

**Department of Computer Science and Engineering  
The Chinese University of Hong Kong**

**2006 – 2007 Final Year Project Semester 1 Report**

**LYU 0604**

**Virtual Dart – an Augmented Reality Game on Mobile Device**

**Supervisor**

**Professor Michael R. Lyu**

**Lai Chung Sum**

**Siu Ho Tung**

# Abstract

---

**Augmented reality (AR)** deals with the combination of real world and computer generated data. At present, most AR research is concerned with the use of live video imagery which is digitally processed and "augmented" by the addition of computer generated graphics. Advanced research includes the use of motion tracking data, and the construction of controlled environments containing any number of sensors and actuators.

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 2<sup>nd</sup> & 3<sup>rd</sup> 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.

<b>Abstract</b> .....	2
<b>Chapter 1</b> .....	5
<b>Introduction</b> .....	5
<b>1.1 Background Information and Motivation</b> .....	6
<b>1.2 Programming Capability Issue</b> .....	8
<b>1.3 Project Objective</b> .....	9
<b>1.4 Project Equipment</b> .....	9
<b>Chapter 2</b> .....	10
<b>Symbian Operating System Overview</b> .....	10
<b>2.1 Basic Architecture of Symbian Operating System</b> .....	11
<b>2.2 Development Environment</b> .....	13
<b>2.3 Limitation of Symbian Phone</b> .....	15
<b>2.4 Features Different in Symbian OS Series 60 2<sup>nd</sup> and 3<sup>rd</sup> Edition</b> .....	16
<b>2.5 Porting Existing Program to Series 60 3<sup>rd</sup> Edition</b> .....	19
<b>2.6 Why Symbian</b> .....	20
<b>2.7 Conclusion</b> .....	20
<b>Chapter 3</b> .....	21
<b>mVOTE Engine</b> .....	21
<b>3.1 What is mVOTE?</b> .....	22
<b>3.2 Block Matching Algorithm</b> .....	22
<b>Chapter 4</b> .....	25
<b>Program Design and Implementation</b> .....	25
<b>4.1 User Interface</b> .....	26
<b>4.2 Program Design</b> .....	30
<b>4.3 Algorithm Comparison</b> .....	35
<b>4.4 How to do feature recognition?</b> .....	36
<b>4.5 Motion Tracking during the Application</b> .....	37
<b>Chapter 5</b> .....	38
<b>Feature Selection Improvement</b> .....	38
<b>5.1 Introduction</b> .....	39
<b>5.2 Harris Corner Detector</b> .....	39
<b>5.3 Fast Corner Detector</b> .....	43
<b>5.4 Select Feature from Corner List</b> .....	47
<b>Chapter 6</b> .....	51
<b>Project Progress, Difficulties and Future Work</b> .....	51
<b>6.1 Project Progress</b> .....	52
<b>6.2 Difficulties We Face during Project</b> .....	52
<b>6.3 Future Work</b> .....	53

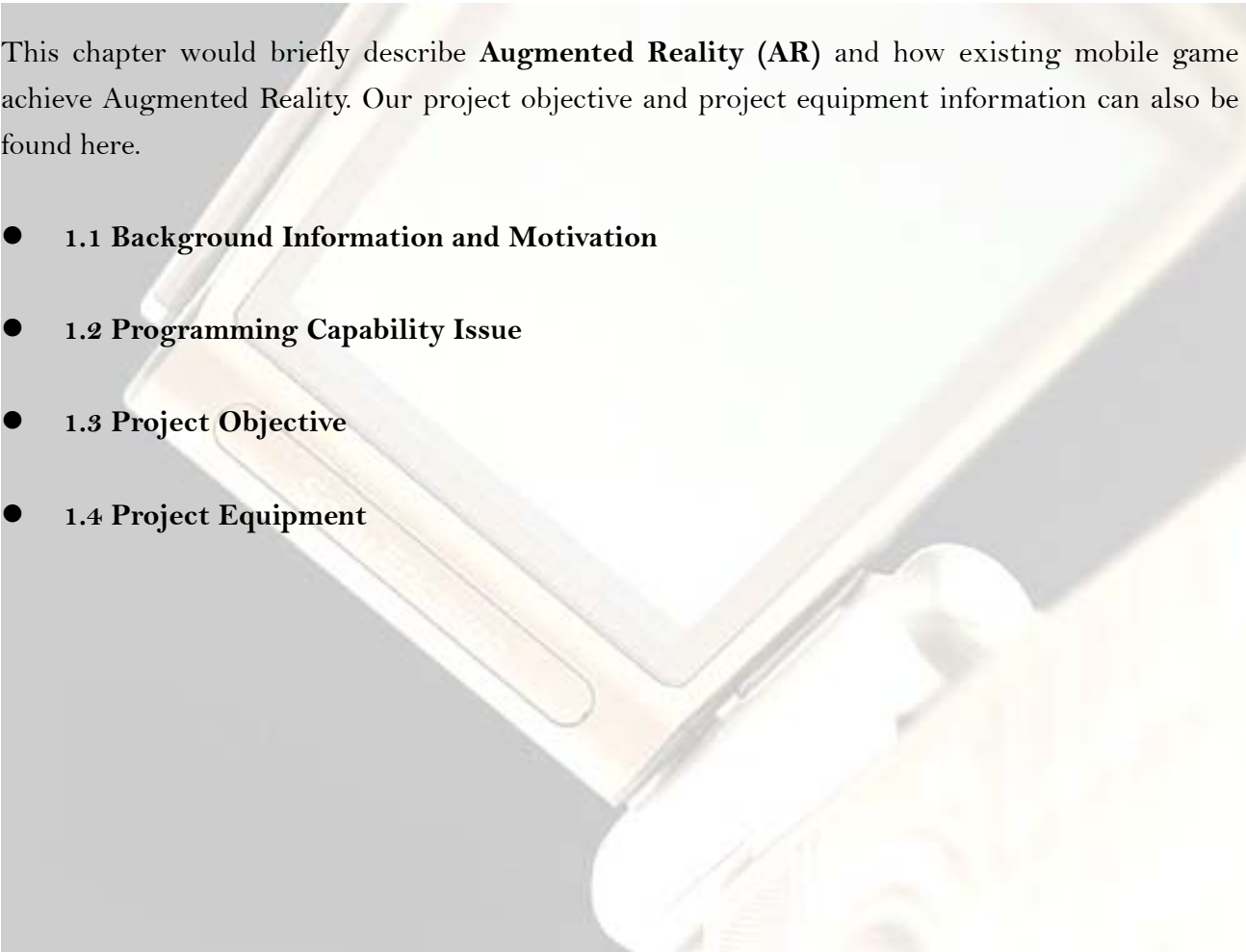
<b>Chapter 7</b> .....	55
<b>References</b> .....	55
<b>Appendix I</b> .....	56
<b>Feature Selection</b> .....	56
<b>AI.1 Pictures under Normal Lighting Condition</b> .....	57
<b>AI.2 Pictures under Insufficient Light Condition</b> .....	64
<b>AI.3 Analysis</b> .....	68
<b>Appendix II</b> .....	73
<b>Parameter Adjustment for Fast Corner Algorithm</b> .....	73
<b>AII.1 Experiment Result</b> .....	74
<b>AII.2 Analysis</b> .....	80
<b>Appendix III</b> .....	81
<b>Accuracy of Feature Blocks Finding for Algorithm 1</b> .....	81
<b>AIII.1 Experiment Result</b> .....	82
<b>AIII.2 Analysis</b> .....	85
<b>Appendix IV</b> .....	87
<b>Accuracy of Feature Blocks Finding for Algorithm 2</b> .....	87
<b>AIV.1 Experiment Result</b> .....	88
<b>AIV.2 Analysis</b> .....	93
<b>AIV.3 Miscellaneous</b> .....	94

# Chapter 1

---

## Introduction

This chapter would briefly describe **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)



**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**

A few years before, users may feel panic to develop programs for mobile phones because they lack supports from the vendors and at that moment, the processing power of a mobile phone is very limited. Now, things become better, there are parties 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 to develop a game. Instead our objective is to develop a way to demonstrate how a program can process memory to “remember” its external environment for interaction of Augmented Reality.

The game is just 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 high 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 2<sup>nd</sup> Edition, Feature Pack 3) as well as Nokia N80 (Symbian OS v9.1, Series 60 3<sup>rd</sup> Edition). As the emulator provided by Nokia SDK does not similar camera function, we use the Mobile Phones as our testing and development platform directly for our project.

# 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 2<sup>nd</sup> Edition and 3<sup>rd</sup> Edition of the Symbian Operating System.

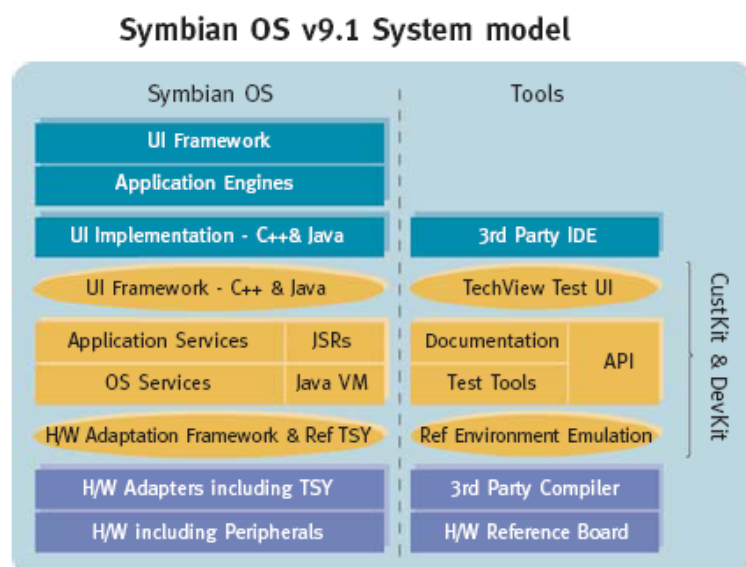
- **2.1 Basic Architecture of Symbian Operating System**
- **2.2 Development Environment**
- **2.3 Limitation of Symbian Phone**
- **2.4 Features Different in Symbian OS Series 60 2<sup>nd</sup> and 3<sup>rd</sup> Edition**
- **2.5 Porting Existing Program to Series 60 3<sup>rd</sup> Edition**
- **2.6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

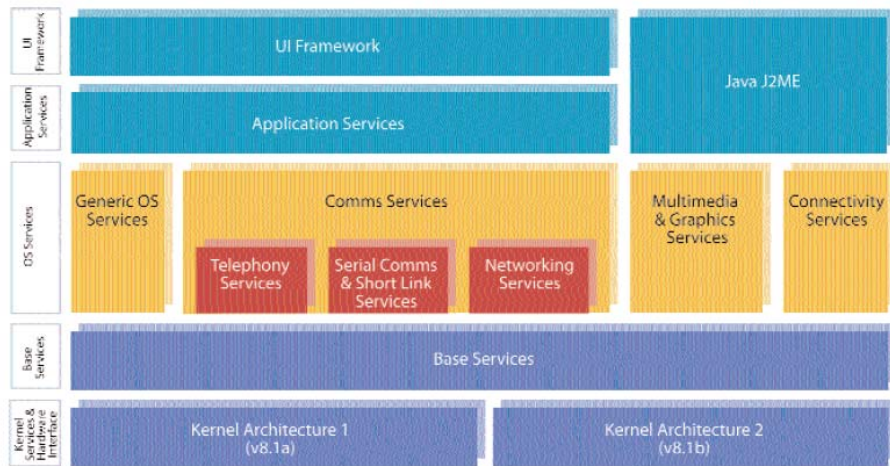
## 2.1 Basic Architecture of Symbian Operating System

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.





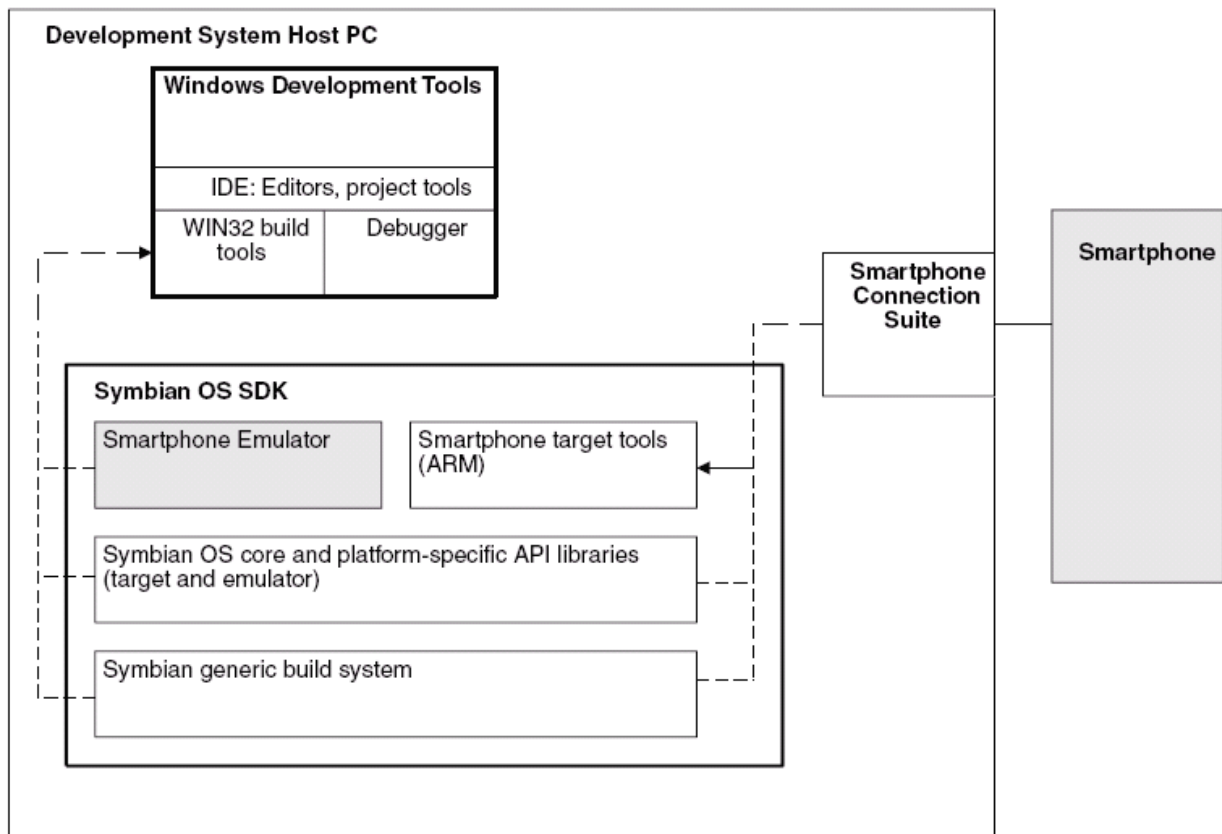
**Symbian OS v8.1 System model**

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 SDK provided by Nokia. The development tools can be formulated like this:

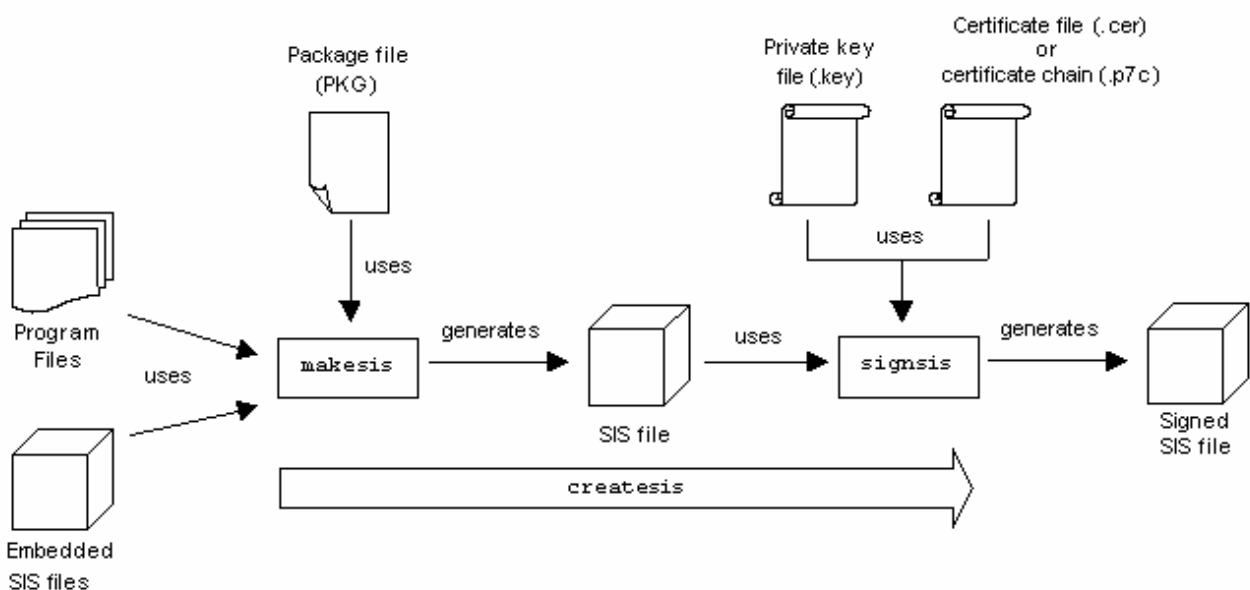


There are emulators provided along with the Nokia SDK which is a Windows application that simulate a Smartphone entirely in software – complete with simulated buttons and display. This allows developers to run and debug Symbian OS software on the PC as opposed to running on a real device. However, there is a major drawback for the emulators. The fact is that emulators for Series 60 2<sup>nd</sup> Edition Feature Pack 3 as well as Series 60 3<sup>rd</sup> 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.

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, as shown in figure 2.3), these make programming on the Symbian phone a very difficult task.

	Nokia N80 Specification	Nokia N90 Specification
Operating System	Symbian v9.1 (Nokia Series 60 3 <sup>rd</sup> Edition)	Symbian v8.1a (Nokia Series 60 2 <sup>nd</sup> 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)

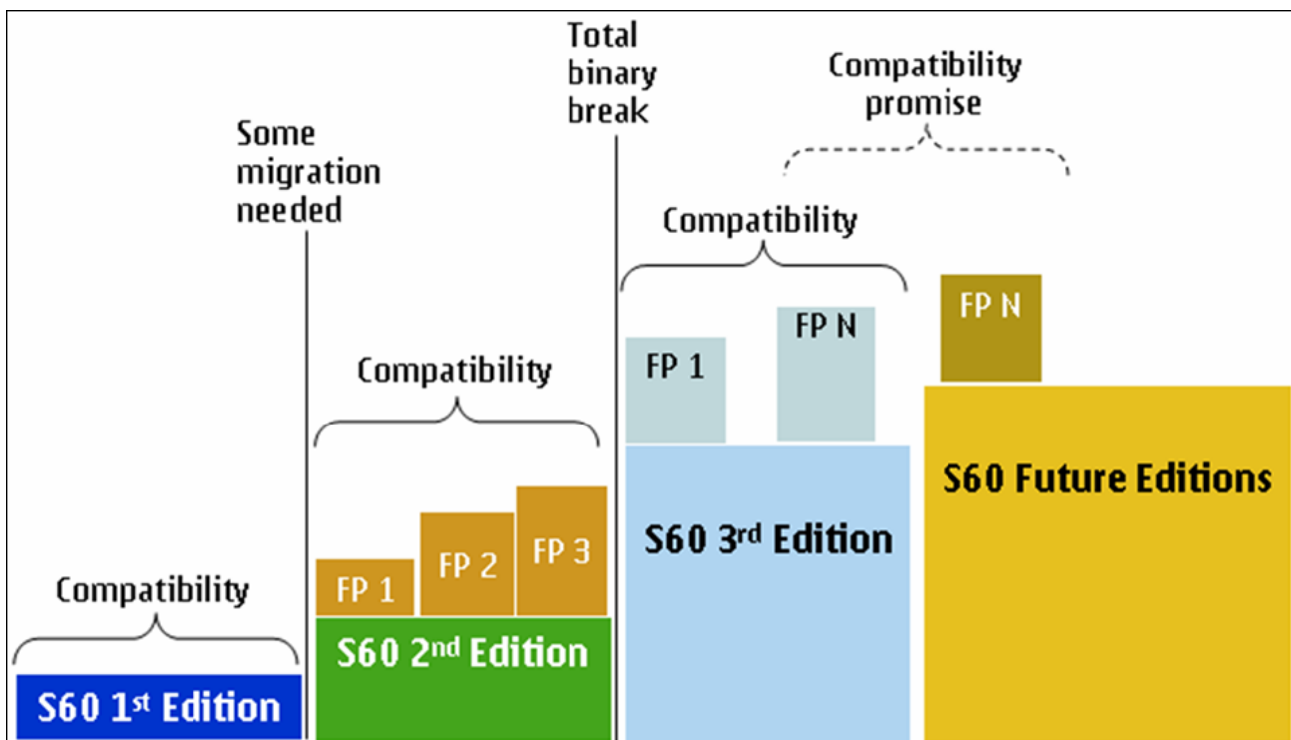
Since computation performance is an important factor in making a realistic augmented reality game. If we take too long time for the calculation of motion tracking, the frame rate will fall off. When ever possible, we would use the following operation to enhance the performance:

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.

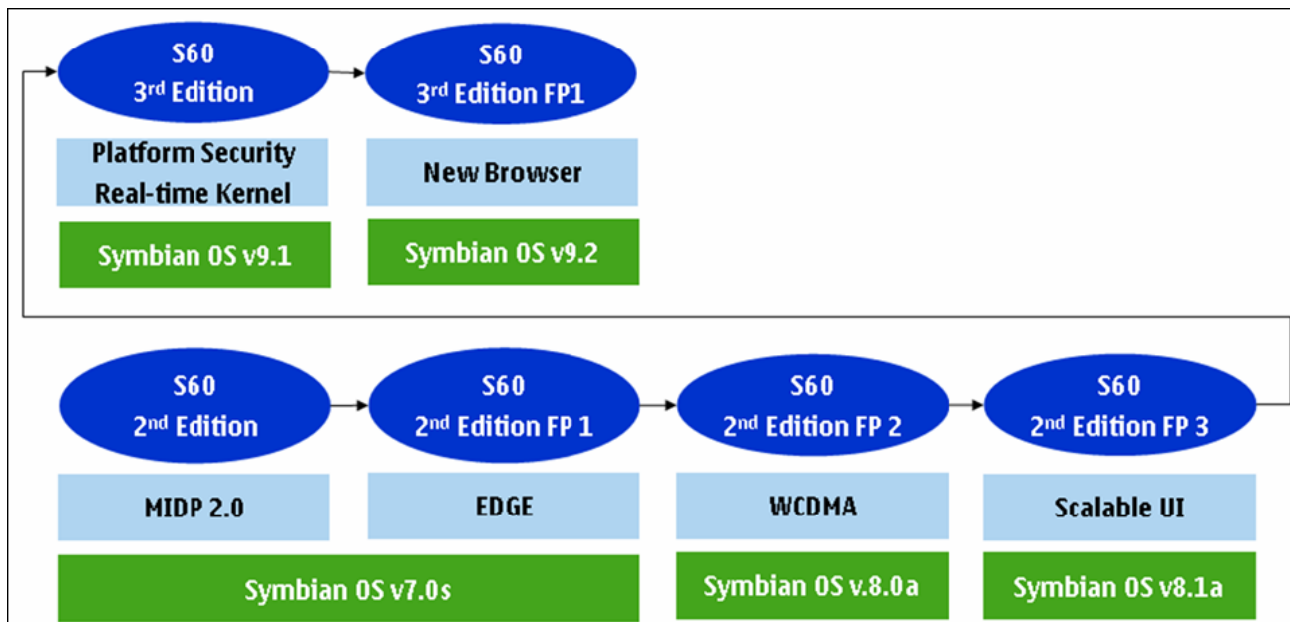
#### 2.4 Features Different in Symbian OS Series 60 2<sup>nd</sup> and 3<sup>rd</sup> Edition

S60 3<sup>rd</sup> 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 2<sup>nd</sup> and 3<sup>rd</sup> Editions, which means that applications need to be compiled using the new tools provided in the S60 3<sup>rd</sup> Edition in order to run on S60 devices based on the new platform edition. S60 3<sup>rd</sup> 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 2<sup>nd</sup> and 3<sup>rd</sup> Edition:





The following diagram shows the Nokia platform development:



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

**Nokia Camera API**

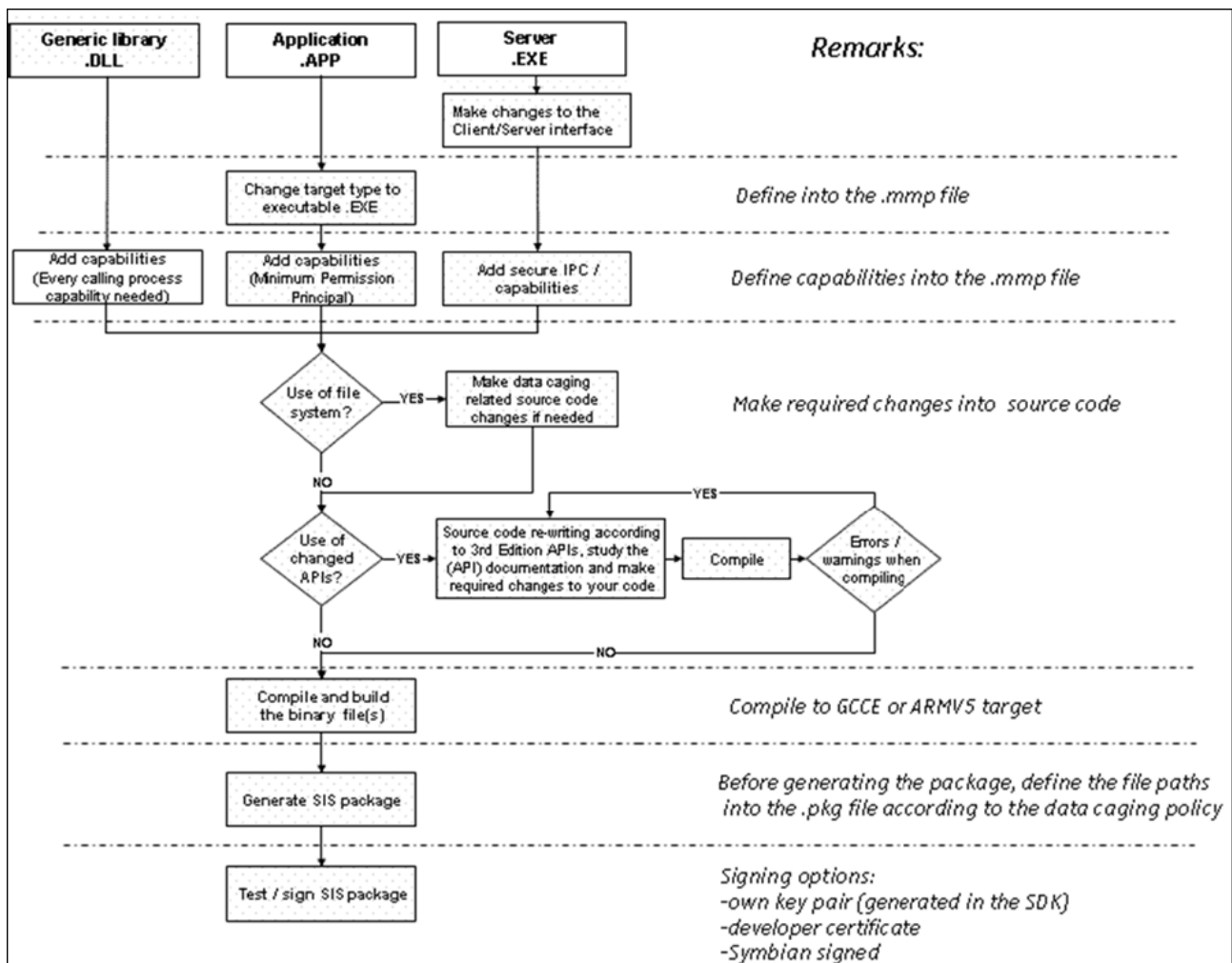
*The wrapper for the Nokia Camera API (Camera Server) is removed. The Symbian Onboard Camera API (ECam) replaces this deprecated API.*

**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 3<sup>rd</sup> Edition

Since our target platform involve both Series 60 2<sup>nd</sup> Edition (Nokia N90 with Feature Pack 3) and Series 60 3<sup>rd</sup> 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 3<sup>rd</sup> Edition Platform can be achieved:



## **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 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 2<sup>nd</sup> and 3<sup>rd</sup> 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 would briefly describe function of mVOTE engine which is created by former team. The mVOTE engine provides the function of motion tracking and feature selection of our project.

- 4.1 What is mVOTE
- 4.2 Block Matching Algorithm



### **3.1 What is mVOTE?**

The Mobile Video Object Tracking Engine (mVOTE™) is software 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™ can convert the corresponding movement of the camera into translational movement and degree of rotation. The mVOTE™ 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](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 divide the search window into blocks with equal size. In each block, it tries to find out which block in the search window is most similar to the feature block that we want to match.

## How to measure the similarity between two blocks

The common way of measure the similarity is calculate the intensity difference 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:

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

Our ancestor choose 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 values 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 a searching algorithm.

## 3. Fast Exhaustive Search Algorithm

The Fast Exhaustive Search Algorithm is trying to reduce the number of values needed 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 calculating the Cost Function.

iii. Partial Distortion Elimination (PDE)

The idea of the PDE Algorithm is to 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 the calculation of SSD on that block is useless and can be stopped. There is the definition of  $k^{\text{th}}$  PSSD:

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

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



# Chapter 4

## Program Design and Implementation

This chapter would briefly describe design and implementation of our program

- **4.1 User Interface**
- **4.2 Program Design**
- **4.3 Algorithm Comparison**
- **4.4 How to do Feature Recognition**
- **4.5 Motion Tracking during the Game**



#### **4.1 User Interface**

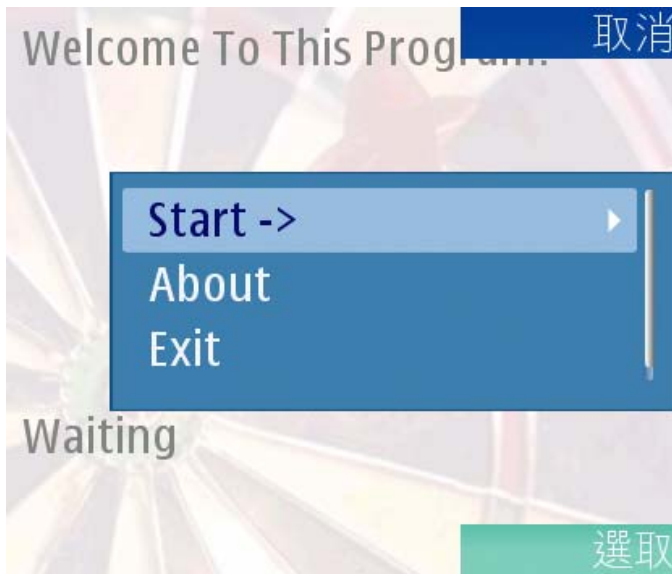
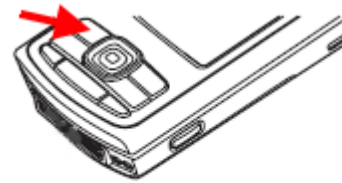
This project was mainly tested and debugged on Symbian phone namely Nokia N80 and N90 mobile phones. We develop the augmented reality game on top of the existing Motion Tracking (MVOTE) Engine which is developed by our ancestor (LYU0404) with some modification.

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.

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



User may press the option button for option selection:



The "Start ->" menu items show two more options, they are, "Take a Sn@pshot", "Yo! Start Playing"



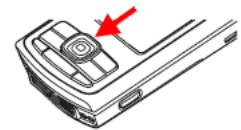
"Take a Sn@pshot" menu item provides a way for users to select a region where dart board is intended to be put.

When "Take a Sn@pshot" is selected, users may see an interface similar to the following diagram:



Please Select Region For Putting Dart Board.

Users may see a selection box in white color. The selection box is the region indicating the location where users would like to put a dart board in. The users may move the location of the selection box by controlling the joystick of the mobile phone.



After the selection procedure, users may press the "enter" key in the joystick for dartboard mapping.



The users may move the mobile phone (i.e. camera) for dart block movement.



“Yo! Start Playing” is in fact directly perform dartboard mapping part and by pass the selection procedure process instead.

## 4.2 Program Design

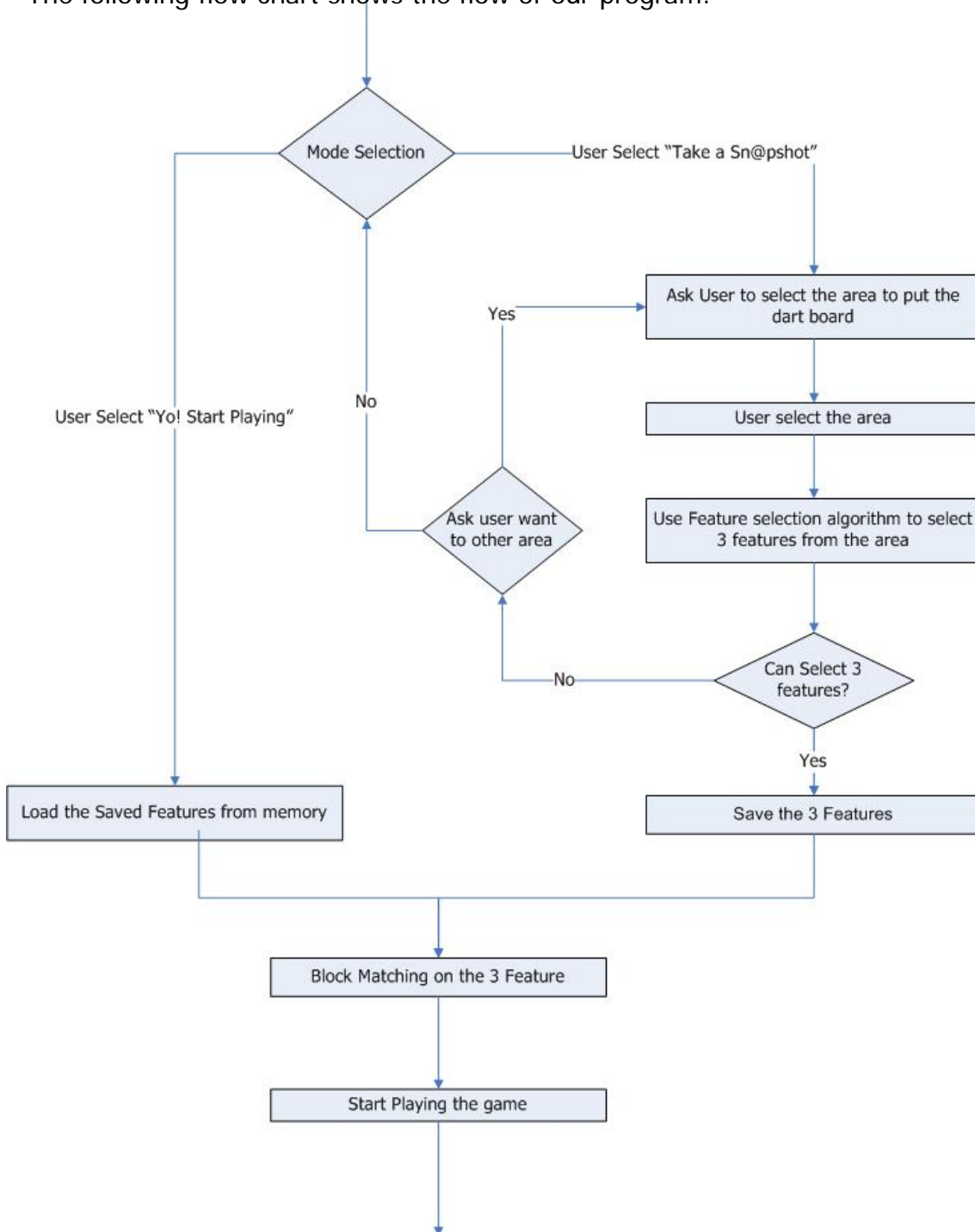
The program consists of these files:

File	Description
AR3rdapplication.h AR3rdapplication.cpp	An Application that creates a new blank document and defines the application UID.
AR3rddcoument.h AR3rdddocument.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
fast.h fast.cpp	Fast Corner Algorithm
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

There are two ways to start our program, the first one is ask user to select the area to put the dart board and other is load the saved features from the memory.

The following flow chart shows the flow of our program:



For "Take a Sn@pshot" mode:

1. The program will ask user to select the area to put dart board.
2. After the selection, the feature selection algorithm selects three features from the area.
3. If it can find three features, the program will save the three features and proceed to next step. If it cannot find three features, it will ask the user to select other area or not. If the answer is "Yes", it will back to step 1, otherwise it will back to mode selection state.
4. It will do the block matching algorithm on the whole screen, in order to get the most matching points for the three features in the screen.
5. We can start playing the game by keep motion tracking on the three matching points on the screen.

For "Yo! Start Playing" mode:

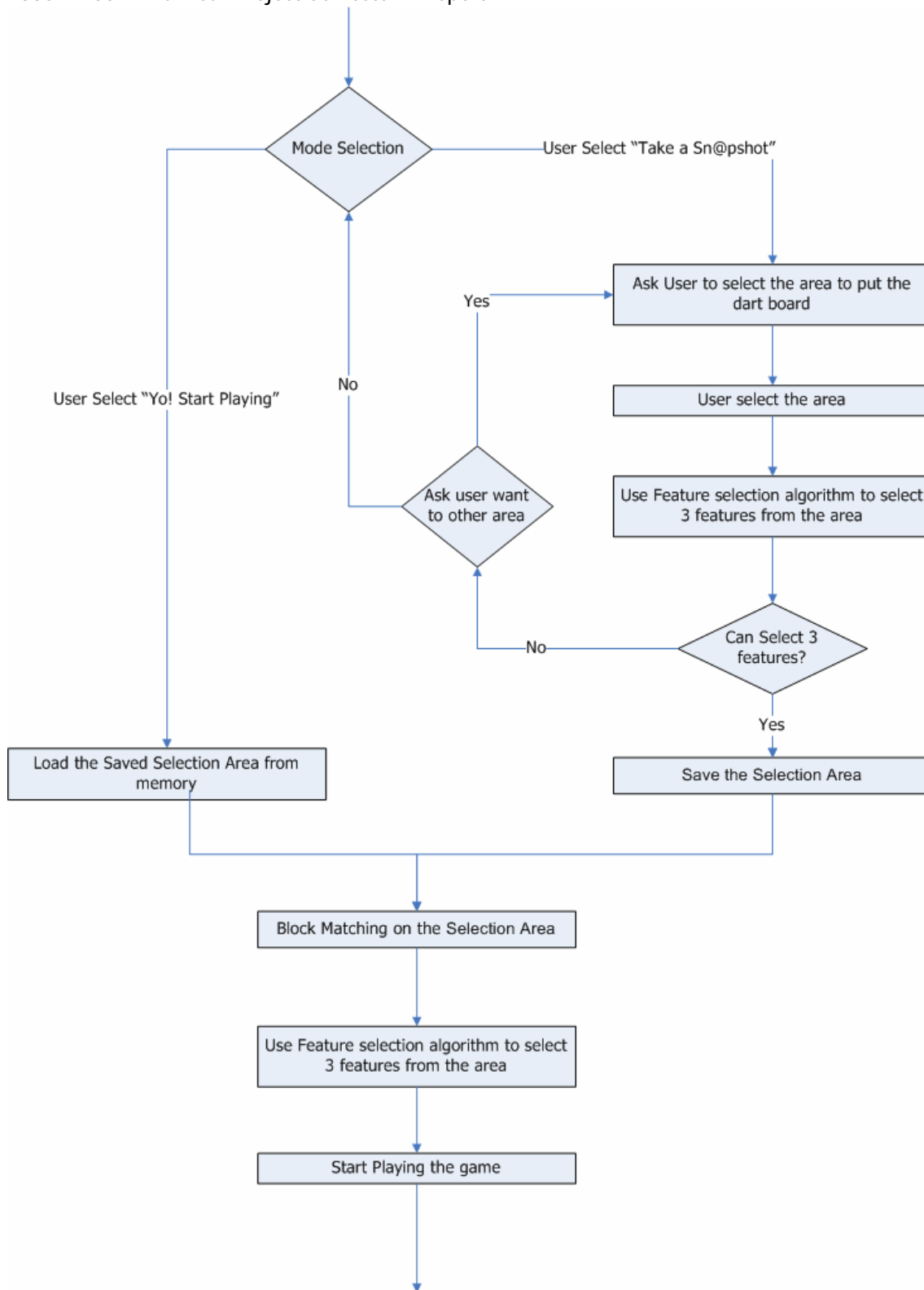
1. The program loads the stored features from the memory.
2. It runs the block matching algorithm on the whole screen, in order to get the most matching points for the three feature in the screen.
3. After finding the points, we can start playing the game.

This is our initial approach to do our program but we find out that the accuracy of this approach is not very high, for more detail please refer to the Appendix III for the our experimental result of the using this approach.

In order to solve the accuracy problem, we have proposed another approach to this problem.

The following is the flow chart of our new approach:





For "Take a Sn@pshot" mode:

1. The program will ask user to select the area to put dart board.
2. After the selection, the feature selection algorithm selects three features from the area.
3. If it can find three features, the program will save the selection area and proceed to next step. If it cannot find three features, it will ask the user to select other area or not. If the answer is "Yes", it will back to step 1, otherwise it will back to mode selection state.
4. It will do the block matching algorithm on the whole screen, in order to get the most matching point of the selection area in the screen.
5. It does the feature selection algorithm on the area selection from the block matching algorithm to get the three points for motion tracking.
6. We can start playing the game by keep motion tracking on the three matching points on the screen.

For "Yo! Start Playing" mode:

1. The program loads the stored selected area from the memory.
2. It runs the block matching algorithm on the whole screen, in order to get the most matching point of the selection area in the screen.
3. It does the feature selection algorithm on the area selection from the block matching algorithm to get the three points for motion tracking.
4. After finding the points, we can start playing the game.

### **4.3 Algorithm Comparison**

The main difference between the new approach and initial approach is instead of storing three features, now we store the whole selection area into the memory. We think that the inaccurate result may due to the size of feature block is not large enough to store enough information to identify feature block from other candidate block when doing block matching. There are two reasons why we choose to store the selection area, not the three features. The first one is the selection area is the most descriptive block in the Selection area, so it can reduce the chance of mismatching. The second reason is larger the block using in block matching algorithm, the number of candidate block will be decrease. So it can increase the efficiency of the block matching algorithm.

You may wonder why we run the feature selection algorithm two times in our proposed approach. The first time running is for the testing purpose, it is running to test for the selection area contain enough feature to do our algorithm. The second time running is for selection purpose, it will select three feature points for playing the game. So, it is different objective for two times running for the algorithm.

The whole experimental result of our new algorithm is in the Appendix IV, please refer it for more detail.

#### **4.4 How to do feature recognition?**

We divide 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 uses the Fast Exhaustive Search Algorithm as searching method and SSD as the Cost Function to calculate the similarity between reference block and candidate block. The block matching algorithm is sped 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 our algorithm apply many techniques to increase the speed of algorithm, but it is still very slow because it searches the whole screen to find the most matching block. So we need to find other ways to improve the performance of it.

We had tried to use a smaller image to do block matching, hope that it can increase the speed of the matching. After the running of the algorithm, we scale the image up to the full screen for display to user. It can increase the speed of algorithm because the search window size is such smaller than the original. It is very time consuming to scale up the image, so we reject this method.

We plan to find other methods or algorithms to speed up the block matching in the next semester.

#### **4.5 Motion Tracking during the Application**

During the game, we need to keep motion tracking on three feature points during our application. Using two feature points is enough to locate an object in the 2D screen, the introduction of third feature point is using as backup purpose.

We use the first two feature points to locate the dart board first. We put the dart board at the mid-point of two points. If one of the feature points fails, we use the third feature point to replace the failure point. If more than one feature points fail at the same time, our program will stop playing the game and switch back to Selection Mode to ask user to select other area to place the dart board.

##### Conditions for Feature Points Fail

There are two conditions for a feature point to regard as failure:

1. The feature point is at the edge region of the screen.

If the feature point is at the edge region of screen, the feature point may be out of screen at next motion. So we need to reject it to keep our program running correctly.

2. The distance between two feature points are too short.

If the two feature points are too closed, the features represent by the feature points may be overlapped. The overlapped features may affect the accuracy of motion tracking algorithm. In this condition, we need to reject both feature points.

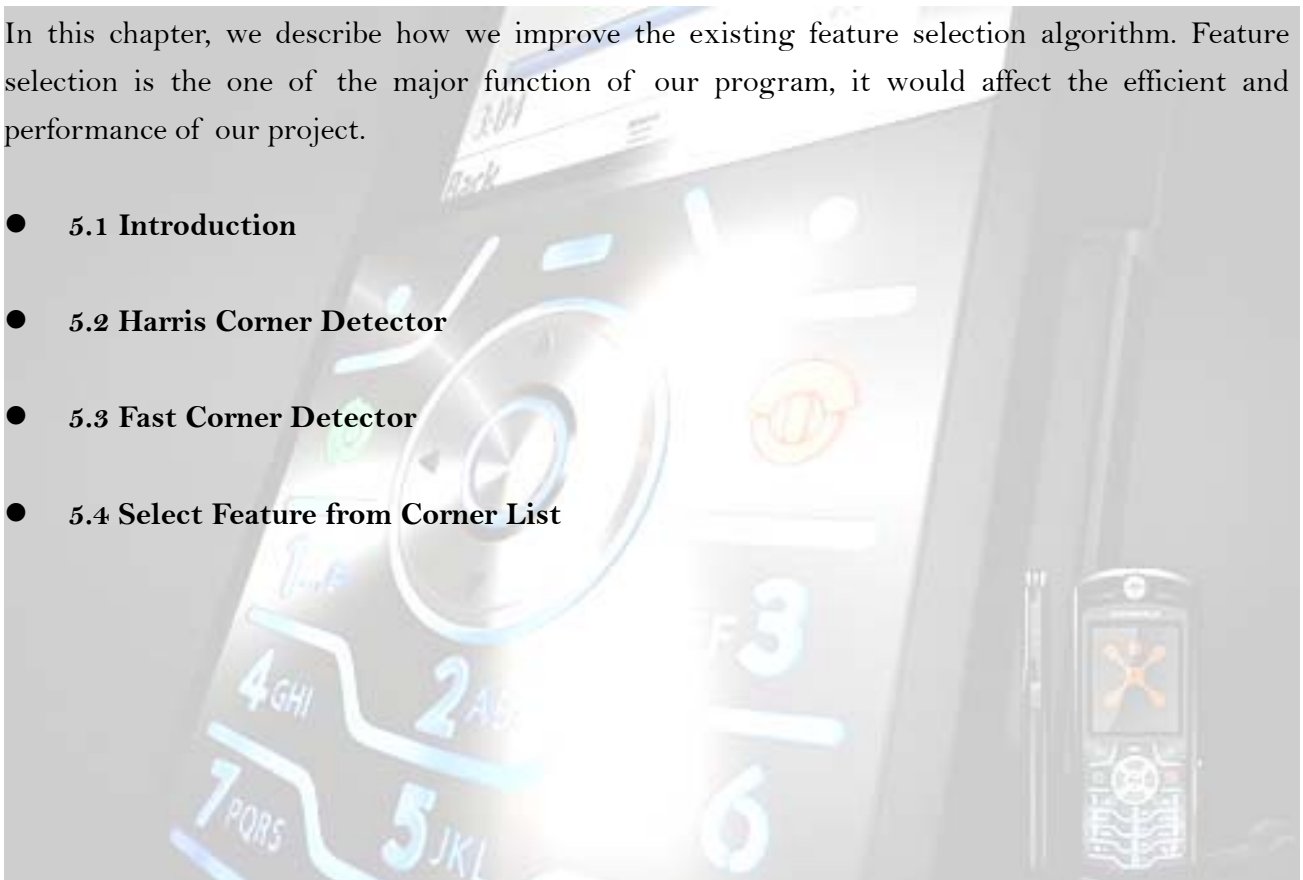
The time need to switch back to game mode from selection mode is very high, so we try to decrease the chance of causing a feature point fail during the motion tracking.

# Chapter 5

## Feature Selection Improvement

In this chapter, we describe how we improve the existing feature selection algorithm. Feature selection is the one of the major function of our program, it would affect the efficient and performance of our project.

- **5.1 Introduction**
- **5.2 Harris Corner Detector**
- **5.3 Fast Corner Detector**
- **5.4 Select Feature from Corner List**



## **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.

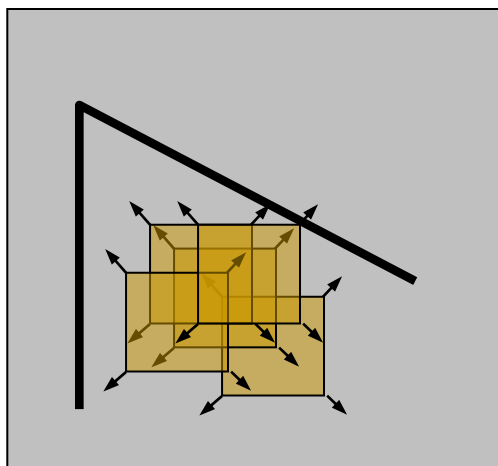
There are two conditions of a “Good” Feature:

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 c a corner. It can record the correct direction of the movement, so it can increase the block matching accuracy.

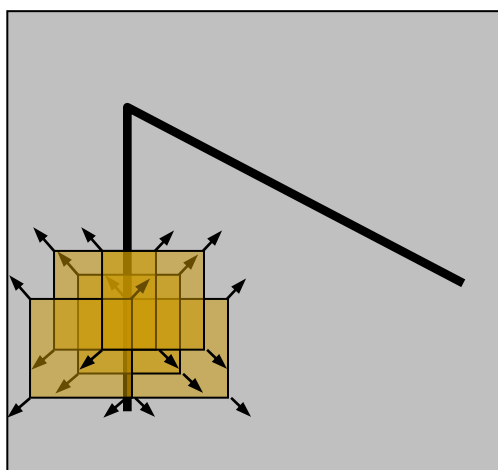
As we mention above corner is a requirement of a good feature, a corner detector to detect the possible corners from the picture is very useful in the Feature Selection Algorithm.

## **5.2 Harris Corner Detector**

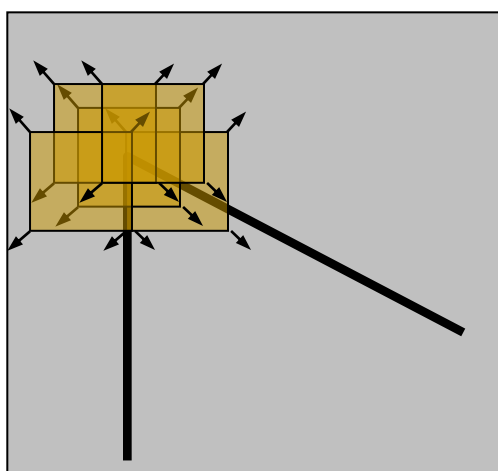
The basic idea of the Harris Corner Detector is calculate the intensity change in shifting all directions. In the “flat” region, there is no great change of intensity in all the direction. In the “edge” region, there is no considerable change of intensity across the edge direction. At the “corner” region, 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  $2 \times 2$  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  $\lambda_1, \lambda_2$  of  $M$ , we can identify the region is flat, edge or corner.

In the flat region, both the  $\lambda_1$  and  $\lambda_2$  are small and  $E$  remains constant in all the direction. In the corner region, one of  $\lambda_1$  or  $\lambda_2$  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  $\lambda_1$  and  $\lambda_2$  are large and  $E$  has very significant increase in all the directions.

The calculation of eigenvalues is computationally expensive because it involves calculation of square root. The following equation is used to calculate the eigenvalues of 2x2 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:

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

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

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

Where:

k is a empirical constant, typical values of k is range from 0.04 to 0.15.

R only depends on the eigenvalues of M and it is less computationally expensive 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.

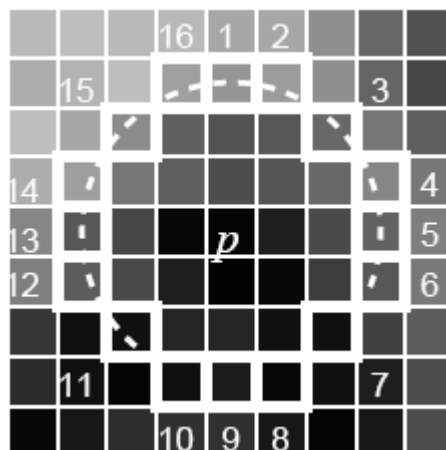
Although calculation of R is less expensive than calculation of eigenvalues, it is still very expensive for our platform, Symbian OS, which has no floating point unit.

### 5.3 Fast Corner Detector

It is suggest by Edward Rosten at his paper Machine learning for high-speed corner detection, May 2006.

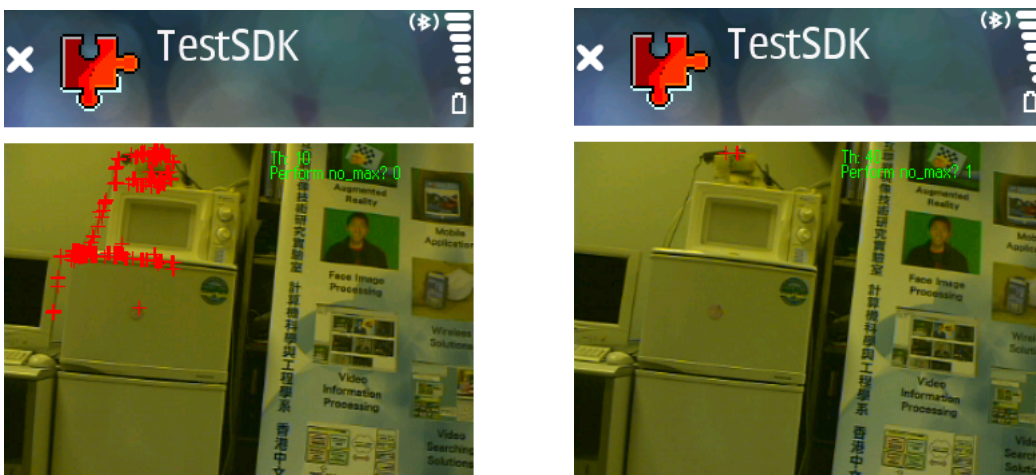
There is a class of corner detector which examines a small patch of image to see it “look” like a corner or not. In this class of corner detector, it doesn’t need a noise reduction step, such as Gaussian Filter, so it is less computation less than other corner detector. The FAST corner detector is inside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

The main conception 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 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.



The 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 ( $768 \times 288$ ) 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 used as an intermediate step in mainly computer vision algorithm. Non-maximal Suppression mean that find 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  $V$ :

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

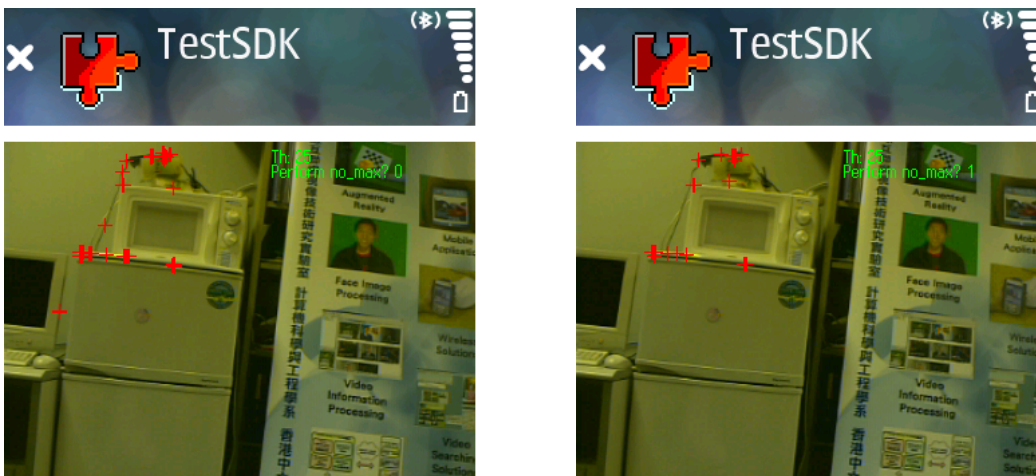
Where:

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 had 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:



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**

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 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-area. 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 I cannot find any feature.

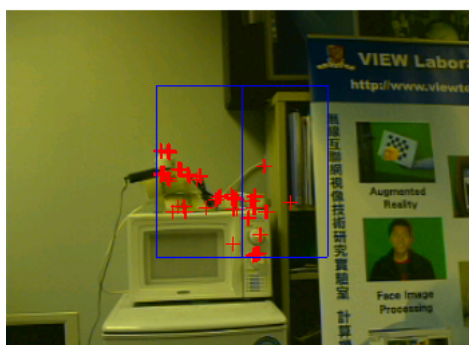
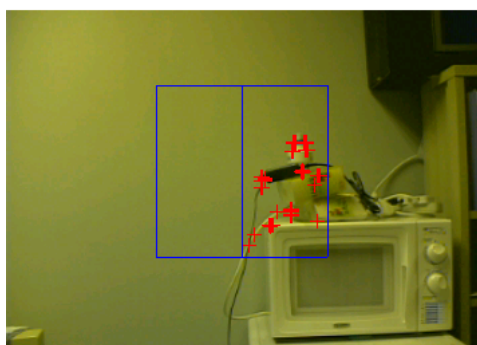


Fig a

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.



# Chapter 6

## Project Progress, Difficulties and Future Work

This chapter would briefly describes progress of our project, difficulties we faces during the project and what will we do in the future

- **6.1 Project Progress**
- **6.2 Difficulties We Face during Project**
- **6.3 Future Work**



## 6.1 Project Progress

There is progress of out Final Year Project:

September 2006	<ol style="list-style-type: none"><li>1. Decide platform of FYP</li><li>2. Study the Symbian C++ Programming</li><li>3. Study the Algorithm use in mVOTE engine</li><li>4. Familiar with the development platform of Symbian</li></ol>
October 2006	<ol style="list-style-type: none"><li>1. Write simple Symbian program</li><li>2. Write the Symbian Testing platform</li><li>3. Study basic idea of Image Processing</li><li>4. Study some corners detector algorithm</li><li>5. Study the possibility of Z motion detection</li></ol>
November 2006	<ol style="list-style-type: none"><li>1. Implement the initial approach of feature recognition</li><li>2. Implement the FAST corner detector algorithm into Symbian Platform.</li><li>3. Implement our proposed approach of feature recognition</li><li>4. Prepare FYP presentation and demonstration</li><li>5. Write FYP report.</li></ol>

## 6.2 Difficulties We Face during Project

There are four major difficulties we fast during our FYP.

1. Before doing our final year project, we have no knowledge about image processing.

So it is very difficult for us at the start of the project. In order to understand how the mVOTE work, we have spent a lot of time to studying the image processing algorithms.

2. Nokia only provide the camera plug-in for Nokia 6600 which is S60 1<sup>st</sup> Edition (Symbian OS v7.0s). As we target on the S60 3<sup>rd</sup> Edition (Symbian OS V9.1) and S60 2<sup>nd</sup> Edition FP3, we cannot use emulator for debugging and testing. We need to test our program on the Symbian phone directly. The debugging is much more difficult and time consuming than debugging on the emulator on PC.
3. There is no floating point unit in processor of mobile phone. It decreases the speed calculation and it also limits our choice to different kind of algorithm.
4. The security measure in S60 3<sup>rd</sup> Edition (Symbian OS V9.1). One of the features of S60 3<sup>rd</sup> Edition is the enhancement of security issue. The new security measure restricts the access of system resources. There are not enough documents about the how security works, we need to spend a lot of time to figure out what happen when we are debugging.

### **6.3 Future Work**

1. Increase the speed of Feature Recognition

As we mention before, the speed of feature recognition is not fast. We want to increase the speed of the feature recognition by using some method to reduce the search window. We may also find other algorithm to do the feature recognition.

## 2. Allow user to load saved featured

In our current mechanism, we will overwrite the previous saved features when the user saves another feature. We want to allow the user to select the name of the saved feature and select which feature he wants to use in the game.

## 3. Add physical calculation engine

As we are doing the augmented reality game, we want to simulate the physical environment as much as possible. We want to add the effect of physical effects, like free falling, projectile motion, etc. So we can have a more realistic game.

# Chapter 7

## References

1. Harris, C., Stephens, M.: "A combined corner and edge detector." In: Alvey Vision Conference. (1988) 147-151
2. Edward Rosten and Tom Drummond: "Fusing points and lines for high performance tracking." In: IEEE International Conference on Computer Vision (2005) 1508-1511
3. Edward Rosten and Tom Drummond: "Machine learning for high-speed corner detection" In European Conference on Computer Vision (2006) 430-443
4. R. Jin, Y. Qi, A. Hauptmann: "A Probabilistic Model for Camera Zoom Detection", Proceedings of ICPR 2002 Quebec, August 2002.
5. 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
6. 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
7. Steve Babin, Richard Harrison, Phil Northam and William Carnegie: "Developing Software for Symbian OS - An Introduction to Creating Smartphone Applications in C++", Wiley 2005
8. Jo Stichbury, "Symbian OS Explained – Effective C++ Programming for Smartphones", Wiley 2004
9. David A. Forsyth, Jean Ponce: "Computer Vision: A Modern Approach" Prentice Hall, 2002
10. Forum Nokia: "S60 2<sup>nd</sup> Edition: Getting Started with C++ Application Development", 2004
11. Forum Nokia: "S60 Platform: Porting from 2<sup>nd</sup> to 3<sup>rd</sup> Edition", 2006
12. Forum Nokia: "S60 Platform: Scalable Screen-Drawing How-To", 2006
13. Forum Nokia: "S60 Platform: Scalable UI Support", 2006
14. Forum Nokia: "S60 Platform: Source and Binary Compatibility", 2006
15. Jani Vaarala, Nokia: "OpenGL ES development on Serires 60 and Symbian",
16. [http://www.viewtech.org/html/mvote\\_tm\\_.html](http://www.viewtech.org/html/mvote_tm_.html)
17. [http://www.symbian.com/Developer/techlib/v70sdocs/doc\\_source/index.html](http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/index.html)

# Appendix



---

## Feature Selection





Appendix I describes the experiment conducted 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 Pictures under Normal Lighting Condition**
- **AI.2 Pictures under Insufficient Light Condition**
- **AI.3 Analysis**







**AI.1 Pictures under Normal Lighting Condition**

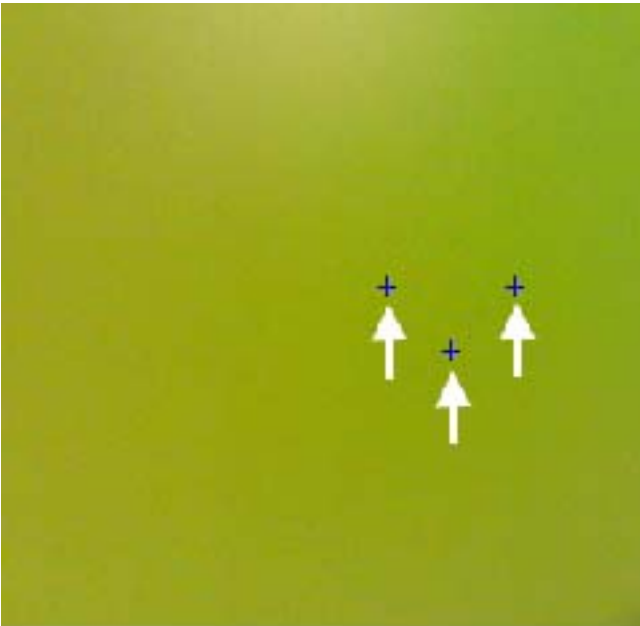


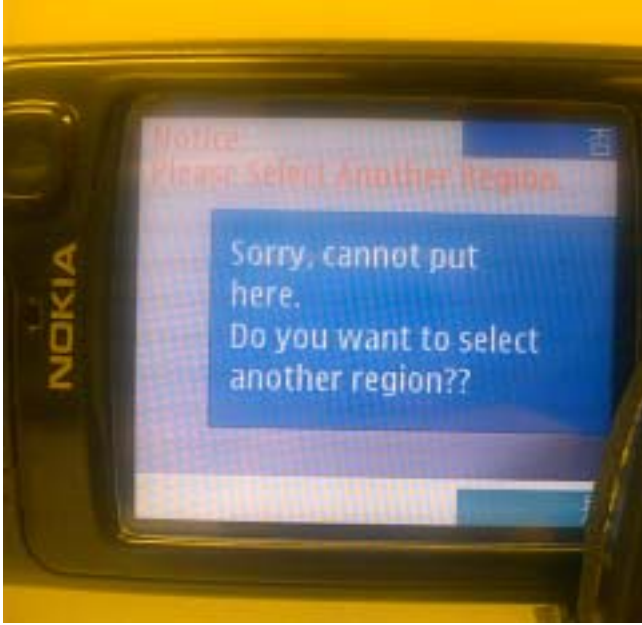
Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
 <p>This image shows a computer monitor and keyboard on a desk. Several feature points are marked with blue crosses and white arrows. The arrows point to various locations on the monitor's bezel and the keyboard, indicating the features selected by the MVOTE engine.</p>	 <p>This image shows the same computer monitor and keyboard scene. Feature points are marked with blue crosses and white arrows. The arrows point to the corners and edges of the monitor and keyboard, indicating the features selected by the Fast Corner algorithm.</p>
 <p>This image shows a poster on a wall. Several feature points are marked with blue crosses and white arrows. The arrows point to various locations on the poster, indicating the features selected by the MVOTE engine.</p>	 <p>This image shows the same poster scene. Feature points are marked with blue crosses and white arrows. The arrows point to the corners and edges of the poster, indicating the features selected by the Fast Corner algorithm.</p>

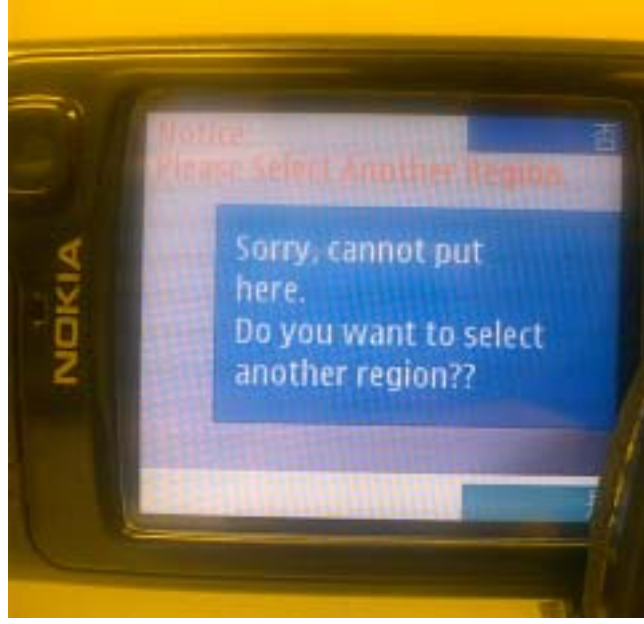

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	
	

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
 <p>A photograph of a wall-mounted interface for the MVOTE Engine. It features a yellow triangular warning sign with a black lightning bolt and the text 'Computerized Voting' below it. To the right is a rectangular panel with the number '101A' and '101A' below it. White arrows point to blue plus signs on the sign and panel, and another white arrow points to a blue plus sign on the sign.</p>	 <p>A photograph of a wall-mounted interface for the Fast Corner system. It features a yellow triangular warning sign with a black lightning bolt and the text 'Computerized Voting' below it. To the right is a rectangular panel with the number 'T01A' and 'T01A' below it. White arrows point to blue plus signs on the sign and panel, and another white arrow points to a blue plus sign on the sign.</p>
 <p>A photograph of a wall-mounted interface for the MVOTE Engine. It features a yellow triangular warning sign with a black lightning bolt and the text 'Computerized Voting' below it. To the right is a rectangular panel with the number '101' and '101' below it. White arrows point to blue plus signs on the sign and panel, and another white arrow points to a blue plus sign on the sign.</p>	 <p>A photograph of a Nokia mobile phone screen displaying a message. The text on the screen reads: 'Notice: Please Select Another Region.' followed by a blue box containing the text: 'Sorry, cannot put here. Do you want to select another region??' The Nokia logo is visible on the left side of the phone.</p>



Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
 <p>The image shows a photo album on a desk. The MVOTE engine has selected four features, marked with blue crosses. White arrows point to these features: one on the left edge, one on the top edge, one on the bottom edge, and one on the right edge. This indicates a selection of corner features.</p>	 <p>The image shows the same photo album. The Fast Corner engine has selected four features, marked with blue crosses. White arrows point to these features: one on the top edge, one on the right edge, one on the bottom edge, and one on the left edge. This indicates a selection of corner features.</p>
 <p>The image shows a photo grid on a desk. The MVOTE engine has selected three features, marked with blue crosses. White arrows point to these features: one on the left edge, one on the bottom edge, and one on the right edge. This indicates a selection of edge features.</p>	 <p>The image shows the same photo grid. The Fast Corner engine has selected four features, marked with blue crosses. White arrows point to these features: one on the top edge, one on the right edge, one on the bottom edge, and one on the left edge. This indicates a selection of corner features.</p>

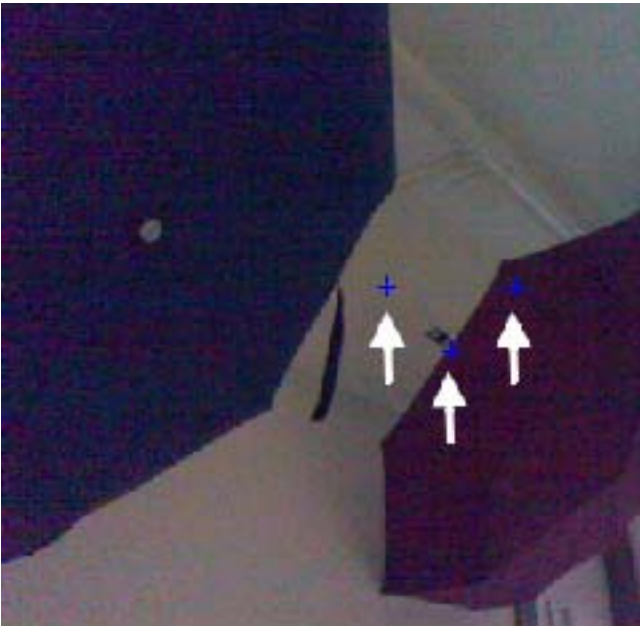

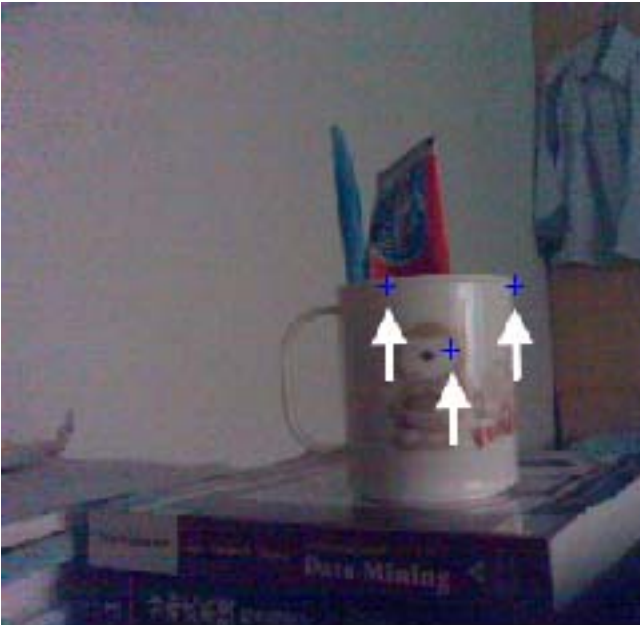

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	
	

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	
	





Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
 <p>The image shows a box of Stropsils Vitamin C 1000mg. The MVOTE engine has selected several features, indicated by white arrows and blue crosses. The arrows point to the top corners and the center of the box, while the crosses mark the corners of the 'Stropsils' text.</p>	 <p>The image shows the same Stropsils box. The Fast Corner engine has selected features at the corners of the box, indicated by white arrows. A blue cross is placed at the top-right corner of the box.</p>
 <p>The image shows a tissue box with a cartoon character on top. The MVOTE engine has selected features at the corners and center of the character, indicated by white arrows and blue crosses.</p>	 <p>The image shows the same tissue box. The Fast Corner engine has selected features at the corners of the character, indicated by white arrows. A blue cross is placed at the top-right corner of the character's head.</p>


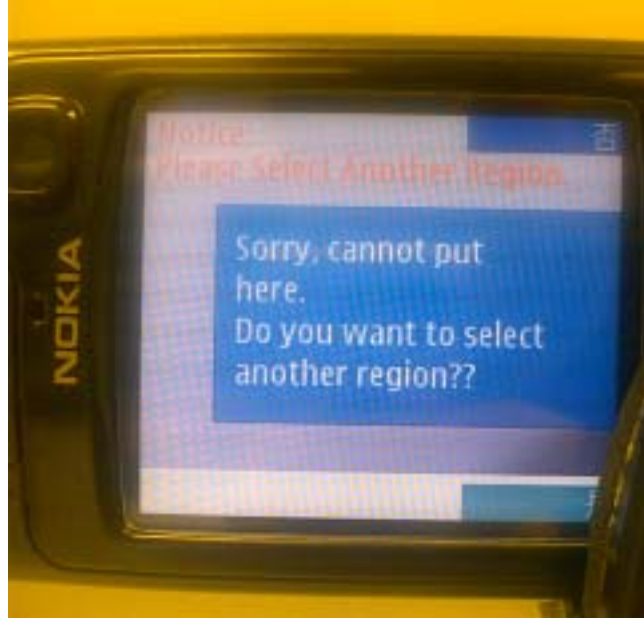

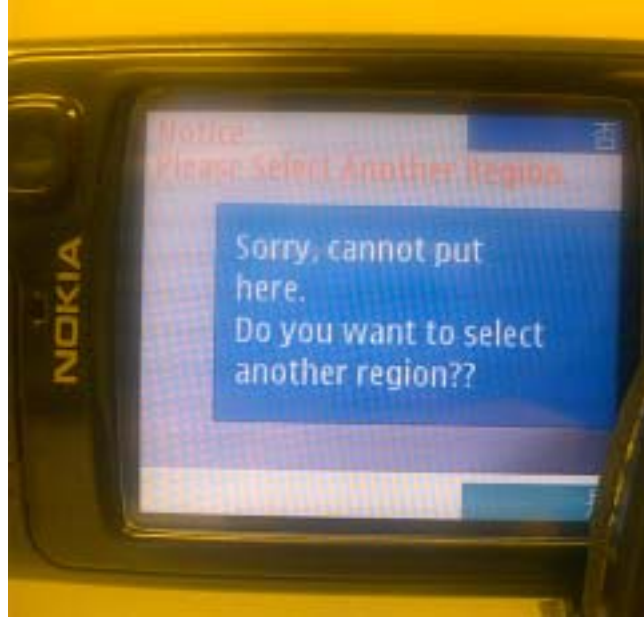



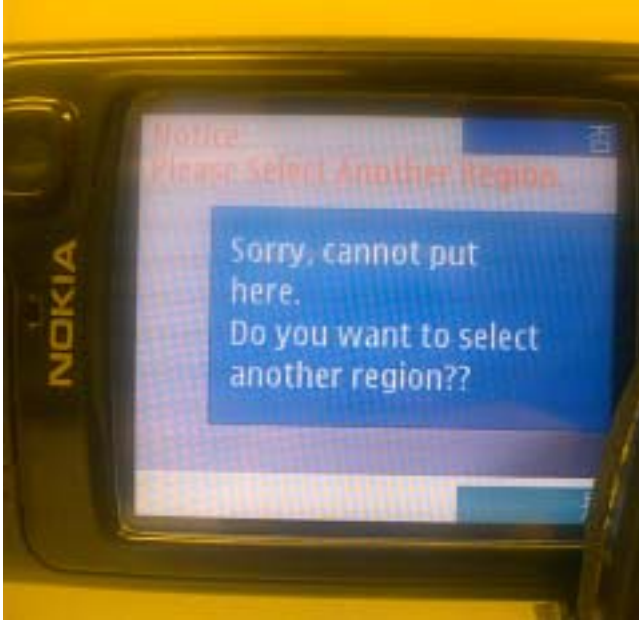

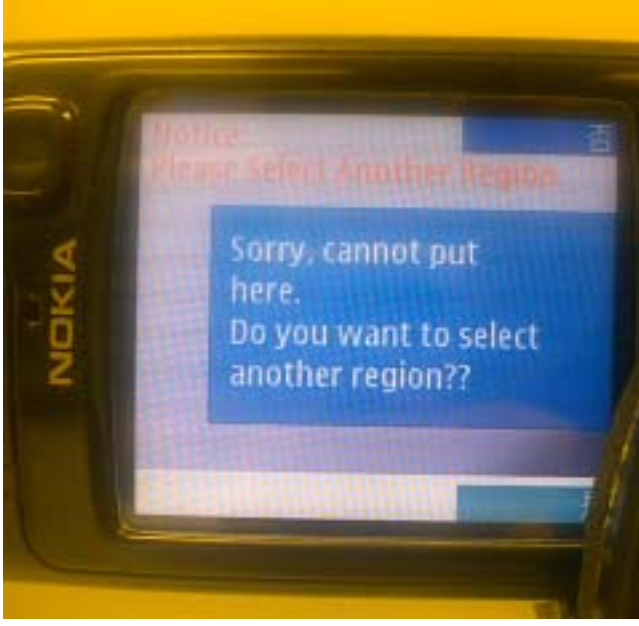
**AI.2 Pictures under Insufficient Light Condition**

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
	
	

Feature Selection in MVOTE Engine	Feature Selection in Fast Corner
	
	

### **AI.3 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.

Features should be selected as descriptive as possible

The chosen block should facilitate the block matching algorithm and increase the accuracy of the algorithm

Feature extraction should be robust enough

In order to facilitate the objective, corner detection is used as feature selection in motion tracking. 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 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  $I(x,y)$  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:

<b>0</b>	<b>1</b>	<b>0</b>
<b>1</b>	<b>-4</b>	<b>1</b>
<b>0</b>	<b>1</b>	<b>0</b>

<b>1</b>	<b>1</b>	<b>1</b>
<b>1</b>	<b>-8</b>	<b>1</b>
<b>1</b>	<b>1</b>	<b>1</b>

<b>-1</b>	<b>2</b>	<b>-1</b>
<b>2</b>	<b>-4</b>	<b>2</b>
<b>-1</b>	<b>2</b>	<b>-1</b>

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			

$$\begin{aligned} & -1 \times 68 + -1 \times 62 + -1 \times 66 + -1 \times 120 + 8 \times 33 + -1 \times 37 + -1 \times 2 + -1 \times 65 + -1 \times 61 \\ & = -227 \end{aligned}$$

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 does not work at optimum. Most of the time, this algorithm would select some flat region as the 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 determinant 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**

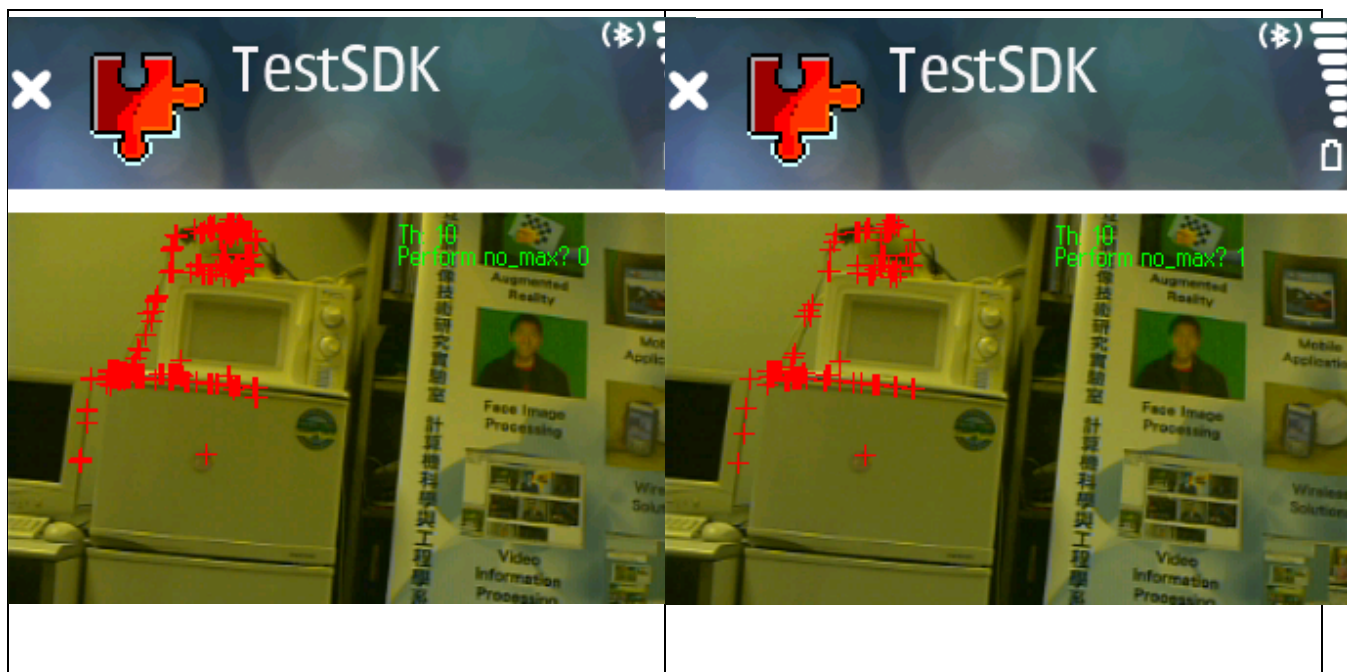
## Parameter Adjustment for Fast Corner Algorithm

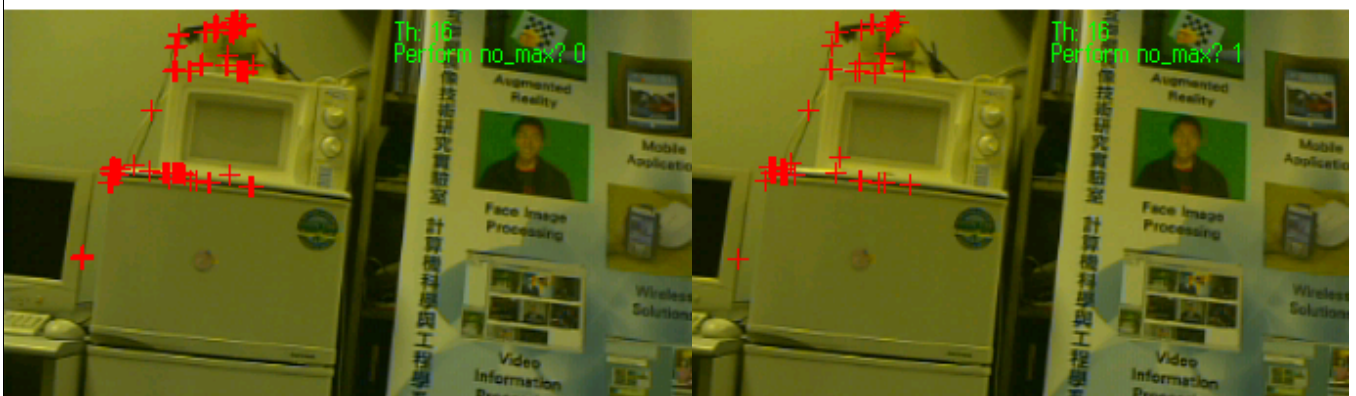
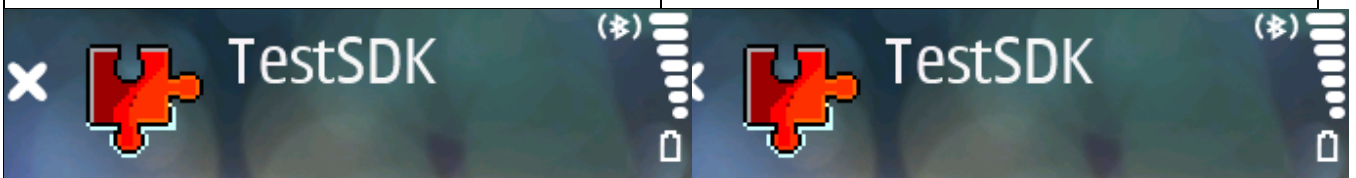
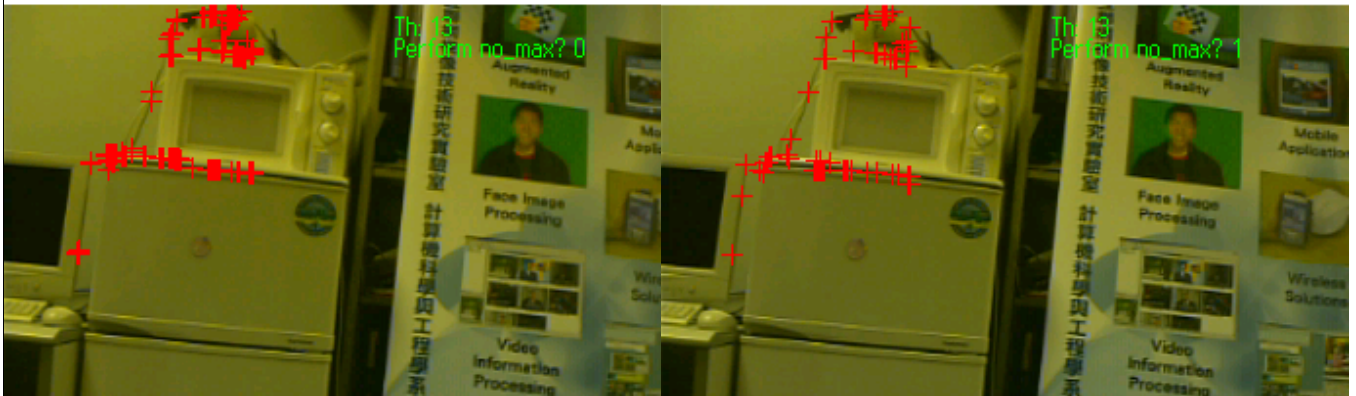
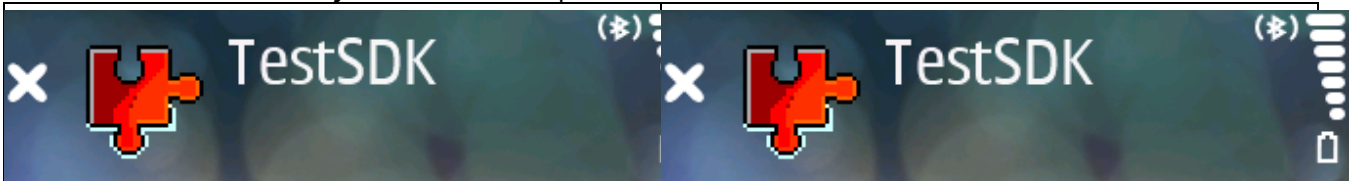
Appendix II describes the experiment conducted for testing how the corner detection algorithm would be affected by adjusting its parameter – barrier. This experiment also shows the different result obtained when running Fast Corner Algorithm with accessory function non-maximal suppression or without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

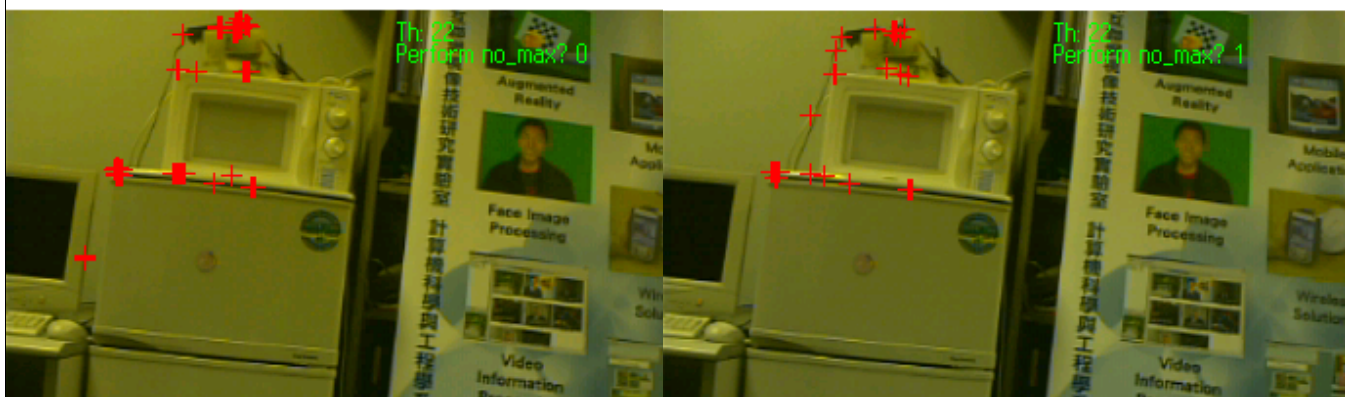
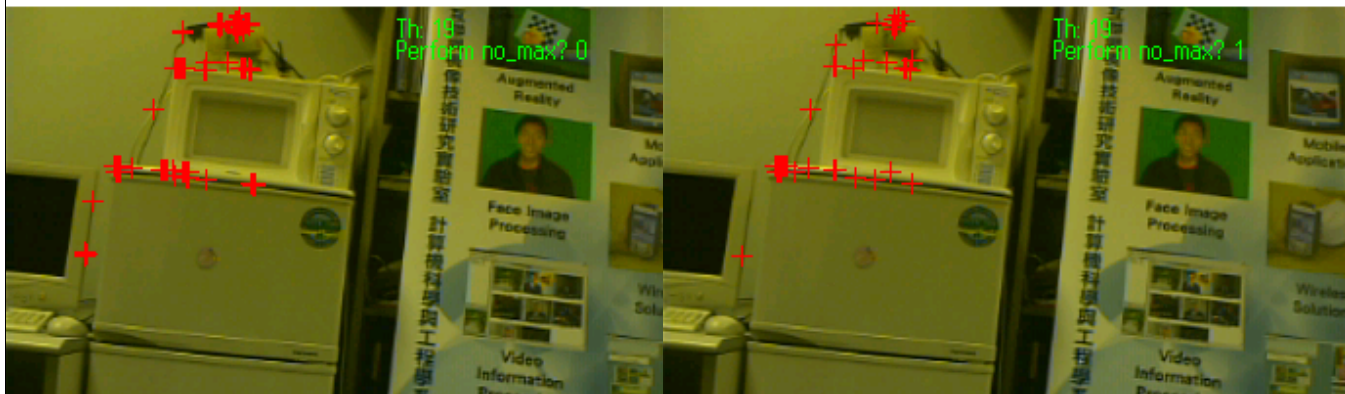
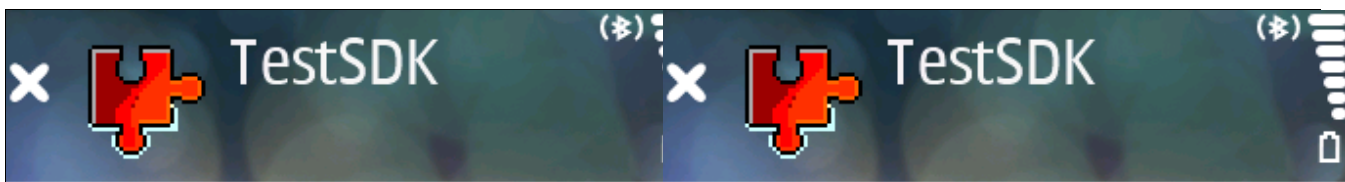
- **AII.1 Experiment Result**
- **AII.2 Analysis**

### AI.1 Experiment Result

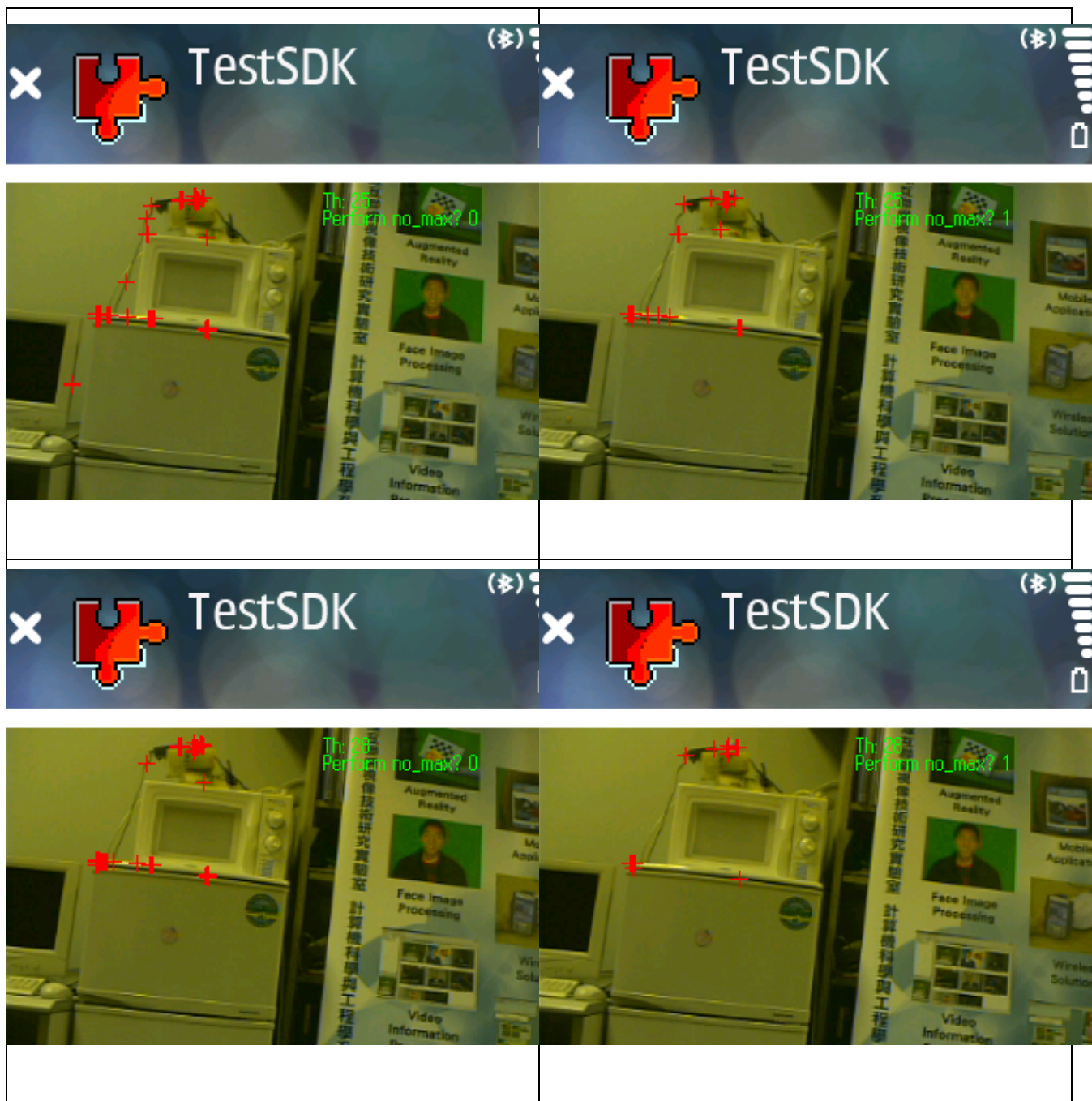
This section would show how different value of the parameter affects the number of corner found by the Fast Corner Detection Algorithm. The corner(s) would be marked as "+" in Red Color in the graphs shown below. The parameter value would be displayed in Green Color in form of "Th: z" where z is the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

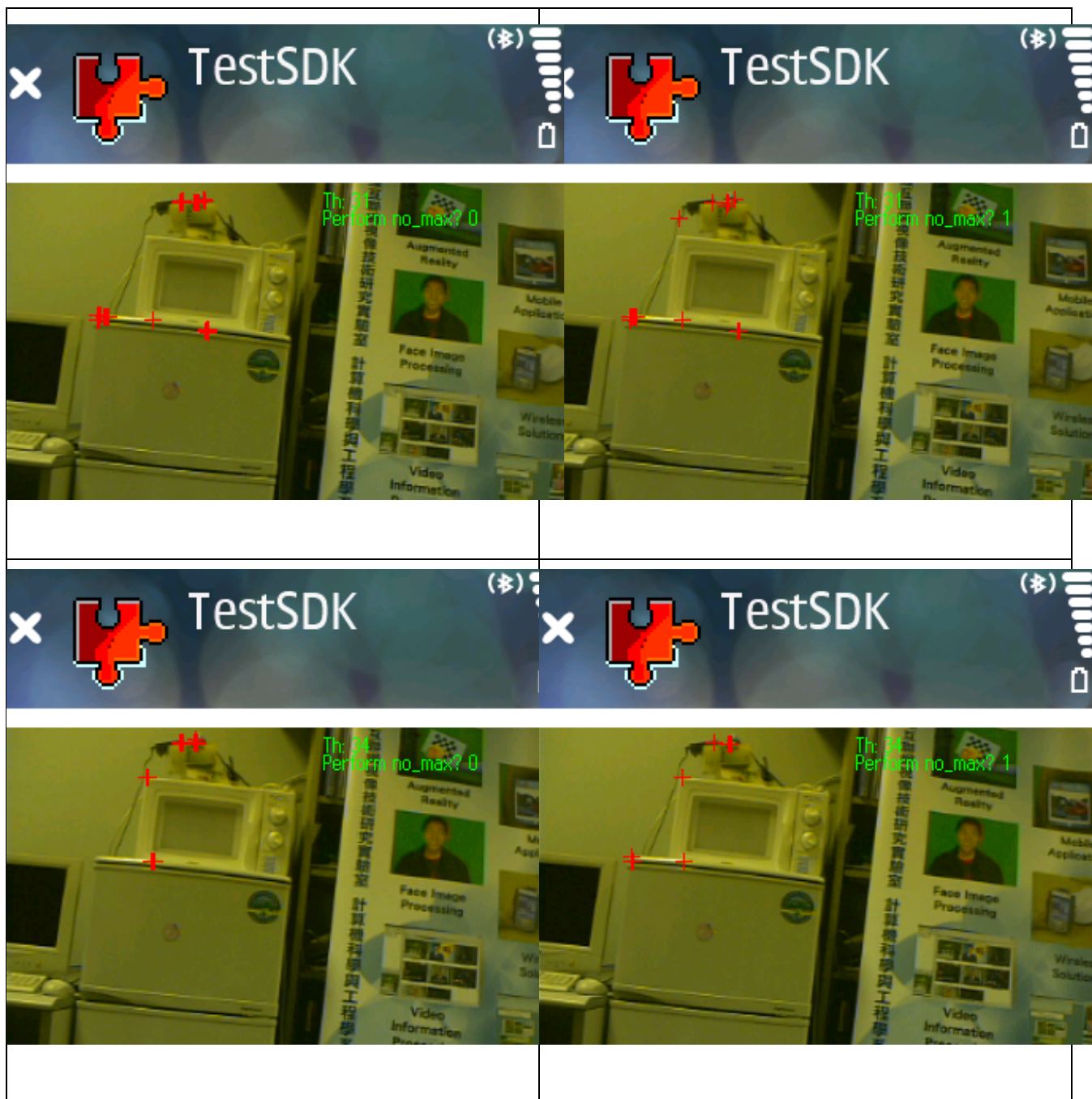


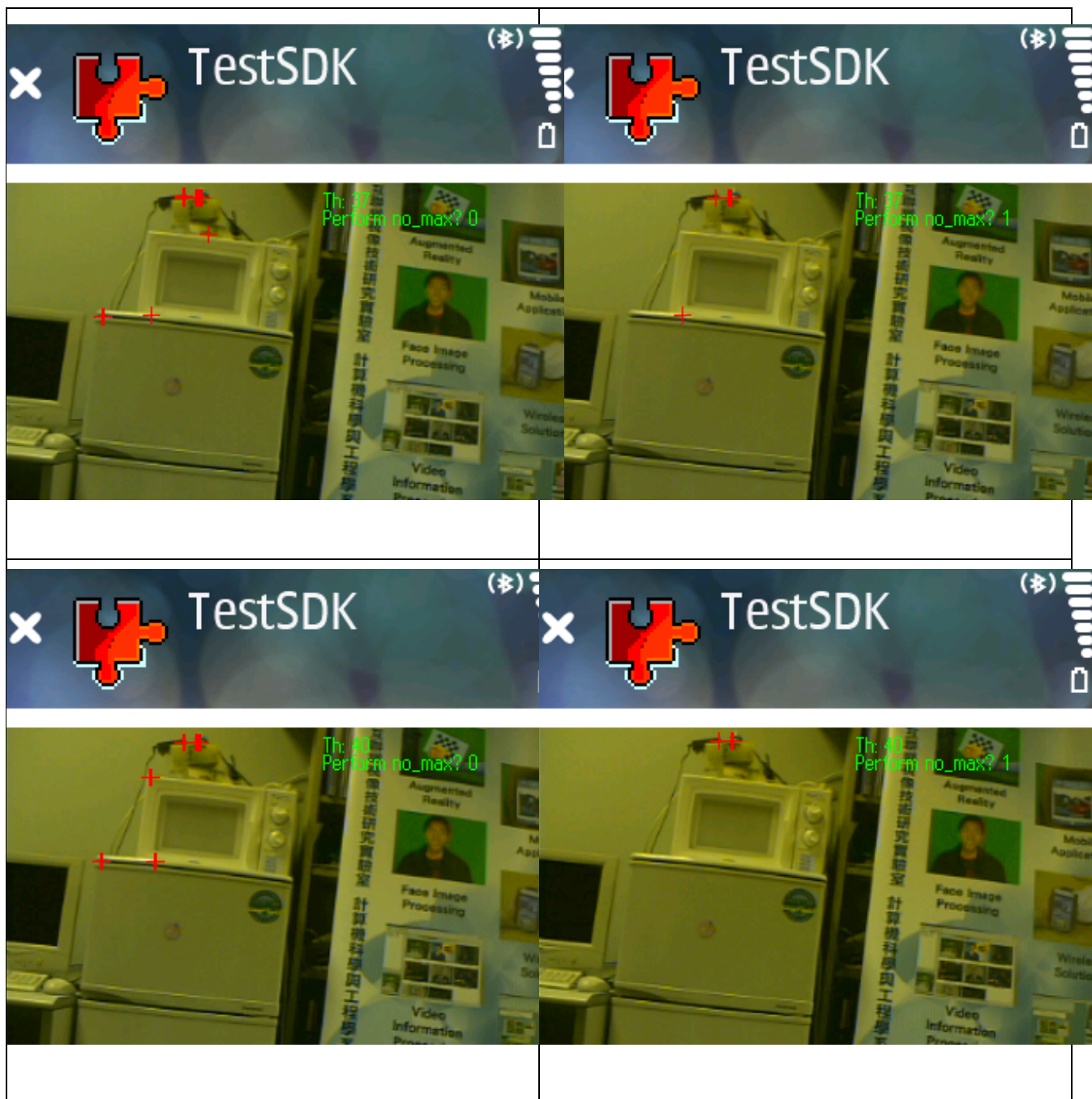












## **III.2 Analysis**

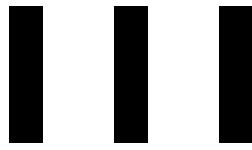
The above graphs show the different in term of corner selection versus value change in the parameter, barrier, in the algorithm. The results obtained on the left column are those using Fast Corner algorithm without the use of Non-maximal Suppression technique. While running Fast Corner algorithm with 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



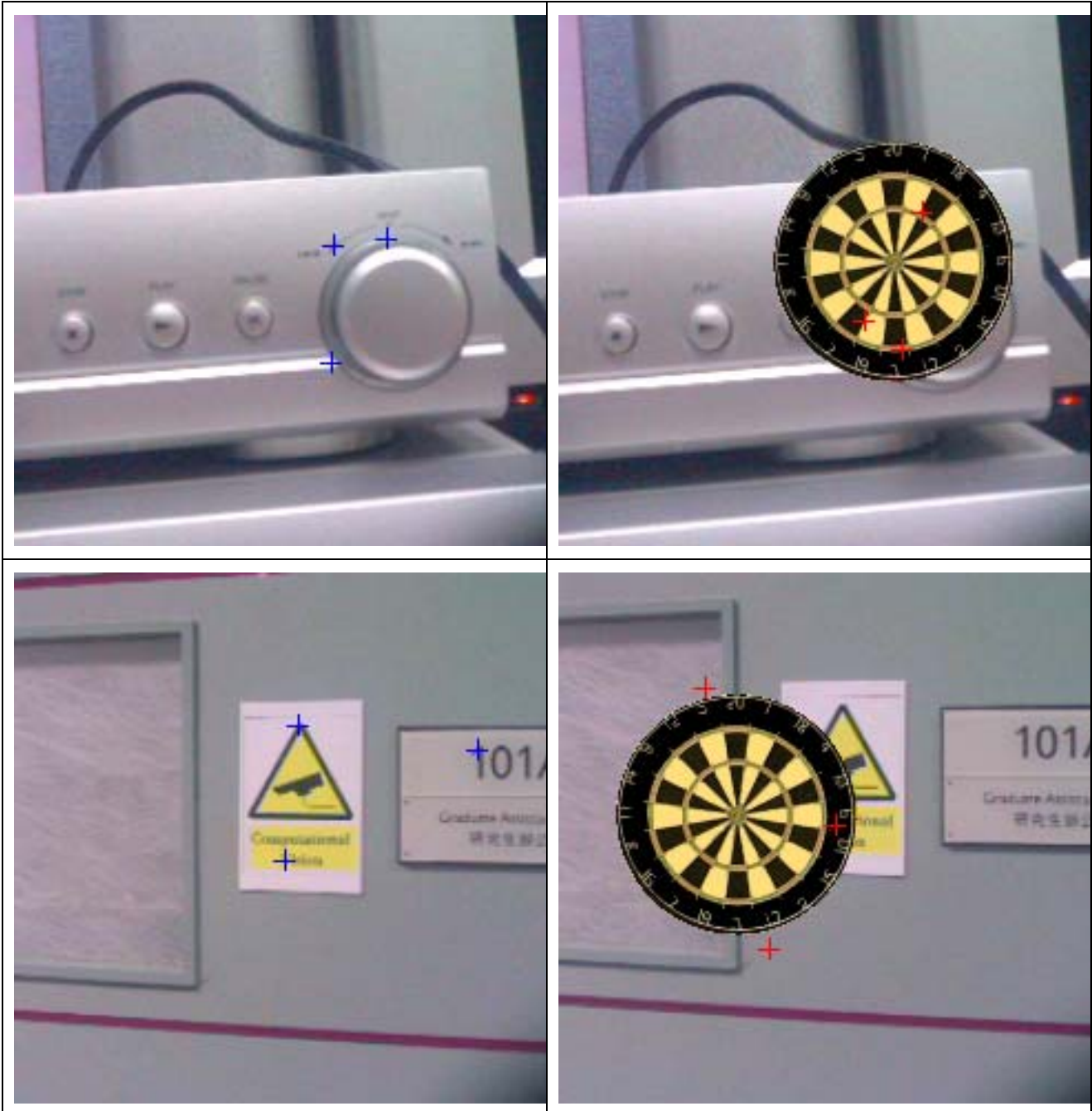
---

## Accuracy of Feature Blocks Finding for Algorithm 1

Appendix III describes the experiment conducted for testing how the saved block is discovered back in the video frame.

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

### AIII.1 Experiment Result







### **AIII.2 Analysis**

The above 6 sets of photos show the result of block matching using our initial algorithm. The markers in blue color of the left hand side are the feature (corner) selected by the Fast Corner Algorithm (Those corners would be saved in the memory card for future usage). The markers in red color of the right hand side demonstrate the matched corner after running the existing MVOTE Engine. In the ideal case, both the red and blue markers should be overlapping. This means that the matched corner is exactly same as the corner found by the Fast Corner algorithm. In other words, the accuracy is 100%.

However, as you may observe from the above photos. The matching accuracy is very bad. In nearly all the cases, the matched corner deviate much from the saved one. In this algorithm, our first step would be using Fast Corner algorithm to seek out 3 key features from the user selected region. Since the Fast Corner just only return a set of Corner co-ordinate, we then produce feature blocks in the following way:

1. The corner (feature) is set to be the center of the feature block.
2. Extends 12 pixels from the corner (i.e. center of the feature block) in Up, Down, Left, Right four directions.

This process would produce 3 25x25 pixels features blocks. These blocks would be saved in the mobile phone for future matching usage. (To allow the program to process memory of external environment)

During the initialization of game playing, the program would load the saved feature blocks. It would try to match those blocks with the current video frame taken from the mobile phone camera.

Suppose originally we have taken the following picture and the red marker indicate the point (corner) proposed by Fast Corner Algorithm. We would extract a 25x25 pixel feature block with the proposed corner being the center and save it into the memory card.



Suppose after some time, we have to find the feature block back from the video frame (on the right hand side). The blue markers indicate the possible regions where the feature block would be mapped to. The save feature block would be mapped to those region because they have the same Sum Square Difference which are the minimum. This also explains why this initial algorithm would not work well. It is because the save feature blocks are just 25 by 25 pixels which is not informative enough. When matching on the video frame, there maybe so many blocks with similar Sum Square Different with the saved one. Since our algorithm takes the block with minimum Sum Square Different, it may easily select a wrong one mistakenly. As we observe this disaster, we modify this algorithm and lead the second algorithm with the experiment results in the Appendix IV.

# Appendix **IV**

---

## Accuracy of Feature Blocks Finding for Algorithm 2

Appendix III describes the experiment conducted for testing how the saved block is discovered back in the video frame.

- **AIV.1 Experiment Result**
- **AIV.2 Analysis**
- **AIV.3 Miscellaneous**



### AIV.1 Experiment Result



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.

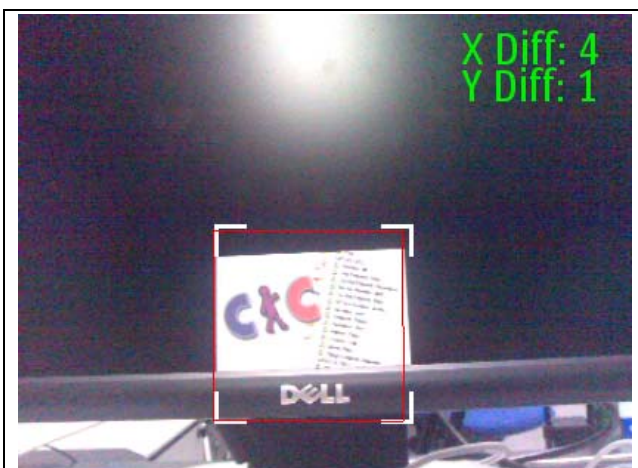


Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.





Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.





Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



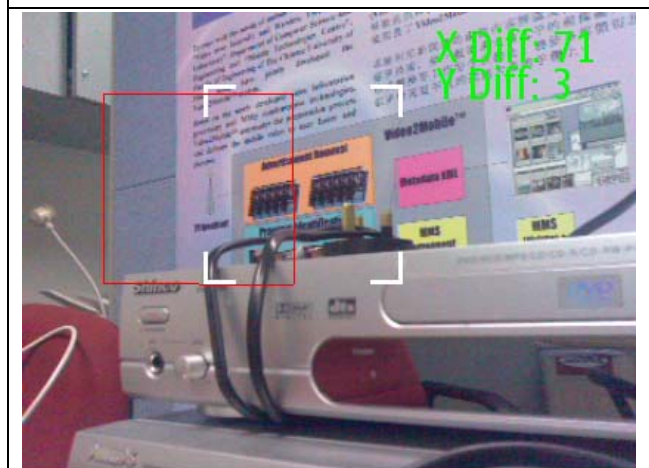
Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.

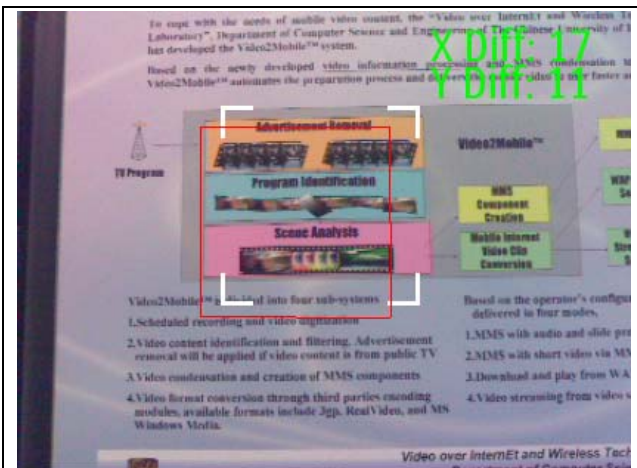


Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.

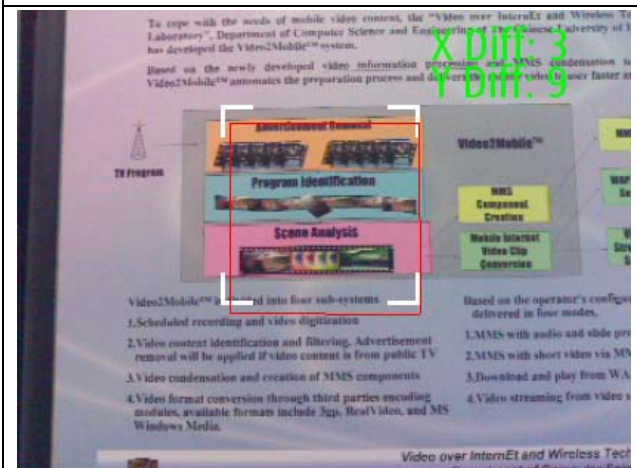




Please Select Region For Putting Dart Board.



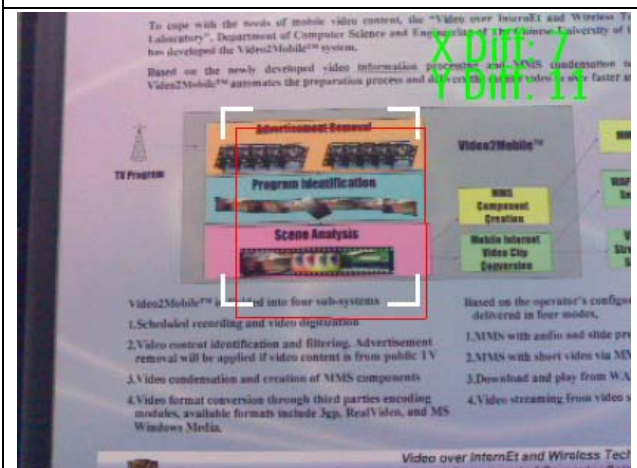
Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.



Please Select Region For Putting Dart Board.

## **AIV.2 Analysis**

There are two boxes appear in the above graphs. There is a box in white color which indicates the location where the users were intended to put the dart board. There is another box in red color showing the location where the block matching algorithm find the best matched image. In other words, the region of the Red color box indicates the location where dart board would actually be put. There are also two lines of messages on the top right corner region. The "X Diff:" message indicates the difference in x axis between the upper left corner of the Red box and the upper left corner of the White box. The "Y Diff:" message is defined in a similar way as the "X Diff:" but in y direction.

The most ideal case would be having both "X Diff:" and "Y Diff:" value as 0. This means that the matched block is exactly same as the one selected by users. In addition, having a low "X Diff" and "Y Diff" values would indicate a higher accuracy of block matched. In short, have a low "X Diff" and "Y Diff" values means having a higher probability that actual dart board would be put in the users selected region.

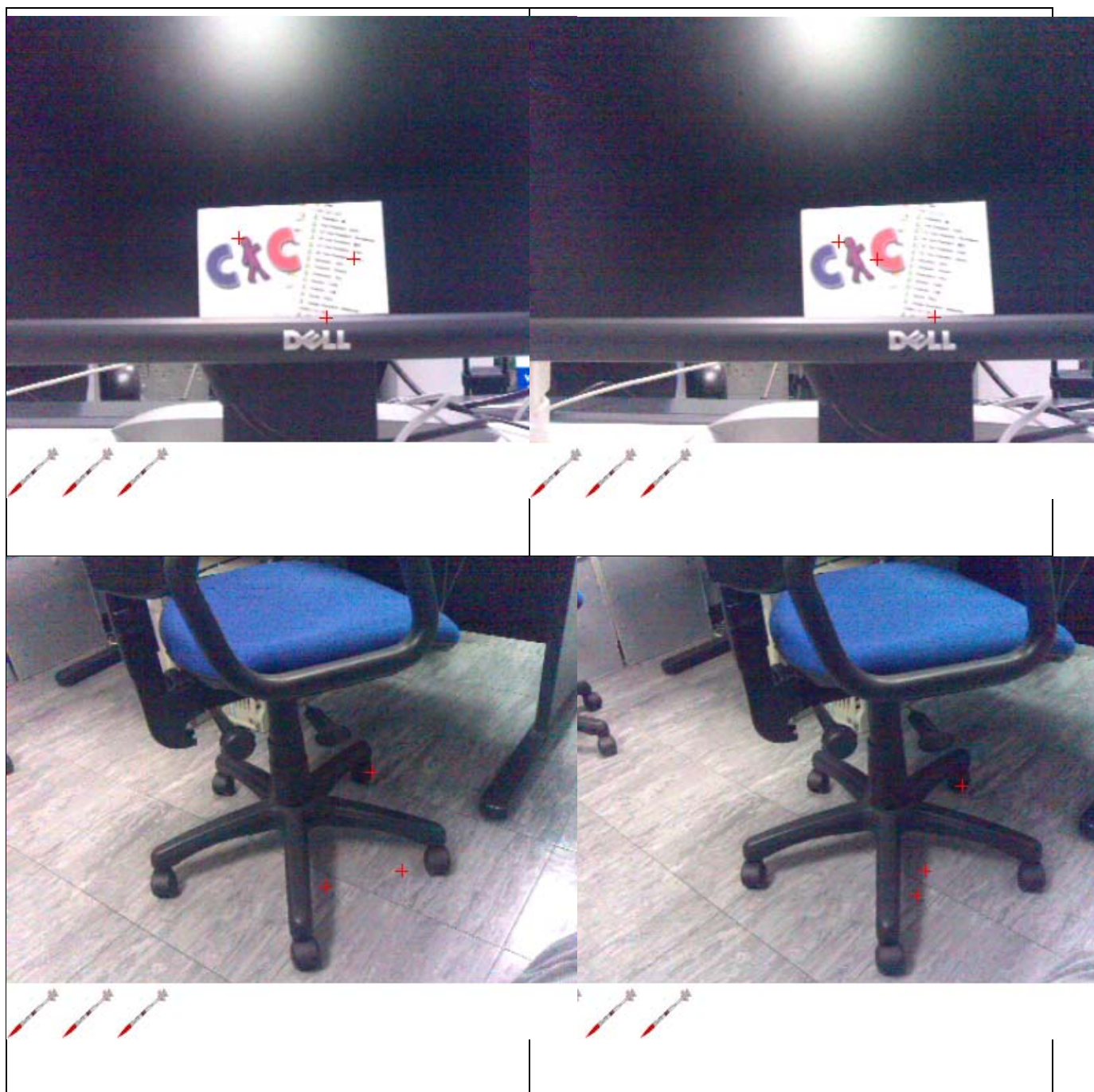
In the above experiment, we took totally 10 set of sample photos. Each set consists of 3 trial runs. It is observed that each run would produce a slightly different result though human eyes may not be able to tell the difference. The different may come from a small vibration during image capturing or maybe due to a small change in light intensity. In general, as the experiment shown, our proposed algorithm works fine.

However, the major disadvantage of our approach is the performance degradation. In the current situation, we make use of the existing block matching algorithm in the MVOTE Engine. We amended the MVOTE Engine such that the block matching would be performed on the whole video frame in a brute force manner. This leads to a performance issue. To perform the block matching, the whole screen would be divided into several blocks. Sum Square Different (SSD) would be calculated for each block. The one with minimum SSD value would be declared as the matched block. As the computation is quite intensive, it affects our program's performance negatively. To improve this area, the previous chapter has mentioned some future works for performance gain.

### AIV.3 Miscellaneous







The above 3 sets of photos shows our motion tracking result using our Algorithm 2. The left columns show the points (corners) returned by the Fast Corner Algorithm. The points in the right columns show the matched point after some horizontal movement. As you can see, in our approach, there are 2 points matched quite well for every set of photos. Comparing this with our experiment results from Appendix III, the results here perform much better.