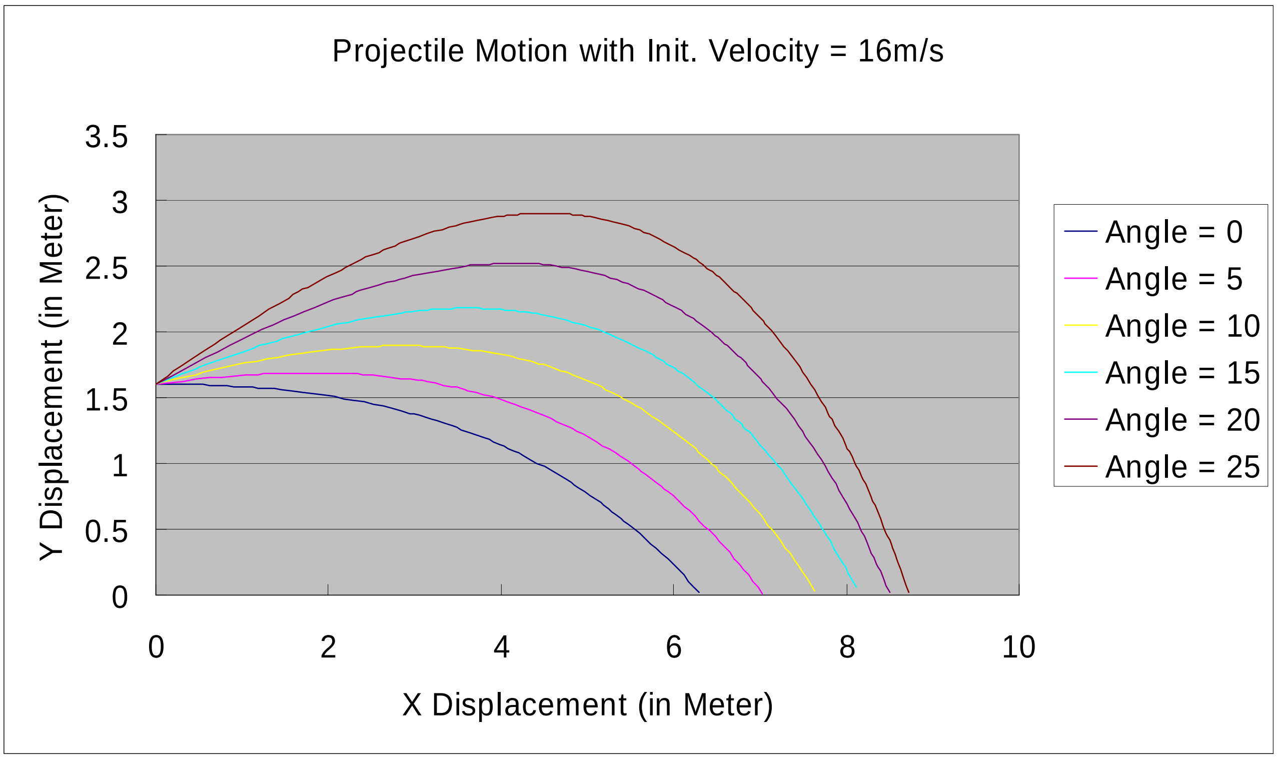
In fact, we performed data analysis in Excel also to find out the possible velocity and vertical angle. In Excel, we calculated the displacement in x and y direction at different time interval under different angle and velocity setting with mass equals to 0.02KG. We plotted some of the graphs for visualization proposes.


Projectile Motion with Init. Velocity = 8m/s
Y Displacement (in Meter)
X Displacement (in Meter)
0
0.5
1
1.5
2
2.5
3
3.5
0
1
2
3
4
5
6
Angle = 0
Angle = 1
Angle = 2
Angle = 3
Angle = 4
Angle = 5
Angle = 6
Angle = 7
Angle = 8
Angle = 9
Angle = 10
Angle = 15
Angle = 20
Angle = 25
Angle = 30
Angle = 35
Angle = 40
Angle = 45
Angle = 50


Projectile Motion with Init. Velocity = 10m/s
Y Displacement (in Meter)
X Displacement (in Meter)
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2
4
6
8
Angle = 0
Angle = 1
Angle = 2
Angle = 3
Angle = 4
Angle = 5
Angle = 6
Angle = 7
Angle = 8
Angle = 9
Angle = 10
Angle = 15
Angle = 20
Angle = 30
Angle = 35
Angle = 40
Angle = 45
Angle = 50
Angle = 25


Projectile Motion with Init. Velocity = 12m/s
Y Displacement (in Meter)
X Displacement (in Meter)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
2
4
6
8
Angle = 0
Angle = 5
Angle = 10
Angle = 15
Angle = 20
Angle = 25
Angle = 30
Angle = 35
Angle = 40
Angle = 45
Angle = 50


Projectile Motion with Init. Velocity = 14m/s
Y Displacement (in Meter)
X Displacement (in Meter)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
2
4
6
8
10
Angle = 0
Angle = 5
Angle = 10
Angle = 15
Angle = 20
Angle = 25
Angle = 30
Angle = 35
Angle = 40
Angle = 45
Angle = 50



We turned out to choose 8ms^{-1} and 18ms^{-1} to be our minimum and maximum velocity respectively. In addition, the vertical angle would be set from 0^{o} to 15^{o}. In other words, in the energy bar of the game, the pure red color would offer 18ms^{-1} initial velocity. Similarly, the pure green color would offer 8ms^{-1} initial velocity. If the user chose the pure green energy level, most probably, we dart would not have enough energy to reach the dart board. On the other hand, if the users chose the maximum energy level, the dart may have too much energy and thus throw over the dart board and cannot hit it.

Appendix V

Color Model

Appendix V describes some general color model.

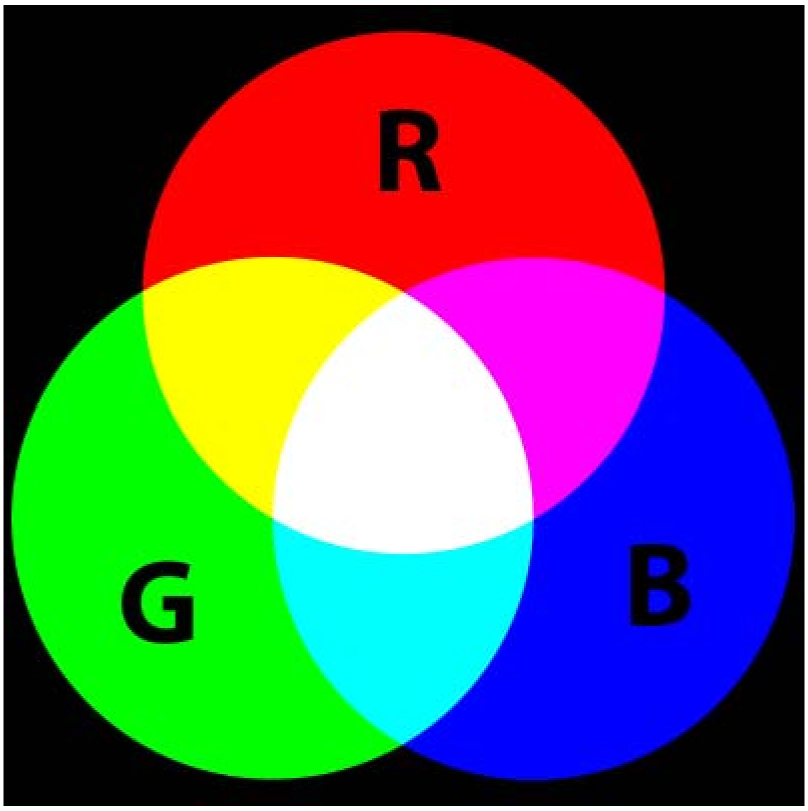
- **AV.1 Introduction**
- **AV.2 RGB Color Model**
- **AV.3 HSV Color Model**
- **AV.4 Transformation between RGB and HSV**

AV.1 Introduction

Color model is a mathematical way to describe colors as a tuple of numbers, each number represent a color component. For example, RGB color model is commonly used in the CRT and LCD displays; YIQ/YUV color model used in the television broadcasting; HSV color model is commonly found in the many computer graphics applications; CYMK uses in the printing process and publish industry.

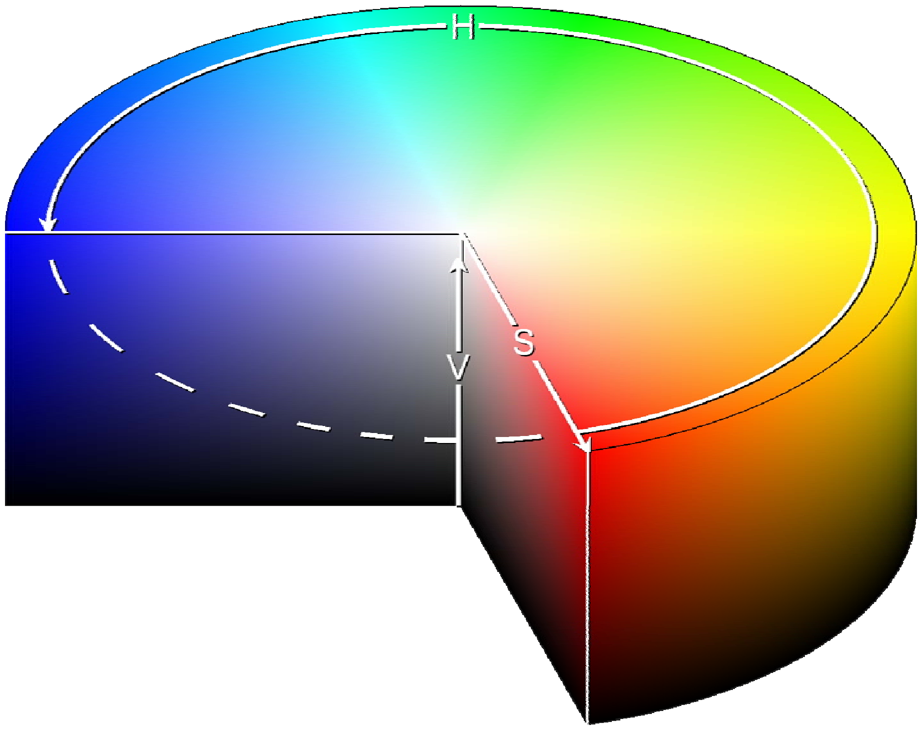
AV.2 RGB Color Model

RGB is the short-hand of the three primary colors used in the color model. RGB color model is an additive model which uses red, green and blue as three primary colors. By mixing different amount of the three primary colors, it can reproduce other colors. It simulates the effect of mixing different amount of 3 primary color lights to form other color. Due to the physical properties of CRT and LCD, RGB color model is very common use in the CRT or LCD display. The figure at the right hand side show the basic principle of mixing color of RGB color model.



AV.3 HSV Color Model

HSV is abbreviation of the three components Hue, Saturation and Value of the color model. Hue represents color type, ranged from 0 to 360. Saturation is purity of the color range from 0 – 100%. Higher value of saturation, the color will be more pure; lower value of saturation, it will increase the "grayness" of the color. Value, also called as brightness, represents the brightness of the color. The following figure shows the basic principle of HSV color model:



AV.4 Transformation between RGB and HSV

We can do a non-linear transformation to convert a color in RGB model to HSV model or from HSV to RGB.

a) From RGB to HSV

First normalize the (R, G, B) value, than let MAX be the maximum and Min be the minimum of these value. Then

$$H = \begin{cases} \text{undefined}, & \text{if } MAX = MIN \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ, & \text{if } MAX = R \\ & \text{and } G < B \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ, & \text{if } MAX = G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ, & \text{if } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{if } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{otherwise} \end{cases}$$

$$V = MAX$$

b) From HSV to RGB

If S = 0, R = G= B = V, otherwise calculate the following equations:

$$H_i = \left\lfloor \frac{H}{60} \right\rfloor \bmod 6$$

$$f = \frac{H}{60} - H_i$$

$$p = V \times (1 - S)$$

$$q = V \times (1 - f \times S)$$

$$t = V \times (1 - (1 - f) f \times S)$$

If H_i = 0, then R = V, G = *t*, B = *p*.

If H_i = 1, then R = *q*, G = V, B = *p*.

If H_i = 2, then R = *p*, G = V, B = *t*.

If H_i = 3, then R = *p*, G = *q*, B = V.

If H_i = 4, then R = *t*, G = *p*, B = V.

If H_i = 5, then R = V, G = *p*, B = *q*.