THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

The Smart Phone as a Mouse

Yinghao Qin

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master of Science at the University of Waikato.

November 2006

ABSTRACT

With the development of hardware, mobile phone has become a feature-rich handheld device. Built-in camera and Bluetooth technology are supported in most current mobile phones. A real-time image processing experiment was conducted with a SonyEricsson P910i smartphone and a desktop computer. This thesis describes the design and implementation of a system which uses a mobile phone as a PC mouse. The movement of the mobile phone can be detected by analyzing the images captured by the onboard camera and the mouse cursor in the PC can be controlled by the movement of the phone.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Bill Rogers, for his valued ideas, helpful advices, patience and continuous support during the time of my study. Thanks also to the Waikato University TSG (Technical Support Group) people for providing the experiment environment.

CONTENTS

List of Figures

List of Tables

Chapter 1
Introduction

1.1 The mobile phone as a Swiss army knife

Nowadays, portable computing devices are becoming more popular in daily use, especially smart phones. Because people usually carry a phone with them at all times, it is interesting to consider alternative uses to which the phone might be put. For example, a phone has a built-in screen light, and it can therefore be used as a torch. Whilst it is not very bright and the light is not focused into a beam, it may help someone to find a key hole on a dark night. The purpose of my research is to explore ways in which we could make use of the features of a mobile device to control a computer.

From the technical view point, the computing capability of embedded devices is steadily increasing. Phones equipped with Bluetooth technology are available, and permit rapid communication with other computing devices. The onboard camera is becoming a standard part of most current mobile phones. It can be used as an alternative to working with a stylus on a tiny screen. Also, mobile phones have similar size and shape to a mouse. These provide the possibility of creating such an application to provide wireless services between a smart phone and a computer. If the idea could be implemented, it could bring a new convenient tool to computer users. Furthermore, if some image processing were used in such a system, the onboard camera in the smart phone could play the role of input device to the computers, which could bring some more interesting features to those mobile devices. For example, a user could use the smart phone as a wireless mouse and handwriting board, sitting wherever he likes. Gently touching his mobile screen by stylus, he can easily control the projector screen or point out the notes for students through the media of a computer.

Our research question then was, could we develop a system which would provide some functionalities based on the connection of computers and handheld devices. For example, the mobile phone could control music player, the slides demonstration or be used as the computer mouse. In this project, we experimented with using a mobile phone as a mouse to control a laptop or desktop PC. We tested our ideas not only of the possibility but also of the usability. The framework of our system includes smart phone touch screen input, onboard camera handling in smart phone and Bluetooth communication between the PC and the mobile device.

1.2 The structure of the thesis

There are seven chapters in this thesis. This chapter explains the original idea and the purpose of this project. In the next chapter, some other projects or related articles will be introduced. The chapter three is intended to introduce the experimental environment and the knowledge required for building the system. In chapter four, our system design is presented and in the following chapter the implementation of the design is explained in detail. In chapter six, some conclusions are drawn from the experiment. In the last part, chapter seven, some advice and possible further improvements are listed.

Chapter 2
Review of literature

There are many examples in the literature in which image processing is used to support navigation tasks of various kinds. In this chapter we review some of this work that is relevant to the project. In most situations where image processing has been used for navigation, an important goal has been to achieve real time performance and low latency – at least 10 frames per second. A number of methods have been used to simplify the image processing to meet the real time criterion. Some are discussed in the review.

A number of system use Fiducial Markers positioned in the environment.

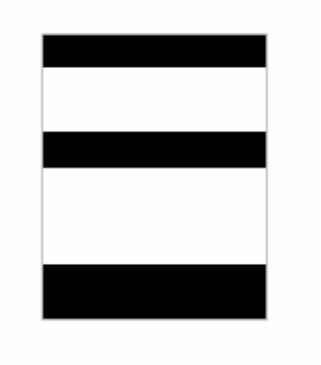


Figure 2.1: An example of Fiducial Marker

A Fiducial Marker is a large simple icon drawn in black on a white background. It is reasonably easy to recognize because of the high contrast (black/white). Also, a reasonable range of shapes is possible to identify different locations.

2.1 Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System (Kato and Billinghurst, 1999)

The system Kato and Billinghurst (1999) introduced provided a new user interface technology, Human-Human interface instead of Human-Computer interface. It

supports the idea of 3D Computer Supported Collaborative Work (CSCW), which let users interact with each other by the computer generated shared virtual object. The system structure is shown below, the figure is from their paper.

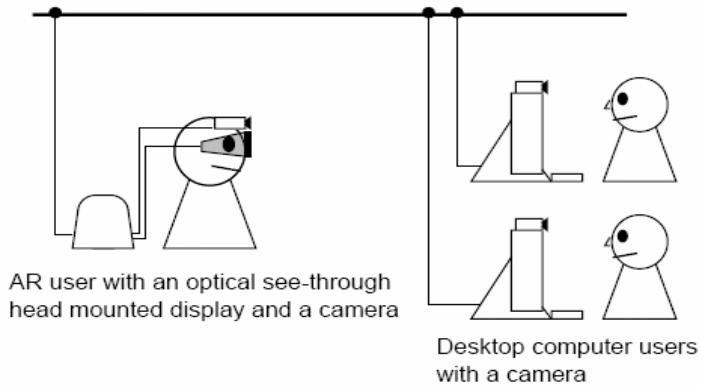


Figure 2.2, CSCW System configuration

The system employed an optical see-through augmented reality technique. The AR user could see the real world directly through the head mounted display with superimposed virtual objects. The share object was marked by six fiducial markers which were used to calculate the virtual whiteboard coordinates and all 3D relative calibration. And also there were some fiducial markers placed out of the shared whiteboard for positioning the desktop users. The video from desktop users' camera can be directly positioned on those markers. Hence, the system could provide a multi-user workplace. All users could work on the same virtual whiteboard at the same time.

Figure 2.3: CSCW AR system. The left is a user who was doing the test with the system, The right is the image the user saw, which was drawn on the HMD

They have solved two major problems for the project, "*calibration of the HMD and camera, and estimating an accurate position and pose of fiducial markers*" (Kato and

Billinghurst, 1999). The essential idea of the solution is to find the corresponding transformation matrices. The shared object coordinates then could be transformed into the camera's coordinate system, hence, the eye's coordinate system. Finally, the HMD (Head Mounted Display) screen coordinates could be calculated by the perspective projection of the objects in eye's 3D coordinates.

The experimental results mentioned in the paper showed that the model they built worked more accurately when the objects were close to the camera. When the camera was moving away from the objects, the accuracy decreased. This problem might have been caused by the size of the markers. As the outline contour of markers were involved in 3D coordinate calculation, the smaller marker would give less accuracy of the shared whiteboard 3D coordinates.

2.2 Location based Applications for Mobile Augmented Reality (Reitmayr and Schmalstieg, 2003)

This is an indoor location navigation system. The user could navigate around a building with some necessary equipment. The equipment included: a HMD; helmet with camera and inertial sensor; a notebook computer for data processing and a pair of special gloves for capturing hand posture. The structure of the building was known by the computer. More importantly, the building was labeled with optical marks (fiducial markers) which can be captured by the camera and recognized by computer. Moreover, the gloves have attached optical markers which are used to locate the position of the user and interact with the user. When a user is walking through the building, the augmented reality technology is used to help the user to navigate the building. They built two location based applications which are based on their mobile augmented reality system. One is Signpost. Signpost is a kind of location finding software. The user could know where he/she is and the shortest path would be shown on the HMD. The other software is a book retrieving system. It could give the location hints about a book.

Figure 2.4: Location based Mobile AR system. The top shows the environment; the bottom left shows the HMD, helmet and the inertial sensor; the bottom right shows the image displayed in HMD

The basic idea of this system is that optical marker tracking is used for locating position and augmented reality is used to interact with the user.

Comparing to our project, location finding could be based on a similar solution, tracking simple physical markers. Augmented reality is not necessary in our case

since we are concerned with navigation on a computer screen, not a real world situation. Also, movement is more important than the actual position to us. In their system, the position of each marker is well-known before navigating whereas, in our system, the position of markers will be unknown it brings us more flexibility as the surface used could be created at any time as long as it satisfies the surface requirement. If we define some less constrained rules, the surface could even be drawn by hand.

2.3 Rapid and Accurate Deployment of Fiducial Markers for Augmented Reality

(Schall, Newman and Schmalstieg, 2005)

This paper discussed a way to rapidly generate the virtual 3D model of environment. Several surveying areas were selected before experiments. Also, some common markers must be positioned between two adjacent surveying areas and the common markers were used for transforming the spatial relationship between markers. During the surveying, each recognized marker would be saved in 3D Cartesian coordinates converted from local polar coordinate system. After all survey points had been checked in sequence, the 3D model was also created based on the saved information.

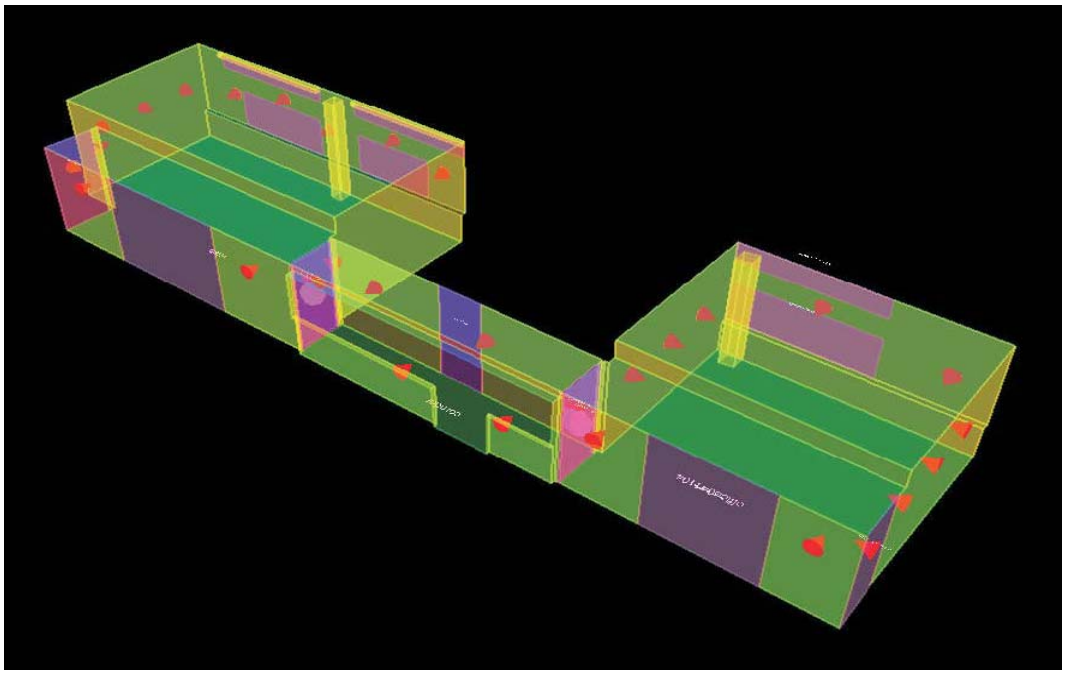


Figure 2.5: Screenshot of BAUML viewer visualizing a section of the building model

The most important technique in this paper is to use the spatial relationship to transform the different local coordinates to global coordinates. In this way, the final 3D model could be generated dynamically and simultaneously while the camera was moving. Common points are essential to this method. They were used to connect all the discrete points (objects) so that all local information could be transformed into global coordinates. The idea of bridging neighbouring areas with common markers could help to solve local-global converting problems.

The Fiducial Marker systems are generally implemented on desktop computers and dealing with the three dimensional space positioning problems. A system designed to work on a mobile phone may use similar ideas. In some systems, one dimensional or two dimensional barcodes are used with mobile devices.

2.4 Toolkit for Bar Code Recognition and Resolving on Camera Phones – Jump Starting the Internet of Things (Adelmann, Langheinrich and Floerkemeier, 2006)

Figure 2.6: Screenshots of the bar code applications

This paper described a system design and some prototypical applications. The system recognizes 1D barcodes using the onboard camera of a mobile phone and displays some relevant information retrieved from a server. As daily use of mobile phone becomes more popular, using the onboard camera of a mobile phone instead of extra scanning devices is a handy way for a user to retrieve information on a

specific product. Basically, the system was designed with two parts, the barcode recognition client and the information server. The client captures the image of the barcode printed on the package of the product and recognizes the code from the still image. The code, is then sent to the information server. The server will query some relevant information from the local database or the Internet when the retrieval request is received. Finally, the information will be sent back to the client and displayed to the user. The information server could be mobile based software or PC software.

As mentioned in the paper, the toolkit, especially the recognition algorithm, was designed for low computing resource devices like mobile phones. Although, there are some more professional barcode recognition algorithms available with more reliable results, those are still too expensive in terms of image processing time. Note that recovering bar codes from images is a different approach to that used in shops to scan products. The shop scanners work by scanning a laser beam over the product and using specialized electronics to process the result. The image analysis was based on a "*scanlines*" method. To improve the results' reliability, they used multiple scanlines instead of one line. The final result was determined by the majority of the multiple scanlines. The "*scanlines*" method could also be used in our project to process images as fast as possible. In our case, we have to deal with video stream continuously instead of some still images. The algorithm we choose must be fast enough to do real time processing.

2.5 Using camera-equipped mobile phones for interacting with real-world objects (Rohs and Gfeller, 2004)

"*The idea described in this paper is to use the built-in cameras of consumer mobile phones as sensors for 2-dimensional visual codes*" (Rohs and Gfeller, 2004). The idea of this paper is to code the items into small images and use the onboard camera to find the items by figuring out the corresponding codes of the small images though an

image recognition algorithm.

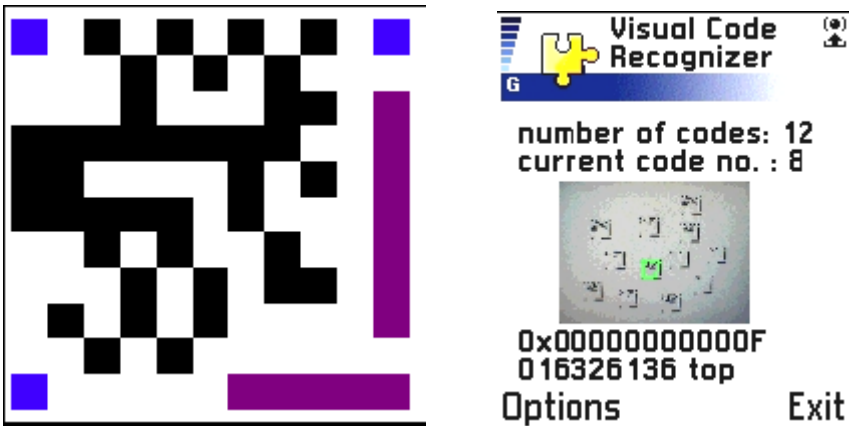


Figure 2.7: Visual code mobile phone application. The left is the visual code example and the right is the running application

To reduce the effect of low quality images, which could be distorted, blurred with strong illumination gradient affection, captured from the onboard cameras, the paper gives some ideas to follow.

1. using binary images (black and white image) instead of colour or grayscale images for further processing
2. using adaptive thresholding method to eliminate the effects of lighting level and gradients
3. the visual coding schema should be carefully designed to reduce the complexity of image recognition algorithm

Also, the steps of achieving the goal is outlined clearly.

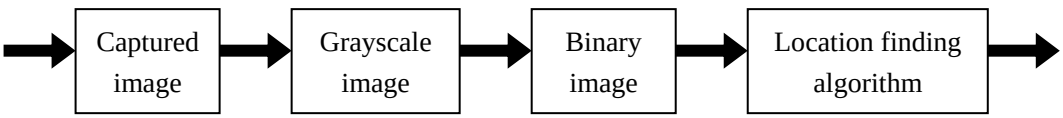


Figure 2.8: Image processing sequence

The steps shown above are very helpful for most image processing projects, especially for the low complexity images. Our goal is to have a similar way to calculate the motion of the mobile phone but a different visual coding schema using less time. For the purpose of mouse control by mobile phone, the high accuracy of code recognition is not necessary; instead, detecting the movement of the mobile is important. Therefore, the code could be simpler, even hand drawn grids could fulfill the task.

An important aspect of the implementation described above was the use of adaptive thresholding.

2.6 Adaptive image thresholding algorithm (Fisher, Perkins, Walker and Wolfart, 1994)

This paper talks about adaptive thresholding. Thresholding is a way to segment a multi-value image to a two-value image. Changing "*The intensity values above a threshold to a foreground value and all the remaining pixels to a background value*." (Fisher, Perkins, Walker and Wolfart, 1994). Thresholding is the most widely used method to generate a binary image for further processing. The simplest way of using thresholding is to calculate the average value of all pixels in an image and then comparing to each pixel with the average. However, in most cases, the light is not evenly spread over the surface. Sometimes, a strong illumination gradient could affect the quality of the thresholding results greatly, which could stop further processing. Some examples from a page image show the effect of illumination gradient.

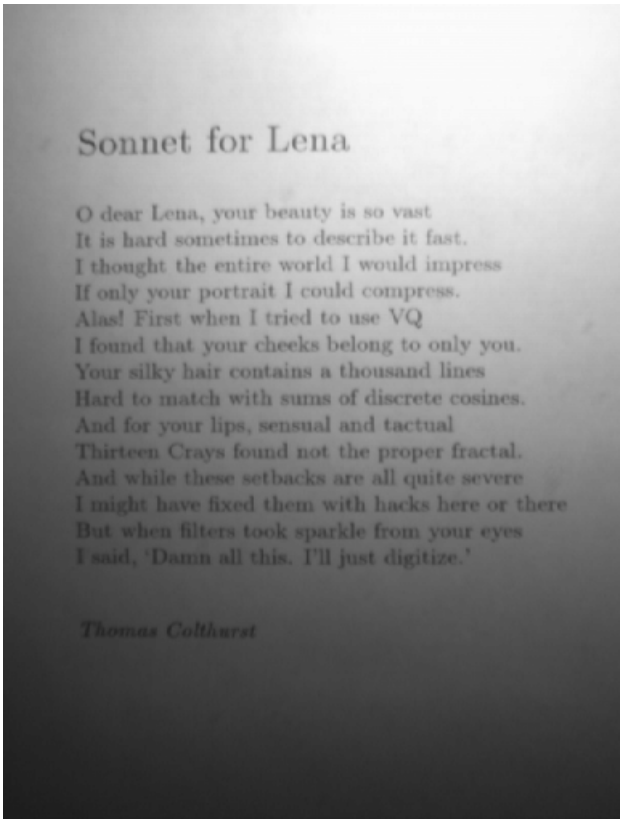


Figure 2.9: The paper with strong illumination gradient

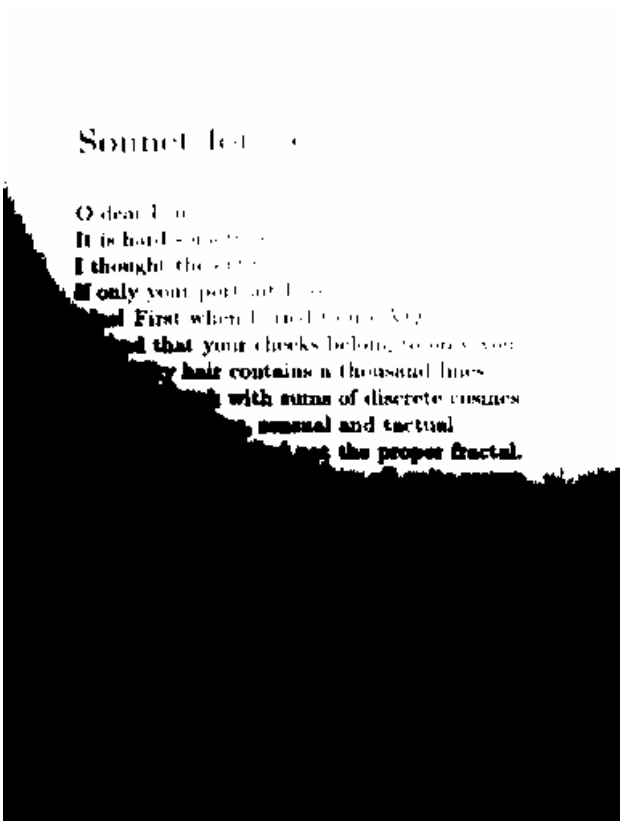
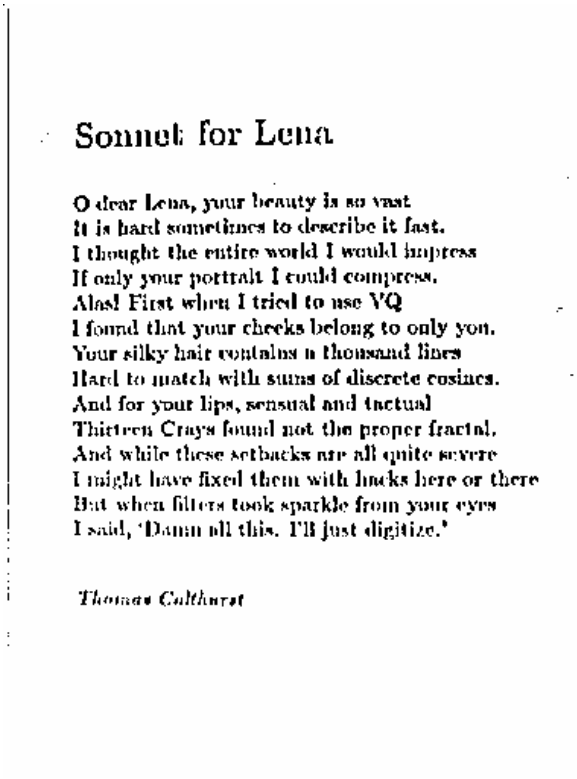


Figure 2.10: The result using a global threshold

As can be seen above, the poor result is caused by the illumination issue. The better solution for this problem is to use local adaptive thresholding instead of global thresholding. For each pixel, only the pixels within certain area around that pixel are taken into account. Therefore, the threshold for each pixel only depends on its neighbouring pixels.



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure 2.11: Result of using adaptive thresholding

In our experiment, real time image processing is needed and the lighting conditions could keep changing during the movement of mobile device. In particular, because the phone is being used close to the scanned surface, there is often shadow from the phone and users hand. Adaptive thresholding, thus, would be the better choice to achieve the greatest quality.

A mobile phone system that is most like our proposed system is a Mixed Interaction Spaces. It allows a user to use a mobile phone as a 3D cursor.

2.7 **Mixed Interaction Spaces** (Hansen, Eriksson and Lykke-Olesen, 2005)

Mixed Interaction Spaces: This paper "*describes a new interaction technique for mobile devices named Mixed Interaction Space that uses the camera of the mobile device to track the position, size and rotation of a fixed-point*" (Hansen, Eriksson and Lykke-Olesen, 2005). The technique is to use the space surrounding the mobile device. In particular, it uses the human face as a physical mark instead of printed or hand drawn mark. All space surround the mobile device is available for use. When the camera is moving, the space is changing, then, the change is converted as 4 dimensional vector as input. The four dimensions include zoom in and zoom out, up and down, left and right, also the rotation. The following shows the space and tracking.

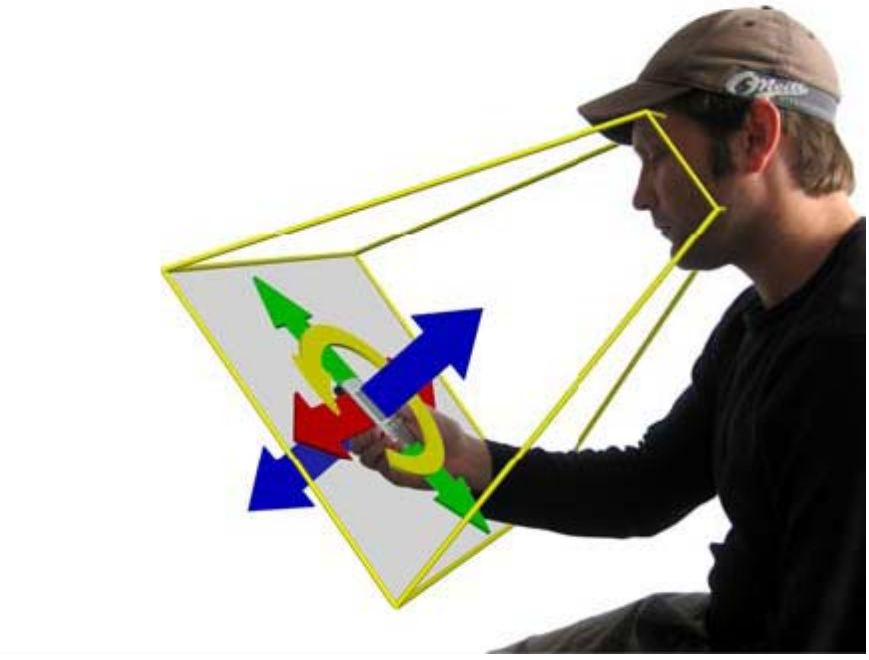


Figure 2.12: The Mixed Interaction Space with face tracking

The highlight of this idea is using the face instead of other fixed-points as the tracing marks. This avoid the need for other devices or related materials. Generally, physical marks like points are selected from some patterned surface or still images. Using

MIXIS is more convenient. When user interacts with mobile device, only thing he/she has to do is moving the mobile in front of his/her face. Whereas, in other systems, there are still more devices involved, for example, the coded patterns on the surface or some kind of circle or cross is used to trace the change of movement. Furthermore, since it explores 3D space instead of 2D space, it could provide more input information by same operations than other tracking methods.

However, the MIXIS technique limits the way of interaction. This works well for applications on the phone. For example, panning or zooming a map display. It is not convenient for controlling something else as it is too easy to lose the face from the camera view. The user has to look at the camera while using it, or at least, the user has to face the camera. As the MIXIS is designed for mobile applications, there are no other devices like computers involved, the mobile device could take advantage of this technique. If the PC and mobile device are combined together as a whole system, the mobile device play the role of an input device, and is not the focus of the user's attention. This system is not ideal for our application.

As the image recognition processes are generally very complex, the high performance hardware is required. The system on mobile phone used CAMSHIFT to reduce amout of image analysis due to its slow processor.

2.8 Computer Vision Face Tracking For Use in a Perceptual User Interface (Bradski, 1998)

People from Intel lab introduced an algorithm, named Continuously Adaptive Mean Shift (CAMSHIFT) algorithm, which is used to dynamically track human faces. As described in the paper, CAMSHIFT is an improved algorithm based on the Mean Shift algorithm. The Mean Shift algorithm was designed to deal with a static model of colour probability distribution whereas CAMSHITF can process the model finding with a dynamically changing colour probability distribution. A simple description of their face tracking process is listed below:

1. record the face image
2. use HSV(Hue Saturation Value) colour system to convert the image to 1D histograms, and save it as a colour model
3. for each frame from the camera, convert all pixels to a probability of flesh image.
4. use CAMSHIFT to locate the centroid of the last search window and calculate the new search window centered at the centroid found for next frame.

The algorithm can track four dimensional changes for a user, X, Y, Z and head roll, which could be used in variety of areas. Also, it has the ability to cope with noise. An example mentioned in their paper showed after addeding 0, 10, 30 and 50% uniform noise data to the original video frames the result was still reliable.

The results generated from their experiments had proved their algorithm is a reliable solution for human face or hand tracking. The CAMSHIFT algorithm is somewhat complex or computational expensive algorithm since a great deal of mathematical calculations is involved. However the idea of performing image analysis in a search window based on the result form the last frame is used in our project.

Chapter 3
Technical Background

Before entering the more complex parts, I will give an introduction of the experimental environment and the knowledge required for building the system. In this chapter, the environment will be described first. Bluetooth technology and the camera related graphics system will be explained respectively. Finally, some methods that I have used to understand these ideas are explained.

3.1 Experiment environment

The experimental environment contains two parts, the hardware and the software. The hardware includes the mobile phone and the PC. The software involves the development tools that are used in this project.

3.1.1 Hardware environment

To make the mobile device interact with the PC, we have to run our software in two systems, the mobile system and the PC system. The development platform for both systems is the PC. Table 3.1 shows the devices.

	Desktop Computer	**Mobile phone SE P910i**
CPU	Intel Pentium® 4 at 2.60GHz	ARM9 at 156 MHz
RAM	504MB	64 MB
Accessory	Bluetooth USB Dongle (Bluetooth Version 1.2)	Onboard camera, built-in Bluetooth (Bluetooth Version 1.2)
OS	Windows XP with SP2	Symbian UIQ2.1

Table 3.1: The devices used for this project

Figure 3.1: A Photo of P910i mobile phone

In our experiment, we used a SonyEricsson P910i mobile phone and a desktop computer. The computer has a normal hardware setup except for a USB Bluetooth dongle which is used to receive and transmitting Bluetooth radio signals. The P910i mobile phone is a PDA like handheld device. It has a large screen and a relatively powerful processor. However, for the embedded CPU, especially for ARM9, there is no FPU (Floating Point Unit). All floating point calculations are done by software on the integer processor. Hence, any floating point calculation would slow down the whole process.

Also, note that we have used a desktop computer rather than a laptop as the computer being controlled. This was convenient in our experimental setup. The software will work equally well with a laptop.

3.1.2 Development environment

To develop the software for Windows XP, Visual Studio .Net 2003 is used with the Microsoft Platform SDK. The MS Platform SDK provides the necessary packages for the Bluetooth development with the Microsoft Bluetooth stack.

The mobile phone software was developed under Windows XP as well. There is a free download of Borland C++ Builder X Mobile Edition, which can work with UIQ 2.1 SDK together. To develop for SonyEricsson P910i mobile phone camera applications, the P800/P900/P910 Specific C++ API for UIQ 2.1 SDK has to be installed as well. This package provides the libraries and header files for video development. With the UIQ SDK, a developer can test their application in the EPOC emulator. However for Bluetooth and camera related projects, the emulator is not reliable enough. There are many posts on the Internet describing a similar situation; applications ran well in the emulator but crashed on the real devices. This problem is probably due to the high dependency on hardware. The emulator is general software provided by Symbian, which simply emulates different devices to the same faked solution. Therefore, the real-time image processing application has to be tested in the real device instead of the emulator. This will make testing more convenient. The real device UIQ application is created on the Windows platform and then installed to the mobile phone directly by the software provided by the phone manufacturer.

3.2 Brief Introduction of Bluetooth

Bluetooth is one of the wireless communication solutions. It was designed for solving the unreliability of infrared beams (IrDA). Bluetooth is mainly used in a short range (up to 10 meters) wireless network. Because of its low power consumption, fast data transmission (compared to IrDA) and better reliability, Bluetooth is commonly built-in to modern portable devices (PDA, Smartphone, Portable computers etc.).

3.2.1 The structure of the Bluetooth network

Basically, there are three different kinds of network topologies for Bluetooth. The concepts, *master* and *slave*, are used for the role of each device. For people who have some network experience, master/slave is somewhat similar to the

Server/Client model except for one thing. The difference is that master/slave is only involved in the physical link layer whereas Server/Client can be used in several different layers. A Bluetooth network is only initialized by the master device. The invited devices will become the slaves. However, in the application layer, the connection requests can be started by any device in the network.

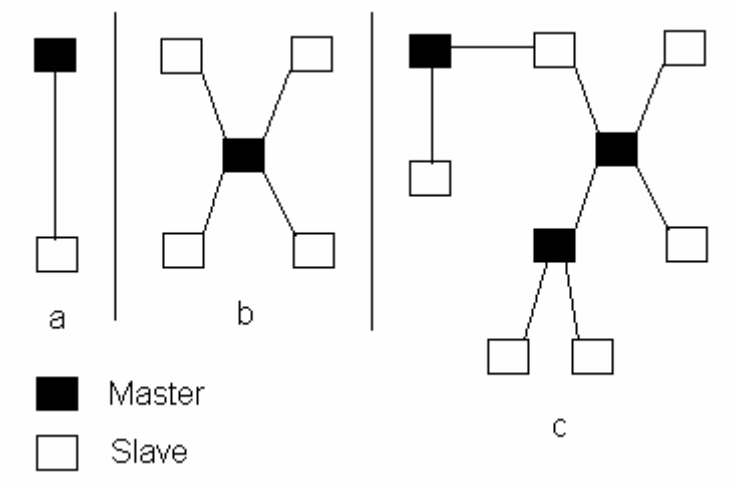


Figure 3.2: Three different Bluetooth network topology.

The kinds of network topology are:

a: Point-to-point is the simplest one-to-one connection.

b: Piconet (point-to multipoint) is the one-to-many connection. A piconet can contain a maximum of 7 slave devices.

c: Scatternet contains several piconets. The master or slave of one piconet can also be the slave of another piconet.

In our case, one mobile phone with a desktop computer, our application will be of type (a).

3.2.2 Bluetooth Protocol

The Bluetooth specification also describes the Bluetooth protocol stack.

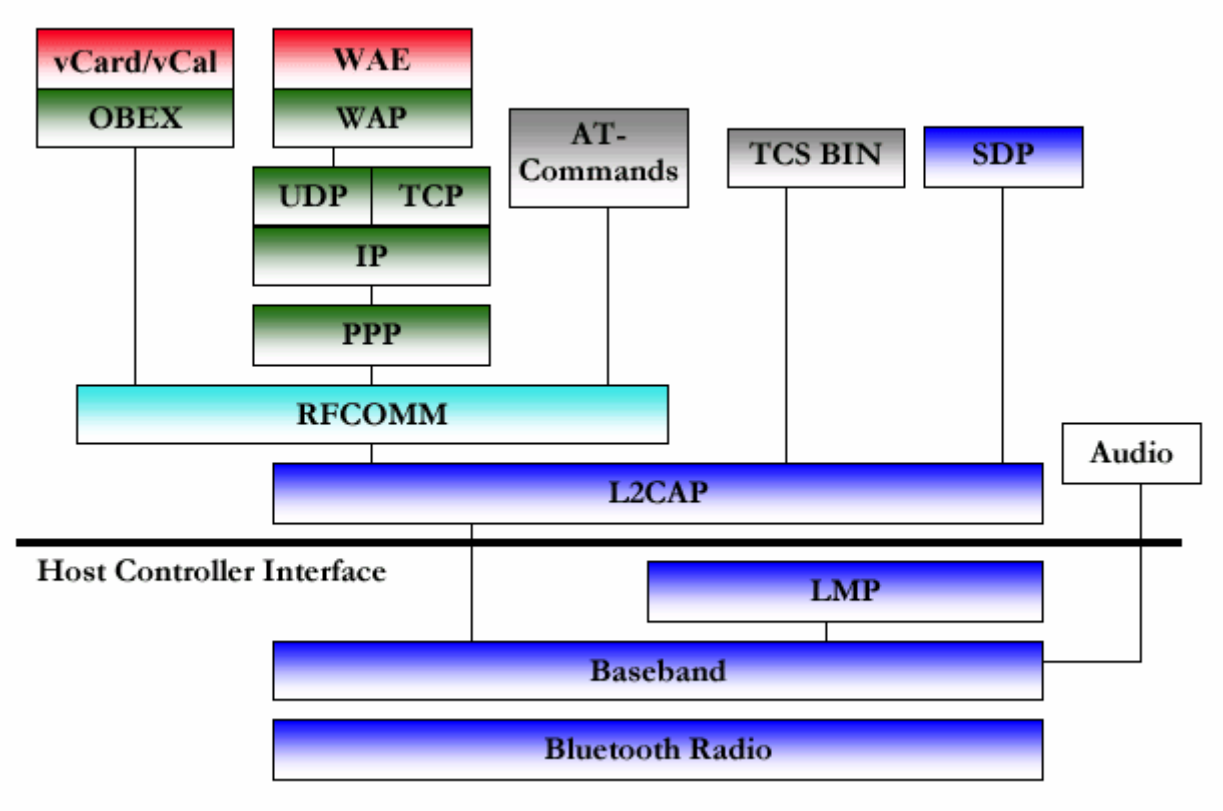


Figure 3.3: Bluetooth protocol stack

Figure 3.3 shows the whole stack from the bottom, low level, to the top, high level. I will give a brief explanation of the most popular protocols.

Bluetooth Radio:

The radio specification. This is the lowest layer for Bluetooth stack and implemented in the Bluetooth hardware. It includes the frequency and power specifications.

Baseband:

The hardware layer which controls the receiving/sending mechanism.

The Link Manager Protocol (LMP):

This layer is used to control other devices. (i.e., switching master/slave)

These three layers together provide the hardware level communication support for

Bluetooth.

Host Controller Interface (HCI) interface:

This is the interface layer for application software. All functions in the lower layer will be encapsulated in Interface.

Logical Link Control and Adaptation Protocol (L2CAP):

This layer provides the connection-oriented and connection-less data services to all the upper layers. As seen in Figure 3.3, L2CAP is the first layer above the HCI interface. L2CAP links uses PSMs (Protocol & Service Multiplexer) to communicate with each other and handle packet segmentation and reassembly.

Radio Frequency COMM (RFCOMM):

This layer emulates a standard RS232 serial port. Applications use this protocol layer and therefore can use the standard AT commands. Furthermore, applications can also use this layer by writing/reading a COM port if a COM port is associated to the RFCOMM connection. Comparing to the L2CAP layer, RFCOMM uses channels instead of PSMs to communicate between applications. Also, there is maximum number of channels limitation for the RFCOMM layer. Only thirty channels are available.

Service Discovery Protocol (SDP):

This protocol is designed for remote device service discovery. As the Bluetooth network is a dynamic wireless network, the topology of the network could change from time to time. Network resources are limited as well (for instance, the RFCOMM protocol only has 30 channels to use). The better solution for applications is to dynamically use resources. If the master device dynamically associates a channel to use, how could the slaves know which channel they should connect to? SDP is the solution for this question. SDP provides many details in service queries and device queries. A device, then, queries for a list of the services another device

provides or for just the one it is interested in, using the SDP protocol.

Object Exchange (OBEX):

This is designed to transmit objects. All objects which operating system can recognize can be sent by the protocol of OBEX. This includes pictures, audio clips, ring-tones, text or binary applications.

To an end user, probably there are no differences resulting from the choice of interface level. However, to a programmer, there are important differences. The protocols below HCI interface are outside software control. For projects which use Bluetooth for a continuous connection to achieve their goals, the protocols above RFCOMM probably are too specific. The most used protocols for software developers are SDP, L2CAP and RFCOMM. Another more important reason is these three protocols are the essential layers of the Bluetooth stack and in general, they are supported well in all Bluetooth devices. For developers with Internet programming experience, the term port probably is much more familiar than the PSMs or channels. The PSM in L2CAP and the channel number in RFCOMM actually follow a similar programming logic. Also, from the programming point of view, L2CAP corresponds to UDP programming and RFCOMM to TCP programming.

3.3 Brief Introduction of Camera Video Format

With development of the integrated circuit industry, the calculation capability of embedded device increases greatly. Based on the increased processing ability, the onboard camera has recently become standard on new generation mobile phones. However, the embedded device still has less processing capability than a desktop PC. Generally, the onboard cameras have less resolution and lower sampling rate than the popular web cameras for the PC. In this section, I will explain some basic concepts about camera video format, especially for P910i.

Before talking about the details of P910i video stream, I will give some general information about the colour system, which is the basis of the entire graphic world including hardware devices and the software applications.

3.3.1 Colour space

Generally, colour space means a set which includes all possible colours. However, there are some different colour spaces. The reason for the differences is that one colour space contains all the possible results from a model, also known as colour model. Therefore, for each colour model there is a colour space corresponding. A colour model describes the way that digits represent a colour. Typically, a colour model is composed of three or four components.

The different colour models were designed for different purposes. For example, the RGB (Red, Green, Blue) and HSV (Hue, Saturation, Value) colour models are mainly used in computer graphic related fields. YUV model is widely used in the television broadcasting systems. CMYK (Cyan, Magenta, Yellow, Key) is a very popular model for the printing industries. Also there are many other colour models like YDbDr, RYB and CIELAB(L*a*b*) etc.. All these models have their own features in different areas. In our project, we will not consider the details for all of them. The ones we will be interested in are the RGB and YUV models.

3.3.2 The RGB colour model

As the name RGB implies, the RGB colour model represents all colours as combinations of red, green and blue components. One colour in RGB colour space, therefore, is formed by three values from the three primary colours, red, green and blue. However, the RGB definition does not define what exactly ‘red’, ‘green’ and ‘blue’ are. Therefore, there are some pre-defined RGB colour spaces for use. sRGB

(standard RGB, created by HP and Microsoft) space and Adobe RGB (developed by Adobe System) space are most widely used in computer graphic processing.

Although there are many details in the specifications of sRGB and Adobe RGB colour spaces, we will not consider any more detail than is necessary for our topic. One important idea to understand is that the RGB colour model is a kind of additive colour model. Any colour in the space is the result by adding the values of three components. The following is a graph shows the results. Areas of overlap show the effects of adding colours. The white in the middle results from adding equal and large amounts of red green and blue.

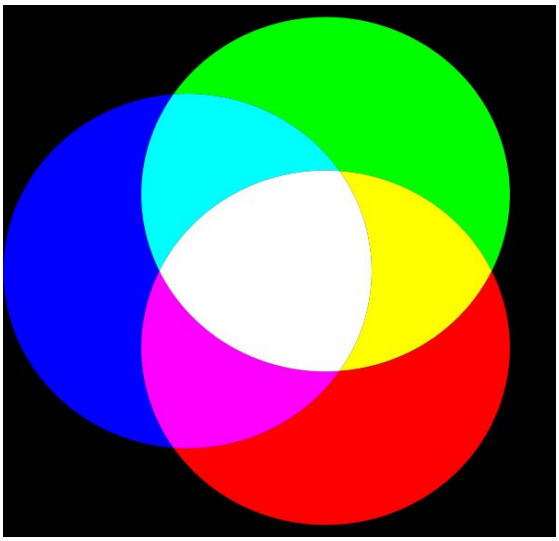


Figure 3.4: Image shows the mixing result of red, green and blue

On a computer we can represent RGB colours in different ways: 16 bit (High Colour) mode, 24 bit (True colour) mode, 32 bit mode and 48 bit mode. The main difference is that the more bits that are used, the more levels the mode can represent. The mode we are interested in is the 24/32 bit mode which is usually represented by a 32 bit integer. Even in the 24 bit mode, most applications still use a 32 bit integer to represent the colour although there are 8 bits left un-used. This is due to the hardware word-aligned addresses accessing mechanism. For the 32 bit mode, the extra 8 bits is used for other purpose like palette entry or transparency level. In the

Windows system, the macro RGB is used for defining the colour. It provides 256 levels for each colour component. The combination of three zero valued components is defined as the colour black and three 255s represents as the white colour.

In some applications, we also see a colour model called RGBA, in which the A represents the transparency channel, also called alpha channel. RGBA uses a 24 bits for RGB data and defines the extra 8 of 32 bits to represent 256 levels of transparency. A well-known use for RGBA is the image format PNG.

3.3.3 YUV colour model

The YUV colour model is to represent the colour space as one luminance component and two chrominance components. The luminance component (Y) is the level of brightness, and chrominance components (U and V) provide colour as a pair of colour difference signals, giving hue and saturation. This model is widely used in television broadcasting standard.

There are many different YUV formats for different kind of signal sampling. YUV formats can be treated in two groups, the packed and planar formats. The difference is that the packed format takes Y, U and V samples for each pixel and treats them as one, then saves all pixels in sequence; whereas the planar format saves the Y, U and V samples in three separate planes. For the packed formats, there are Y411, Y211, Y41T, YUYV and more formats. The planar formats have YVU9, YV12, YV16, Y41B, I420, Y42B and more.

3.3.4 P910i Video Format

As the format used in our project is what we are interested in, this section will explain the details of SonyEricsson P910i camera video format. P910i mobile phone uses a different sampling system to all standard formats, so called user defined YUV

4:2:0 data format. As described in User Guide to the Sony Ericsson Camera API, the actual pattern follows the sequence,

for each even row, only the Y value included, 1 byte Y per pixel

for each odd row, 2 bytes Ys, one byte U and one byte V,

where U and V represent 4 pixels, the 2 Ys before UV and 2 Ys above this row.

The format can be shown in Figure 3.5:

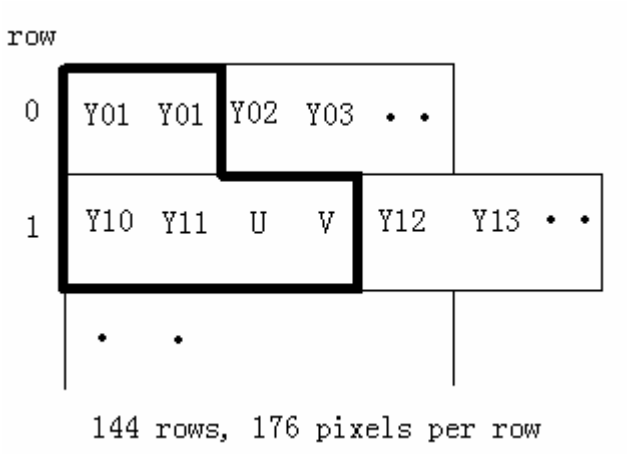


Figure 3.5: P910i video format. UV values shared with 4 Ys in the bold area

The frame size is also defined in the Camera API document, it is the same size as QCIF (Quarter CIF). QCIF is part of the CIF (Common Intermediate Format) video format standards in videoconferencing. The CIF standard describes the pixel horizontal and vertical resolutions for YUV sequences video signals. QCIF is defined as 176 pixels in horizontal size and 144 pixels in vertical size.

3.3.5 RGB YUV conversions

As the P910i video is packed using the YUV model and both the systems, MS Windows XP and Symbian OS UIQ use RGB model to build all the user interface

parts, conversion from YUV to RGB is necessary for our experiment.

After some research, several conversion formulae were found:

Formula from Keith Jack:

```
R = 1.164(Y - 16) + 1.596(V - 128)
G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128)
B = 1.164(Y - 16)                  + 2.018(U - 128)
```

Julien formula:

```
R = Y +             1.403 * V
G = Y – 0.344 * U – 0.714 * V
B = Y + 1.770 * U
```

Microsoft formula:

```
C = Y - 16
D = U - 128
E = V - 128
R = clip(( 298 * C           + 409 * E + 128) >> 8)
G = clip(( 298 * C - 100 * D - 208 * E + 128) >> 8)
B = clip(( 298 * C + 516 * D           + 128) >> 8)

Where clip is to ensure a value is in range of 0 to 255
```

Avery Lee's JFIF formula:

```
R = Y +                    1.402 * (Cr-128)
G = Y - 0.34414*(Cb-128) - 0.71414*(Cr-128)
B = Y + 1.772 * (Cb-128)
Where Cr = V and Cb = U
```

In the implementation, we have tested all the three formulas above and they all reproduced the mobile phone captured images correctly in the PC side applications. However, in the final implementation, the Microsoft is selected for its slightly better colour expression, and also because it does not use floating point arithmetic.

3.4 Methods used

To solve our problem, some research and testing was done before designing the

system. As this was the first time I had touched embedded system software development, especially the Symbian UIQ application, I used some general methods to make myself familiar with the system.

Study other work

Some other graphic processing projects, especially the papers they provided, were studied. The focus for the studies was the image recognition methods. For example, the methods were used to locate the camera and how the image analysis works?

The Internet research

I used Google to search for references to the Symbian OS, UIQ platform. The most time searching was spent on Bluetooth related issues. Also, I have done some research on image processing related issues, like colour spaces and the conversion between different colour spaces.

Read and Write

There are many articles talking about Bluetooth programming and mobile phone onboard camera programming. Sony Ericssion also provides an example in the camera API package. I have read many articles and much example code. To make some technical details more clear, I have written some test programs which include the image binary algorithm, position finding algorithm and Bluetooth communication tests. To understand how the UIQ program is organized, I have written two small test applications. All the work above is to prepare for the better completion of this project.

Chapter 4

System Design

The system design for our project will be explained on the basis of the information provided earlier. To achieve our goal, a smart phone acting as a wireless mouse, the system is obviously composed of two parts, the mobile phone application and the PC side application. From the point of development, mobile phone camera access, image processing and Bluetooth network are the main parts of this project. The whole system was designed using a traditional Client/Server model. On the PC side, it has a server and a client, which is listening to the mobile device and processing the data or commands from the mobile device. The mobile device has client software running, which captures the raw video stream, does some processing to locate the position and sends the result through Bluetooth to the PC.

The reason the image processing was implemented in the mobile client side is that the transmission capability of Bluetooth is still not fast enough to transmit a raw video stream. Thus, we must send as little data as possible through the Bluetooth network.

The following graph shows the final design idea:

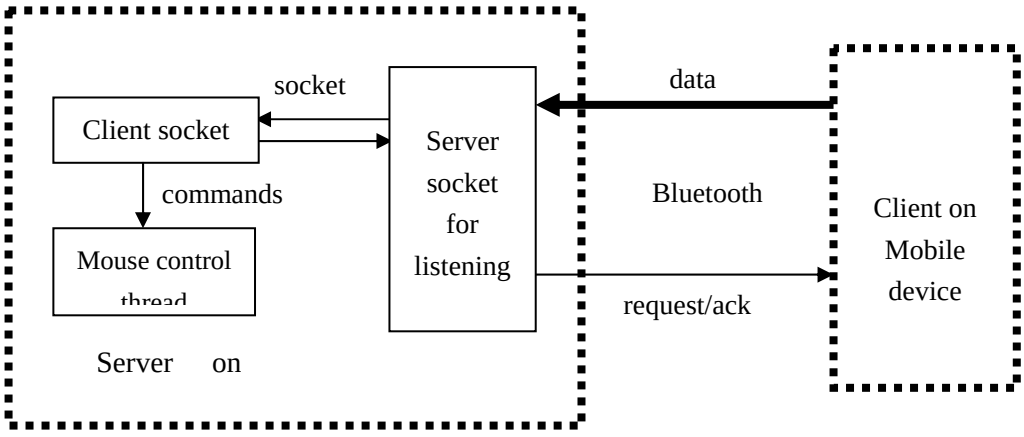


Figure 4.1: The system architecture

In this chapter, I will explain in three sections: the image recognition; the Bluetooth network; and the revised design based on the results from the first implementation.

4.1 Image Recognition

Image recognition will be discussed in three parts, the surface we are going to use, the image to binary algorithm that generates a binary version of the image and how the location is found based on the binary data (the intersection finding method).

4.1.1 Patterned surface

To help the location finding, physical markers are used in a similar way to that used in many other projects including the mentioned projects in chapter 2. In our case, a patterned surface is used to locate the position. The surface, a white paper, is filled with parallel horizontal and vertical thick lines. Any white piece of paper can be the surface. It is not necessary for it to be pre-prepared and the lines can be drawn with any thick pen. The example shown in Figure 4.2 has lines drawn with a ruler. However, hand sketched lines would suffice.

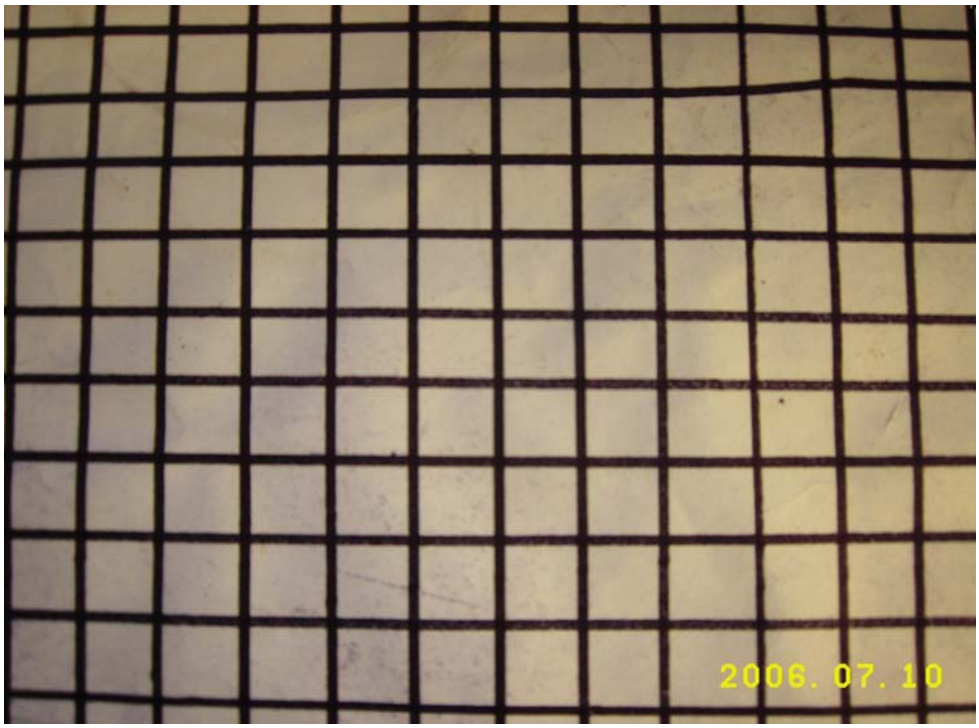


Figure 4.2: The hand drawn pattern used to help position recognition

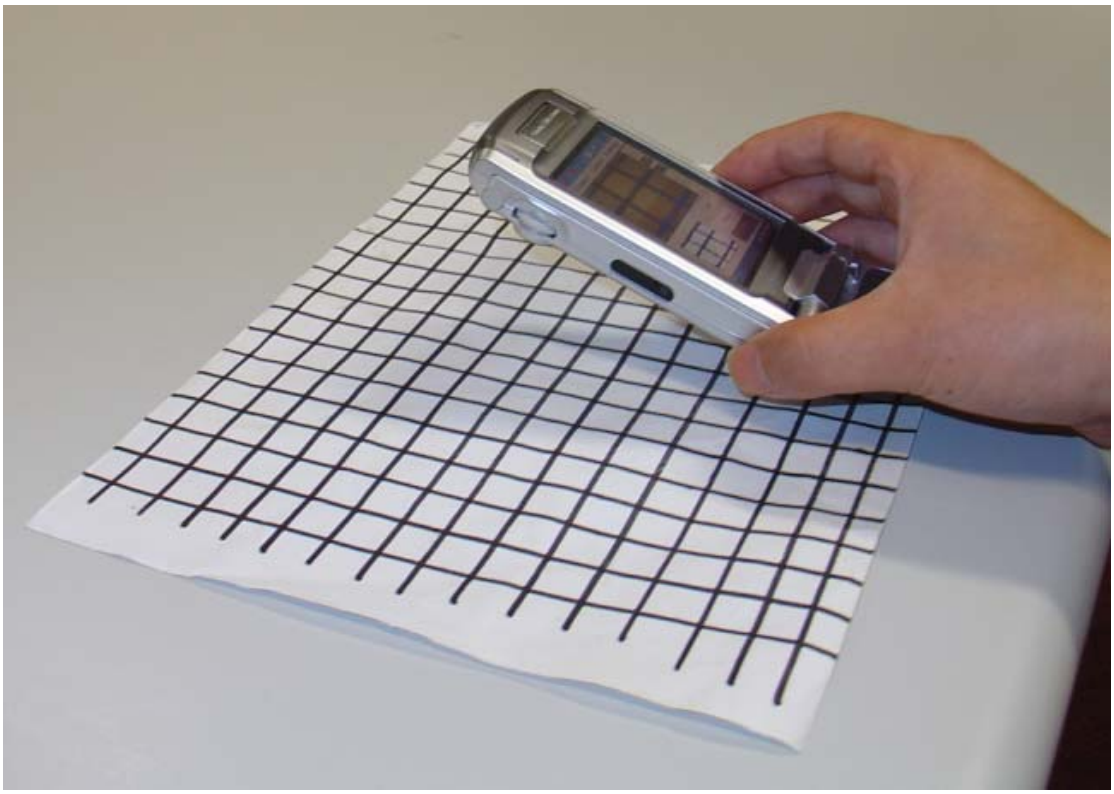


Figure 4.3: Mobile phone was tracking the intersections.

The distance between two lines need not be equal. Roughly spaced lines are enough

to figure out changes of position. There are some constraints the surface should follow,

1. The lines with same direction, horizontal or vertical, should not overlap
2. Those horizontal or vertical lines should not be too far apart. This is to ensure that, at any time, there is at least one intersection existed in a captured frame
3. The lines should be thick enough. Ensure that clear lines (without breaks) occur in the camera image.

When the camera is moving, the coordinate of the left top line intersection in the image is calculated. The coordinates will be sent to the PC. The idea of using a grid as the pattern was considered to provide a simple way to recognize and track location because in normal operation, the camera is close to the surface and movements will quickly take it out of range of single objects. The reason for thick lines is that the camera is too close to the surface. Also, using grids could give some extra information about the position, for example, the rotation to the nearest 90 degree.

4.1.2 Conversion to Binary Image

Many image processing methods start by converting a colour image into a binary image. As compared to a colour image, a binary image only contains 2 colours, which will simplify the problem and require fewer calculations in the following image processing steps. Another important idea is that the binary image contains two kinds of information, the one we interested in as the foreground colour and the one we do not care about as the background colour. Based on these points the binary image contains the information we need and binary processing is much easier and

therefore much faster for the position finding process. Converting the YUV plane to binary image should be done before further calculation.

4.1.2.1 Using Y plane only

As explained before, the raw captured video stream is encoded of YUV space. Before the binary image is generated, we need an extra step, which is grayscale image generation. The binary image will then be created from the grayscale image. To convert a colour image into a grayscale image, the formula calculating the grey level from the RGB values can be used directly. Therefore, the straightforward way is to convert YUV space into the RGB space, and then create the grayscale image from the RGB image. However, in our design, YUV is a special case. Y samples are the set of luminance, which could be treated as grey levels directly. Thereby, the design is to use the Y samples directly as the grey levels. The chroma components, U and V will be ignored. In this way, the processing time will also be saved.

4.1.2.2 Using a Fixed Threshold Value

Once we have the grayscale image, thresholding will be done. At the beginning of design, a fixed number was used as the threshold. The threshold value was taken to be the grey level average of all pixels. The following images show original and the converted images

Figure 4.4: Zoomed out image. Similar effects as Zoomed in image

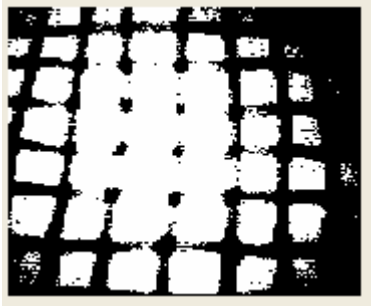


Figure 4.5: Converted binary image (threshold 85)

The quality of the binary image will affect the final result greatly. The converted image shown above has a poor quality. The central area has several broken lines which will make the next step very difficult. The problem is caused by the strong illumination gradient. The distribution of luminance sample values is not as expected which ideally would have a big gap between two groups of values.

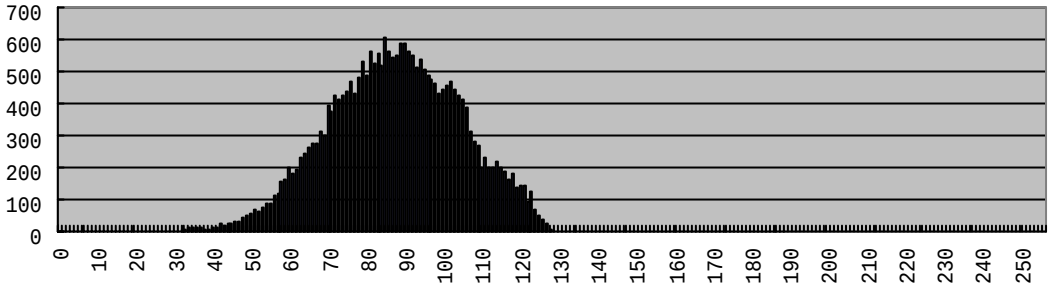


Figure 4.6: Luminance distribution

Figure 4.6 shows the luminance distribution of Figure 4.4. The horizontal axis represents the range of sample values. The vertical axis represents the number of sample values occurring. As seen above, the range of samples takes only about one third of 256 different levels (between 40 and 130). Also, the threshold cannot reasonably be selected from the distribution.

To reduce the effect of strong illumination gradient, an adaptive threshold method (mentioned earlier) was used. The original adaptive threshold method is to calculate an average value of a rectangle area around each pixel.

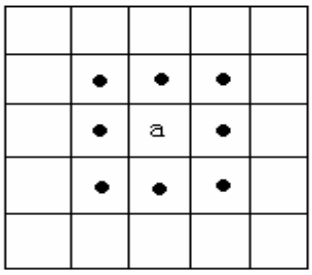


Figure 4.7: Adaptive thresholding

For example, the average of nine pixels, or more, including pixel ‘a’ itself are calculated. The average will be used as the threshold for comparison. If the average is greater than the level of ‘a’, ‘a’ will become a white point, otherwise, it will be converted to black.

Figure 4.8: One frame of captured stream

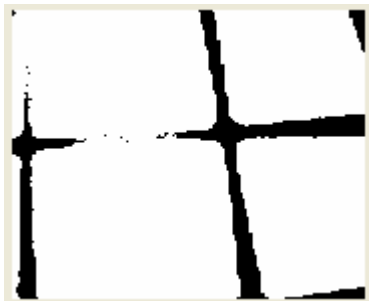


Figure 4.9: Using fixed value of threshold

Figure 4.9 shows the result using fixed value 85. The broken part of the horizontal line will cause an inaccurate result in further processing. The worst part of the converted result will always be the part which has the strongest illumination gradient

affection. When the adaptive thresholding method is used, the result turned to the following.

Figure 4.10: Result of using adaptive thresholding

Results were more reliable if the neighborhood size was increased from 3*3 to 21*21, which means the threshold is the average of 21*21 pixels around the target pixel. The mean constant value is 7, which is used to eliminate the black noise on the background.

However, the calculation is computationally expensive. It took too long to process one frame in the mobile phone. The solution for this problem is using the fixed threshold and adaptive thresholding together. In a certain area, the average is used for processing each pixel within this area. In this way, the computing time is saved and the quality of results is still acceptable as shown below.

Figure 4.11: The final result

Figure 4.9, 4.10 and 4.11 above also have been subject to a fine tuning process

which fills one white pixel gaps in black areas, and speeds up next processing in some cases.

4.1.3 Intersection finding

The goal of this step is to find the top left intersection of drawn lines in the binary image generated from the last process. The algorithm used in intersection finding is designed as following way,

```
start at top row of the binary image
for each row from left to right
   for each angle within certain range (-60 degree to 60 degree from horizontal line)
      for each pixel in the line at the current angle
         if it is white, break
         if it is the n-th black pixel, record the position as P and
               try the line through P at 90 degree rotated from the current angle
            if the n-th black pixel from P without any white found in both direction(up and down)
               the intersection found, which is P
            otherwise, continue
         otherwise, continue;
      end for
   end for
end for
```

The basic idea of this method is that it is trying to scan each row at all possible angles. If there is a line found, it tries to determine whether the corresponding vertical line exists. If all conditions are matched the left top intersection is found. The only problem with this method in practice is that if the edges of the image match horizontal or vertical lines the algorithm above will return an unreliable result. This problem can be easily solved by selecting the point within central part of the image. The image below shows the final result.

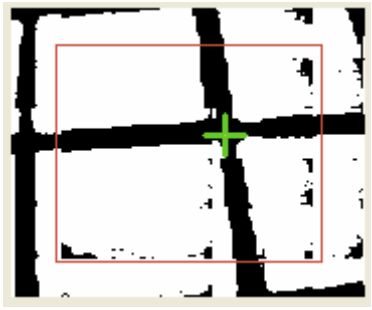


Figure 4.12: Intersection finding

The scanning area is within the gray rectangle and the gray cross is the point the method found.

4.2 Bluetooth network

4.2.1 Bluetooth protocol

Before designing the connection structure, some experiments were done. The mobile phone package includes some PC based software which can be used to access a mobile phone by Bluetooth. The bluetooth driver it used is from a company called Broadcom and the bluetooth software is called Widcomm. During the first test of the Bluetooth connection, everything was working as expected, including download and upload of files between mobile phone and PC. Connections could be created easily. However, the smartphone software package does not include the Widcomm software development kit. At the time of looking at the Broadcom web, it charged more than one thousand US dollars for its SDK, whereas Microsoft Service Pack 2 provides bluetooth stack development SDK for free. Therefore, the Microsoft SP 2 was chosen for implementing our bluetooth connection. At the time of writing this thesis, I found Broadcom had changed their price strategy and the SDK can now be freely downloaded after filling some user information.

As Bluetooth provides several protocols for use, L2CAP is the one most applications choose, as mentioned in the UIQ documentation. However, in early experimentation, an connection using L2CAP between mobile phone and PC could not be created properly. When the Widcom Bluetooth software is used, the connection was quite easy as expected, whereas when the Microsoft Bluetooth stack was used, connection with L2CAP layer was impossible. The socket can never be created in this case. It might be due to some Bluetooth stack implementation or compatibility issues as Widcomm has supported its product for years where Microsoft SP 2 is much younger. Also I found one article written at the end of 2005 which said that the Microsoft Bluetooth stack only support RFCOMM protocol. Based on the fact that only RFCOMM could work for us, finally, the protocol RFCOMM was used. As mentioned in the earlier chapter, RFCOMM was designed to emulating serial port communication over L2CAP protocol. In practice, all methods used are mostly same as using TCP programming, especially under Windows XP.

On the PC side, we intended that the server would setup a service called “Mouse Camera”. This would have allowed the mobile device to query the device information and discover, for example which channel the server was using. The UIQ software in P910i would then have used the SDP protocol to look for the “Mouse Camera” service and obtained the channel number from the server. However, because SDP inquiries do not work in our environment, the final decision was to use a fixed channel for communication. This will be explained in the next chapter.

4.2.2 Simple Video Transfer Protocol (SVTP)

Over Bluetooth communication, the project needs its own way to communicate data. For example, position information, a raw video stream or a compressed video stream could be transferred to the PC and some feedback could be sent back to the mobile device. Therefore, a simple protocol, Simple Video Transfer Protocol was designed and used as the method of communication in our system. The protocol is defined as

follows.

a) Packet Header:

```
typedef struct {
   unsigned char tag;
   unsigned char type;
} SVTPHeader;
```

1. Byte 00: SVTP packet mark, always 0x18, meaning this is a SVTP packet
2. Byte 01: packet type, can be one of the followings,
   00: flow control, 01: data packet, 02: Camera Info (not implemented)

b) For each packet type:

. Flow Control (FC, 00):

1. Byte 0: type of command, one of the following values:
   00: success (ACK), 01: error, 02: raw video, 03: position

. Data Packet (DP, 01):

1. Byte 0: command type
   00: reserved, not defined, 01: reserved, not defined, 02: raw video,
   03: position

. Camera Information (CI, 02): (not implemented)

1. Byte 0: size of captured frame. Currently only
   0: QCIF defined (176 * 144)
2. Byte 1: YUV format, Only
   0: 4:2:0 defined.

c) For each data packet:

Raw Video Stream (RVS, 02):

Fixed data size (including header): 640, (for one frame, needs about 60 packets)

1. Byte 0 - 3: number of processing frame.
2. Byte 4: current fragment of this frame
3. Byte 5 - 8: length of the following data
4. Byte 9 – end: video raw data. Maximum buffer size is 0x18C0 (6336) bytes

Position Data (PD, 03)

1. Byte 0: horizontal coordinate x
2. Byte 1: vertical coordinate y
3. Byte 2: the camera rotation angle

4. Byte 3: reserved, used in the final implementation for which button clicked

All control packets are sent from the PC to the phone and all Data packets sent from the phone to the PC. The Tables below might display the packet structure better.

0	1	2
Mark	CTL	0x00(success)
Mark	CTL	0x01(error)
Mark	CTL	0x02(raw video)
Mark	CTL	0x03(position)

Packet Mark: 0x18 (Mark), CTL: 0x00 (control packet)

Table 4.1: Control packet structure

0	1	2	Byte: 3 - 6	Byte: 7	Byte: 8 - 11	Byte: 12 - end
Mark	DATA	CMD type	Frame No.	Fragment No.	Length of followings	Data (fixed size of 600B)

Mark: 0x18, DATA: 0x01, CMD: 0x02, fixed size 12 + 594 = 606 bytes

Table 4.2: Raw video data packet structure

0	1	2	3	4	5	6
Mark	DATA	CMD	X	Y	Angle	Reserved

Mark: 0x18, DATA: 0x01, CMD: 0x03, fixed size 7 bytes

Table 4.3: Position data packet structure

The reason for the raw video packet including frame number and fragment number is that, if a whole frame is put into the buffer to send, the PC side receives the data not in one packet, but in several small packets sequentially. Therefore, the design was changed to send smaller sized packets each time and include the frame number and current fragment number with each.

As the initial idea was to do all possible calculations on the PC side, raw video transmission was tried first. However, the transmission time was too long for one frame. We decided to put the image processing part in the mobile device. Hence, in the defined Simple Video Transfer Protocol (SVTP), finally, only the position data packet structure is used.

4.3. Revised Design

After some experiments, we met some problems. The location finding took longer than expected. We required the mobile phone to process 15 frames within one second. After some optimisation, the number of frames processed per second was still within the range 0 to 4. Another annoying problem was when the intersection found was leaving the frame or searching area, the algorithm sometimes found the wrong intersections, or, in other words, when the target point was changed, the new target point might not be the expected left top one. The situation is shown below.

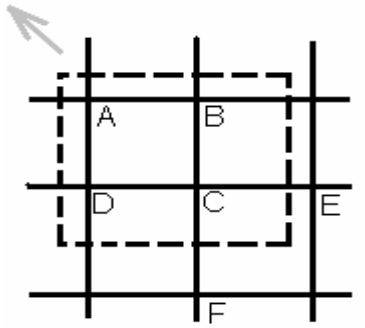


Figure 4.13: Before A went out

The dashed rectangle is the searching area (camera image). The black horizontal and vertical lines are the patterns the algorithm is looking for. The surface is moving toward the left top direction, i.e. the camera is moving diagonally down to the right. The left top intersection A in the searching window is found as the target point.

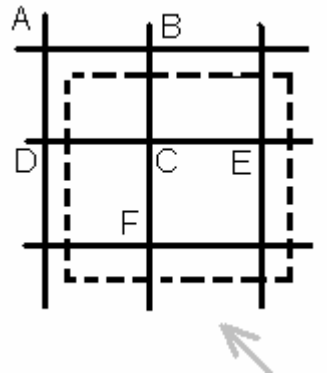


Figure 4.14: After A went out

Figure 4.14 shows the next frame the camera captured. Intersection A now is out of the searching window. The result for our algorithm will find the target point C. In the PC, the mouse should keep in the same direction, but the actual target is changed to C now(C is actually on the right bottom direction towards A). Inaccuracy in recognition made it very difficult to detect and properly handle this situation. To make mouse keep in the same direction, the last motion direction has to be recorded for next move. Also this will require the server software to do some extra calculation to determine if the target point has changed or not, in which case it should ignore current target point information.

As the onboard camera is very sensitive to the changing of the light, the binary image could sometimes become worse. When the camera is moving, the light conditions are gradually changing. In the worse cases, the target was incorrectly calculated as E or F. This makes the situation even more complicated. The mis-calculated point would control the PC mouse to follow an un-expected behaviour.

To avoid such situation, the design was changed in a major way.

1. the pattern is changed to a solid circle (dot) with certain size. Again, this can easily be drawn on a piece of blank paper, whenever it is needed.

2. the algorithm now is tracking the centre circle instead of finding the left top intersection.

Figure 4.15: The new pattern, a solid black circle

The advantage of using this pattern is that the target will no longer change, and finding the circle needs less calculation than finding an intersection. The rotation calculation is ignored which means floating point calculation in mobile device is no longer needed.

The algorithm for finding the centre point of the circle is as following.

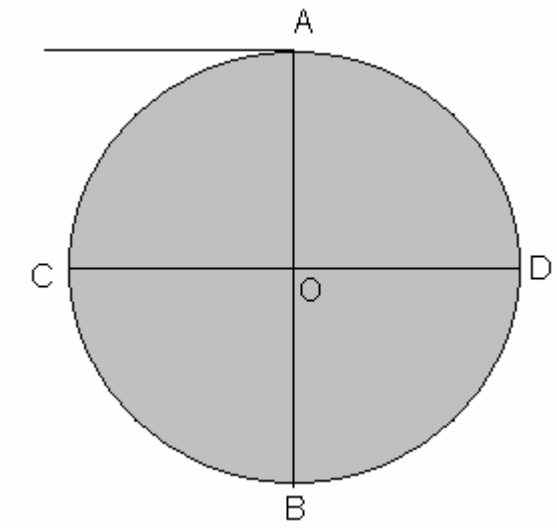


Figure 4.16: Tracking algorithm. The target dot shown as the gray solid circle

try each line from top to bottom

scan through the current line from left to right

the top point A of the target should always be found first.

count the length of from A to B vertically. The half length of AB will be the radius

based on the position of A, calculate the position of point O.

This is a very simple algorithm, and it gives the server more reliable results.

Chapter 5
Implementation

The design of the system is as stated in the previous chapter. In this chapter, we will focus on how the design was implemented and improved based on test results. As an experimental project, during the implementation, the functionality achieved is our main interest. Our design ideas will be proved practically possible or not. The functions we designed will work as expected or not function at all. We will not be concerned with the usability of the software too much at this stage.

Following the design, the system will have two separate applications running on the P910i and Windows XP respectively. The P910i application runs as a client and the XP application sets up a Bluetooth server. The server, then, waits for position information from the P910i to control the mouse pointer. Before the final implementation, video data transmission was also been tested. The camera in the mobile device, SonyEricsson P910i mobile phone, captures 15 frames every second and each frame contains 38016 bytes. To transfer 15 raw frames in one second would need a data transfer rate reaching at least 38016 * 8 * 15 bits per-second, which is about 4.35Mbps. However, the USB dongle we used only provides at most 721kbps and the result we achieved was at the rate of one or one half frame per second. This was too slow for our mouse control application, but it did allow us to use the mobile phone as a (low frame rate) web cam for the PC. This test program was not developed further for the mouse application, but might have some interesting applications of its own. Therefore, due to the low speed issue, we stop trying to send raw video data. Instead, all calculations will be performed on the mobile side. Only calculated coordinates will be sent to the PC.

In the following parts of this section, I will explain the two applications. In some parts where we revised the design, I will give more details about the implementation.

Also, some source code is presented for better understanding.

5.1 Mobile device application implementation

The client application should implement the following functions: accessing onboard camera; converting raw video frames to binary images; finding the centre point of the target dot; creating the Bluetooth connection and sending the position information to the server. The processing sequence is very straightforward. The main process flow can be simply shown in the following chart.

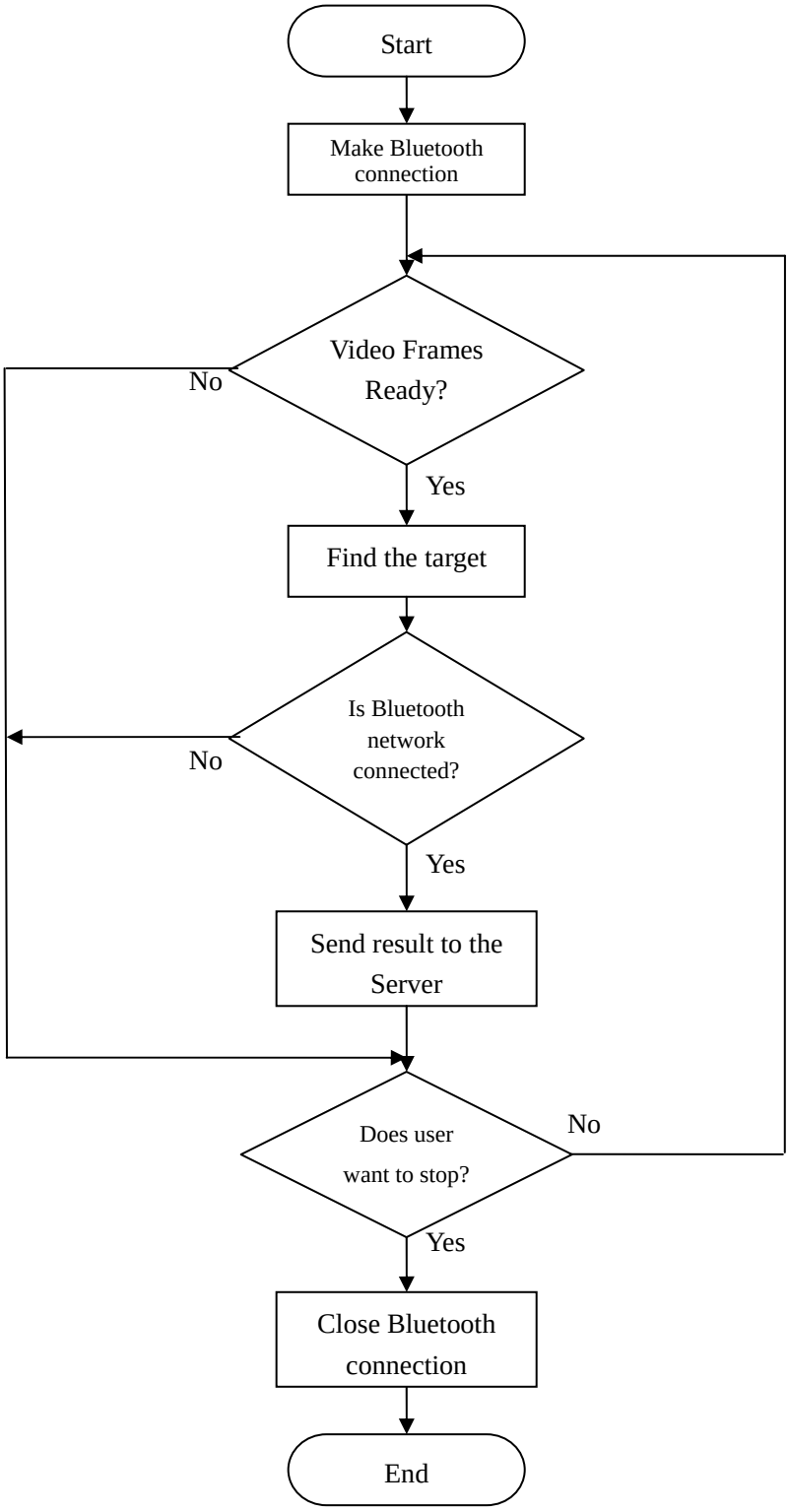


Figure 5.1: The main process flow for the client

Before explaining each class of the client application, I will introduce some general Symbian GUI programming concepts.

5.1.1 Symbian GUI programming

Symbian GUI applications are written in C++. However, due to the fact that embedded hardware has the less calculation and storage capability, the Symbian system has some features to reduce resource requirements.

5.1.1.1 Naming Conventions

Symbian programming has its own naming conventions. They are not necessary to build the applications, but their use will help others to better understand the code.

T… classes (class names start with a capital T) are simple classes and are usually allocated in the stack. These classes are data classes and do not contain other external object references. They can be simply declared as normal C++ type or structures.

C… classes are derived from CBase class. These classes are allocated in the heap. To use these classes, developers use ‘new’ to allocate a C… object instance, and manipulate them via pointers.

R… classes are the resource handling classes. The resources are usually owned elsewhere and the R classes are the handles to access them.

M… classes are the interface classes. They contain pure virtual methods only. To use these M classes, developers have to subclass the M classes and implement the virtual methods.

5.1.1.2 Exception handling

Symbian C++ does not use the standard exception handling mechanism (try and

catch). The main purpose of the Symbian exception handling mechanism is focused on memory consumption. For example, if the application crashed, how could the system release the memory which the crashed application had allocated?

The **Leave** mechanism was designed for this. "*Leaving means that an exception (or an abnormal event) has occurred*" (the "Starter's Guide" in the UIQ SDK package).

All the methods with a suffix L in their names contain some operations which could cause leave. When leave happens the developer should call the static function User::Leave() to invoke the system leave mechanism. In Symbian coding, all C… classes should be initialized by a NewL or NewLC function. The names will tell the developers this operation might cause Leave if an allocation operation failed and developer should use the Symbian method of catching the exception. The difference between the suffices L and LC is L is only means leave but LC means the created object has been pushed onto a cleanup stack already.

The Cleanup Stack was designed to de-allocate memory when leave occurred. The idea is, when allocating a C… object, the pointer of the object should be pushed onto a system maintained cleanup stack. If a leave happens, the static method User::Leave() would invoke the cleanup mechanism to de-allocate the objects in the cleanup stack. An example of using the cleanup stack is:

```
CMyClass * object = new CMyClass();
CleanupStack::PushL(object);// before some operations
                            // which could lead to leave
                            // the object is pushed onto
                            // the system cleanup stack
SomeDangerousProcessL();
CleanupStack::PopAndDestroy();
```

To catch errors, Symbian system uses **TRAP** and **TRAPD**. TRAPD is a macro for

TRAP that declares a return variable automatically. When the program leaves, the cleanup stack mechanism will de-allocate any objects pushed onto the cleanup stack in that method and then, look for the most recent TRAP. The TRAP, therefore, will be used to handle the exception and retrieve the error reason or state. A typical use of TRAP will be like the following:

```
TInt error = KErrNone;
TRAP(error, SomeDangerousMethodL());
If(error != KErrNone){
    …
}
```

or simply using the macro TRAPD,

```
TRAPD(error, SomeDangerousMethodL());
If(error != KErrNone){
    …
}
```

5.1.1.3 Two-phase construction

The Symbian system introduced a different way to ensure that even if the constructor of an object leaves, the system can still handle the exception. As in the normal C++ design, if a leave occurred in the constructor, the destructor would not be called. Therefore, the other objects created in the constructor will not be released. To handle the “new” situation, Symbian OS uses a two-phase construction. In the constructor, it is required that there are no memory allocation operations. All the allocation operation are required to be in a member function, conventionally called `ConstructL()`. Any C… class (allocated on the heap) should have a public static function to do the two step construction and return a pointer to the object. The public static member function is usually called `NewL` or `NewLC`, depending on whether or not the object is pushed onto the cleanup stack.

5.1.1.4 Symbian GUI application framework

All Symbian GUI applications are DLLs(Dynamic Link Libraries). The type of a GUI application is declared as UID in the project specification file. The framework of GUI application follows the MVC (model view control) pattern. For any GUI application there are three classes which have to be implemented to satisfy the MVC pattern.

A model class with the suffix “Document” is used to create the control class and is in charge of all external resources.

The control class usually has the suffix “AppUi” and handles all user input. The developer should notice that this class does nothing about interface operations. However, it creates the view class to do the UI operations.

The view class handles all the user interface related operations. All the paint routines should be implemented in this class. “View” is the most commonly used suffix by this class.

Besides MVC classes, we still need an application class which is used to create our document class and return the UID of this application. Also, for each application, the MVC model needs an entry point since it actually is a DLL. There are two functions that have to be provided. One is to initialize the DLL, named `E32Dll`. Another is `CQikApplication::NewApplication`, which is used to create our DLL instance;

5.1.2 Mobile Phone Class Details

The client has 11 classes including 2 interface classes. The structure is shown in the following class diagram.

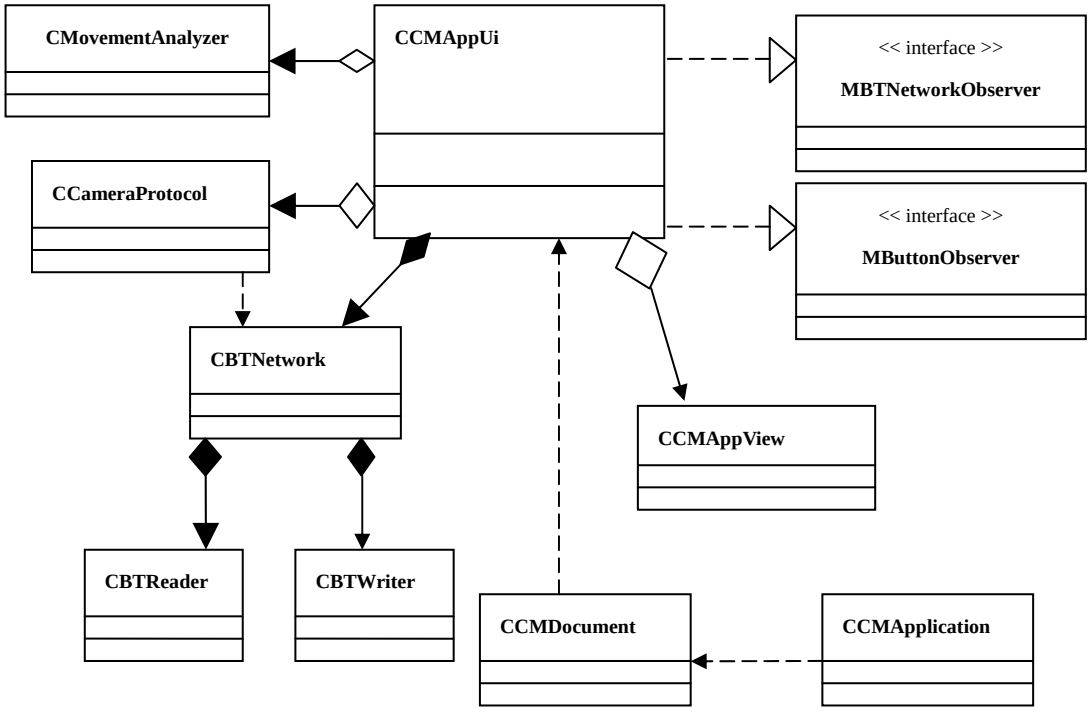


Figure 5.2: Client side class diagram

The code follows the UIQ 2.1 MVC framework.

`CCMApplication`: the main application which inherits from `CQikApplication`. It will create our `CCMDocument`.

`CCMDocument`: the model class which is a subclass of `CEikDocument`. Since we do not need external resources, this class is only used to create the `CCMAppUi` class.

`MBTNetworkObserver`: an interface class. All network related events will correspond to its pure virtual methods respectively.

`CBTNetwork`: the Bluetooth network handler. It does all the network related operations. The final result will be sent to the server by this class. In the case of sending raw video frames, this class also receives the ACK packets from the server.

CBTWriter: an asynchronous network sending class. It is a member object of CBTNetwork.

CBTReader: an asynchronous network reading class. It is a member object of CBTNetwork.

CCameraProtocol: this class implements the SVTP protocol as described earlier. This class will accept the calculated results, pack them into SVTP format, and then gives the buffer to CBTNetwork.

CMovementAnalyzer: the image recognition class. It does all the image analysis including conversion into grayscale, generation of binary, and the target finding.

CCMAppUi: the subclass of CEikAppUi. It handles the GUI input events. Also this class inherits several other classes to receive camera, network, touch screen events.

MButtonObserver: an interface class. Its purpose is to notify the CCMAppUi class the touch screen user input.

CCMAppView: the user interface handler. All paint processes should be done in this class. It is a subclass of CCoeControl.

5.1.2.1 Bluetooth network handler

CBTNetwork is the class which encapsulates all Bluetooth related operations. CBTNetwork inherits from CActive class. CActive class is a system provided asynchronous object. It is controlled by CActiveScheduler class. When a request is completed, CActiveScheduler will call CActive's member method RunL. CActiveScheduler is an asynchronous service provider and handles all

events of CActive objects in sequence. For each thread in a Symbian system, there is only one CActiveScheduler which can handle many active objects. This is the Symbian asynchronous call solution which is much less costly than having one active object per thread. The steps for using CActive object are shown below.

. Add the class onto the CActiveScheduler before requesting any actions

```
CActiveScheduler::Add(this);
```

. Issue a request

```
SetActive();
```

. Process the results when request completed as shown before in void CBTNetwork::RunL() method. The status of request completion is checked and further action can be taken based on the results

Having the description of CActive object, we will explain how a CBTNetwork object connects to the server. To connect to the server, requires the following steps:

. Connecting to system socket server. This will establish the sub-session communication for our RSocket object iBTSocket to use:

```
iSocketServ.Connect();
```

. Set up the protocol to use. In our case, we use RFCOMM as the base protocol:

```
iSocketServ.FindProtocol(_L("RFCOMM"), pInfo);
```

. Select the remote device using a system provided dialogue. UIQ provides a ready-to-use Bluetooth device selection dialogue, CQBTUISelectDialog, which will list all Bluetooth devices the hardware can find. The user can then select the one the server application is running on. When the LaunchSingleSelectDialogLD returns, the remote device address will be assign to the one selected. The main

purpose of this operation is to retrieve the remote device address.

```
CQBTUISelectDialog * dialog =
       new (ELeave) CQBTUISelectDialog(
                iRemoteDevAddr,
                iRemoteDevName,
                iRemoteDevClass,
                CQBTUISelectDialog::EQBTDeviceFilterAll,
                serviceFilter /*, ETrue */);
// popup Select Bluetooth device dialog
if(dialog->LaunchSingleSelectDialogLD()){
// if comes here, all parameters passed
// in CQBTUISelectDialog should be updated already
    return KErrNone;
}else return KErrCancel;
```

. Connect to the PC. To connect to the server, we use the remote device address just obtained. Also, the channel that the server uses is needed as well. In our experiment, to simplify the process, we use a fixed channel number, channel 1.

```
// iRFCOMMPChannel is 1
iRemoteSockAddr.SetPort(iRFCOMMPChannel);
iRemoteSockAddr.SetBTAddr(iRemoteDevAddr);
// open the subsession create before for communication
Tint aRetValue =
            iBTSocket.Open(iSocketServ, _L("RFCOMM"));
// connect to the remote device
iBTSocket.Connect(iRemoteSockAddr, iStatus);
```

. Event processing. As mentioned earlier, we have to check the values returned in `RunL` function. In the method `RunL`, we use a switch statement to check the return states (network connected, network broken, data arrived and others). Based on state found, different events are sent to the `CCMAppUi` class.

For non-blocking communication, this class has also included two active attributes. One is for writing data to the `iBTSocket`. Another is for reading data from the `iBTSocket`. They are objects of the `CBTWriter` and `CBTReader` classes.

To send network events to the control class `CCMAppUi`, `CBTNetwork` also has a reference to an `MBTNetworkObserver` class. `CCMAppUi` is the actual object which implements this interface class (as will be explained later). The event notifications are simply done by calling implementations of the virtual methods of `MBTNetworkObserver`.

As stated before, a fixed channel number is used for communication instead of using the SDP protocol to retrieve the remote device channel number. The SDP queries have been implemented in our code. However, using SDP to query the information of the device which we selected will always return a system error. The error number is -6312, which is documented in the Symbian OS error code listing as KErrL2CAPRequestTimeout. There is no further information about this error in the UIQ SDK documentation. Some posts on the Internet talk about similar situations but in the S60 SDK which is slightly different from UIQ SDK. The solution suggested for S60 SDK was to avoid keeping RHostResolver open once the remote device was found. RHostResolver is the class used to retrieve remote device addresses. It does the same work as `CQBTUISelectDialog` does, but without the elegant user interface. However, using `CQBTUISelectDialog,` it is not possible to alter the way in which RHostResolver is operated. Therefore, their solution cannot be implemented in our system. So far, the SDP query function in our experimental environment does not work. To check whether the SDP could ever function properly, we have tested an open source program, BeMused, which provides Bluetooth services on a PC, allowing a user to use the mobile phone to control Media player and PowerPoint on the PC. BeMused is supposed to work in a UIQ system. When experimenting with this application, the “find server” menu always came out with exactly the same error number as we have. To make progress, finally, we set the communication channel to a fixed number.

5.1.2.2 Image recognition

`CMovementAnalyzer` is the class which implements all necessary image processing routines. It does several steps, including converting to grayscale, thresholding, intersection and centre point finding. Intersection finding was used in the first implementation. As mentioned, the pattern in the final design was changed to a solid circle. However, the method for calculating the left top intersection is still in the code.

No matter which pattern is used, binary conversion is always needed. The method `ConvertToBinary(…)` is used to do the conversion. The parameter for this method is the raw frame buffer pointer. After processing is done, this class will hold a buffer which contains binary data for this frame. The most time consuming process in this method is thresholding. As explained in the design section, all pixels within a certain area will use the same threshold for this area. To make the process faster, in the implementation, a large area (size of 21 * 21 pixels) was used. After further experiments, to achieve better binary result, the area for calculating the threshold value was selected to be a larger rectangle (size of 41 * 41 pixels) which covers all the to be calculated pixels.

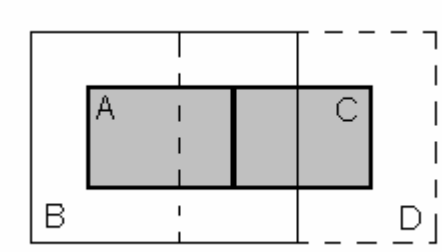


Figure 5.3: Our threshold algorithm. when converting area A to binary, area B will be used to calculate the local constant threshold value. Likewise, area D will be used to calculate the threshold for converting area C

The result still gives us an acceptable binary image as shown below.

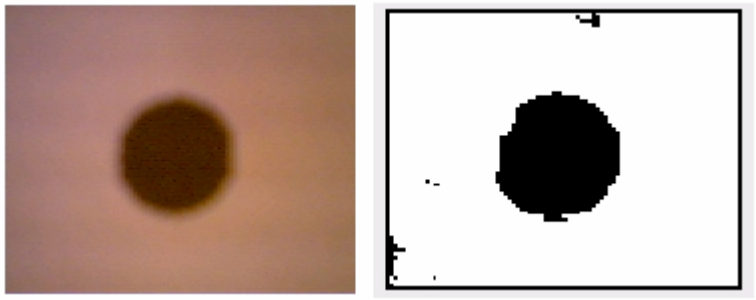


Figure 5.4: Binary result. The left is a captured raw frame; the right is the binary result

Also a searching window was used to reduce the thresholding calculation. The initial size of the window is the actual size of video frame. It is used to define the conversion area. Only the pixels within this area will be converted to binary and the rest will be ignored. When a target point is found, the searching window will be centered at this point for the next search. The size will be changed to the size of the square with twice length of the target diameter. In some situations, the binary image turns out to be of very poor quality. The algorithm, hence, might not find the target. In this case, the searching window will be set back to the initial case for the next frame. The idea of this design was similar to CAMSHIFT algorithm described in chapter two.

Once the binary image is obtained, another method of `CMovementAnalyzer`, `TrackTarget()` can be used to find the centre point of the black circle, and its diameter. `TrackTarget()` does line by line scanning. In practice, it scans every alternate line and checks the first pixel of each pair of pixels in the line. If a black pixel is found, the pixels above and before the current checked one are checked to see if they are black or not. In this way, the first black pixel will be found with nearly a quarter of the full scanning time. To determine whether the pixel found is the top point of the target or just noise data, a minimum circle size is needed. Within the minimum size, a number of evenly distributed points were tested.

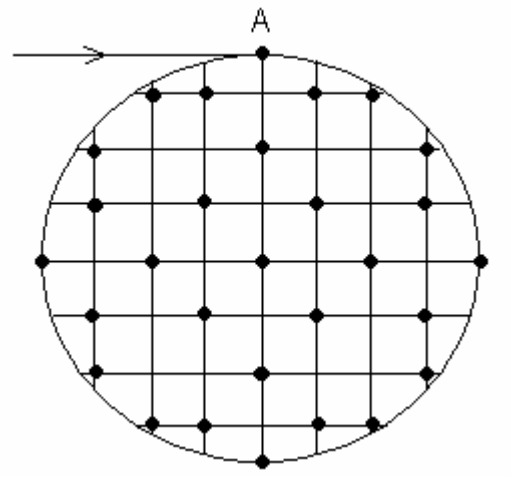


Figure 5.5: Minimum size test. All black pixels will be tested

In our implementation, there are 29 pixels within the minimum circle size (diameter is 18 pixels) tested. If 20 or more out of 29 tests turned out the true results (blacks), the point (A) is thought as a target top point. This process tests whether the black area has big enough size, therefore rejecting small areas of noise.

Based on the point we have found, a binary search used to detect the size (diameter) of the target circle. Only the column starting at point A is tested. As the minimum length is known, the next pixel below the minimum length starting at A will be the first one to test. If it is black, the test will jump down with a fixed length. Then, the second check will be done. If it is black the test will jump down with half size of the fixed length or jump up with the half length if it is white. Further tests will be made until the jump length becomes 1. In the implementation, only 4 tests were need as starting jumping length was chosen to be 18 pixels. After 4 tests, the jump length will be $18/(2^4) = 1.125$ which is close enough for our needs. After this process, the circle information (centre position and diameter) are available.

As mentioned in the design section, intersection finding is no longer used in the project. Since it was implemented in the first design version and the code is still in the client source, I will explain briefly how it works. Basically, the `CMovementAnalyzer` uses one simple call, named

`CalculateLTIntersection`(), to perform all necessary calculations. It takes 3 steps to generate a result. The implementation tries `TopDownScan(…)`s on each row and `TopDownScan` tries `LeftRightScan()`s at all possible angles (interval of one degree). If the current pixel is black, a `VerticalScanAt()` will be done at this point. These three private methods, `TopDownScan()`, `LeftRightScan`() and `VerticalScanAt`(), have the same return. They return false if a white pixel is discovered. The purpose of the last two methods is to find lines that match the minimum length requirement.

In our experiments, we have found that camera in mobile phone generally has a low shutter speed. This will cause a blurring problem when the mobile phone is moving rapidly. The blurred images would make image processing stop early and our algorithm would fail in these situations.

5.1.2.3 SVTP (Simple Video Transfer Protocol) implementation

The class CCameraProtocol implements Simple Video Transfer Protocol, encapsulating point and raw video stream transfer. It provides a simply public interface.

`ReceivedData(TDesC8& aBuffer)`: processes ACK or CTL packets from the Server.

`SendFrame(TDesC8 &aBuffer)`: sends a raw frame to the server.

`SendPosition(TUint8 x, TUint8 y, TInt8 theta, TUint8& aButton)`: sends the target position to the server. For sending the circle tracking results, the third parameter will be the diameter of the circle. The last parameter is the clicked button id.

The implementation of these functions focuses on byte sequence organization. This class also has references to the interface class `MBTNetworkObserver` and the

network handler `CBTNetwork`. The first reference is to notify the `CCMAppUi` if something is not obeying the protocol. The second one is for the network sending and receiving tasks.

5.1.2.4 The control class

CCMAppUi is the control class of the MVC model. Actually all the classes are connected and controlled by this class. It inherits from several super classes.

`CEikAppUi`: the base class for all control objects.

`MBTNetworkObserver`: the interface used to receive network events.

`MCoeView`: which receives view change events. If another application comes to the front (which is called active), the client would be told and release control of the camera so that other application could use it.

`MCameraObserver`: is to receive camera events

`SonyEricsson::MViewFinderObserver`: P910i specific interface which is used to receive the event, “view finding finished”. The camera processing was designed to find the current view first. The view, then, has the video painted directly onto it.

`MButtonObserver`: is to catch those touch screen events.

Also, this class controls all other parts of the application. It has a network, a camera, a protocol and a moving analyzer object as members. All these objects are organized by this class and the UI notifications which will be shown by the view class are sent via this class as well.

One important thing to mention here is the number of frames captured each time. The callback function FrameReady is provided by SonyEricsson API. This method is called after capturing five frame. In the API document, it clearly says only five frames is the only valid number for video capturing. It does bring us some difficulties to catch up with the most updated frame. There will always be up to a five frame latency for the video processing. We were never able to correct this problem. Hopefully the next release of the API will work better.

5.1.2.5 The view class

As mention before, this class is created by the control class. It handles all the painting routines for user interface. All touch screen events are received in this class. To notify the control class about the user actions, `MButtonObserver` was provided. `MButtonObserver` is an interface class which contains pure virtual methods only. The control class, therefore, inherits this interface in order to receive these clicking events.

A screen shot of the running application is shown below.

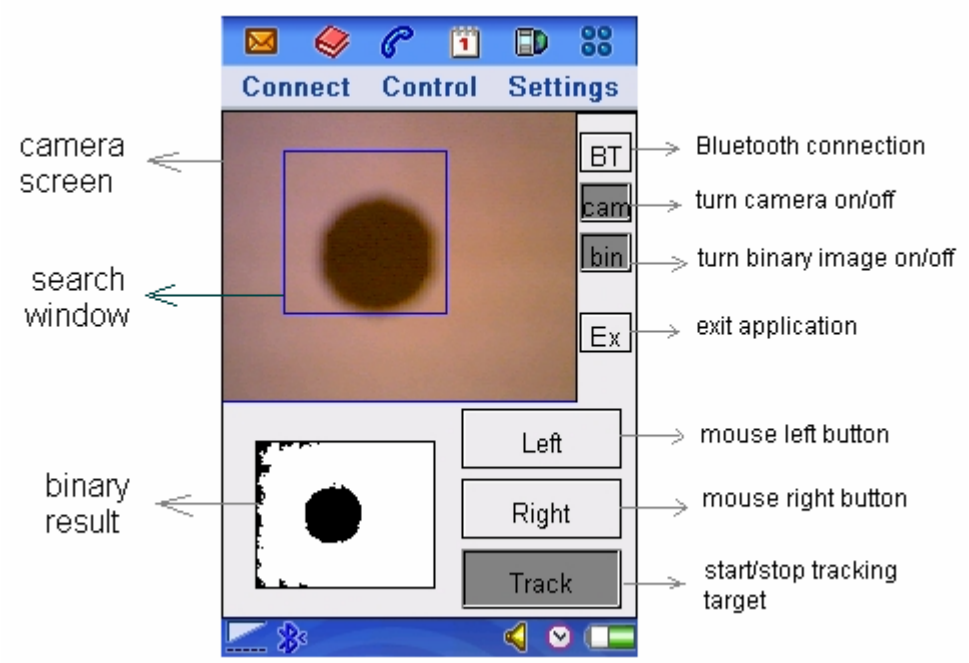


Figure 5.6: Screenshot of a running mobile phone application. The application was tracking the solid circle and the rectangle in the camera screen is the search area. Note: this is not intended as an end user interface. It provides features that are helpful in developing and debugging the application.

5.2 Windows application implementation

The Windows application is playing the role of a server and runs on a desktop PC. It was developed in Visual Studio 2003 with Microsoft platform SDK. As stated earlier, a fixed channel was used for our project Bluetooth communication instead of setting up a Bluetooth service. The sequence of operations is that this application sets up the Bluetooth socket and listens on the channel. A separate thread is provided for controlling mouse movement. Once the server receives a connection request, it creates a socket and accepts the connection. Then, the application waits for information sent by the client device. When information is received, the server checks the type of the information. If it is a position command, the application sends the command to the mouse control thread. If it is a raw video data packet, the frame will be displayed on the PC screen. The process can be shown in the following

flowchart.

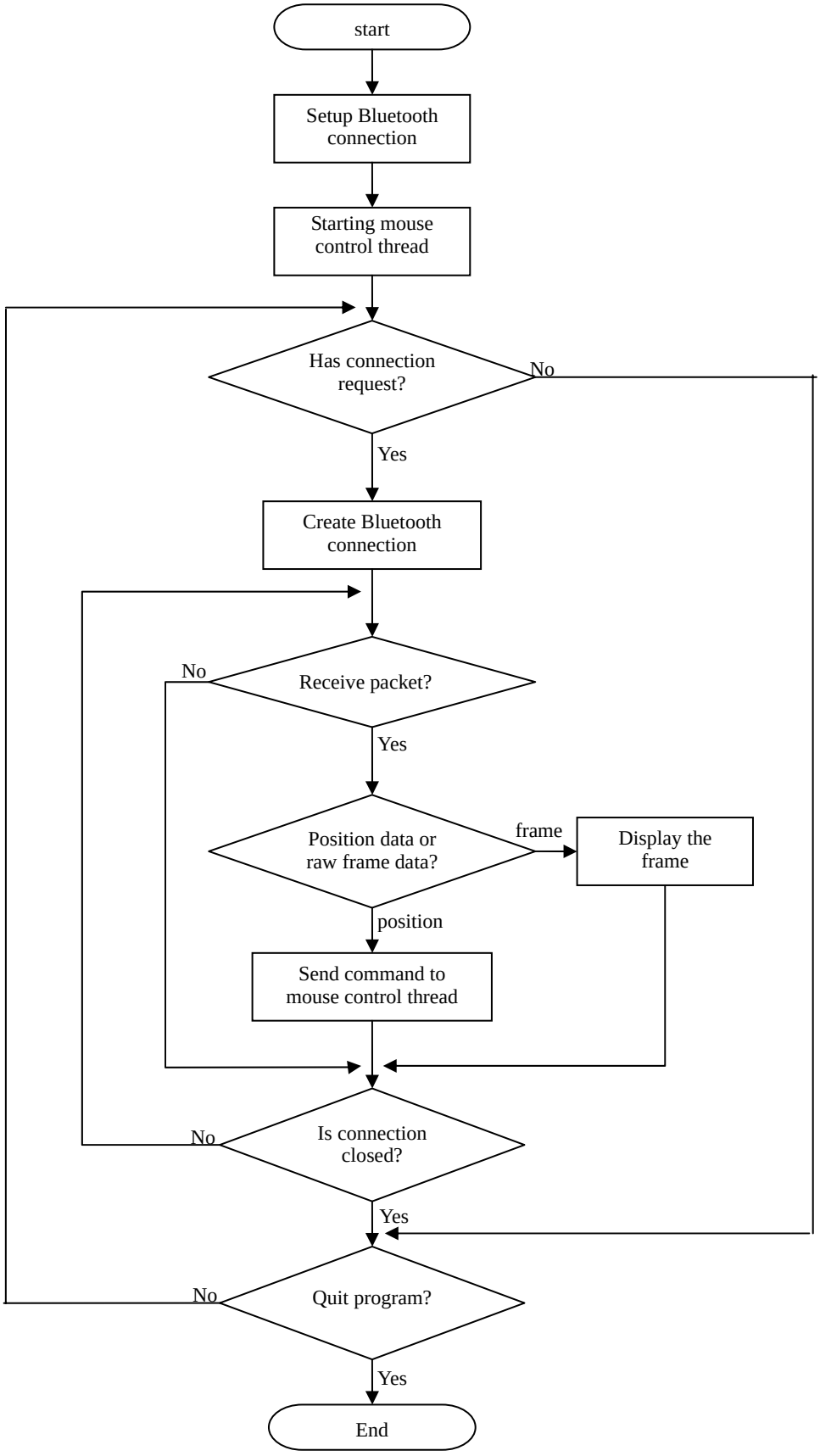


Figure 5.7: Process flowchart of the server application

Before explaining the class details, I will provide a general outline of Bluetooth programming on MS Windows. More detail about Windows XP Bluetooth programming can be read in the MSDN (Microsoft Developer Network) website. There are also numerous forum postings by people who have worked with MS Bluetooth software.

5.2.1 Windows Bluetooth programming

In the chapter on system design, I mentioned that development in Windows would be based on the Windows native Bluetooth libraries and APIs (Application Programming Interfaces). All the server code uses native Windows Bluetooth stack support. Microsoft Windows XP actually did not natively support Bluetooth devices until Service Pack 1 was released. To write Bluetooth applications, developers have to install the Microsoft Windows XP Platform SDK (which can be freely downloaded from Microsoft website). To make the MS Bluetooth stack work properly, the Bluetooth driver that came with the Bluetooth dongle (aerial) was not installed to the system. We assume that Windows socket programming is familiar. Developers who have no experience with Windows socket programming can easily find plenty of articles describing the methods used.

Basically, the Microsoft Bluetooth API is an extension of the Window Sockets 2 API. One of the differences is that Bluetooth socket uses some different data structures than those used in TCP/IP programming. There is a basic structure `SOCKADDR_BTH` which is used to initialize Bluetooth specific socket details. When binding the address to a socket, a `SOCKADDR_BTH` will be passed in as a `SOCKADDR` pointer. When creating the Bluetooth socket, the usage of the socket function follows this form:

```
socket(AF_BTH, SOCK_STREAM, BTHPROTO_RFCOMM);
```

The first argument is the socket type which means, in this case, it is a Bluetooth socket. The second and the last arguments should be as shown. Currently, only the RFCOMM protocol is supported. The last two arguments can not take any other values. I have searched for MS supported Bluetooth protocols on the web (including the MSDN website), but none of results turned out to be helpful except for one article which said that the Microsoft Bluetooth stack only supports RFCOMM protocol. After some experiments, only the RFCOMM protocol could be successfully used to create the Bluetooth socket. Although there is another protocol type defined in ws2bt.h header file, BTHPROTO_L2CAP, using this type would always raise a socket error.

Another difference between Bluetooth socket programming and ordinary socket programming is the Bluetooth service setup and query. As stated in the last chapter, SDP is usually used for Bluetooth applications. A remote device can retrieve device details from the service provider and show the services available. To setup a Bluetooth service in Windows XP, the method `WSASetService(…)` is used to register or deregister services. The function has a structure `WSAQUERYSET` as one of its arguments and all service related details are specified in that structure.

To query for remote services, the `WSALookupServiceBegin` is used to start the query. `WSALookupServiceNext` is used to retrieve the next available service. Device details are returned if `WSALoopupServiceNext` is called successfully. However, the code for setting up Bluetooth service was commented out for the reason that the service inquiries always failed on the client (mobile phone). Channel 1 was always used for our system.

5.2.2 Structure of the server application

The class diagram below shows the structure of the server application.

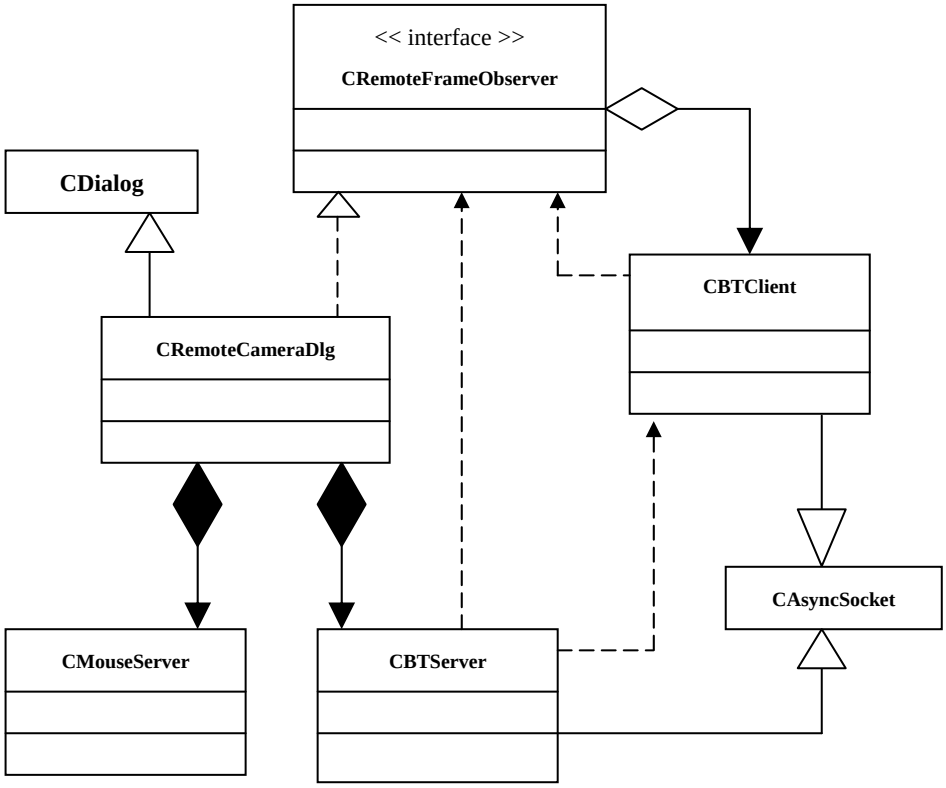


Figure 5.8: Class diagram for the server application

CBTServer: the server socket implementation. It initializes the Bluetooth network and waits for client connection requests.

CBTClient: a socket which does all the protocol related tasks.

CRemoteFrameObserver: an interface class used to notify the user interface of network event.

CMouseServer: a mouse control object which will control the mouse movement based on the commands it received.

CRemoteCameraDlg: the user interface class. All other objects will be organized from this class. This object will control all other objects.

5.2.2.1 Bluetooth handler

`CBTServer` is the class which implements all the operations that a server needs. The class inherits the MFC (Microsoft Foundation Class) class, `CAsyncSocket`, in order to provide the non-blocking socket functionality. `CAsyncSocket` itself provides some virtual methods which can be overridden to perform specific tasks. In our case, the Bluetooth socket used these as follows:

. in the constructor, the Bluetooth address and the Bluetooth service information were initialized.

. in method `Create`(), the Bluetooth socket was created and attached to the `CAsyncSocket` member `m_hSocket`. Then, Bluetooth service is started after registration and the socket is bound to the address initialized in the constructor.

. in method `OnAccept`(…), a socket (`CBTClient`) would be created for client connection once this method was called.

. in method `OnClose`(…), the class deregisters the Bluetooth service and notifies the user interface since this method is called if the connection has stopped for some reason.

. in method `Stop`(), the Bluetooth server stops all its services and the any active connection socket (`CTBClient`) is deleted as well.

This class handles all actions including starting the Bluetooth server, stoping the server and creating/deleting a socket for the connection. As mentioned in the client implementation, the communication mechanism of objects was an implementation of an interface class, `CRemoteFrameObserver`. It provides some pure virtual

methods. `CBTServer` holds a reference to the interface class. Whenever a network event occurs, `CBTServer` call `StateChanged`() of the `CRemoteFrameObserver`. All objects inheriting the interface, `CRemoteFrameObserver`, then, would know of the change by checking their `m_State` members.

5.2.2.2 Protocol handler

All protocol related functions are implemented in the class `CBTClient`. `CBTClient` is a subclass of `CAsyncSocket`. There was no need to do any extra work here for the remote Bluetooth connection. After the `OnAccept`() of `CBTServer` was called, this object would have been created and could be used in the same way as in other TCP socket in Windows.

This class provides a couple of methods corresponding to different functionalities. When the class receives data, it checks the packet head. If it was a control packet, `ProcessCtrlPacket`(…) would be used for processing. Or, `ProcessDataPacket`(…) would be called if it was a data packet. As for receiving raw video frames, the server has to acknowledge (send an ACK message) each packet received from the mobile phone. The method `SendCtrlPacket`(…) was implemented for the purpose. Once the whole frame has been received, `GenerateFrame`() would create the image for the frame. To notify the user interface, this class has a pointer pointing to the implementation of `CRemoteFrameObserver` object.

Actually, in the final implementation, the raw frame related methods were not used. Only point position information was used.

5.2.2.3 Mouse controller

Because of the mobile system limitation of the five frame latency mentioned earlier, and the fact that in one second, the times of position information the server received was in the range from 3 to 15. The mouse control class, `CMouseServer`, therefore, was implemented to keep the mouse moving smoothly as expected.

`CMouseServer` was implemented as a separate thread. It provides four methods for other objects to control the mouse.

`StartServer`: start the mouse control server and waiting for commands. After calling this, the thread should start running.

`StopServer`: stop the mouse control server.

`IsRunning`: return true if the thread is running, false otherwise.

`MoveMouse`: send a command to the mouse control server. The mouse server then will set the mouse pointer to follow the command and fire the mouse button event if the command includes appropriate flags.

In method `MoveMouse`(…), a thread lock was used to prevent updating the mouse status with the half changed member variables. In the implementation for the moving speed, the distance to the centre point of the camera screen was taken into account.
If the circle was away from the centre point, the moving speed would become faster than for closer positions. If the black circle is right on the centre point, movement would stop.

For the thread programming, there was a global static C style method, named `MouseControlThreadProc`(…), used as a callback function when starting the

thread. When the function was executed, the current object was passed in this callback function. The member `Run`() of the object passed in (`CMouseServer`) would be called. In the method `Run`(), it keeps checking for a command and controlling the mouse pointer until the exit condition occurred. In our project, the mouse would keep moving for one and half seconds and the speed was gradually changed to zero. Whenever, a new command was set by `MoveMouse`(…), all status about the mouse moving would be changed immediately. In the experiments, the usability improved after the mouse thread was implemented.

5.2.2.4 User interface object

The user interface class, `CRemoteCameraDlg`, is a subclass of MFC class `CDialog`. It provides the buttons to start/stop the Bluetooth server, ask the mobile phone to send raw video, or track the circle. A video frame can be displayed on the form. Furthermore, this class implements the interface, `CRemoteFrameObserver`, in order to receive network events.

`CRemoteCameraDlg` implements four pure virtual methods and overrides the log method. When a whole raw video frame has been received, the `FrameReady`(…) method is be called. The `CImage` would, then be displayed on the form. In the method `PositionReady`(…), this object would the information to the mouse server. Also, if any network status changed, `StateChanged`() would be called.

The following image shows the display of received raw video.

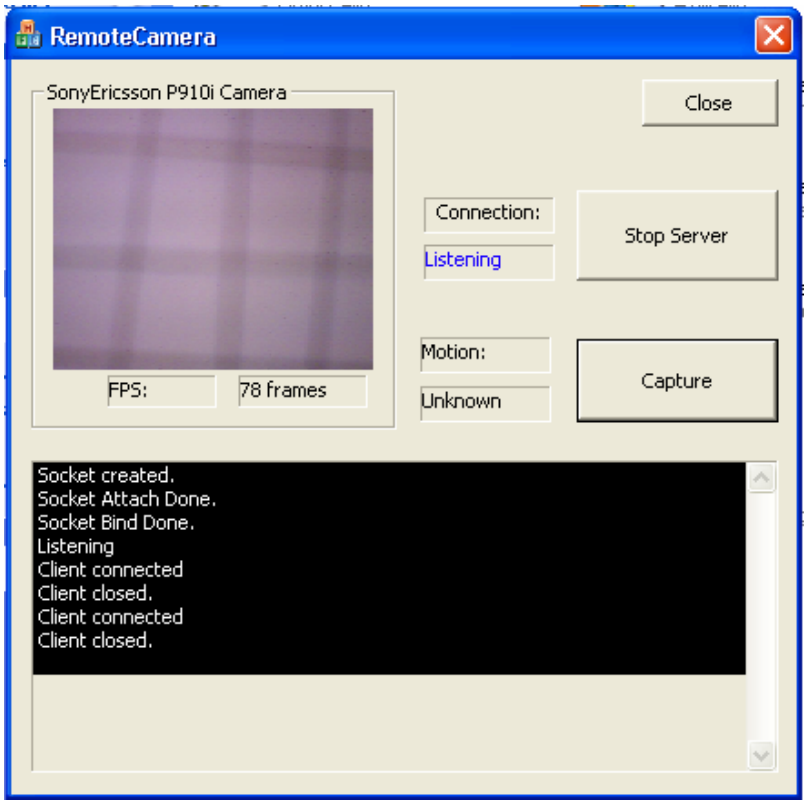


Figure 5.9: Screenshot of early implementation. Only raw video frames were sent. This image was taken after the connection closed.

In the final implementation, the server receives position information only.

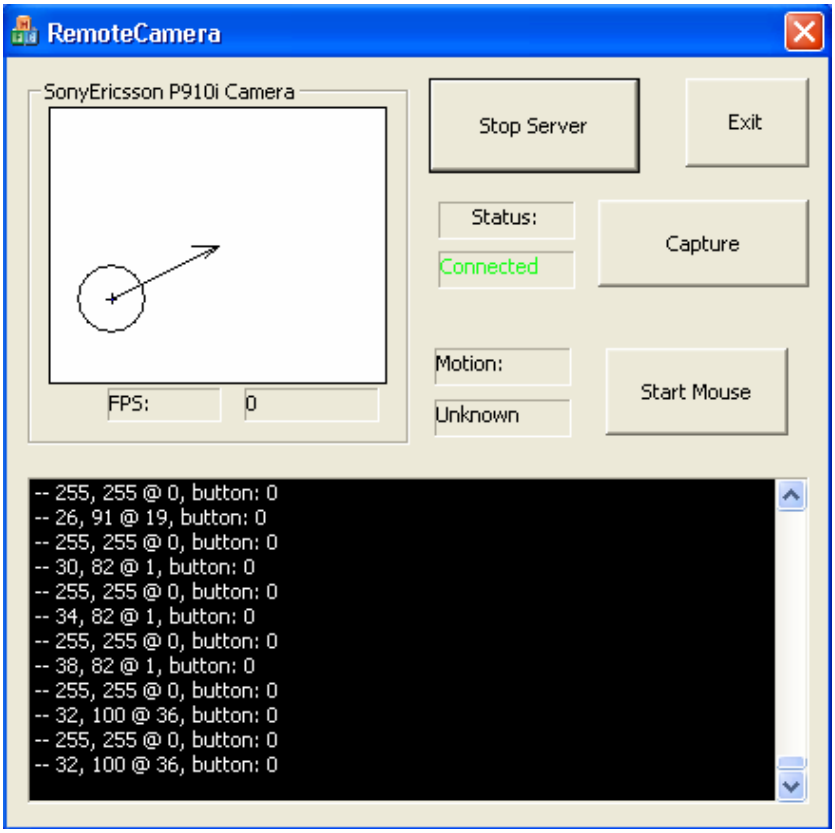


Figure 5.10: Screenshot of the server application. The bottom edit control with black background was used for debugging. In the left top area, the circle was shown based on the information received including the position and diameter of the circle. The arrow to the centre point means the direction the mouse would move to.

Chapter 6
Conclusion

In our experiments, the mobile device (phone) was successfully used as a PC input device. As the main purpose of this project was to examine the possibility of using a mobile device camera as input device, connected to a PC through Bluetooth wireless communication, our conclusion is that such usage is possible. During the experiments, video frames were successfully displayed in a PC application and more importantly, the PC mouse could be controlled by the mobile phone. Based on our tests, systems that extend the PC's input devices to include handheld equipment can be built using current hardware.

However, from our results we see that the usability of such a system still needs a lot of improvement to be suitable for daily use. There are three main reasons for poor usability. One problem is the limitation of P910i system software. The camera API that the Symbian OS in SonyEricsson P910i provides is missing some features which we really need in real-time video processing. The program always adds five-frame latency from the time that action starts. Ideally, the program in the mobile phone should start processing a frame immediately after the frame is captured. Another issue is that the transmission speed of Bluetooth for most current Bluetooth devices is not fast enough. Doing the image recognition in the PC could bring some more interesting features to the system. The mobile phone camera could also act as a web camera device for Windows PC, which could be used in any Windows applications. As the PC has more calculation capability than the embedded devices, the result of image recognition could be more accurate as well. As our experiments show, transferring raw frames in real-time is not possible. The peak transmission rate for Bluetooth 1.1 or 1.2 devices is 721 kbps. Even that was not reached during our experiments. Also, we have found that in the most cases, the speed of transmission depends on the position and direction of the Bluetooth dongle. This, therefore,

finally forced the decision that image recognition could only be implemented in the mobile phone.

Finally, as the hardware limitation, it is very common that the images captured become blurred if a user moves the camera (mobile phone camera or other cameras) at high speed. The problem cannot be solved by the software developers. Therefore, the mobile phone camera may not be suitable for those applications which need high speed mouse movement, for example, some shooting games. However, the blurring effect could be used to advantage. A real mouse can be lifted off the surface and moved without altering the curser position. High speed blurring offers a way of achieving a similar effect with the phone “mouse”. Rapid movements would return no position hits and could therefore be used for position shifting. We have not implemented this idea, so cannot draw conclusions about its practicability.

All in all, to build such a system is feasible, though there are some limitations. The five frame latency, about one third of a second, still may be acceptable in some applications.

Chapter 7
Further Work

Since the usability of the system needs improvement to provide a better user experience, further work would mainly focus on these issues. To improve usability, improvements could be made based on the conclusions in the last chapter.

At the hardware level, the Bluetooth specification version 2.0 provides a high speed transmission of 2.1 Mbps. There are already some mobile phones implementing Bluetooth version 2.0 and also some USB Bluetooth dongles. Using the faster devices would make the server receive information from mobile device more frequently.

Apart from the hardware issues, some software level improvements could be done in the future.

To make more reliable Bluetooth connections, some other Bluetooth SDKs could be used. For example, the Widcom Bluetooth SDK is more mature than the one Microsoft provides. It may also be possible to dig deeper into the mobile phone camera related APIs, allowing a system to process one frame at a time. This is supposed to be possible with the “FrameReady” callback function. Later releases may repair this bug or by doing some programming tricks, like pointer address analysis, setting some flags in the passed in frame data buffer, a method of dealing with one frame at each call could be found. If it was possible, mouse control would be more responsive. Also, the target finding algorithm could be more accurate for the circle information if some more lines instead of only the vertical and horizontal diameters were checked. By changing the light conditions, the best or better light condition could be found and by emulating the lights to make the binary algorithm

work better. The user interfaces of the client and server could be designed again for a better user experience.

Bibliography

Adelmann, R., Langheinrich, M., & Floerkemeier, C. (2006). *A Toolkit for Bar-Code-Recognition and -Resolving on Camera Phones – Jump Starting the Internet of Things.Workshop Mobile and Embedded Interactive Systems (MEIS'06) at Informatik 2006*. Retrieved October, 2006 from World Wide Web: http://www.vs.inf.ethz.ch/publ/papers/barcodes2006.pdf

Bluetooth SIG, Inc. (2006, January). *Specification of the Bluetooth System: Host Controller Interface.* Volume 04. Washington.

Bluetooth SIG, Inc. (2004, November). *Specification of the Bluetooth System: Master Table of Contents & Compliance Requirements, Covered Core Package version: 2.0 + EDR.* Volume 0-3. Washington.

Bradski, G. R.(1998). *Computer Vision Face Tracking For Use in a Perceptual User Interface. Intel Technology Journal,* Q2, p15.

Dasgupta, K. (2000). *Bluetooth Protocol and Security Architecture Review, Project Submitted for Computer and Network Security course*. The United States of America: University of Tennessee.

Davies, E. (1990). *Machine Vision*, Academic Press, 1990, pp91 - 96.

Fisher, B., Perkins, S., Walker A., & Wolfart, E. (1994). *Hypermedia Image Processing Reference, Adaptive Thresholding*. UK: University of Edinburgh. Retrieved May 11, 2006 from World Wide Web: http://www.cee.hw.ac.uk/hipr/html/adpthrsh.html

Gonzales, R., & Woods,R. (1992). *Digital Image Processing*, pp443 - 452. Addison-Wesley Publishing Company.

Hansen, T. R., Eriksson, E., & Lykke-Olesen, A., (2005). *Mixed Interaction Spaces – a new interaction technique for mobile devices*, UbiComp 2005. Tokyo, Japan.

Jain, A. (1986). *Fundamentals of Digital Image Processing*, p408. Prentice-Hall.

Kato, H. & Billinghurst, M. (1999, October). *Marker tracking and hmd calibration for a video-based augmented reality conferencing system. Proc. IWAR*, 10, pp85-94.

MSDN from Microsoft Official Website. (2004*). Converting Between YUV and RGB*. Retrieved November, 2005 from World Wide Web: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wceddraw/html/_dxce_converting_between_yuv_and_rgb.asp

Reitmayr, G. & Schmalstieg, D. (2003). *Location based Applications for Mobile Augmented Reality***.** *Proc. 4th Australasian User Interface Conference*. pp65-73, Adelaide, Australia.

Rohs, M.& Gfeller, B. (2004). *Using camera-equipped mobile phones for interacting with real-world objects.* Advances in Pervasive Computing. Retrieved January 9, 2006 from World Wide Web: http://www.vs.inf.ethz.ch/res/papers/rohs-gfeller-visualcodes-2004.pdf

Schall, G., Newman J., & Schmalstieg D. (2005, June). *Rapid and Accurate Deployment of Fiducial Markers for Augmented Reality,Proc. 10th*

Computer Vision Winter Workshop. Retrieved March 16, 2006 from World Wide Web: http://www.icg.tu-graz.ac.at/pub/pdf/schall05cvww.pdf

Sony Ericsson Mobile Communications AB. (2003, August). *User Guide to the Sony Ericsson Camera API.*

Sony Ericsson Mobile Communications AB. (2004, October). *Developers Guidelines: UIQ C++ and PersonalJava-P800, P900 and P910 series.* Retrieved October, 2005 from World Wide Web: http://developer.sonyericsson.com/getDocument.do?docId=65023

Symbian Ltd. (2002). *How to use the Camera API*. Retrieved October, 2005 from World Wide Web: http://www.symbian.com/developer/techlib/v70sdocs/doc_source/devguides/cpp/multimedia/onboardcameraguide/HowToUse.guide.html

Symbian Ltd. (2002). *System panic reference.* Retrieved October, 2005 from World Wide Web: http://www.symbian.com/developer/techlib/v70sdocs/doc_source/reference/SystemPanics/index.html#PanicsReference%2eindex

Symbian Software Ltd. (2004). *Symbian OS v7.0 help documentation: UIQ 2.1 SDK.*

Wikipedia Encyclopedia. (2006). *Absolute color space.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/Absolute_color_space

Wikipedia Encyclopedia. (2006). *CMYK color model.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/CMYK_color_model

Wikipedia Encyclopedia. (2006). *Color model.* Retrieved January, 2006 from World

Wide Web: http://en.wikipedia.org/wiki/Colour_model

Wikipedia Encyclopedia. (2006). *Color space.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/Colour_space

Wikipedia Encyclopedia. (2006). *OCIF.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/OCIF

Wikipedia Encyclopedia. (2006). *RGB color model.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/RGB_color_model

Wikipedia Encyclopedia. (2006). *YCbCr.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/YCbCr

Wikipedia Encyclopedia. (2006). *YUV.* Retrieved January, 2006 from World Wide Web: http://en.wikipedia.org/wiki/YUV

Wilson, D. (2006). *RGB/YUV Pixel Conversion.* Retrieved November, 2005 from World Wide Web: http://www.fourcc.org/fccyvrgb.php

Wilson, D. (2006). *YUV Formats*. Retrieved November, 2005 from World Wide Web: http://www.fourcc.org/yuv.php#Planar%20YUV%20Formats