University of Southern Queensland

Faculty of Engineering & Surveying

Development of a Laser Pointer Mouse (Volume 1)

A dissertation submitted by

Mr Anthony Gibbons

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Software Engineering)

Submitted: October, 2010

Abstract

This dissertation is a comprehensive report on the initial stages of the development of a laser pointer mouse for use in computer driven projector based presentations. The initial aim of the project was to develop software that enables the user to control a computer using only a webcam and laser pointer in a presentation environment.

The outcome of comprehensive background studies in this dissertation has found the existence of projects with working laser pointer mouse devices, but none of which had any of the features proposed for this project, especially with tracing a circle around an object to be clicked and auto-calibration.

This system in this project uses the DirectShow applications programming interface in Microsoft Windows to interface with the camera, obtain a frame and detect through the use of a "high-pass filter" the presence of the laser. The system is then able to return the position of the laser in terms of its corresponding pixel position. Reliable error tolerant laser detection, calibration, circle clicking and system integration was never achieved at this stage of the project due to a shortage of time and energy.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen

Dean

Faculty of Engineering and Surveying

Certification

I certify that the ideas, designs and experimental work, results analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Anthony John Gibbons

Student Number: 0050009494

____
Signature

____
Date

Acknowledgments

I would like to especially thank Dr. John Leis, my supervisor, for his continued encouragement and support not just during this project, but also through my entire program.

I would like to thank the members of my family for being patient and understanding during the challenging undertaking of this project, and I would also like to thank St Patrick's College, Mackay for providing me with the lab environment I needed for certain aspects of this work.

Mr Anthony Gibbons

University of Southern Queensland
October 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of modern slide show presentations to communicate to an audience is becoming increasingly popular. The equipment used to conduct these, what is commonly referred to as "power point presentations", is typically a computer running presentation software such as Microsoft Power Point, which is connected to a data projector that projects what is on the computer screen to a large screen that everyone in the audience can see.

In the majority of the situations, the person or person's conducting the presentation is commonly confined to where the computer is in order to control the presentation. This prevents the presenter from walking around the room and communicating more effectively with the audience. One way that this problem is solved, is through using wireless presenters, which are portable devices much like a remote control that enables the presenter to move between the slides remotely from anywhere in the room by pressing typical navigation buttons such as next and previous. Some more dedicated wireless presenters have a little joystick on it that is capable of controlling the mouse in what can be a slow and cumbersome experience for less experienced users.

The use of laser pointers for pointing things out on the screen is also becoming increasingly popular as many wireless presentation devices have these integrated in them. But laser pointers are more commonly purchased as small separate devices that uses button style batteries and have become very cheap over the last ten years, dropping from one hundred dollars down to just about five dollars. Now everyone uses them, not just as pointers but also for fun, and is sometimes even used to amuse pets.

If the laser pointer was able to be used as a mouse, and the presenter can control the mouse from anywhere in the room and even click on things, then it would greatly reduce the need for having to be near the computer, and also make interaction with the computer's mouse from a distance more natural.

A small collection of people have been able to produce such as system with various capabilities, most of these systems requiring a calibration process and also the use of dwell clicking to click on items on the screen. This makes it hard to differentiate between double clicking, single clicking and left and right clicking.

One way of solving this problem is to trace a circle around the object to be clicked; anti-clockwise for left clicking, and clockwise for right clicking, and possibly even a triangle for double clicking which would speed up the process of interaction.

To have a laser pointer mouse system that the user can just connect up and use with an ordinary laser pointer, and using a web-cam to detect the laser spot on the screen, without the need for calibration would be an enhancement of what has already been achieved.

1.1 Project Aim

The aim of this project was to develop software that can enable a user to control the computer of a projected presentation using only a basic laser pointer and webcam exclusively.

In order to enhance what has already been achieved by others, this project also aims to implement a system that can automatically calibrate, as well as enable the user to quickly click on items through tracing a circle around the item to be clicked.

1.2 Specific Objectives

- Investigate the concept of a laser-pointer mouse for use with data projectors by investigating literature on the topic, patents, and possible existing commercial devices.

  Obviously there is no point in ‘reinventing the wheel’, and so it would be appropriate to do some research into what other’s have already been able to do with regard to a laser pointer mouse. Then use some of their ideas to assist in achieving the outcomes of this project and also perhaps build on to what they have achieved.

- Investigate user interface technology and the need for such a device in the commercial market or specialized markets such as to aid persons with certain disabilities.

  In doing the above, it will be useful to know what other types of devices or systems exist that makes it easy for people to interact with projected presentation other than using the computer, and to determine whether the laser pointer mouse could make it easier for people with disabilities.

- Investigate color point detecting algorithms using a webcam-based vision system, and determine if the recognition of a laser spot can be performed in real-time.

- Investigate motion-tracking algorithms for the laser spot which account for the variability of the pointer motion, including circle-tracing to emulate mouse clicking.

  - If it is possible to track the laser pointer and determine around which item a circle has been traced, ways of detecting clockwise and anti-clockwise motion as well as even a triangle would be aimed for in order to make it possible to perform left, right and double clicking as well.
  - An additional feature will also be attempted that will allow the user to flash the laser on either side of the screen that will tell the system whether to move to the next slide or to move to the previous slide.
  - Another feature will be aimed for that will enable the user to perform a certain motion on the screen that will turn the laser pointer mouse to ei-

ther interaction mode, or pointing mode to prevent the user from making accidental clicks.

- Develop prototype code for testing the above algorithms, and assess the feasibility of the concept.

  Efforts will be made to develop code that can enable the laser to be used as a mouse, at least in its ability to determine the location of the laser on the screen and move the mouse there as well as being able to click on the item. This prototype code will be demonstrated in this dissertation, but due to time constraints, an independently working system may not be completed.

- Investigate the requirement for calibration of the pointer system, and implement prototype test code.

  An attempt will be made at developing a system that is capable of auto-calibration. This means that the user can start using the laser pointer mouse straight away after connecting the camera and starting up the software.

As time permits:

- Integrate the code into a system which can interface with the Windows API in order to inject mouse events into the system.
- Integrate into a complete standalone application which can use the laser as a mouse.

1.3 Overview of the Dissertation

This dissertation is organized as follows:

Chapter 2 Introduces the concept and delivers a comprehensive literature review of work done on the laser pointer mouse, and provides the motivation for the project.

Chapter 3 The development of the methodology and approaches is covered in this Chapter as well as the assessment of risks and consequential effects.

Chapter 4 This chapter describes the processes that was involved in setting up the working environment for this project, as well as setting up the integrated development environment.

Chapter 5 This chapter explains the work done on developing the usage of the webcam interface so that the software can retrieve a frame for processing.

Chapter 6 This chapter explains the development of the laser detection algorithm and shows some of the outputs generated by the software.

Chapter 7 This chapter discusses the results and covers some of the proposed improvements for the laser detection system.

Chapter 8 This chapter concludes the project and suggests future directions for the project.

Chapter 2

Background

2.1 Chapter Overview

This chapter explores some existing technologies that are being used to control presentations. It also provides a comprehensive review of some of the existing laser pointer mice devices that have been developed and provides a brief explanation of their function as well as what is perceived to be the good and bad aspects of each, and which of the good aspects could be included and enhanced upon in this project.

2.2 Existing Presentation Control Devices

Currently projected presentations can be controlled by a variety of devices other than the mouse and the keyboard. These have been found to fall broadly into two categories; wireless presenters and interactive white boards.

2.2.1 Wireless Presenters

Wireless presenters are commonly in the form of a device similar to a remote control; in fact they are sophisticated specialized remote controls that control the computer.

Most of them contain the typical buttons needed to control the presentation such as the next and previous buttons for navigating slides. One such example is the Professional Presenter R800 (Logitech 2010*c*) which is one of the many wireless presenters developed by Logitech. Other more expensive devices even has a way of controlling the mouse of the computer such as the Targus Basic Presenter (TechBuy 2010). And then there are other designs such as the Wireless Notebook Presenter Mouse 8000 (Microsoft 2010*b*) developed by Microsoft that is basically a wireless mouse with presentation controls underneath. The mouse is wireless so all the presenter needs to do is find a flat surface nearby to use the wireless mouse in the normal way to control the presentation remotely within the theatre.

2.2.2 Interactive White boards

The other method of controlling the presentation is in the form of an interactive white board which in most cases looks like an ordinary white board used in class rooms. These white boards have the added ability of the user being able to use the white board as a touch screen when the screen of the computer is projected onto it using a data projector. This enables the user to control the computer by touching the white board where various interactive elements such as menu items and buttons are projected onto from the computer. In most cases the white boards are connected to the computer either wirelessly or physically using the Universal Serial Bus (USB) interface. Commonly these boards have a grid of copper wires behind them that are joined together and eventually connected to the computer which sends the signal of where the user has touched the screen that the computer can use to perform interactive events (Nan Wodarz October 2005).

Usually at the start of the setup between the computer and the white board some sort of a calibration process has to be performed in order to allow the device driver software on the computer to determine the correct pixel coordinates of where the user has touched it. Typically the software at the start displays a set of markers on the board that the user must physically touch so that the computer can understand which points on the white board corresponds to certain critical points on the computer screen, such as the four corners of the computer screen (Nan Wodarz October 2005). From these points the software can calculate the corresponding pixels on the computer screen from the point pressed on the white board so that the user can accurately interact with the computer.

Types of Interactive White boards

Generally there are three categories that interactive white boards belong to. The first is a touch sensitive membrane that consists of two layers of flexible surfaces with a small gap in between the two (Nan Wodarz October 2005). When the user touches this two layered surface a signal is generated and sent to the computer that tells it where on the screen the user has touched it. This is similar to how an iPod Touch works (Wilson & Crawford 2010).

The second type of interactive white board is the electromagnetic design which is similar to the traditional white board in that it has a hard durable surface, but underneath this surface is a grid of wires that that detects the signal that comes from a special pen that transmits an electromagnetic signal (Nan Wodarz October 2005, Wikipedia 2010*c*)

The third type of interactive whiteboard is a little less dedicated than the last two and the cheapest option. This is basically an ordinary white board that has been upgraded with the inclusion of an ultrasound kit. The ultrasound kit uses scanners that are mounted on the top corners of the board (Nan Wodarz October 2005),or can also come in the form of a bar that is affixed on the top of the white board or even a wall (Wikipedia 2010*c*). A special pen or pen housing is used to interact with the board which is detected by the scanners which sends the coordinates back to the computer. All three of these types of white boards have some sort of integrated circuitry attached to them that allows for the detection of the position of a pointing device of some sort.

Another type of interactive white board solution that is very close to the functionality of the laser pointer mouse is the use of a Nintendo Wii remote. Though they may be considered a bit of a "hack" they do work, and are based on sound electronic and optical principals, invented originally by Johnny Chung Lee, PhD. in 2007 (Wikipedia 2010*c*). This system was developed by Johnny Chung Lee (Lee 2008) which uses the infrared

(IR) camera on the front of the Wii remote control to capture and detect the IR pen in front of the white board. The user simply uses this pen to interact with the computer from the computer screen image projected on the white board.

The reason why this design is very close to the proposed laser pointer mouse is because it uses a camera to detect the source of the infrared light at the tip of the IR pen relative to the projected image of the computer screen. So some form if image processing needs to be performed on the image captured by the camera in the Wii remote, similar to what needs to be done with the location of the red dot on the screen produced with the laser pointer.

2.3 The required qualities of the solution

The quality of the presentation control system can be assessed against four easily understood qualities such as user friendliness, availability, performance and cost. How well this system performs in these four qualities will ultimately affect its popularity, and the criteria that the device needs to meet in each of the four qualities will be discussed below.

2.3.1 User Friendliness

For the system to be user friendly the user needs to be able to use the system requiring very little training or need for reading instructions. To be able to interact with the computer the actions to be performed needs to feel natural, and the installation of the system needs to be quick and straight forward.

Wireless Presenters

Of all of the wireless presenter devices reviewed here, the Microsoft Notebook Presenter Mouse 8000 (Microsoft 2010*b*)and other similar devices that come in the form of a wireless mouse is probably one of the most user friendly wireless presenter devices because of the fact that it is a mouse and can be used wirelessly up to 10 meters away from the computer which enables the user to quickly and comfortably control the mouse on the computer. It has the standard presentation control buttons underneath such as next, previous, volume up and down and the fact that they can be customized to control anything makes this device very flexible. And since it is a mouse, if the user needs to control something else other than the presentation on the computer, the mouse can be used wirelessly. This device also has a laser pointer integrated into it which allows the user to highlight things on the screen. Because of this mouses plug and play capabilities it is easy to install as well which adds to the user friendliness of this device.

The next most user friendly device would be the wireless presenter remote controls that have mouse control capabilities such as the Targus AMP17AP Wireless Presenter with Cursor Control (Targus 2010). This device also comes with the presentation control buttons and laser pointer but the reason why it is not as user friendly as the wireless mouse presenter is because even though the mouse is controllable from this device, it is more cumbersome and lower because of the small joystick. The joystick simply doesn't provide really fast and comfortable control over the mouse as the Microsoft Presenter Mouse.

The other wireless presenter devices that do not contain mouse control would rank last in terms of user friendliness because there is no control of the mouse whatsoever and the user can only control the slides of the presentation.

Interactive White boards

From the information available in the literature about the interactive white boards (Lee 2008, Nan Wodarz October 2005, Wikipedia 2010*c*), no matter what technology they have they are all equally as user friendly because the user interacts with them in the same way by using some sort of special pen and tapping the desired interaction elements on the screen to interact with them. If the cost and availability wasn't an issue these systems would be more user friendly than the wireless presenters because the user interacts with the presentation directly.

Other factors that will influence user friendliness will be the ease of installation and calibration, but more detailed research into this is beyond the scope of this project since its not concerned with the development of the laser pointer mouse.

2.3.2 Performance

When looking for good performance in this interaction system, attributes such as speed, reliability and robustness is sought. Measuring the speed for this system is the same as measuring the time it takes for the computer to react to a mouse click, or starting up a program such as a word processor or game. So when the user selects a menu item from the projected image on the white board, the time taken to react and cause the menu to drop down compared with just a normal keyboard or mouse interaction for the same item is an instance of the indication of the speed of the system. The closer the reaction time is to a normal mouse or keyboard interaction the faster the system.

Reliability translates to the system being able to detect the correct position of where the user is pointing at and not cause false positives, which is when the computer reacts in a way that is not expected by the user such as the mouse pointer being purposely positioned away from where the user is pointing.

Robustness is demonstrated when the system can handle unexpected inputs from the user and not cause an error to occur, or the system to behave unexpectedly or even crash.

Wireless Presenters Performance

Speed wise the wireless presenters would be higher performing than interactive white boards but not by a really large amount. This is because the signals generated by the wireless presenters do not require a large amount of processing of information and also do not require to be calibrated unlike the interactive white boards. So their performance would be similar to that of the wireless keyboard and mouse.

Because of the lack of calibration requirements and processing of information to determine the correct position of the mouse and point of interaction, the wireless presenters although limited in capability, compared with interactive white boards are more reliable and more robust because there is not much that can go wrong.

Interactive White boards Performance

Because of the calibration requirements of interactive white boards and delicate and sophisticated methods of detection of interaction, the interactive white boards can be seen as less reliable and robust than the wireless presenters but are capable of more in terms of interactivity and user friendliness.

The speed of interaction with the interactive white boards would vary based on their design. The Nintendo Wii adaptation although the cheapest upgrade (Lee 2008, Wikipedia 2010*c*), would be the slowest and less reliable because of the use of a camera and the need for the processing of the frames captured of the screen and the IR pen. On the other hand, the touch sensitive membrane implementation of the interactive white board should be the fastest and more reliable (but less robust) because of straight signals being sent to the computer and no need for any further processing other than calibrating the coordinates received. The most robust of the interactive white board designs would be the ultrasonic upgrade and electromagnetic designs because their detection mechanisms can be damaged by aggressive interaction such as with the touch sensitive membrane design.

2.3.3 Cost

For the system to be cost effective means that generally any individual can acquire it without much financial difficulty and can get hold of it fairly quickly such as within one or two weeks based on an average low to middle income salary. So the sooner a person can acquire the system without putting any real strain on finances, the more cost effective the system is. Another costing factor to take into account is the cost of upkeep such as the cost of replacing the batteries or lamps. When taking cost into consideration of course, it is assumed that the user already has a computer, projector and screen (although the screen is not absolutely essential).

The wireless presenters are obviously the cheapest option for controlling presentations being around less than $200 some even as cheap as $50 (TechBuy 2010). The Microsoft Wireless mouse presenter (Microsoft 2010*b*) would be the best because of its user friendliness and price. Although the associated running costs of wireless presenters would be the cost of replacement batteries, but this is not really a serious issue.

Interactive White boards on the other hand are more expensive because of their dedicated role and sophistication. The cheapest option for the interactive white board would be the hack alternative of using the Nintendo Wii remote (Lee 2008, Wikipedia 2010*c*). If one already have the remote as part of a Nintendo Wii entertainment system then a working basic interactive white board can be achieved for very little cost, even cheaper than the laser pointer mouse would be.

A dedicated interactive white board would be far more expensive. Eyo Technologies retails a Panasonic UB-T760 Interactive Panaboard for nearly $2000 (Eyo 2010). So this is definitely not an attractive method of interacting with the presentation for an individual.

2.3.4 Availability

Availability means how quickly the system can be obtained and how easily it is accessible when money is not a factor. It is more likely to find a wireless presenter in a department store in your everyday shopping center than an interactive white board which would only be available from specialized suppliers or shops.

2.4 The need for the laser pointer mouse

The problem with the interactive white-board is that the presenter can't move around the room while talking to the audience and although there aren't any real serious problems and disadvantages with the wireless presenters, just being able to use a simple laser pointer and being able to directly interact with the computer using this laser pointer while using it as a normal pointer, just makes it easier and more convenient than the wireless presenters. This project aims to develop a laser pointer mouse which enables the presenter to use it as a pointer and an interaction device. The laser pointer should be an ordinary red beam laser pointer and the detection device a relatively cheap web cam.

2.4.1 The user friendliness of the Laser Pointer Mouse

Upon successful development of the laser pointer mouse the device will be very user friendly in the sense that all the user has to do to navigate between slides is to flash the laser on either side of the screen, on the left side for previous slide and on the right side for next slide.

For the user to move the mouse or to interact with the computer, all the user has to do is move the laser pointer to the desired button or menu item and trace a small circle around the object. This will cause the computer to click on the object.

This is the quickest method of interaction compared with a wireless presenter because if using the Microsoft Wireless mouse, the user has to find a flat surface first in order to use the mouse (which may not look very professional in front of an audience), and if using the Targus presenter, moving the mouse is even slower because of the interaction with the joystick which is uncomfortable.

When compared with the interactive white board, it may seem that the interactive white board would be more user friendly, but that means that the presenter needs to stay near the screen and talk to the audience. With the laser pointer mouse, the screen can be controlled from a distance allowing the presenter to move around the room and talk to the audience.

The laser pointer mouse seems better than both the wireless presenters and the interactive white boards, but there might be a problem with using the laser pointer when trying to click on things by tracing a circle. It is very hard to keep a laser pointer steady so tracing a circle around an object will require a serious lack of coffee in the body, strong hands and some practice, and being nervous in front of an audience and trying to use a laser pointer can be really difficult.

2.4.2 The performance of the Laser Pointer Mouse

Getting the computer to detect the position of the laser on the screen will require a fair bit of image processing. One of the other factors that will influence the performance of the laser pointer mouse as pointed out by Popovich (Popovich n.d.) in his paper on the topic is the performance of the webcam. Low resolution and automatic brightness control will make it very difficult to detect the laser and slow frame rates will make it hard to keep track of laser movements in order to perform certain interactive actions on the system.

The quality of the laser detection technique, the quality of the webcam and laser as well as the processing ability of the computer will determine the performance, speed, robustness and reliability of the laser pointer mouse.

2.4.3 The cost and availability of the Laser Pointer Mouse

For what this system will be able to do, it will be the most cost effective system because laser pointers can be found at any electronic stores and can also be found on a lot of these wireless presenters. The cost of a laser pointer ranges from $15 to about $50 at places like Jaycar Electronics or Dick Smith Electronics and these stores are present in most regional cities and some of the larger towns around Australia. The difference in the quality of the laser spot between the cheapest and most expensive laser pointers is negligible.

The other component that this system will need is a webcam. Webcams ranges from about $20 to $100 and are more readily available than laser pointers at any computer or electronics store. Generally the more expensive the webcam, the better the quality of the image and the frame rate as well as manual controls and perhaps even the availability of an Application Programming Interface (API) for the webcam to automate the system.

So it is possible to have a laser pointer mouse put together for around $40 using the cheapest and nastiest of laser pointers and webcams. But this will provide the ability to remotely control the presentation from anywhere in the room without having to go back to the computer.

2.5 Current Developments on the Laser Pointer Mouse Concept

There have been a number of projects being conducted in the development of the laser pointer mouse. During the research conducted for this project, three projects or systems in particular stood out to be the closest to the requirements of this project. But other projects have also been found to contain helpful information for the development of this project. Two patents on the concept have also been found, but none of them have anything about tracing a circle around interactive objects on the screen. Currently there are no commercial devices that are being sold that are capable of working as a laser pointer mouse. In the next section the findings of the research into these existing projects will be summarized.

2.5.1 Patents

The two patents that have been found during the research are (Raynor 2007) and (Finley 2003). Both of these patents are very generic and tries to cover everything that could be developed using a laser pointer and a web-cam.

(Raynor 2007) is a European patent with the inventor being Jeffrey Raynor. This patent proposes a laser pointer mouse system that uses a laser pointer and a web-cam, but the laser pointer is a special one that has been modified to modulate it's light according to certain frequencies that dictates which button has been pressed. This means that one can not use this system with an ordinary laser pointer, but need to get hold of one that has been designed to output light signals so that clicking events can be performed.

The other patent that have been found is one by (Finley 2003) which is an American patent with the inventor being Michael Cain Finley. This patent proposes or covers a system that doesn't require specialized hardware and can read the laser pointer of any display device, not just the projector screen. This system also proposes to have the ability to read hand writing patterns off the screen so that it can be determined as character input signals like what one would get from the keyboard.

This is to prove that patents definitely exists and that the idea is not new. None of the patents however covered tracing a circle around an object in order to be able to click it and there is also nothing mentioned about being able to 'auto-calibrate' the system.

2.5.2 The current existence of the device

A lot of interesting work on the concept of using a laser pointer as a mouse has been done by various groups and individuals. Some of these systems developed perform really closely to the requirements of the laser pointer mouse for this project, with the exception of being able to circle objects on the screen in order to be able to click on them. One of the laser pointer mice that have been developed was developed by a team of four people from the Saha Institute of Nuclear Physics (Atul Chowdhary n.d.) the principal author of the paper being Atul Chowdhary. This particular system uses a red filter in front of its camera to make it easier to detect the laser on the screen and detects the laser at the blue layer of the resulting image. This system however like most of other systems developed requires manual calibration before it can be used effectively.

Richard de Bruijn (de Bruijn 2008) from Eindhoven University of Technology developed a system that is capable of automatically calibrating and uses hue, saturation and value detection methods to detect the laser which means that this system is not limited to having a physical filter in front of the camera.

Another system that matches closest to the requirements of this project has been developed by Kelvin Cheng and Kevin Pulo from the University of Sydney (Cheng & Pulo 2003). This system is capable of performing click events by tracing circles around objects (which is a requirement of this project) and uses an approach that seems really simple, but unfortunately this system can only detect the objects being clicked on if their coordinates are known. So this project will be improving on their development by allowing anything that falls within the mid point of the circle that was traced to be clicked.

2.5.3 A very brief overview of the main requirements of the laser pointer mouse

The functional and non-functional requirements was covered in Chapter 1, but in order to set the context of this section, the main requirements for the laser pointer mouse which will be developed in this project is given below. The most essential of them being the use of an ordinary web-cam, ordinary laser pointer, an average computer, an average projector, no special hardware, and the ability to fully control the mouse.

1. Ordinary red laser pointer
2. Use a webcam
3. No special hardware
4. Ordinary Projector
5. Fully control the computer in the same way as with using a mouse
6. Be able to click on objects
7. Clicking to be performed by tracing circles around the object
    (a) Clockwise for right click
    (b) Anticlockwise for left click
8. Auto calibration

So the closer the existing devices matches the essential requirements given above, generally the more useful it would be to study the device to see how the authors managed to get the laser pointer mouse to work. How Atul Chowdhary, Richard de Bruijn, and Kelvin Cheng's projects matches the requirements for this project is briefly explained in the section below along with some other projects developed.

2.5.4 How existing devices meets the requirements for this project

(Atul Chowdhary n.d.), (de Bruijn 2008), and (Cheng & Pulo 2003) all use laser pointers to to control the mouse and use web-cams to capture the laser pointers on the screen, however (Cheng & Pulo 2003) uses an infrared laser pointer rather than an ordinary red laser.

Using an infrared laser pointer means that the actual laser point is not going to be visible to the human eye. Cheng's reason for this design is that when anyone uses a laser pointer, the immediate problem that is noticed, is that of the instability of the human hand, which results in the user not being able to accurately point at something, which will confuse the audience (Cheng & Pulo 2003). The other problem is that of cursor lag, which means that because of the time taken by the computer to determine the location of the laser pointer and consequently the new location of the cursor, the cursor will always drag behind the laser point almost as if the cursor is following the laser. With this in mind the slower the computer and the faster the laser pointer point is being moved around on the screen, the further behind the cursor will be. So Kelvin's solution is to just hide the cursor, which is a good idea to easily solve the cursor lag problem without too much effort. The benefits of hiding the cursor will be discussed a bit further on in this section.

(Atul Chowdhary n.d.), (de Bruijn 2008), and (Cheng & Pulo 2003) also uses ordinary data projectors which is a requirement of this project, and none of these systems make use of any other hardware other than the web-cam and laser pointer, and these systems are being used with an ordinary computer system and data projector, which enables for a cost effective and readily available solution with no special hardware. The only minor exception to the above point is that Chowdhary (Atul Chowdhary n.d.) uses a red filter in from of the camera to make the detection of the laser easier and Cheng (Cheng & Pulo 2003) uses an infrared laser pointer instead of an ordinary one.

In all three of the projects (Cheng & Pulo 2003, de Bruijn 2008, Atul Chowdhary n.d.), the system enables the user to perform mouse clicks using the laser pointer. Chowdhary (Atul Chowdhary n.d.) and de Bruijn(de Bruijn 2008)uses a method known as dwell clicking, which is performed by holding the laser point at the desired point of interaction for a small amount of time which allows for the system to detect that this is where the user wants to click and then performs a click event system call to perform the click. Holding the laser at that point for a bit longer causes a double click to occur.

The advantage of this method is that it is easy to implement, but the disadvantage is that it slows down the process of interaction which means that the system requires for the user not to be in a hurry. The other problem with this is that if the user wanted to keep the laser at a word on a slide or a diagram to emphasize something the system will cause an unwanted click event to occur. Chowdhary's way of solving this problem is to enable the user to switch into different interaction modes, one of the modes enabling the user to disable the mouse interaction so that the laser can be used as a pointing device.

The method of clicking to be achieved by this project is through tracing a circle around the desired object to be clicked. So far in this literature study, no project on the development of the laser pointer mouse, developed a method of clicking on elements through tracing a circle. One project however, that came close was carried out by Cheng and Pulo (Cheng & Pulo 2003) who's system uses an infrared laser pointer. Their system relies on known and predefined bounding boxes or shapes around the interaction element. When the point of infrared light passes over this shape, the shape will change color to show that it is highlighted. The user then traces a circle around this shape to perform a click. How this is done will be discussed in section ??? below, but their method will be extremely helpful in developing an algorithm that can enable clicking on anything, even if its location is not known.

Of these three projects (Cheng & Pulo 2003, de Bruijn 2008, Atul Chowdhary n.d.), only one talks about being able to automatically calibrate. This means any sort of manual calibration at the start is not required and that just being able to put the camera down and point at the screen is all that is required. Richard de Bruijn (de Bruijn 2008) managed to design a system that automatically detects the shape of the screen in the image captured by the camera, and finds the four corners of the shape. It then uses these four corners to produce a matrix that then gets used to multiply with the captured coordinates from the camera to obtain a close approximation of the actual coordinates of on the computer screen being projected. The master's thesis by de Bruijn (de Bruijn 2008)contains a lot of information on how this is done.

The calibration method being used by Chowdhary (Atul Chowdhary n.d.) uses roughly the same principle in (de Bruijn 2008) referred as projective transformation which uses a transformation matrix also generated from obtaining the four corners of the screen, Chowdhary's system requires the manual specification of the coordinates of the corners of the screen whereas (de Bruijn 2008) does this automatically.

Cheng and Pulo's infrared laser pointer mouse requires the exact alignment and adjustment of the camera so that the entire screen fits its entire field of view in order to obtain the right coordinates. At the end of (Cheng & Pulo 2003) there is a mention of future work to be done which will include automatic keystone calibration. But no specific calibration techniques have been given in (Cheng & Pulo 2003).

Of the three projects, the most helpful sources would be in (de Bruijn 2008) and in (Atul Chowdhary n.d.) because the calibration techniques described appears simple and effective enough to implement without too much difficulty. The automatic edge detection techniques explained by (de Bruijn 2008) would be a good starting point for this project's auto-calibration processes and the information in both (de Bruijn 2008) and (Atul Chowdhary n.d.) on the projective transformation matrices would be useful in determining the actual location of the laser pointer dot.

2.5.5 Other similar computer vision based interaction technologies

Use of Laser and Web Cam

Some other projects undertaken that also aims to develop some sort of a laser pointer mouse system have been reported in papers by (Toshiharu Wada 2007, Popovich n.d., Olsen & Nielsen 2001, Carsten Kirstein 2002). All of these systems aims to control the computer through using a computer vision system to locate the dot of the laser pointer on the screen. Other projects that uses vision systems in other ways that doesn't include the usage of a laser pointer have been described in papers by (T.V.Thati 2005, Kelvin Cheng 2006, Dekel & Kirkpatrick n.d.).

Toshiharu Wada's system has got some more specialized hardware that is used to detect the laser. The camera that is being used in this project is a frequency demodulated CMOS image sensor which can detect intensity modulated light (Toshiharu Wada 2007). As a result of this technology a special laser pointer is used that modulates its light according to a certain frequency. The frequency demodulated CMOS image sensor is operated at the same frequency which enables it to easily capture the location of the laser point on the screen. Unfortunately because of this specialism hardware, this system would as a result be more expensive and not as readily available as a system that uses an ordinary web-cam and laser pointer.

The system by Carsten Kirstein (Carsten Kirstein 2002) detects the laser pointer by calculating the difference between a reference image and the current frame, and then uses a pattern recognition technique to detect the laser in this changed region. Because of Kirsteins method of only attempting to find the laser in the changed regions of the image, this system may be fast and efficient and as a result may be a good technique to be used in this project.

Popovich's system uses a Logitech presenter mouse that has a laser pointer built into it. The method that is being used in this project (Popovich n.d.) to detect the laser pointer is the application of a high pass filter focusing on the red component of each image. Popovich's paper contains some pretty nice mathematics that can be of really good help when implementing the laser pointer mouse being developed for this project. Another nice feature of Popovish's project is that the program itself runs as a daemon in the background and doesn't interfere with anything and can be enabled and disabled by using keyboard shortcuts. This paper (Popovich n.d.)also includes some detailed matrix algebra that explains how to find the position of the laser point from the captured image so that it corresponds with the computer screen mouse position.

Olsen's project (Olsen & Nielsen 2001)also uses a web-cam and laser pointer, but what makes this project unique is the method being used to detect the laser which enables for more efficient laser pointer detection. This system uses a two level technique, where it first searches for the brightest red spot in the image and if it is acceptable it returns it as the position of the laser, but if it finds several red sports or none at all, the method being used then is the application of a convolution filter which makes the detection of the laser slower, but this is simplified by just searching in a region around where the last laser was detected.

Some other projects that uses computer vision systems but not laser pointers for interaction are projects by (Dekel & Kirkpatrick n.d., Kelvin Cheng 2006, T.V.Thati 2005). (Dekel & Kirkpatrick n.d.)produced a system which has got more to do with how various lecture theater equipment is controlled. Most modern lecture theaters have some sort of a touch screen that enables the presenter to control the projector, lights, sound system etc. Dekel's system extends this system to using a laser pointer and computer vision system to point at various devices that needs to be controlled around the room. So by pointing the laser at a light will cause the light to either turn on or off. Since this system is not concerned with controlling the computer mouse, its usefulness in this project is limited and as a result will not be discussed any further.

Cheng's invention (Kelvin Cheng 2006)enables the user to control the mouse through just pointing at the screen in mid air. The computer vision system uses the distance between the finger and the user's eye to determine the location of the mouse. So from the user's view, the mouse will be close to the tip of the user's finger if it is pointing at the screen. This system is quite nice and clever but does not fulfill the requirements of this project because a laser pointer is not being used and as a result it would be hard to be able to point to items on the screen with just the finger.

Another interesting application of computer vision techniques is described in (T.V.Thati 2005). This system is able to turn any monitor or screen into a touch screen through computer vision. This project employs complex mathematical techniques to accomplish calibration which may come in handy in this project if required. But due to the fact that no laser pointer is involved, much of (T.V.Thati 2005) would not be useful for this project.

2.6 Chapter Summary

With the increase in the use of technology, especially with its use in presentations, the demand on quality delivery of the presentations are quite high, and the use of technological aids to fulfill this need is becoming increasingly popular. Wireless presenters coupled with a laser pointer is a common tool for controlling presentations and for the more wealthy organizations the use of the electronic white-boards makes for a more productive interactive experience for the presenter.

The development of a laser pointer mouse in this project aims to make it possible for the presenter to control the entire presentation using just a single simple everyday red laser pointer. What would make this project unique from other devices and systems already developed by (Toshiharu Wada 2007, Popovich n.d., Olsen & Nielsen 2001, Kelvin Cheng 2006, Carsten Kirstein 2002, de Bruijn 2008, Atul Chowdhary n.d.) is it aims to make it possible to click on interactive objects by circling them with the laser, anti-clockwise for a left click and clockwise for a right click. If time permits, the development of auto-calibration along with all of the good qualities from the existing devices mentioned in this review, would make for a truly user friendly and reliable laser pointer mouse system, that if it was open source software and published as a free download would make it immensely popular.

Chapter 3

Methodology and Approaches

3.1 Chapter Overview

This chapter covers in a great depth the proposed methodology for the development of the laser pointer mouse. Firstly the most obvious tasks is explored and from that the overall system architectural design is presented.

After this the work break down structure is explained in more detail which is then followed by the project time line.

A few sections at the end of this chapter is dedicated towards exploring the risks and consequential effects that the project will have.

3.2 Determining the methodology

Determining the tasks to be carried out for this project started with a dependence on current knowledge and experience. The first thing that needed to be done was to come up with the obvious steps to be taken. From there, these steps needed to be broken down into smaller components which exposed some tasks which were uncertain, and needed some more research and experimentation to become clear for further planning later on in the project. This breakdown of tasks was performed to a level where the amount of time that could be spent could be easily estimated. This resulted in the development of the work breakdown structure (WBS). Which was used to determine the time lines for this project as will be covered in section 3.7.

3.3 The obvious tasks

The first step in determining what needs to be done is to know what some of the obvious things are that needs to be done which starts at a high level. Obviously all of the necessary equipment both hardware and existing software needs to be obtained.

The first step is to get the picture from the camera, then some code needs to be developed to find the dot of the laser in the picture. This code will then find the laser point and then provide the raw location information of the laser point in terms of pixels. After this, the raw location information then needs to be calibrated in order to find the actual position of the laser relative to the borders of the screen. After this, this information needs to be translated into coordinates that are ready to be passed onto the operating system (OS) in order to tell the OS where the mouse should be pointing at next. This constitutes the laser pointer finding component of the software.

The next component of software that needs to be developed is the software that is able to determine whether a circle was traced and if so, where the center of the circle is in order to perform a click on whatever is in the center of the circle. How to perform this is not that obvious so the amount of time that will be allowed for this will be substantial, and reference to existing sources such as (Cheng & Pulo 2003) which explains their development of circle tracing through using a set of four boxes joined together could be useful. More on this will be discussed in section 3.6.5. There is also the requirements of other slide navigation techniques such as next and previous that needs to be developed that also requires motion tracking.

Another component is the to do with calibration and hopefully if time allows, auto-calibration. This will have a lot to do with knowing the location of the four corners of the screen on the image captured by the camera, and being able to use this information to adjust the location information of the laser to yield the true coordinates in the screen where the mouse needs to be pointing at next.

So the obvious components in developing the laser pointer mouse is outlined below:

3.3.1 Components of this project

1. Setting up and preparation
2. Interfacing the camera
3. Finding the position of the laser point
4. Calibrating the position of the laser with the screen
5. Circle clicking
6. Slide Navigation
7. Automatic calibration
8. System integration into one standalone package
9. Reporting Requirements

These components can be seen as some of the milestones of the project. The last item is the reporting requirements for this project which is the production of this dissertation. This item is necessary because the production of this document is a major task and needs to be included in the time lines.

3.4 The overall system architectural approach

This laser detection system will consist of three main subsystems, and the relationships between the subsystems is shown in the top level system design diagram shown in Figure 3.1. The main subsystem will be the laser detection subsystem, which will be quite complicated in that it deals with the interface to the webcam, the detection of the laser and the calibration of the laser. The other large subsystem will be the circle

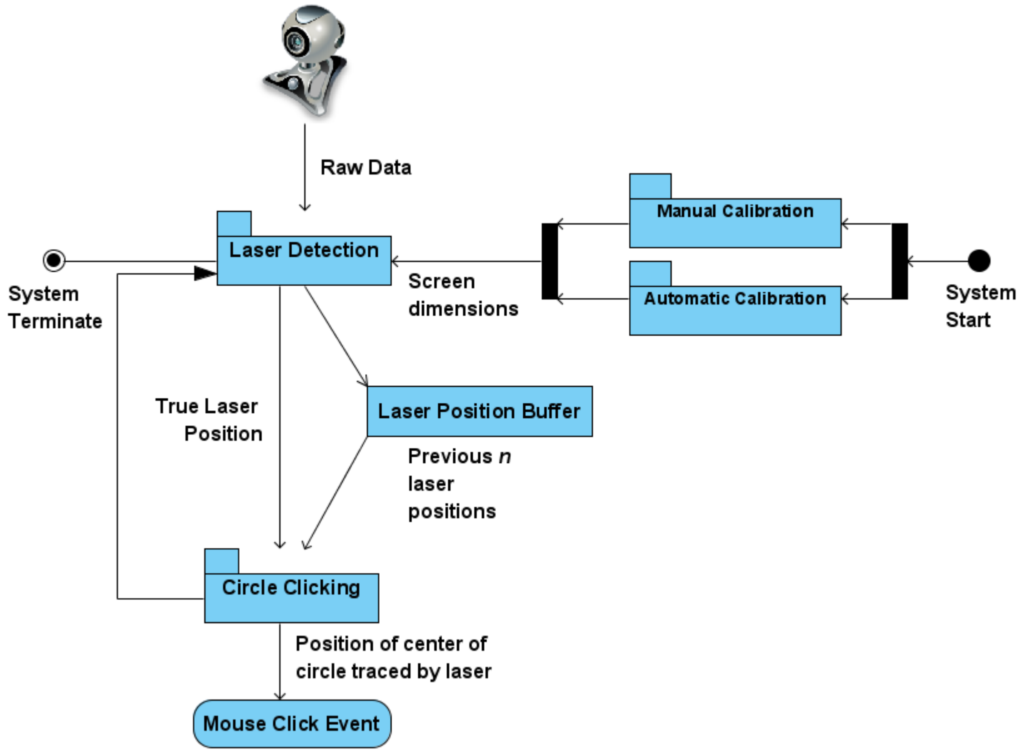


Figure 3.1: Laser pointer mouse overall system design showing the various subsystems

detection subsystem which will deal with the motion tracking of the laser in order to detect the presence of a circle being traced and return the coordinates of center of the circle. The other subsystem is the calibration subsystem which will consist of manual and automatic calibration.

3.4.1 Calibration Subsystems

The aim is to have automatic calibration, but before this is achieved, manual calibration will need to be achieved. Therefore it is shown in the system design diagram (Figure 3.1) as two separate subsystems close together and may both contribute to the calibration of the laser.

Manual calibration might involve the system having to project a special calibration image onto the screen that displays four bright dots, one in each corner of the screen and the rest of the screen just black. The system then obtains an image from the webcam of this projection and then detects the position of the four dots on the screen in the form of pixel coordinates that then later gets used to determine the true position of the laser. This is a manual process that takes some time at the beginning of a session before the system can be used effectively.

With auto-calibration the system would be able to dynamically detect the position of the four corners of the screen and then return the pixel coordinates that can be used in calibration. This could be done before the session like with manual calibration, or it could be done "on the fly" during each laser detection, or every predetermined number of frames being read in. This latter method may result in a drop in response time due to the additional graphics processing that will be required for this.

Both the manual and automatic calibration methods could be used. The manual calibration method could be used as a backup if the automatic calibration doesn't work. But for the project, achieving the manual calibration would be more important than the automatic calibration. The calibration subsystems provide the laser detection subsystem with the information required to refine the position of the laser by supplying it with the dimensions of the screen.

3.4.2 Circle Clicking Subsystem

The circle clicking subsystem will be responsible for taking information about last predefined number of positions of the laser and determining whether a circle has been traced, and if so, return the coordinates of the center of the circle. These coordinates is used in conjunction with the Windows API to perform a click event that would click on the area of the screen where the laser traced a circle.

This subsystem will also detect whether the circle was traced clockwise or anti clockwise. This information will be used along with the coordinates to perform either a left or a right click depending on whether the circle was traced anti-clockwise or clockwise respectively.

As shown in Figure 3.1, there is a laser position buffer in between the laser detection and the circle clicking subsystem. This buffer will contain the predefined number of

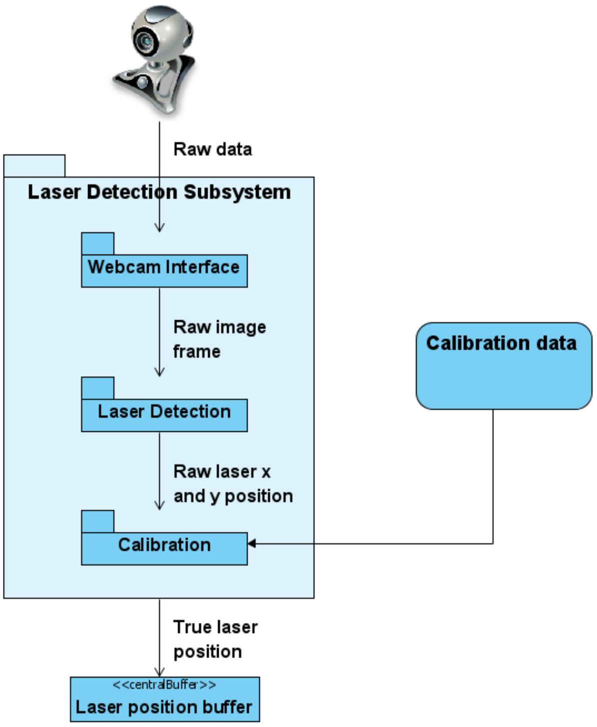


Figure 3.2: Laser detection subsystem with its three layers

previous laser positions that will be accessed by the circle clicking subsystem.

3.4.3 Laser Detection Subsystem

The laser detection subsystem will be a large subsystem consisting of three separate subsystems as shown in Figure 3.2. The inputs to this subsystem are the pictures taken by the webcam and the calibration data.

The laser detection starts with the webcam interface, which is a system that has been developed by ¡reference to john leis system and other references to this system¿. This system is only concerned with obtaining a frame from the webcam so that its contents can be analyzed and scanned for the presence and location of the laser.

The frame is then passed onto the laser detection layer which does all of the work to do with finding the location of the laser in the frame.

Once the location has been found, the coordinates are passed onto the calibration layer which uses the calibration data to produce the correct laser position which then ends up in the laser position buffer, which is used by the circle clicking subsystem for analysis and detection of a circular motion.

These three layers, the webcam interface (JL) layer, the laser detection layer, and the calibration layer makes up the laser detection subsystem.

3.4.4 System components meeting the specifications

The various subsystems in this laser pointer mouse system, once completed, would meet the specific objectives outlined in Chapter 1. Already the laser detection subsystem meets the objective for investigating the color point detection, and future work on the calibration and circle tracking subsystems would meet the rest of the specific objectives outlined in Chapter 1.

3.5 The work breakdown structure

The work breakdown structure begins with the major components given in the previous section which is the top level expectation of how this problem will be solved.

In this section the expected work that will need to be performed to fulfill the outcomes of each component will be briefly explained and will lead to the development of a more detailed WBS that will be the first stage of planning the time lines for the project. The explanation of each component is given below starting with its expected outputs and inputs and then what will be done to achieve the outputs.

3.6 Top Level WBS explained in more detail

3.6.1 Setting up and preparation

During this component which will be the first phase of the practical and technical work for this project, all of the necessary tools and facilities that will be needed to complete this project needs to be obtained, both hardware and software. The obvious hardware and software requirements are listed below:

Hardware Requirements

- Projector and Screen
- Laser Pointer
- Webcam
- Computer or laptop

Software Requirements

- Generic applications programming interface (API) for interfacing with the webcam.
- API for the processing of the graphics
- Programming language and/or integrated development environment (IDE) for writing the software

So there are two types of facilities that needs to be obtained, hardware and software. The selection of each item will depend on the requirements of this project. The aim will be to select items that are the most common and the most likely to be used by most people, because this project aims at developing a system that does not require special facilities that aren't readily available.

The aim with selecting software and software development tools and environments, is to select items that provide the most functionality so that as little as possible new code needs to be developed to solve this problem. New code is of course inevitable, but it would be a great advantage to lessen the effort involved in tasks such as graphics processing by finding tools that allows for common algorithms such as edge detection, or applying a particular type of filter. The main programming language that will be used will depend on whatever language the graphics processing API will end up being.

The other aim in selecting the software tools for this project is to select tools that are freely accessible and not commercial. But if this is not possible, commercial tools will be used. The reason for this is, free software especially open source software is easier to get hold of and not restricted by expensive licenses which will make this solution easy to distribute and share.

The other problem in the selection of hardware will be in the selection of the webcam. As described in (Popovich n.d.) the type of webcam and its capabilities will affect the performance of the laser pointer mouse. So if an expensive more functional webcam was chosen for this project, the resulting software of the solution will depend and make use of the advanced features of the camera, which means that other more common and cheaper cameras may render the solution useless. Therefore it is important that a cheap and popular camera needs to be used for this project.

Work Breakdown Structure for Setting up and Preparation

As a result of the above analysis a work breakdown structure for this section has been derived and is presented below, along with explanations where needed.

Setting up and preparation

1. Find and Test Graphics Processing API
2. Find and Test Web Cam API
3. Find and Test System Mouse API

4. Preparing Equipment

    (a) Determining and Setting Up IDE

        The determination of the IDE will depend on which languages and development environments are being used for the API's.

    (b) Preparing a Laser Pointer

        This involves obtaining a laser pointer and getting a plenty full supply of batteries.

    (c) Choosing and Purchasing a Webcam

        This involves finding out which webcams are the most popular and also finding out the availability of API's for it. These factors will determine which camera will be chosen

    (d) Arranging Access to Equipment

        This involves making arrangements with known owners of presentation facilities such as the projector and the screen.

    (e) Experimenting with Laser Photos

        This involves using the obtained webcam to take photos of the laser pointer and determining the best settings that will maximizes the detection of the laser point on the screen or even the wall at this stage of the project.

3.6.2 Interfacing the camera

Interfacing the camera should be a pretty easy and quick component of the project. There are mainly two operations that needs to be performed with the camera, only one of those operations being essential. The first is to write software that would grab a frame or a picture from the camera instantly. This is very important.

The second operation would help with making it easier to detect the laser by adjusting the exposure and sensitivity of the camera. If generic software can be written that can set these settings on the webcam it would be an advantage but not essential.

The code that needs to be written to obtain a picture from the camera needs to be generic and independent in the sense that the same code can be used to interface all possible webcams.

Work Breakdown Structure for Interfacing the Camera

As a result of the above analysis a work breakdown structure for this section has been derived and is presented below, along with explanations where needed. By now the experimentation and familiarization of the camera interfacing API would have been done and the implementation and design of the system can take place immediately.

Interfacing the Camera

1. System Design
2. Picture Request
3. Camera Settings
4. Finalizing and Testing Code

3.6.3 Finding the position of the laser point

This part of the project will be challenging and will require a lot of uncertainty therefore the amount of time to be spent on it must not be underestimated. It is very likely to involve being able to read in and keep in memory the value for each pixel of each line of the photo taken and performing mathematical operations on it, perhaps several times in order to capture the location of the unique characteristics of the laser dot on the screen.

This operation will have to be done in an efficient manner so that it does not take up all of the computer's central processing unit (CPU) computing power in order for the user to be able to use the computer for running the presentation.

One of the first things that have to be done in this phase of the project is to draw from the methods and conclusions from previous projects such as (Toshiharu Wada 2007, Popovich n.d., Olsen & Nielsen 2001, Cheng & Pulo 2003, Carsten Kirstein 2002, de Bruijn 2008, Atul Chowdhary n.d.). If a method of detection works well enough it would be good enough for this project and there would be no need to investigate any further.

If none of the methods described by (Toshiharu Wada 2007, Popovich n.d., Olsen & Nielsen 2001, Cheng & Pulo 2003, Carsten Kirstein 2002, de Bruijn 2008, Atul Chowdhary n.d.)works well enough then a new laser detection method will have to be developed which will result in some experimentation until the laser point can be detected efficiently enough.

This component of the project is the most important in that if the laser can't be detected then controlling the mouse using the laser would not be possible. So being able to detect the laser would be a great outcome of the project.

Work Breakdown Structure for Finding the Position of the Laser Point

As a result of the above analysis, a work breakdown a work breakdown structure for this section has been derived and is presented below, along with explanations where needed. This work breakdown structure is a little fuzzy at the moment, because it depends on whether the laser pointer detection technique from other works actually work using the chosen graphics processing API. If it doesn't work, then an original algorithm for detecting the laser pointer will have to be developed which will add to the amount of time that will be spend on this section.

Finding the Position of the Laser Point

1. Choosing a Source of Information

    Choosing a source of information requires having a look at all of the previous work done and determining whether the techniques being described will be useful in this project or not. If a source is found that works with the chosen graphics processing API, then those techniques will be used.

2. Source Reverse Engineering and Testing

   This involves studying some more material on the concepts and techniques covered in order to gain a better understanding of the concepts, and then using this knowledge to design an algorithm that can perform what the author achieved.

3. Improvements

   Making improvements on the code involves enhancing the code to make it more reliable and stable, and perhaps more responsive if it is required.

4. Developing Original Algorithm and System

5. Testing and Improving

   The above two items will only be performed if the technique from the author of whatever project does not work. This means that some more research will be required on graphics processing in order to derive an algorithm that can perform successful laser detection

6. Performance Analysis

   This piece of work will require the development of some code that will perform successive laser point detections and measure the time taken between successive detections and then perform some mathematical operations in order to measure the overall performance of the laser detection system

7. Finalizing Code

   Finalizing the code involves encapsulating the code into an object oriented object such as a class or a subsystem so that the laser detection system can be independent and be added to other projects if required.

3.6.4 Calibrating the position of the laser with the screen

Calibration is the process of removing the surrounding image data from the image of the screen in order to accurately calculate the position of the laser, and hence the next position of the mouse. This area is an area of uncertainty that will also just like with the detection of the laser require some research and experimentation before achieving the desired result.

The first thing that would need to be done in this section is to find the edges of the screen in the picture. There are two ways that this can be done. The first is to use a manual calibration technique where the user tells the software where the edges of the screen are, or second, the software automatically finds the edges of the screen using an edge detection technique which will allow for the feature of auto-calibration to come into effect. More on auto-calibration will be discussed later.

Obviously the first thing to know is where the edges of the screen is in the image. If the edges were to be found through a manual calibration process at the beginning of the session, then the camera needs to stay in the same position for the rest of the session or until it is moved again. The software would use the same data for the location of the edges of the screen for the whole of the session and if the camera was to be moved, the laser detection would be inaccurate. If auto-calibration was available, then the calibration could be performed each time the image is captured from the camera, making it appear as if the system doesn't require to be calibrated.

A basic calibration technique would only rely on the location of the left edge of the screen and the top edge of the screen. In this basic calibration scheme, the x value for location of the left edge of the screen would be subtracted from the x value of the location of the laser, and the y value (when dealing with computer screen coordinates, usually measured form the top of the image not the bottom) of the location of the top of the screen would be subtracted from the y value of the laser.

The resulting co-ordinates would then be converted to values that is consistent with the input requirements of the OS API function call that would reposition the mouse.

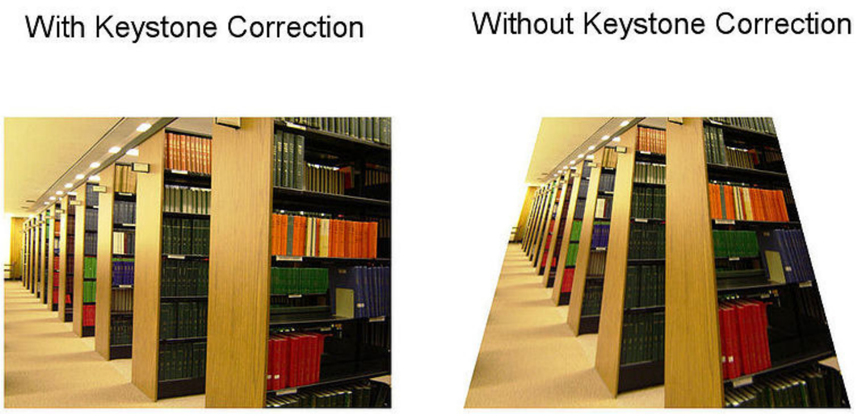


Figure 3.3: The effects of Keystone (Wikipedia 2010*d*)

The above would only work if the projected image did not suffer from the keystone effect (Wikipedia 2010*d*). This happens when a projector projects its image at an angle where the light rays projected by the projector is not perpendicular to the screen surface. The result is an image where the sides are not parallel to each other resulting in a distorted image (See Figure3.3).

In order to calibrate the coordinates of the laser, more advanced calibration techniques will have to be used such as what is described in (Popovich n.d., de Bruijn 2008).

Work Breakdown Structure for Calibration of laser pointer mouse

As a result of the above analysis, a work breakdown a work breakdown structure for this section has been derived and is presented below, along with explanations where needed. The work breakdown structure for this calibration subsystem is much the same as with the detection of the laser pointer mouse since the processes that needs to be performed is very much the same as they both involve using techniques already developed, and they both will require original solutions if the solutions by the authors of the existing works aren't suitable.

Calibrating

1. Choosing a Source of Information

Choosing a source of information requires having a look at all of the previous work done and determining whether the techniques being described will be useful in this project or not. If a source is found that works with the chosen graphics processing API, then those techniques will be used.

2. Source Reverse Engineering and Testing

   This involves studying some more material on the concepts and techniques covered in order to gain a better understanding of the concepts, and then using this knowledge to design an algorithm that can perform what the author achieved.

3. Improvements

   Making improvements on the code involves enhancing the code to make it more reliable and stable, and perhaps more responsive if it is required.

4. Developing Original Algorithm and System

5. Testing and Improving

   The above two items will only be performed if the technique from the author of whatever project does not work. This means that some more research will be required on graphics processing in order to derive an algorithm that can perform successful laser calibration.

6. Performance Analysis

   This piece of work will require the development of some code that will perform successive laser point detections and calibrations and measure the time taken between successive detections and calibrations and then perform some mathematical operations in order to measure the overall performance of the laser detection system

7. Finalizing Code

   Finalizing the code involves encapsulating the code into an object oriented object such as a class or a subsystem so that the laser calibration system can be independent and be added to other projects if required.

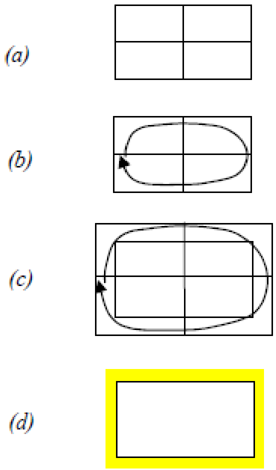


Figure 3.4: Possible circle detection approach (Wikipedia 2010*d*)

3.6.5 Circle clicking

Clicking on objects through circling them, is an aspect of this project that is intended to be original work, because no similar project has such a functionality. A project that incorporates a function which is close to this is the project by Cheng and Pulo (Cheng & Pulo 2003) has a concept of using a bounding box around an item.

Cheng and Pulo has implemented their circle tracing feature by predefining which items will be clicked on, and then defining box around each item which is divided into four smaller boxes with the central point being shared by the corner of each box. In this way, if the laser has passed through each box in a small period of time, it would be identified as the user having traced a circle around that object, which then initiates a click event around that it.

Because of this example by Cheng and Pulo, the concept of having a bounding box will be used but modified to meet the aims in this project. The modifications include

- Being able to click anywhere on the screen, therefore not restricted to predefined items
- Being able to determine clockwise and anti-clockwise movements

If this method doesn't work, then there would be even more uncertainty, and as a result, would require more research and experimentation, which will take time.

Because this is one of the major features of the project, achieving this would be the second most important priority, after being able to detect the laser pointer location, and therefore a longer length of time would need to be allowed for it.

Work Breakdown Structure for Circle Clicking

As a result of the above analysis, a work breakdown a work breakdown structure for this section has been derived and is presented below, along with explanations where needed. Just like the WBS for the calibration and the laser pointer detection, the work involved with performing the circular clicking will also follow the same steps. Except for this time there is not as much information available for circular clicking so more time will be spent on doing research and development of a method of detecting whether the laser has traced a circle.

Circle Clicking

1. Choosing a Source of Information
2. John Leis source examination
3. Source Reverse Engineering and Testing
4. Improvements
5. Developing Original Algorithm and System
6. Testing and Improving
7. Performance Analysis
8. Finalizing Code

3.6.6 Slide Navigation

In order to make overall system control and navigation easier for the user, some other interface control features will be implemented. One of these features will be the ability to navigate to the previous or next slide easily. This will be done by pointing the laser for a brief period on either side of the screen. On the left side of the screen to navigate to the previous slide and the right of the screen to navigate to the next slide.

The other feature is to be able to turn the laser pointer mouse system on or off. In order to turn the system on or off, the system will be programmed so that when the user keeps the laser point above the screen for a brief period of time, the system will activate or deactivate.

Implementation of these features should be easy to do because a lot of the code that will be developed for detecting the laser and also calibration will also be used to perform these tasks.

To detect the user's intention to move to the previous slide, the system will detect that the laser is out of range, and then it will detect whether it is on the left or the right of the screen and based on this it will navigate accordingly. In a similar fashion the system will detect the user's intention to turn the system of by detecting the presence of the laser above the screen and out of range.

Work Breakdown Structure for Slide Navigation

As a result of the above analysis, a work breakdown structure for this section has been derived and is presented below, along with explanations where needed. The slide navigation will depend on the subsystems of laser detection and calibration. These components are needed in order to be able to detect that the laser was outside of the screen's area and the user's intention to navigate the slides or turn the laser pointer mouse application on or off.

Slide Navigation

1. Develop Algorithm
2. Perform Detection of Laser
3. Determine where the laser is
4. Execute API Command for Slide Navigation

3.6.7 Automatic calibration

The main objective of the automatic calibration subsystem, is to be able to automatically find the four corners of the screen in the image captured by the camera. After the coordinates for these four corners have been obtained the coordinates gets passed onto the calibration subsystem. So the auto-calibration subsystem doesn't do much except for finding the location of the screen.

The hardest part of this automatic calibration system will be the graphics processing involved in oder to find the edges of the screen. (de Bruijn 2008) has some coverage of the idea, and mentions some algorithms that can be used to accomplish this.

Once again, if the techniques covered in (de Bruijn 2008)doesn't meet the objectives of this system, or if it fails to work properly some time will need to be allocated towards the uncertainties involved in finding a solution through research and experimentation.

Work Breakdown Structure for Automatic Calibration

As a result of the above analysis, a work breakdown structure for this section has been derived and is presented below, along with explanations where needed.

Slide Navigation

1. Research some edge Detection and Related Algorithms
2. Experiment with these on some Existing Infrastructure
3. Develop the Algorithm

4. Implement the Algorithm

5. Test the Algorithm

3.6.8 System integration into one standalone package

The aim in the development of this laser pointer mouse system is to design it in such a way that the code can be reused through the development of separate sub systems. In this way whichever subsystems or components haven't been completed yet won't matter because the sub systems that have been developed can still be integrated into the system. Of course essential sub systems such as the interface to the camera and the laser detection subsystem will have the be completed in order for this project to be a success.

The key to integrating the system is the development of the main driving subsystem and the interface, and perhaps also calls to the Windows API for controlling the mouse and other system events.

One other items that might be implemented if time permits is a setup package to make the process of installing the system easier for the user and also make it easier to distribute the software.

Work Breakdown Structure for System Integration

As a result of the above analysis, a work breakdown structure for this section has been derived and is presented below. Since this tasks is straightforward no more detailed work breakdown structure will be required.

System Integration

1. System Integration into a standalone package

3.6.9 Reporting

The task of reporting on everything that is done on this project is major and will consist of three subtasks, the actual dissertation production at the end of this project, the proofreading, collating and presentation tasks, and the weekly task of reporting on work done during the week.

The weekly task of reporting will be done in order to make it easier at the end of the project to put together the dissertation, and to make sure that everything is accurately reported. Most of this content will be on the actual implementation of the various phases of the project and the testing of the various different components.

The dissertation production towards the end of this project will involve checking over the what has been done on the document, as well as further completion of the document which will involve reporting on test results and the final chapters of the dissertation.

The last part of the dissertation production will be the final proofreading as well as the physical production of the document which is predicted to take around four to five days to do.

The Work Breakdown Structure of the Reporting work

As a result of the above analysis, a work breakdown structure for this section has been derived and is presented below. Since this tasks is straightforward no more detailed work breakdown structure will be required.

Dissertation Production

1. Weekly work on Dissertation
2. Dissertation Production
3. Collating and Publishing

3.6.10 The Complete Work Breakdown Structure

This project obviously consists of nine components or phases of work that needs to be completed. In the previous section, the work to be done in each phase of the project has been covered and the work break down structure for that phase has been presented and justified.

In this section the work breakdown structure for each phase of the project is combined to become the complete work breakdown structure for this project, which will be used to determine the time lines for this project. The work breakdown structure is presented below.

3.6.11 The Work Breakdown Structure for this Project

1. Implementation Start

2. Setting up and preparation

    1 Find and Test Graphics Processing API

    2 Find and Test Web Cam API

    3 Find and Test System Mouse API

    4 Preparing Equipment

      i. Determining and Setting Up IDE

      ii. Preparing a Laser Pointer

      iii. Choosing and Purchasing a Webcam

      iv. Arranging Access to Equipment

      v. Experimenting with Laser Photos

3. Interfacing the camera

    1 System Design

    2 Picture Request

    3 Camera Settings

    4 Finalizing and Testing Code

4. Finding the position of the laser point

   1 Choosing a Source of Information

   2 Source Reverse Engineering and Testing

   3 Improvements

   4 Developing Original Algorithm and System

   5 Testing and Improving

   6 Performance Analysis

   7 Finalizing Code

5. Calibrating

   1 Choosing a Source of Information

   2 Source Reverse Engineering and Testing

   3 Improvements

   4 Developing Original Algorithm and System

   5 Testing and Improving

   6 Performance Analysis

   7 Finalizing Code

6. Circle Clicking

   1 Choosing a Source of Information

   2 John Leis source examination

   3 Source Reverse Engineering and Testing

   4 Improvements

   5 Developing Original Algorithm and System

   6 Testing and Improving

   7 Performance Analysis

   8 Finalizing Code

7. Automatic Calibration

   1 Research some edge Detection and Related Algorithms

2 Experiment with these on some Existing Infrastructure

3 Develop the Algorithm

4 Implement the Algorithm

5 Test the Algorithm

8. Slide Navigation

    1 Develop Algorithm

    2 Perform Detection of Laser

    3 Determine where the laser is

    4 Execute API Command for Slide Navigation

9. System Integration into a standalone package

10. Dissertation Production

    1 Weekly work on Dissertation

    2 Dissertation Production

    3 Collating and Publishing

11. Project Performance

This work breakdown structure is used to determine the time lines for this project. The resulting time line is covered in the next section

3.7 Laser Pointer Mouse Project Time line

The time line for this project was done in the form of a Gantt chart (refer Appendix B). The implementation phase of this project will start at the setting up and preparation phase which will be on the 5th of July 2010 and the project is aimed at being completed by the 28th of October 2010.

This is a very tight timescale and it may not be likely to achieve all of the objectives of this project (refer Project Specification Appendix A). The aim is to at least be able to achieve the detection of the laser, as well as the manual calibration of it and the circular clicking. Because the completion of these items will result in a system that would be useful and a real contribution towards the laser pointer mouse concept.

The resulting Gantt chart is included in Appendix B.

3.8 Assessment of Consequential Effects

3.8.1 Introduction

The work involved in this project and the outcomes of it will result in consequential effects than may have an effect on present and future societies and environments. The three universal areas of consequential effects include sustainability, safety and ethical dimensions (Hancock 2010). These areas and how they affect the project will be explained in this section in more detail.

3.8.2 Sustainability

The effect that the development of the laser pointer mouse and its use will have on issues of sustainability should be minimal. The requirements of sustainability can be found in a document entitled, "Towards Sustainable Engineering Practice: Engineering Frameworks for Sustainability" which contains ten aspects of sustainability. The Engineers Australia Sustainability Charter also contains three major sustainability objectives that could be used to assess the effects of the laser pointer mouse on sustainability.

Most of the items in the Engineering Frameworks for Sustainability (InstitutionofEngineers1997) and the Engineers Australia Sustainability Charter does not really apply to the outcomes and consequences of this project. Items one to seven in (InstitutionofEngineers1997) (refer **??**) is concerned about the effects that the product and activities would have to the sustainability of the environment and how this affects current and future societies. This project does not really have any major effects on the environment, but because it utilizes technology and hence electrical power, there is of course the issues associated with pollution and wastes.

Pollution generated by this laser pointer mouse system (as well as from the work done on developing it in this project) is generated by the greenhouse gases emitted by coal fired power stations that most of Australia's energy still comes from (WWF-Australia2010,EIA2010). The level of pollution depends on how much power the electrical equipment such as the computer, the projector, and the web cam would consume, and how much power is needed depends on how the equipment is used. But because the computer will be turned on in any way for the presentation, whether using the laser pointer mouse or not, the effect that the laser pointer mouse would have on the environment in terms of using up fossil fuels and producing green house gases would be minimal.

The other form of pollution that would be generated by the laser pointer mouse system is the use of batteries. Laser pointers are power hungry and they run through batteries fast. The waste and pollution from this would be the amount of batteries that would get used.

Even if a laser pointer mouse wasn't available, the computer and the projector would still be used for the same amount of time. Having this fact in mind, it is clear that the introduction of some software that enables the use of a laser pointer mouse would not make a difference on the power consumption of the system. The software however may need more processing power which depending on the type of processor and computer system being used, may cause a little more power consumption than normal, but even then the result would be negligible.

For the rest of the points in (InstitutionofEngineers1997), the laser pointer mouse project does not apply. For point eight in (InstitutionofEngineers1997) (refer **??**) the use of the laser pointer mouse does not have any effect or application towards the reduction of poverty and does not affect certain groups of people more than others, so point eight does not apply and the laser pointer mouse project is not affected by it.

For point nine in (**?**), the use of the laser pointer mouse in developed countries, due to advances in technology, may result in cleaner energy use through the use of solar energy or nuclear power. In undeveloped countries the use of the laser pointer mouse just like any other electrical system may result in the use of unclean energy sources which would affect the environment. But then again, Australia is a developed country and it still uses coal fired power stations, so this particular point also doesn't fully effect the laser pointer mouse project.

Point ten of (InstitutionofEngineers1997) also does not effect this project, because the use of the laser pointer mouse will not affect the degree of international understanding and hence the likelihood of warfare occurring.

The other major Australian Engineering document on sustainability, the Engineers Australia Sustainability charter (**?**), has three main objectives. The first is to do with the well-being of society and viability of the planet, i.e. the environment. This is similar to the first seven objectives of (InstitutionofEngineers1997) which hardly applies to the laser pointer mouse except for its energy use which also also negligible.

The second objective in (**?**) is similar to objective eight of (InstitutionofEngineers1997) which is to do with equity among different groups of people, and this also doesn't apply to this project. The third objective in (**?**)is very abstract in that it encourages that development issues be solved holistically and pro-actively, which basically covers all of the sustainability issues and how to react to them.

The effect that the development and use of the laser pointer mouse would have on sustainability issues is mostly negligible because the only issue associated with the laser pointer mouse system, is its use of power which is also negligible. So sustainability is not a major issue in this particular engineering project and will not be discussed any further.

3.8.3 Safety

As mentioned before most of the safety aspects in the execution of this project and the usage of the laser pointer mouse is to do with the intensity of the laser beam and its effect on eye sight should one stare into the beam. A lot of these safety aspects have been covered in the risk assessment. **??**

3.8.4 Ethical

The ethical aspects of the consequential effects of this project can be assessed with the use of the standards set by the Institution of Engineers Australia known as the Code of Ethics (**?**). This code of ethics is common to all engineers as it has been emphasized upon during their training. The code of ethics contains nine rules more commonly referred to as the tenets of the code of ethics, and all engineers who are members of the Institution of Engineers of Australia are required to uphold these tenets are a requirement of continual membership and the ability to practice as a professional engineer.

The ethical aspects of this project will be assessed against the code of ethics. The ways in which the development of the laser pointer mouse and its effects in its use will adhere to the code is explained below:

The Application of the Code of Ethics to the Project

1. *Members shall place their responsibility for the welfare, health and safety of the community before their responsibility to sectional or private, interests, or to other members;*

   Obviously the main aspect of the laser pointer mouse that this applies to, would be the safety issues associated with using a laser pointer. The beam of light that is emitted from a laser pointer is intense enough to cause serious eye damage as mentioned in **??**. As well as the laser emitting intense light there is also the light emitted form the projector that is a safety issue. In order to address this problem the software will warnings incorporated in the user interface that will inform the user of the correct handling of the laser pointer.

2. *Members shall act with honour, integrity and dignity in order to merit the trust of the community and the profession;*

   The aspects in this project that requires the point above to be applied with is when dealing with customers should they ask any questions about the capabilities of the laser pointer mouse. This means that no lies need to be made up in order to sell the product, but selling the product is not within the scope of this project so it will not be an issue.

The other point here is when reporting the outcomes of this project to be careful not to make any false statements about the laser pointer mouse's capability and the work involved in developing it.

3. *Members shall act only in areas of their competence and in a careful and diligent manner;*

   This tenet is applied to areas that would require work to be done outside of the software and computer related disciplines. If what needs to be achieved can not be achieved through software and basic hardware interfaces, other ways of working around it will be sought such as getting a different engineer specializing in the particular area to provide assistance or to find another way of solving the problem that does not fall outside of the software engineering aspects of the project.

4. *Members shall act with honesty, good faith and equity and without discrimination towards all in the community;*

   The aim of this project is to ensure that everyone can use the laser pointer mouse, which should also benefit the disabled, so that they can use the laser pointer mouse on a part of their body that is mobile if they can't use their hands to control the mouse.

5. *Members shall apply their skill and knowledge in the interest of their employer or client for whom they shall act with integrity without compromising any other obligation to these Tenets;*

   The employer of this project is USQ and obviously substantial marks will be lost if the outcomes of this project was achieved in an unethical manner.

6. *Members shall, where relevant, take reasonable steps to inform themselves, their clients and employers, of the social, environmental, economic and other possible consequences which may arise from their actions;*

   Issues relating to this tenet has been covered in the previous two sections

7. *Members shall express opinions, make statements or give evidence with fairness and honesty and only on the basis of adequate knowledge;*

   This is unlikely to be required in this project unless a client for this product wants advice on something or there are questions form the audience during any conferences or presentations to be done regarding the laser pointer mouse.

8. *Members shall continue to develop relevant knowledge, skill and expertise throughout their careers and shall actively assist and encourage those with whom they are associated, to do likewise;*

   This project will enable its implementor to learn more about the practices of engineering project execution and technical material such as graphics processing, and applied mathematics towards solving the problem of finding the red dot of the laser pointer mouse in the graphic captured by the camera.

9. *Members shall not assist in or induce a breach of these Tenets and shall support those who seek to uphold them if called upon or in a position to do so.*

   Everything will be done in a professional manner that will not breach the code of ethics

3.8.5 Conclusion

Because this project is quite small and the product or system being developed is also fairly simple, and because most of what will be developed will be mostly computer program code, the effects that this will have on the environment will be minimal. Due to the use of a laser in this project, even though its only a small laser pointer, there are still safety issues involved with its beam, and because of its use in an environment that involves an audience, this safety aspect will be dealt with through informing the customers of the dangers of a laser pointer by including this information in the user interface in the form of warnings. Because of the simplicity of this project, the ethical issues associated with it will also be minimal.

3.9 Risk Assessment

3.9.1 Identifying the risks in this project

The risk identification involves two steps (Hancock 2010). The first step is the identification of the hazard, which is a source of physical harm (Hancock 2010). The second step is the identification of the associated risk of the harm occurring from the hazard (Hancock 2010).

There are two areas where this risk identification needs to take place. The first is during the execution of the project and the second is after the project has been completed.

Risks during the execution of this project

Identification of hazards

The potential hazards that will be encountered during the execution of the project is given below:

1. Workstation
2. Laser Pointer
3. Data projector
4. Screen/White board
5. Project Work

Identification of the risks associated with each hazard

The associated risks for each hazard is given below.

1. Workstation

1 Repetitive Strain Injuries

2 Back injuries due to poor posture and ergonomics

3 Electric Shock

2. Laser Pointer

1 Eye Sight Damage

3. Data projector

1 Eye Sight Damage upon staring into beam

2 Electric Shock

4. Screen/White board

1 Physical injuries to due maneuvering of mobile screens or white-boards

5. Project Work

1 Stress related health problems

2 Poor performance at work for employer which may lead to loss of job

Risks after the completion of this project

Identification of hazards

The potential hazards that will be encountered by the users after the completion of this project is given below:

1. Laser Pointer; Sight Damage
2. Data projector, Sight Damage
3. Screen/White board; Physical injuries

Identification of the risks associated with each hazard

The risks associated with each hazard is given below:

1. Laser Pointer

    1 Eye Sight Damage

2. Data projector

    1 Eye Sight Damage upon staring into beam

    2 Electric Shock

3. Screen/White board

    1 Physical injuries to due maneuvering of mobile screens or white-boards

3.9.2 Risk Quantification

Quantification of the risks involves a process of determining the amount of exposure and the scale of the consequences of the risk. The exposure and consequences is measured according to certain levels as outlined according to the Project Reference Book (Hancock 2010).

Exposure is the likelihood of being exposed to the hazard and is categorized at six levels:

Level	Descriptor	Situation
1	Very Rarely	Once per year or less
2	Rarely	A few times per year
3	Occasionally	Perhaps once or twice a month
4	Regularly	Perhaps Weekly
5	Frequently	Perhaps Daily
6	Continuously	All the time

Likelihood of Risk

Exposure is the likelihood of being exposed to the hazard and is categorized at six levels according to the table:

Level	Descriptor	Situation
1	Very Rarely	Once per year or less
2	Rarely	A few times per year
3	Occasionally	Perhaps once or twice a month
4	Regularly	Perhaps Weekly
5	Frequently	Perhaps Daily
6	Continuously	All the time

Consequences of Risk

Consequences are considered for both property and personnel (Hancock 2010), and falls into five levels of severity as shown in the table below which has been adapted from (of New South Wales 2010):

Level	Descriptor	Situation
1	Insignificant	Minor Equipment / Component Damage
2	Minor	Major Destruction of Equipment
3	Moderate	Minor Injury / Illness (Small Burn, bruise, cut or abrasion or headache)
4	Major	Major Injury / Illness
5	Catastrophic	Possible Death

Risk Analysis Matrix

The Risk Analysis Matrix assists with determining overall level of risk involved based on the level of likelihood and consequence combined. This matrix has been adapted from (of New South Wales 2010).

Likelihood	Consequences				
	1 Insignificant	2 Minor	3 Moderate	4 Major	5 Catastrophic
1 (Very Rarely)	Low	Low	Medium	High	High
2 (Rarely)	Low	Low	Medium	High	Extreme
3 (Occasionally)	Low	Medium	High	Extreme	Extreme
4 (Regularly)	Medium	High	High	Extreme	Extreme
5 (Frequently)	High	High	Extreme	Extreme	Extreme
6 (Continuously)	High	Extreme	Extreme	Extreme	Extreme

Risk Assessment

The risks that were outlined in the section above have all been evaluated and tabulated into the table below with the overall rating given based on the combination of likelihood and consequence.

Risk Type	Likelihood	Consequences	Overall rating
Repetitive Strain Injuries	1	3	Medium
Back injuries due to poor posture and ergonomics	1	3	Medium
Electric Shock	1	5	High
Eye Sight Damage	4	4	Extreme
Physical injuries (heavy screen)	1	4	High
Stress related health problems	3	3	High
Poor performance at work	2	3	Medium

3.9.3 Risk Control

Repetitive Strain Injuries

1. Take plenty of breaks when at the workstation

2. Follow good workplace ergonomic practices

Back Injuries

1. Ensure that a good posture is adopted when sitting and working at a workstation
2. Try and keep the back so that it is shaped like an S instead of a C to keep the backbone in its natural position

Electric Shock

1. Never open up a computer unless appropriately qualified
2. Never open any other electrical device
3. Care needs to be taken when eating and drinking at a workstation around a computer
4. Always turn the plug off at the wall before plugging in and unplugging
5. Never work near any electrical equipment plugged into the mains during a thunderstorm

Eye Sight Damage

1. Never look at the exposure or opening of any laser pointer or device emitting laser radiation
2. Never point a laser pointer at a person
3. Never point the laser pointer at the audience
4. Always have signs up to warn the audience or any other people involved about the risks of looking into a laser beam
5. It should be good practice to give the audience a friendly reminder of the dangers of looking into a laser beam

Physical Injuries

1. Never attempt to carry any heavy equipment on onces own

2. When lifting objects, bend the knees and keep the back straight

Stress Related health problems

1. Ensure a healthy lifestyle of decent food, exercise, and sleep
2. Never continuously work laser than 11 PM each night
3. Stay calm as much as possible and focus on the task
4. Break large ambiguous tasks into smaller more manageable tasks
5. Stay ahead of time, don't fall behind

Poor Performance at Work

1. Concentrate on the job when at work and don't think about the project
2. Don't take days off from work to do project work

3.10 Chapter Summary

In this chapter the WBS was explained and the resulting time lines were developed. The basic architecture of the system has also been explained.

In general the risks associated with this project is quite small and does not affect many people. There are some equipment however that poses to be a reasonable hazard. Such devices such as laser pointers, data projectors, as well as electrical equipment can cause serious harm if common sense is not taken. The biggest hazard in this project is the laser beam since it is quite easy to accidentally stare into the beam.

Chapter 4

Setting Up and Preparation

4.1 Chapter Overview

This chapter covers everything that was involved in the preparation and setting up for the implementation phase of the project. Obviously, the project can not be proceeded any further without the right webcam and laser pointer, so the first two sections discusses the decision making and selection of the hardware for this project.

After the discussion of the selection of the hardware for the project, the processes involved with selecting the right software to interface the camera is discussed. Then what was required to test the system was covered next, which covers mostly the advantages and disadvantages of different testing setups, such as the bedroom and the classroom environments.

And finally, the choice of integrated development environment (IDE) is discussed, which covers the selection of the compilation environment as well as the tools to be used to write and modify the software.

Figure 4.1: The Logitech Webcam C200 chosen for this project

4.2 Hardware Selection

4.2.1 Overview

In this section the reasoning behind the choice of hardware is discussed, especially in relation to its popularity to the public and cost. The two most important hardware items was the webcam and the laser pointer. Firstly the reasoning behind the webcam will be discussed followed by the choice of the laser pointer.

4.2.2 The Selection of the Webcam

The webcam that was chosen for this project was the Logitech Webcam C200 (see Figure 4.1), which represented the sort of typical type of webcam that would normally be purchased due to its availability and cost. This web cam's cost was $39.95 at Dick Smith Electronics (DSE) in October 2010 (refer Appendix C.1), and was also available for around the same price at other places such as K Mart and Officeworks for around about the same amount. Logitech is a popular brand for webcams as well as Microsoft, and as it is evident, due to it's availability and cost, this webcam would be found in many households and businesses.

The Logitech Webcam C200 has got a 640 X 480 VGA sensor capable of capturing up to 30 frames per second (see Appendix C.2) .It connects to the computer via a USB 2.0 interface, and when plugged into a computer with Microsoft Windows XP or above,

Figure 4.2: High performance Logitech Webcam C910

will work immediately without the need for any drivers.

The reasons for the choice of this particular webcam is because its developed by Logitech which is a major webcam brand, and its capabilities represents the capabilities of almost all webcams in this price bracket of below $100. There are many other webcams that can do the same as the Logitech C200, and therefore any other webcam especially those of Logitech and Microsoft would also have been good candidates for this project. Most of them have around the same frame rate of around 30 frames per second, and around the same resolution. Not much time has been spend on researching which camera would be the best due to the fact that the everyday webcam that is purchased from DSE or KMart is not going to be the best. Therefore, just randomly walking into the everyday department store such as DSE and picking out a budget camera would be good enough for its selection for this project.

The serious problem, as mentioned in Section 3.6.1, with selecting a high performance camera such as the Logitech HD Pro Webcam 910 (see Figure 4.2), would be that it would work a lot better than a cheaper more common camera such as the Logitech C200. This camera can provide full HD 1080p video capture (see Appendix C.3) at a resolution of up to 1920 X 1080 pixels. Due to its higher resolution and frame rates, detection of the laser would be much more accurate due to the presence of more pixels in the image, and performing motion detection for gestures such as circle tracing would be easier due to more frames being available in the same period of time. Due to the availability of higher performance from a high end camera such as the Logitech C910, the software that could have been developed in this project may become too reliant on this high performance, rendering the outcome of the project potentially useless when used with the more common budget camera such as the Logitech C200.

Figure 4.3: A class 3R green laser pointer used for aligning an astronomical telescope at the target star. These beams are clearly visible, unlike the red laser pointer

The selection of a common budget webcam such as the Logitech C200 is appropriate due its representation of the capabilities of most budget cameras. It is essential for a low end camera like this to be selected so that the detection of the laser can work successfully on all webcams, rather than just the more specialized higher performance cameras such as the Logitech C910.

4.2.3 The Selection of the Laser Pointer

The aim of the selection of a suitable laser pointer for its use in this project was to choose a device that, just like the webcam, represented the common laser pointer that would typically be used in computer driven presentations. Such laser pointers are mostly Class 2 red lasers which are limited to 1mW continuous wave (Wikipedia 2010*d*). More information on the ratings of lasers can be found on (Wikipedia 2010*d*).

There are laser pointers that are more powerful, which are rated as Class 3R or Class 3B lasers. In its common form, they are mostly green laser pointers rated from <5mW up to <30mW, and are commonly used in astronomy for pointing out the stars and as finders for telescopes due to the green laser beam being easily visible to the unaided human eye at night (see Figure 4.3). These green laser pointers are far too bright to be

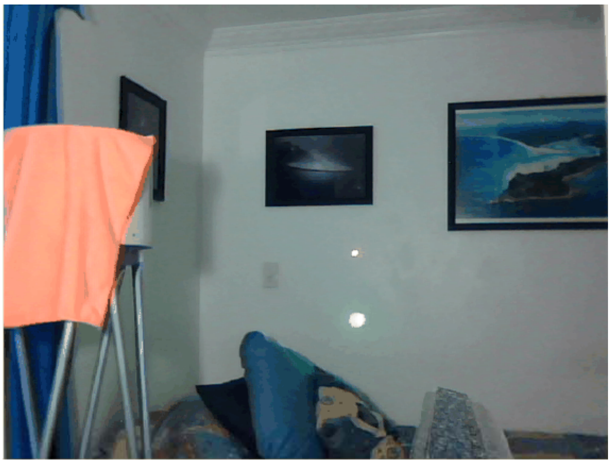


Figure 4.4: As can be seen in this figure, the green laser is significanlty brighter than the red laser!

used as a pointing device indoors for computer driven projected presentations, because the resulted laser dot that reflects of an ordinary wall is like staring into a green halogen lamp as can be seen from Figure 4.4.

The laser pointer that was chosen for this project is a typical red laser pointer rated at <1mW and falls under that Class 2 category. It was purchased from DSE for about $45, and is powered by a single AAA battery which is desirable because of the laser pointer's heavy power consumption and its heavy use in this project. But more commonly, these red lasers are powered by three button style batteries. The only difference is that the laser powered by a AAA battery lasts slightly longer, but does not affect the brightness of the laser.

4.2.4 Summary

The two main hardware items that needed to be obtained for this project was the webcam and the laser pointer. It was important to select hardware that would be used most commonly in day to day use by the public so that laser pointer mouse system can cater for these devices. The Logitech C200 was chosen to be an appropraite and easily available webcam for use in this project and an ordinary red laser pointer rated <1mW was chosen for this project.

4.3 Software Selection

4.3.1 Overview

In this section the webcam interfacing and graphics processing software that was found is discussed. Firstly the software requirements will be discussed in order to point out the aspects of the software that would be required to enable programming to start on the system. After this two alternatives will be explored, OpenCV and a system developed by Dr. John Leis, after which reason for the selection of the code developed by Dr. Leis will be discussed.

4.3.2 Software Requirements

There are two main software requirements for this project. The first is the need for software to interact with the webcam, especially with the ability to obtain a frame from the camera, and the second is the need for software that allows for the manipulation of graphics so that operations can be performed on the picture frame data structure in order to obtain the laser position data from it.

The other important factor with choosing software would be the degree of difficulty associated with using the software. Some APIs can be overly complicated even for the simplest of tasks such as obtaining a picture from the camera and loading it into a matrix of pixel data. The more time required for learning how to use a particular API, the less time there will be left for project work.

Another important factor is the cost of the software. Some API's can cost a lot of money, and the budget for this project is restricted, so if there are no real major benefits in having a package that costs money as compared with an open source package other than some special feature that would not be necessary for this project, then there would be no point in spending money for an API if there is one that can do roughly the same stuff but is free.

During the starting phases of this project, there were two alternatives that were the most attractive. The first and most attractive was OpenCV (Wikipedia 2010*h*) which presented complications, and the other option was to build on to a webcam interface program developed by Dr John Leis from the University of Southern Queensland (USQ) which uses DirectShow (Wikipedia 2010*b*), which is an API provided as part of the DirectX Software Development Kit for Microsoft Windows.

4.3.3 OpenCV

During the beginning phases of the setting up for this project, there was really only one attractive option for interfacing with the camera discovered from the works of (de Bruijn 2008) and (Kelvin Cheng 2006), and that is a package called OpenCV. OpenCV is a comprehensive computer vision library originally developed by Intel (Wikipedia 2010*h*), and has toolkits for facial recognition, gesture recognition, motion tracking, and even stereopsis (depth perception from two cameras). Not only would it have been possible to interface with the webcam, but it would also have made it easy due to the extensive graphics processing toolkits available.

Although OpenCV is packed with features and could be very useful, within the context and time frame of this project, getting OpenCV to work quick enough was problematic. There was a lack of simple enough documentation on how to get OpenCV to compile properly, and on how to use it. OpenCV apparently requires a system known as CMake (Kitware 2010, Wikipedia 2010*a*) which is in itself a very complicated system to set up when the user, as with the case of this project, is not experienced enough with make systems. The learning curve involved in setting up CMake and getting OpenCV to compile is large and would have required a lot of time. The OpenCV API, is also very complicated and is suited to advanced graphics processing operations. It would also have taken a while to learn how to use this API.

4.3.4 Existing Webcam Interface System Using DirectShow

The software developed by Dr. John Leis is able to successfully retrieve an image from a webcam using DirectShow, and has a function called `ProcessFrame` that made it easy to start straight away with the processing of the image from the camera. This function is found in the C source code file named `WebCam.c` and is listed in Appendix D.1, on line number 867. Having this made getting started easy as the webcam interface subsystem is essentially complete.

This code that was developed by Dr. Leis was adapted to C (refer Appendix D.1 line 7 of `WebCam.c`) from the original source code that is available from a popular on-line developers resource called `Code Project` (CodeProject 2010). The original code was written in C++ and needed to be adapted to C in order for it to be compiled by the LCC compiler, which is a small retargetable compiler capable of producing very fast and small executables (Wikipedia 2010*e*), which makes it ideal for compiling a laser pointer mouse application.

There are no real complications in compiling this code as it requires a traditional makefile and the make utility to compile the code.

4.3.5 The Chosen Webcam Interface API

Due to the advantages of having an already completed and well performing interface to the webcam and being able to use the `ProcessFrame` function to process the graphics from the camera, the webcam interface developed by Dr. Leis using DirectShow did the job for this project. There was no need to spend hours on learning how to use the code, and not too much trouble trying to get it to compile. And since it used simple makefiles and simple C code, it was a lot easier to read.

If there was more time available for this project, it would have been worthwhile to get OpenCV to work and to experiment with its interfaces. OpenCV looked to be a very comprehensive, tried and tested package that could have made it easier in the long run to develop the laser pointer mouse. But when considering what needed to be achieved for this project, it would just have been a bit of an overkill.

4.3.6 Summary

Only two main camera interfacing software packages were found, and although OpenCV looked promising, the amount of preparation and setting up involved to get it to work properly would have taken too long to do. In the end, the code developed by Dr. Leis from USQ was found to be good enough for the purposes of this project, and also a bonus considering that most of the webcam interfacing was already implemented and that an immediate start could be made on processing the frame from the camera.

4.4 Test Bench

4.4.1 Overview

This section talks about the requirements for testing the laser pointer mouse, such as whether a projector and a screen is initially needed or not to successfully develop the code the detects the laser. It covers two different environments namely the bedroom environment and the classroom environment.

4.4.2 Requirements for Testing

In order to be able to successfully test various laser pointer mouse prototypes for this project, it was essential to understand what sort of environment and equipment would be needed. Since nearly all of the work that was done on this project was done at home in a bedroom, it would not have been easily possible to test the prototype code in a lecture theater or a classroom, since these environments would not have been easily accessible.

Obviously a screen and a data projector would be used with the laser pointer mouse system, but for testing purposes all that was really needed was a white wall about the same distance away from the webcam as a screen in a classroom would be.

During the initial stages of the development of the laser detection system, the only

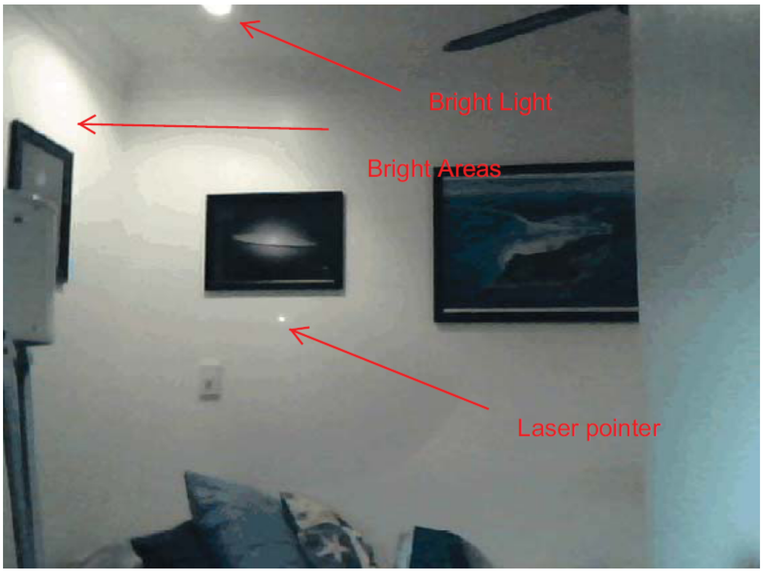


Figure 4.5: In this figure, the bright areas generated by the lighting in the room are visible, and can be compared with the laser dot

important thing was to be able to detect the reflection of the laser dot off the wall. The presence of a projected image from a projector of a screen would have very little affect on the appearance of the laser. As a result of this, a screen and a projected image was not needed until the system could successfully detect the laser.

The requirement of a proper classroom setting with a screen and a projector would only start to become important during the later stages of the project, when the laser detection layer is complete, and the next phase of the development would be the calibration of the laser which will rely on the edges of the projected image on the screen.

4.4.3 The Bedroom Lab

Since the initial development of the laser detection didn't require a screen and a projector, the prototype code was easily tested in a bedroom at the desk. The webcam was mounted on top of the computer monitor and pointed at the wall on the other side of the bedroom.

There were some problems with this setup, since there were other bright sources of light such as the light on the roof and its reflection off the paintings (refer Figure

4.5). This resulted in big bright spots of light at the same intensity as the laser, therefore complicating the laser detection algorithms to be developed. But this was good, because it introduced some problems that may as well be encountered in the real use of the system. During the day the curtains had to be closed and sealed from leaking light from outside in order to avoid its appearance in the webcam image as additional bright spots.

Due to the webcam adjusting its sensitivity to cope with the lack of light, even the smallest sources of light appeared as bright as the laser. This made the camera over sensitive which caused random white pixels popping up everywhere. This would not have been a problem if an image from a computer was projected on the wall, because the camera would have adjusted its sensitivity settings accordingly. This problem was solved by turning on the desk lamp and pointing it at the wall to emulate the light produced by the projector.

For most of the work that was done on the developing the laser detection layer, the setup in the bedroom was adequate enough.

4.4.4 School Classroom

A high school senior college computer classroom was available for further testing of the laser detection system. The classroom had a projector mounted on the roof and the screen went over the white board. The webcam was mounted on the computer monitor of the teacher computer at the front of the classroom and pointed at the screen. The distance between the webcam and the screen was just good enough to fill the webcam's field of view with the screen.

This classroom setup was useful to simulate the behavior of the system in a real practical setting with a real projected image on a real screen. This enabled for the testing of the system's ability to pick out the laser from bright white areas of the screen.

The other result from this classroom settings was to see the shape of the edges of the screen in order to get a better idea of how the calibration could be done.

4.4.5 Summary

It was found that for the most part of the development of this system, it was only required for the laser to be detected of a white wall and therefore the need for a proper projector and screen was not necessary. Later on in the project however, it was found that a projector and screen was needed for the development of the calibration and fine tuning of the laser detection algorithms.

4.5 The Integrated Development Environment

4.5.1 Overview

This section discusses the choice of IDE to be used for the development of the system. This was done by starting with the compiler of choice, then followed by the method used to compile it. After this, obtaining `make` and an LCC compiler for Windows was discussed then followed by the editor that was used for the development of the code. At the end of this section the reasoning behind the choice of IDE is discussed.

4.5.2 Compiler To Be Used

As mentioned in section 4.3.4, the compiler that was chosen to be used with this project was the LCC compiler. This compiler was developed by Chris Fraser and David Hanson and was used to compile the popular computer game, Quake III (Wikipedia 2010*e*).

Because of the compiler's ability to produce small executables that are really fast, it was chosen for this project due to its need for fast detection and calibration of the laser.

4.5.3 Compilation Method

Due to the large number of different source files and dependencies required for the successful compilation of the webcam interface, the traditional method of compilation through the use of a makefile was required.

Through the use of the makefile and the `make` (Wikipedia 2010*f*) utility, the simple `make` command issued from the Windows command prompt (`cmd.exe`) compiled the project. This makefile was supplied as part of John Leis' webcam interface code and did not need any modifications to be made to it. A copy of this makefile is included in Appendix D.4.

4.5.4 Make for Windows

Since Make is included with most versions of Linux, compilation of C source code using make would be quite easy. But this webcam interface system needed to work in Windows and Windows does not come with a make utility. So a version of Make needed to be sourced for Windows.

The version of make that was eventually used for this project to compile the code in Windows actually came as part of the LCC package and was found in the `/lcc/bin` directory. In order for make to work in the same way as it would in Linux by just issuing the `make` command, an entry in the Windows `Path` environment variable pointing to the path of make had to be entered. Anything in the Windows `Path` environment variable can be executed from anywhere by just entering the name of the program and not requiring the full path.

This modification to the Windows environment variable enabled `make` to work in the same way as it does in Linux and Unix.

4.5.5 LCC Compiler For Windows

The LCC compiler that was used for this project was a version of it known as the lcc-win32 compiler system, and can be downloaded from `http://www.cs.virginia.edu/~lcc-win32/` (Navia 2010). The `makefile` for this project was specifically implemented to use the LCC compiler.

The LCC compiler works like any other common C compiler in that it is called in a command prompt with the source file names as its arguments. The path to the LCC compiler, just like with `make`, had to be included in the `Path` environment variable of Windows for it to be called in a way that just requires its name and not the whole path to the executable.

4.5.6 Text Editor

One of the most important items in any software developer's toolkit is of course the advanced text editor. For something such as C programming, a simple Windows `notepad.exe` just would not suffice. The advanced text editor that was used for this project was Notepad++.

Notepad++ (HO 2010) is free and has been designed specifically for the Windows environment and can be downloaded from `http://notepad-plus-plus.org/`. It is extremely feature rich in that it supports syntax highlighting and syntax folding for 48 different languages (Wikipedia 2010*g*). It also has a tabbed document interface, meaning that different documents can be open at the same time in the form of tabs. It also supports auto-completion, drag and drop editing, split screens, spell checker, find and replace over multiple documents, as well as the inclusion of an FTP browser which is useful when developing websites.

Notepad++ can also allow the user to expand and collapse functions, and displays line numbers for easy debugging and does a great job of printing out source code as is evident for the included source code for this project found in Appendix D.1.

4.5.7 Reason For Choice

There are other much more sophisticated IDE's such as Microsoft's Visual Studio (Microsoft 2010*a*) and Code::Blocks (Code::Blocks 2010) just to name a popular few. Microsoft Visual Studio has an IDE for C++ called Visual C++, but is very sophisticated and would require a lot of time to learn how to use properly. Code::Blocks is a free dedicated C++ IDE which boasts compiling and debugging facilities, but would also require quite some time to learn how to use.

The other problem with the selection of specialized IDEs was to try and configure them to compile with LCC properly. Setting these IDEs up to be able to do this would have also taken a long time to do.

At the end of it all, it was decided that it would have been best to just stick with what would work, and that was the traditional old Makefile compiling and advanced text editor.

4.5.8 Summary

The selection of the IDE was heavily dependent on how much time was available. Sophisticated IDE would have taken a lot of time to set up to get them to work properly with very little to gain from it.

This system had to compile with LCC which also added to the complication, and since it was easy enough to just use the traditional methods using makefiles and advanced text editors, given the amount of time available, it was decided to choose a basic IDE using basic tools.

4.6 Chapter Summary

This chapter discussed everything that was involved with setup up for the implementation phase of this project. Firstly the hardware requirements was discussed and it was found that hardware that represented the sort of items that would be mostly used by the public would be the best, and as as result it was found that the Logitech C200 was one of the best candidates for the webcam. For the testing of the laser pointer mouse, it was found that the initial development work could be done in a bedroom with the webcam pointed at the wall.

The chapter also covered the requirements for the software to be used, initially the requirements for system that would enable communication with the webcam in order to retrieve image data from it. Then the IDE was discussed, and it was found that a straight forward, simple and traditional approach was going to be the most economical with time. This meant that the normal `makefile` system was to be used for compiling the project and that a simple text editor was good enough for coding.

Chapter 5

Interfacing the Webcam

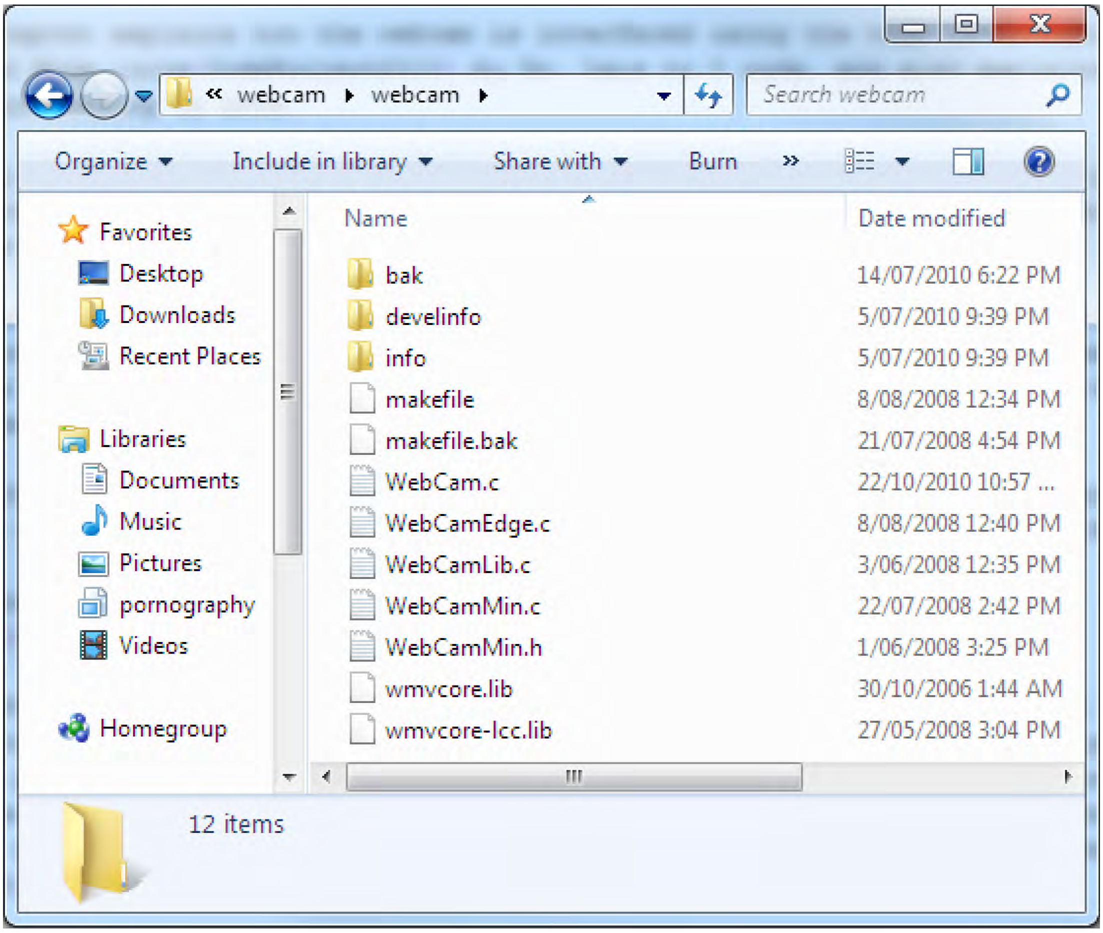
5.1 Chapter Overview

This chapter explains how the webcam is interfaced using the high level webcam interface adapted from (CodeProject 2010) by Dr. Leis to C code, and also explains how the image processing is done.

Firstly the code that does all of the work in this webcam interface system is introduced in section 5.2, then followed by an explanation of the user interface and its usefulness for this project in section 5.3.

The next section, section 5.4 explains the workings of the two main functions that sets up the application and handles the events of the application. The chapter then follows onto section 5.5 which explains how the webcam is interfaced tracing the code back to where DirectShow is used.

After this in section 5.7 the main engine of the image processing system is explored which explains how the image is processed using `for` loops.



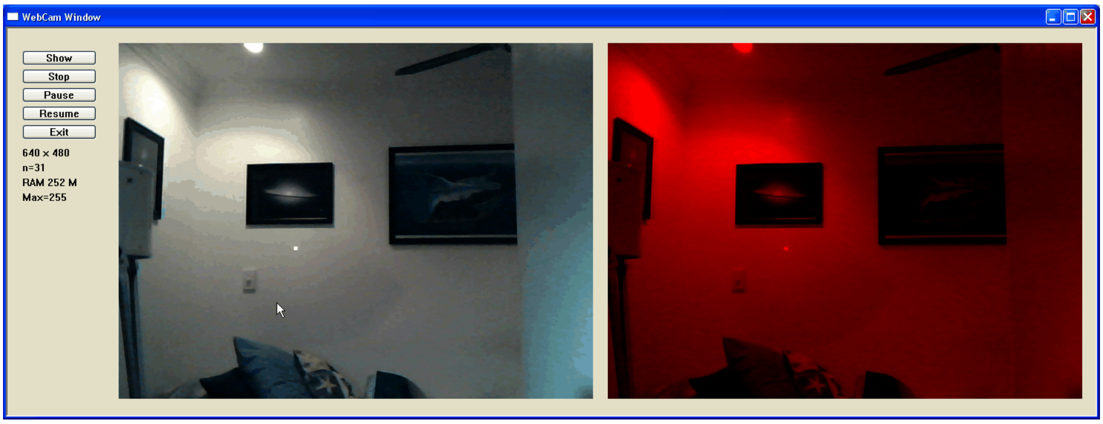
Name	Date modified
bak	14/07/2010 6:22 PM
develinfo	5/07/2010 9:39 PM
info	5/07/2010 9:39 PM
makefile	8/08/2008 12:34 PM
makefile.bak	21/07/2008 4:54 PM
WebCam.c	22/10/2010 10:57 ...
WebCamEdge.c	8/08/2008 12:40 PM
WebCamLib.c	3/06/2008 12:35 PM
WebCamMin.c	22/07/2008 2:42 PM
WebCamMin.h	1/06/2008 3:25 PM
wmvcore.lib	30/10/2006 1:44 AM
wmvcore-lcc.lib	27/05/2008 3:04 PM

Figure 5.1: Files of the webcam interface system

5.2 The Code That Does All The Work

The software that was developed or adapted by Dr. John Leis interfaces with the webcam using the DirectShow API. The files associated with this system comes in a package, and a screen print of these files are shown in figure 5.1. `WebCam.c` is the top level source for this software, because it deals with setting up the Windows GUI, as well as the associated event handling, and declaration of the main data structures such as `imageBuffer`, and `imageBuffer1`.

`WebCamLib.c` is the high level interface adapted by Dr. Leis from (CodeProject 2010), which presents high level control functions for the camera such as taking a picture, starting, stopping, and pausing. `WebCamMin.h` is the main h file that contains all of the definitions for the project. This file is substantial, please refer to Appendix D.3 for



```
640 x 480
n=31
RAM 252 M
Max=255
```

Figure 5.2: The user interface

more details.

The software uses a timer to set the frame rate of the webcam which is set at 400, which is currently manually adjusted according the the speed capabilities of the system.

For each frame that is captured from the camera a special function called `processFrame` is called (refer D.1, line 944). This function steps through each pixel of the frame and performs various operations on it. In its original non modified condition it performed a conversion to grayscale of the original color image.

One of the main aims of the project is to be able to process a frame from the camera and be able to detect the presence of the laser. It was in this function, `processFrame`, that the detection work was performed in. So it is reasonable to say, that the main work done by this system is in this function.

5.3 Overview of the WebCam interface Application

5.3.1 Overview

To the user, the software provides an interface that allows the user to control the capturing of the webcam (see Figure 5.2). On the left of the interface are five buttons that allows the user to choose these operations.

When the user presses the show button, the image that is captured from the camera is shown in the frame on the left. The resulting output from the processing done in the `processFrame` function is displayed in the frame on the right. The rate of change of the frames, or more commonly, the refresh rate at how often a newer image is shown from the camera is set in the code on lines 95 and 98 of `WebCam.c`, as shown below.

```
//-------------------------------------------------------------
#define ID_TIMER        350 //350

// frame rate in ms
#define FRAMERATE       400 //100
//-------------------------------------------------------------
```

Below the buttons, some information is displayed about what is happening. At the very top of this information, the current resolution of the capture from the camera is displayed. Below this is the number of frames that have been processed, and below this is the amount of memory that the application is taking up, and below this, the maximum value of intensity of the current frame that has been detected is displayed, where 0 is the minimum and 255 is the maximum.

5.3.2 The Usefulness Of The Application

The fact that this application has an interface that shows the outputs of the camera and the resulting processed frame makes assessing the outcome and effectiveness of the processing algorithm in `processFrame` very convenient.

This reason for this will be seen later where the image was processed into becoming a gray scale image, and only pixel values above a certain threshold was passed on to the output frame. The rest of the values was set to `0` resulting in a black pixel, and the values above the threshold kept the intensity value of the pixel. This resulted in an image that showed areas of light above the threshold.

5.3.3 Summary

This section introduced the graphical webcam interface and how it works. It also explained that this interface is very useful for testing an image processing algorithm by viewing its result in the output frame on the right of the interface.

5.4 How the Application Works

5.4.1 Overview

As mentioned earlier, the code in `WebCam.c` is what controls the application. It sets up the user interface, and data structures, and also handles the events from the user and the operating system.

The the two main functions in `WebCam.c` that controls the application is:

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
                   LPSTR lpszCmdLine, int nWinMode)
```

and

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam,
                         LPARAM lParam)
```

The main function, namely, `int WINAPI WinMain(...)` is the very start of the program, and sets everything up like the windows and the main data structures.

The event handling function, namely `LRESULT CALLBACK WndProc(...)` handles input from the user and the system (such as the timer events or interupts).

Of course the other main function in `WebCam.c` is the `processFrame` function which does the processing of the image.

5.4.2 The Main Function and what it does

As mentioned before, the main function sets the application up and drives it. It has a few sections.

Firstly everything is initialized and the webcam state variables, `WebCamRunning` and `WebCamPaused` is initialised to `0` as seen below (for the entire main function refer to Appendix D.1 lines 206-271, meaning that the webcam is not running yet and as a result is not paused either.

```
WebCamRunning = 0;
WebCamPaused = 0;
```

The next two lines of the main function sets up the window containing all of the camera outputs and the status of the images and the camera.

```
//---------------------------------------------------------
hMainWnd = CreateMainWindow(hInstance, hPrevInstance, lpszCmdLine, nWinMode);
if( ! hMainWnd )
    return 0;
//---------------------------------------------------------

//-------------------------------------------------------------
hCapWnd = CreateCaptureWindow(hMainWnd);
//-------------------------------------------------------------

//-------------------------------------------------------------
hCanvasWnd = CreateCanvasWindow(hMainWnd);
//-------------------------------------------------------------
```

The Image Buffers

The picture that is captured from the camera is stored in a buffer of bytes called the `ImageBuffer`. This `ImageBuffer` is set up in the following lines of the `main` function.

```
    ImageBufferLen = MAXIMAGE_WIDTH * MAXIMAGE_HEIGHT * 3L;
```

`malloc` is used to allocate memory to the image buffer.

The next buffer of importance is `ImageBuffer1`, which contains the result of the processed image after being processed using `processFrame`. This buffer eventually goes into the frame on the right of the output screen

The two image buffers are arguments of the `processFrame` function as shown below (for more details refer to Appendix D.1).

```
    int     ProcessFrame(unsigned char *pFrameIn, unsigned char *pFrameOut,
                         int Width, int Height)
```

Enabling Event Handling

The next few lines in the `main` function gets a message from the operating system containing information about user interfaction and other events.

```
    while( GetMessage(&msg, NULL, 0, 0) )

        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    //----------------------------------------------------------
```

The message is translated and is later caught by the callback function, which is the second major function of `WebCam.c`

5.4.3 The Callback Function

As shown below, the callback function accepts a number of arguments, but the only argument of concern in this function is `UINT message`, which contains the information about events.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
```

The events are handlelled by a `switch` (refer Appendix D.1) statement with a number of cases. The cases or events of interest for this application are `WM_TIMER`, `ID_BSHOWBUTTON`, and `IDB_STOPBUTTON`.

The `WM_TIMER` Event Handler

The `WM_TIMER` event is an event triggered by a timer created earlier on. This timer has a certian interval which captures images from the camera at that interval. This interval is dependent on the frame rate of the webcam as well as the speed capabilities of the system.

The `processFrame` is called from this event handler, basically asking for the system to start the detection of the laser on the captured image.

Before the `processFrame` function is called, the `GrabWebCamFrame` function belonging to `WebCamLib.c` is called to capture a frame from the camera.

5.4.4 Summary

This section explained how `WebCam.c` works and how the two main functions, `int WINAPI WinMain(...)` and `LRESULT CALLBACK WndProc(...)` works within this file.

This section also briefly introduced the the other main function `processFrame` which processes the image captured from the webcam.

5.5 Interface to the Webcam

5.5.1 Overview

This section explains how the image is captured from the camera by tracing the code back to where DirectShow API is used. It uses a top down apprach by starting at the high level code and tracing into the low level code.

5.5.2 Investigation Approach

The approach into investigating how the system interfaces with the webcam will be done from a top down approach starting from the point at which the image is captured from the `processFrame` function and then going all the way down to where the DirectShow interface is.

It is beyond the scope of this dissertation to go any further than the point at which DirectShow is used, which occurs in `WebCamMin.h`. Refer to Appendix D.3 for more details.

5.5.3 The ImageBuffer Object

The image buffers (`ImageBuffer` and `ImageBuffer1`) are major objects in the webcam interface system. The first image buffer, `ImageBuffer` is where the capture from the webcam end up and gets passed into the `processFrame` function.

`ImageBuffer1` is also passed as an argument into the `processFrame` function, but in the case of the current project doesn't actually get used beyond the `processFrame` function, but the data in `ImageBuffer1` is passed on to the output in the `processFrame` function which eventually ends up in the right pannel of the display.

5.5.4 The Starting Point

To capture a frame from the webcam, a call is made to

```
GrabWebCamFrame(ImageBuffer, ImageBufferLen, &WebcamImageWidth,
&WebcamImageHeight)
```

on line 421 in `WebCam.c` D.1. `GrabWebCamFrame` puts the captured image into `ImageBuffer`, and the rest of the arguments are used for checking that the image that was captured from the camera are the right dimensions.

5.5.5 GrabWebCamFrame function

The `GrabWebCamFrame` is found in the main upper level interface to the webcam adapted from Dr. Leis, `WebCamLib.c` (Appendix D.2). This function is quite simple. It gets the image from the DirectShow interface and manipulates it into a form that is usefull for this application, and then assigns it to `*pFrameOut` which goes into `ImageBuffer`

The main operations of interest in this function are shown below and can be referred to on lines 612 to 620 of `WebCamLib.c` (Appendix D.2).

```
if( ! m_pWC )
    {
        return 0;
    }

    if( m_pWC->lpVtbl->GetCurrentImage(m_pWC, &lpCurrImage) != S_OK )
    {
        return 0;
    }
```

`m_pWC` Is an object defined in `WebCamMin.h` and has a member function called `GetCurrentImage` which gets the image from the camera. `m_pWC` Is declared at the beginning of `WebCamLib.c` as a `IVMRWindowlessControl`.

The definition of `IVMRWindowlessControl` is found in `WebCamMin.h` on lines 906 to 995. Please refer to the details of this definition in the source in Appendix D.3.

To understand why the deffinition of `IVMRWindowlessControl` exists is beyond the scope of this project and therefore would not be investigated any further.

5.5.6 High Level Webcam Interface Subsystem

The high level interface to the webcam is `WebCamLib.c` and provides various operations on the webcam. The main operations are shown below

```
int     InitWebCamCapture(HWND hWnd, int iDeviceID,
                          int *pWidth, int *pHeight);
int     GrabWebCamFrame(unsigned char *pFrameOut,
                        unsigned long FrameBufferLen,
                        int *pWidth, int *pHeight);
int     StopWebCamCapture(void);
int     PauseWebCamCapture(void);
int     ResumeWebCamCapture(void);
int     CloseWebCamCapture(void);
```

The above functions are quite self explanatory.

5.5.7 Summary

This section explored how an image is captured from the camera using a top down approach by tracing the code back to where reference is made to the DirectShow API.

5.6 The ImageBuffer

As mentioned before, there are two image buffers in this software, `ImageBuffer`, and `ImageBuffer1`. The image buffer just like all data structures is basically a long string of bytes.

Its natural to think that the image would be represented as a matrix of values with dimensions consistent to the width and height of the image. After all, this is how it is done in MATLAB (a numberical computing package used for complex mathemicatical calculations).

Every three bytes of the image buffer represents a pixel, each byte in the set of bytes represents an intensity value of one of the three primary colours; blue, green, and red, in that order.

The way that `WebCam.c` extracts the colours from the image buffer is evident in the following lines of code. On line 960 of `WebCam.c` (Appendix D.1),

```
pByteIn = pFrameIn;
pByteOut = pFrameOut;
```

`pFrameIn` Is the first image buffer argument of `processFrame` function and gets assigned to `pByteIn`. The same happens for the `pFrameOut` buffer.

The extraction of the individual colour values occurs on lines 978 to 980 of `WebCam.c` and an extract is given below:

```
// retrieve image bytes
BlueByte  = *(pByteIn+0);
GreenByte = *(pByteIn+1);
RedByte   = *(pByteIn+2);
```

5.7 The ProcessFrame Function

5.7.1 Overview

As mentioned earlier, the `processFrame` function is the main function of interst because it is in this function that the actual image processing is done for the laser detection system.

The two main parts of interest in the `ProcessFrame` function are the `for` loops and the acutal processing of the image inside the second nested `for` loop. In this section each of these two parts will be explained in detail.

5.7.2 The For Loops

The main engine of the `ProcessFrame` function are the two nested `for` loops that cycles through each pixel of the image in the image buffer (`pByteIn`) (refer Appendix D.1).

Firstly the image buffer is loaded into the local image buffer data structure, `pByteIn` as explained in section 5.6. Then each pixel is processed through the two `for` loops that follows, as shown below;

```
//------------------------------------------------------------
for(y = 0; y < Height; y++)
{
    // Reset the x min
    curXmin = NOTSET;

    //------------------------------------------------------------
    for(x = 0; x < Width; x++)
    {
        //...........
        //...........
```

The image is scanned from left to right, row by row. The `y` variable in the first `for` loop deals with the rows of the image, and the `x` variable in the second `for` loop deals with each pixel (column) in a row.

But since the image buffer is a string of bytes, the `for` loops are necessary to treat the image as a matrix. The `Height` variable of the first `for` loop tells the `for` loop how many rows there are in the image, and the `Width` variable in the second `for` loop tells it how many pixels there are in each row.

So when the end of the second `for` loop is reached, the end of a row is reached, then control goes back to the first `for` loop which increments the `y` variable, telling the system to progress to the second row of the image.

In this way, a linear buffer of image data can be treated like an image matrix.

5.7.3 The Image Processing

Within the second `for` loop, the code that does the processing on the image is located. So in the first version of `WebCam.c`, the image is turned into a grayscale image by calculating the average of the three primary colours and assigning it to `PixelValue`;

```
// calculate new pixel value
PixelValue = (int)RedByte + (int)BlueByte + (int)GreenByte;
PixelValue /= 3;
//-------------------------------------------------------------
```

and then the value of pixel value is assigned to each of the primary colour bytes and then assigned to the output by assigning the pixel values to `pByteOut` as shown below;

```
// restore pixel value
BlueByte  = (BYTE)PixelValue;
```

```
GreenByte = (BYTE)PixelValue;
RedByte   = (BYTE)PixelValue;
//-------------------------------------------------------------

//-------------------------------------------------------------
*(pByteOut+0) = BlueByte;
*(pByteOut+1) = GreenByte;
*(pByteOut+2) = RedByte;
//-------------------------------------------------------------
```

5.7.4 Summary

In this section the two main parts of the `ProcessFrame` function was explained in detail. The reason why the other parts of this function was not explained is because its beyond the scope of this dissertation.

5.8 Chapter Summary

This chapter explained how the webcam is interfaced using the high level webcam interface adapted from (CodeProject 2010) by Dr. Leis to C code, and also explained how the image processing is done.

Chapter 6

Developing Laser Detection

6.1 Chapter Overview

This chapter discusses the steps taken to develop the initial system that can detect the position of the laser. Firstly, the work breakdown structure is revisited as a starting point for the development of the laser detection so that other projects can be examined for ideas on how to do this.

Firstly the concept of colour filtering was examined and found to be inadequate, then the high pass filter method was investigated and found to be good enough for the purposes of this project.

After this a basic system was designed to allow for the location of the laser to be output to the screen and some testing was performed on it where screen shots are shown. After the testing of the laser detection, the need for testing the performance of the system became apparent and as a result the following section discusses the development of a data collection system, so that testing of the motion of the laser and the speed of the system could be performed.

6.2 The Work Breakdown Structure

A good place to start with developing the laser detection for this project was to begin with the proposed work breakdown structure covered in section 3.6.3.

The first thing that was done was to acknowledge the approaches in other projects and to try and immplement them for this project. Two possible appraches were found and was attempted for this project, after which the next step was to make improvements to these approaches so that they can work successfully in this project.

6.3 Laser Detection in Other Projects

After having looked at the approaches by other works on the concept of the laser pointer mouse, it was found that the approaches towards the detection of the laser in (Atul Chowdhary n.d.), and (de Bruijn 2008) were the most suitable for this project since they seemed easy to implement and since the time that was available for this project was limited, it was better to choose approaches that would be simple to implement.

Chowdhary's approach (Atul Chowdhary n.d.) was to attach a physical red filter, like a piece of red cellophane, over the lens of the webcam, to amplify the red light and then apply a blue filter processing algorithm to only allow blue light which apparently showed the laser easily as the most intense pixels in the image. The problem with this approach is that it is a disadvantage against the requirements of this project to use additions such as physical red filters for the camera, but it was decided to test whether applying a colour filter to the image processing algorithm will make the laser stand out.

de Bruijn's approach (de Bruijn 2008) was to apply a hue, saturation and value algorithm to the image processing, and then look for the pixels with the brightest value. This approach seemed even simpler than Chowdhary's approach and as a result it was decided to test this approach as well.

The first method that was tried out for this project was the colour filter approach of

Chowdhary, but this method didn't work so well, so the approach that was found to have worked better was the approach by de Bruijn.

6.4 Colour Filtering

6.4.1 Overview

This section explores the approach by Chowdhary to filter through red light. Firstly the theory behind this method is explained, and then the changes that was made to the code is shown and explained.

After this, the program is compiled and run to view the results of the modification and then the problems that was found with this approach was explained in detail, which will lead to the next section.

6.4.2 Looking for Red

The idea behind applying a red filter was that since the laser is red, only the red's in the image would stand out, therefore it was assumed that the red dot of the laser was going to be one of the very few areas of the image remaining. If this was the case, then looking for pixels with only red would reveal the position of the laser.

So the aim was to implement a red filter into the existing `processFrame` function and test the output in the second frame of the application.

6.4.3 Implementing the Red Filter

Implementing the red filter using the existing code in the `processFrame` function was a simple task. To do this, only the value of the red byte was allowed to be assigned to the `PixelValue` variable, and the line of code below that line that calculated the average of all of the values for red, blue, and green was commented out.

```
// calculate new pixel value
PixelValue = (int)RedByte;//(int)RedByte + (int)BlueByte + (int)GreenByte;
// PixelValue /= 3;
```

So the exact intesity of the red pixels is present in the `PixelValue` variable. The next few lines of code that was modified to enable the red filter to work is shown below

```
//--------------------------------------------------------------
*(pByteOut+0) = 0; //BlueByte;
*(pByteOut+1) = 0; //GreenByte;
*(pByteOut+2) = RedByte;
//--------------------------------------------------------------
```

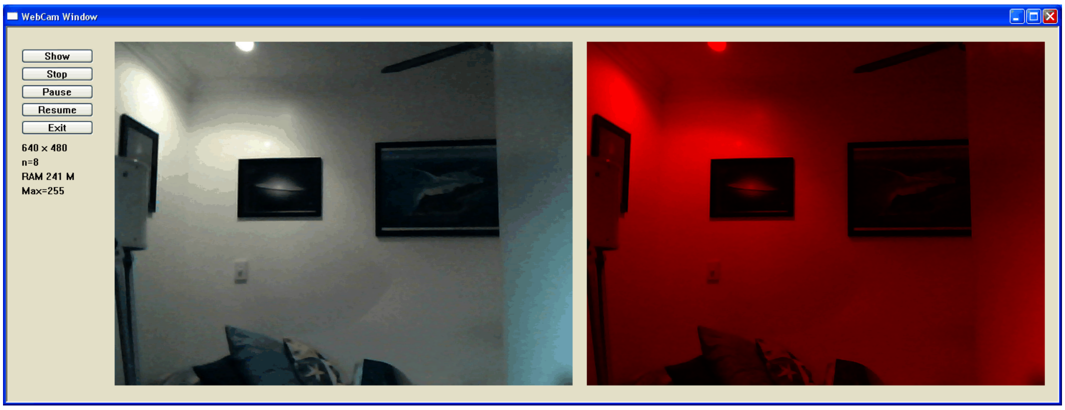
Where the `BlueByte` and `GreenByte` variables (which would have just been the average values) was assigned to `pByteOut`, `0` was assigned instead, effectively setting the blue and green bytes to become black. This modification therefore only allows the red to go through to the output.

6.4.4 Compiling Project

Compilation of the project was simple in that the command, `make` was given at the command line in the same folder as the project. This compiled the project and displayed any errors and warnings if it found any. At the end of the compilation, the file `WebCam.exe` was produced, which is a binary executable that the user can just double click on in order to start the program.

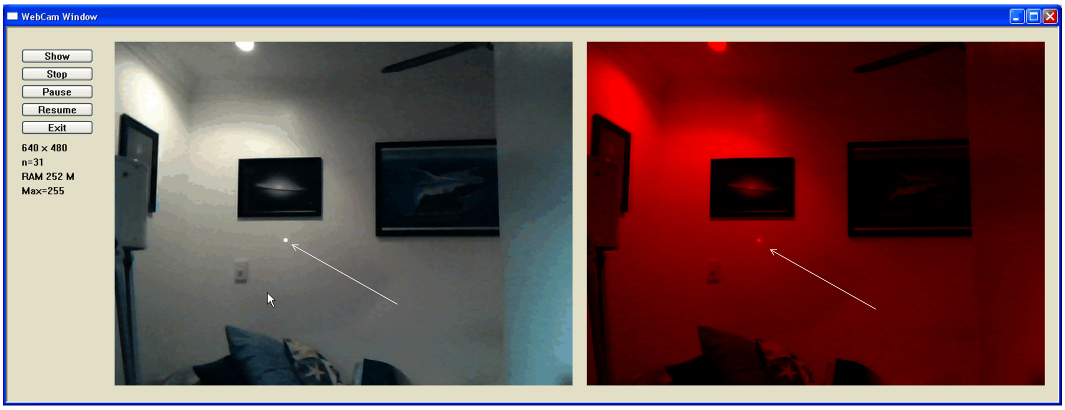
6.4.5 Results

This particular test, along with the rest of the tests in this project was performed in the author of this dissertation's bedroom where the webcam was attached to the LCD screen at the desk and pointed to the wall above the bed where all of the testing for this project was done.



```
640 x 480
n=8
RAM 241 M
Max=255
```

Figure 6.1: Red Filter Applied



```
640 x 480
n=31
RAM 252 M
Max=255
```

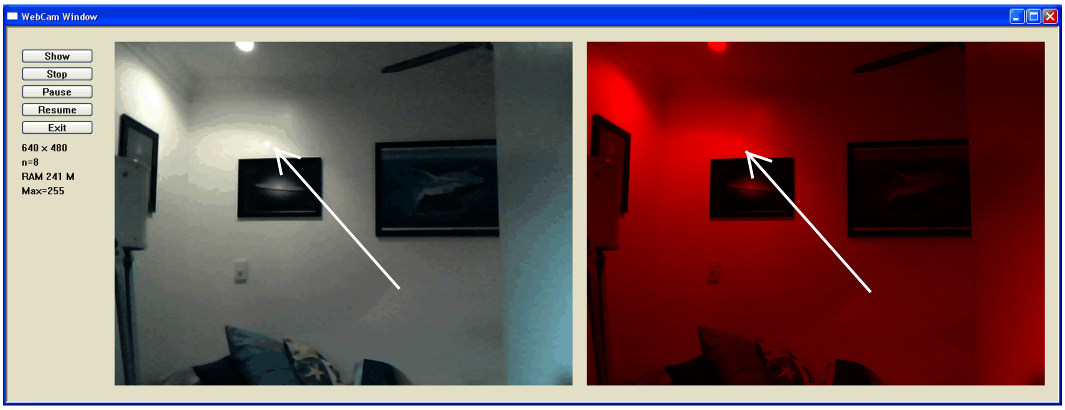
Figure 6.2: Presense of the Laser

The program `WebCam.exe` was started and displayed the output of the webcam allong with the red filtered result in the right frame as can be seen in figure 6.1. As it is evident in figure 6.1, the red is allowed through, but a lot of the laser surrounds have also gone through such that the entire wall of the bedroom was visible.

The next step that was performned was to point the laser at the wall to see what would happen, and the result of this is visible in figure 6.2. Note that the laser is an easily visible bright point and that the colour of it is white, not red!

6.4.6 Problems With the Red Filter Approach

The biggest problem with the red filter approach is that a lot of red is let through as is evident in figure 6.1. This is because normally white light which contains red, blue



640 x 480
n=8
RAM 241 M
Max=255

Figure 6.3: Laser not Visible

and green (RGB) will reflect all of its colours off an object. The only difference in the reflection is that due to the properties of the object, it would no longer reflect equal amounts of RGB but at varying levels. So this means that every object will reflect a red component and therefore explains why the red filter wont work.

The other problem that was noticed, is that the laser is not red, but white! This is because the webcam is over exposing the image to compensate for a lack of light. So any bright light in the image will be white, irrespective of the colour of it. The webcam came with software that enables the user to turn off all of the automatic features of the webcam and manually adjust settings such as gain, brightness and contrast and exposure time. It was found that if the gain was reduced and exposure time set to minimum, that the laser would actually appear to be its correct colour, and that is red. But the rest of the image was black.

One thing that was noticed is that the laser stands out in the image and always appears white. Therefore, the algorithm can be adjusted to look for the brightest pixels. But this leads to another problem as is evident in figure 6.3. The laser is actually turned on and pointed at the wall, but in the picture it is not visible. This is because the laser is actually pointed in an area where a lot of bright light is reflecting off the wall as pointed in the figure. Trying to detect the laser in these circumstances would be extremely hard because there is no difference in the pixels or the laser and surrounding the laser. The only thing that can be done to solve this problem is to develop code that is able to adjust the camera's parameters so that the image isn't over exposed.

6.4.7 Summary

It was found in this section that the red filter approach didn't work as it was expected due to the fact that everything will reflect red light. It was found however that the laser still stood out a fair bit and was white in colour and as a result it may be possible to filter through light above a certain level, which is referred to in this dissertation in the next section as the high pass filter.

6.5 The High Pass Filter Method

6.5.1 Overview

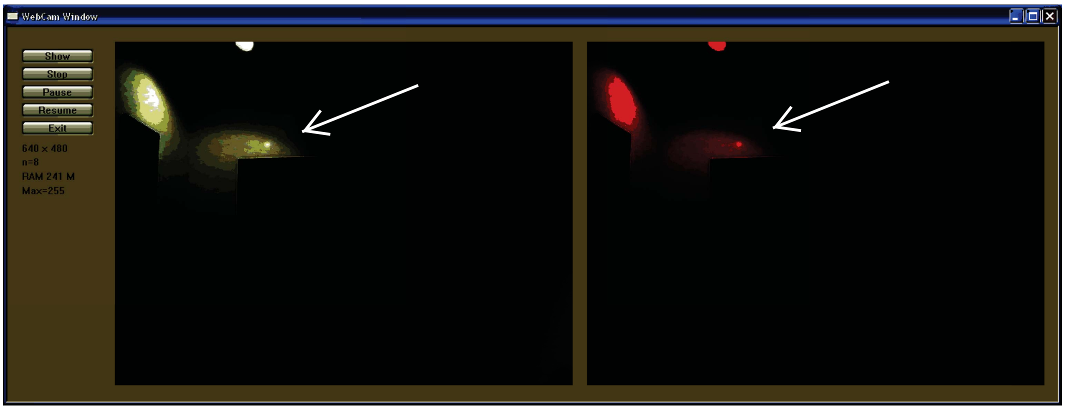
In this section the the concept of the high pass filter is explained a bit further and a test was done using a computer graphics application to test the theory of the high pass filter. After this it was found to work well, and steps were taken to implement this in the processFrame function. The results are shown and further problems are identified.

6.5.2 The High Pass Filter

As it was seen in the previous section, the laser appears as a bright white dot, not a red dot. Therefore, the next approach was to create a filter that would only allow pixels above a certain threshold through. In this dissertation, it is called a high pass filter, because it only passes through pixels that are really bright and above a certain high intensity threshold value.

In computer graphics the most common values of intensity ranges from 0 to 255, where 0 is black and 255 is white. So for this project, the value that was chosen was an educated guess of 250.

In order to test this theory, a screen shot was taken of the webcam program and imported into Adobe Photoshop, which is an advanced graphics editing application, and very well known in the web development world. It has a whole suite of different



```
640 x 480
n=8
RAM 241 M
Max=255
```

Figure 6.4: Levels in Adobe Photoshop

filters in it for applying all kinds of effects on the graphic.

In Photoshop there is an option to adjust the levels of the image, and the cutof was chosen at 250. The result of this image is shown in figure 6.4. The image that was used for this operation was the image in figure 6.3. It can be clearly seen that the laser that appeared invisible in figure 6.3 is now easily visible, with the level set at 250.

So with this knowledge, the initial algorithm to detect the laser should test for a pixel above a certain level and return the position of this pixel. This is the easiest way of doing it, and if the system was used in a proper Power Point presentation with no other bright lights, then the system would have a good chance of working.

6.5.3 Implementing The High Pass Filter

In order to make the high pass filter work, a test needs to be constructed to test whether the pixel is above the threshold of 250. But before this is done, it was decided to create this high pass filter so that it only lets through pixels above the threshold and set the rest to 0. This was done in processFrame just after the averaging of the RGB bytes in WebCam.c as shown below;

```
//--------------------------------------------------------------
// calculate new pixel value
PixelValue = (int)RedByte + (int)BlueByte + (int)GreenByte;
```

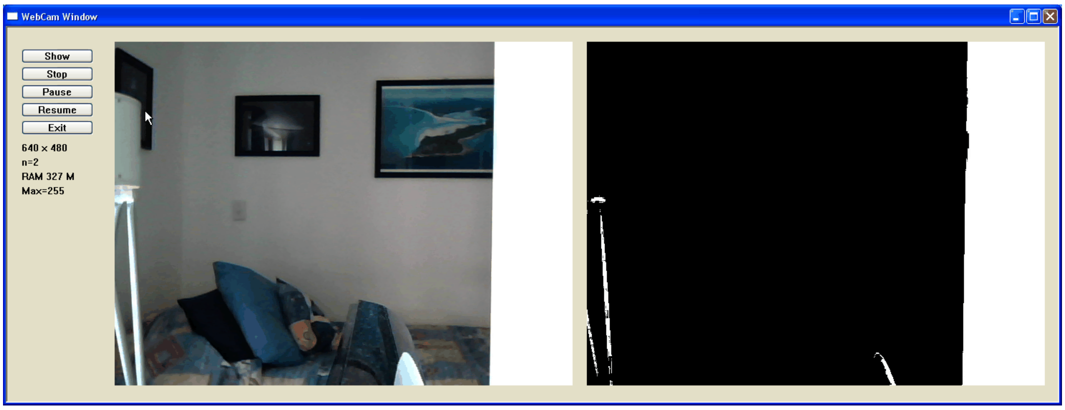


Figure 6.5: Application of the High Pass Filter

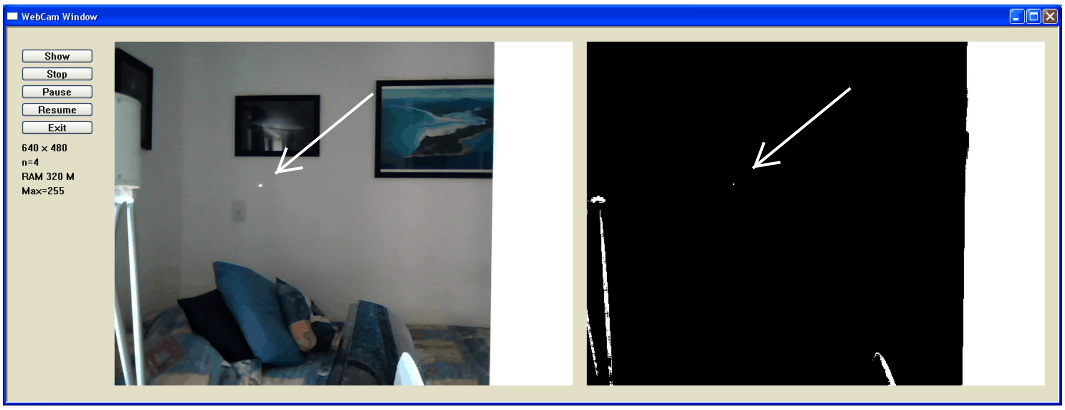
```
    PixelValue /= 3;
//------------------------------------------------------------

// "High pass" filter
if (PixelValue < 250) {
    PixelValue = 0;
}
```

This implementation would result in the second frame in the WebCam.exe to only display pixels that are above 250.

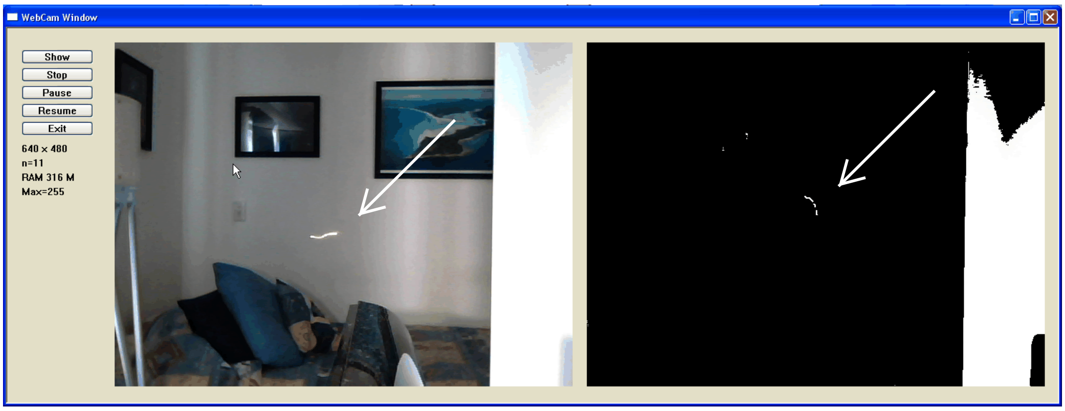
6.5.4 Initial Test of the High Pass Filter

The program was compiled and executed and resulted in the output as can be seen in figure 6.5. As it is evident in this figure, there are areas of bright light shown, mostly on the right hand side of the image where the light from the window was reflecting of the cupboard wall, and just next to it where light was reflecting of a computer screen and then an area of white light on the left of the image, which is light reflecting of the trusses of the telescope which was standing next to the window.



640 x 480
n=4
RAM 320 M
Max=255

Figure 6.6: Laser captured and clearly visible!



640 x 480
n=11
RAM 316 M
Max=255

Figure 6.7: Exposure compensation causes laser trails

6.5.5 Testing the Presence of the Laser

The initial outcome of the code looked promissing as only the brightest areas of the image came through. So the next step was to take the laser and point it at the wall to see if a little white dot was going to show up. This was done and as it can be seen in figure 6.6, the laser is clearly visible as a white dot!

This means that all the system needs to do in a perfect environment is to return the location of the first pixel above the threshold. But there was another problem as shown in figure 6.7 where if it was too dark in the room, the camera would compensate by increasing the exposure time. This results in a trail instead of a laser dot, meaning that the code needs to be able to pick this up and calculate the correct position.

6.5.6 Summary

In this section the first stage in the development of the high pass filter was performed by creating an algorithm to just let light above 250 in intensity through, and proved to be successful. A problem with exposure compensation was identified as well, as well as the presence of other areas of bright light. In the next section, a perfect environment was created to test the performance of the system.

6.6 Creating a Controlled Environment

6.6.1 Overview

In this section the problems associated with other sources of light on the laser detection system is introduced and measures taken to reduce these light sources in a testing environment are explained. Two seperate testing environments are also introduced and the significance of each is explored. This section concludes with the expectations of the performance requirements of the system in an uncontrolled non testing environment.

6.6.2 Problems with Other Light Sources

As it was evident in the previous sections especially if referred to figure 6.5, a problem is encountered when there are other sources of light, especially if the sources are intense and not spread over a uniform area of the image.

At this stage of the project, while developing a laser detection algorithm, it was best to set up an environment where there are no sources of bright light other than the laser, so that testing on the algorithms for detection can be performed.

In the next section, steps taken to remove these areas of light will be explained.

6.6.3 Reducing Light Sources

For testing the laser pointer mouse, as explained earlier, there were two test environments. The main testing environment was the author's bedroom environment, simply for convenience, and the second testing environment was the school classroom environment. The classroom environment was more realistic in that it had a real projector and screen and was in an environment that was a more likely place for the laser pointer mouse to be used, than the bedroom environment.

Due to the fact that the bedroom environment was the more frequently used and accessible environment, measures had to be taken to emulate a lecture theater or board room environment where there wouldn't be any sudden sources of light. So adjustments were made in the bedroom to get rid of unwanted sources of light.

During the day, the curtains were closed and any areas where the curtains weren't joined properly, was sealed closed with the aid of a few paper clips. If these areas weren't closed properly and some light was allowed to leak from the curtains, the webcam would compensate and these areas would appear as great big unwanted areas of light. Therefore it was important to seal these areas so that absolutely no light was allowed to leak through the curtains. Other sources of light such as the roof lights were turned off during testing as well.

6.6.4 A really sensitive webcam

Now that all sources of light was removed, the scene is completely dark, and as a result the webcam would start to apply exposure compensation. This would cause the laser to develop trails, which was an unwanted trait due to the difficulty to determine the position of the laser.

In a real projected presentation, this would not be a problem because most of the light over the field of view of the camera would be spread out, and as a result the camera would not perform exposure compensation and the laser would not appear as trials. To counteract this, the desk lamp which was behind the camera anyway, was turned on and pointed at the wall.

This dramatically improved the performance of the camera, since the light was spread over a large area, the camera would turn off its exposure compensation. This resulted in faster frame rates from the camera, and hence no laser trails.

This was the test environment initially for the laser pointer mouse system.

6.6.5 Testing Laser in Controlled Environment

There were two environments for testing this system; the bedroom and the classroom. Firstly the laser was tested in the bedroom environment to test its frame rate, then the laser was testing in the classroom environment, so that it could be confirmed that the camera would not perform exposure compensation.

The Bedroom

Once all of the adjustments were made to remove unwanted areas of light from the field of view of the camera, as well as removal of exposure compensation by using the desk lamp, the laser pointer was pointed at the wall for testing of frame rates. It was found that the frame rates were good enough for the laser to maintain its shape without developing a trail.

The Class Room

It was necessary for this high pass filter to be tested in a real presentation environment such as a classroom to ensure that the laser is still easily detectable without any other bright areas of light showing. It was found that the laser was still clearly visible even if it was positioned over a bright white area of the screen.

6.6.6 System Expectations

In a perfect environment where the laser is the brightest point of light, all that is needed is for the system to look for the first bright pixel and return its position to work successfully. But in the real world, there will always be other areas of bright light, and the system needs to be able to detect this.

In order for this to be done, the system needs to look for certain characteristics of the laser. For example, if an area of light is too large, then it can't be the laser and therefore needs to be ignored. So the system needs to look for areas of light where the width or height doesn't exceed a certain predefined limit. This limit depends on the laser being used and the resolution of the camera, because for a camera with a larger resolution, the amount of pixels that the laser point will take up will be more than for a camera with a smaller resolution.

There are of course more powerful laser pointers, as discussed in section 4.2.3. If a more powerful laser was to be used for this application, then the system needs to be able to detect this because the area of light that a green astronomical laser pointer will take up will be a lot more than a normal red laser pointer. (see Figure 4.4.

It is the aim of this project for a system to be developed that could pick out the laser from multiple areas of bright light, but if the laser was to be in one of those areas, the system would not be able to detect this at all.

6.6.7 Summary

This section explains the problems encountered with other sources of light and the camera's tenancy to apply exposure compensation in dark areas which causes a lot more unwanted light to appear on the high pass filtered image. Two testing environments were also introduced and the significance of each was explained.

It was found that in order for this system to perform reliably, it needed to be able to detect other areas of light and ignore them be able to find the laser successfully in these areas of light. In the next section the system will be enhanced even further to display the detected location of the laser.

6.7 Detecting Pixel Location

6.7.1 Overview

In this section the implementation of the initial basic laser detection system will be explained and then followed by some really basic tests to see if the location of the laser that was returned appears correct.

6.7.2 Implementation of Laser Location Detection

The laser detection system was modified even further to locate the brightest pixel, and display the location of the this pixel on the left hand side of the laser detection window. Obviously for this to work a fair bit of code modification had to take place and these modifications are explained in the following sections.

6.7.3 Setting Up the Output

In order to have the easiest way to output a simple result such as the `x` and `y` location of the laser, one of the lines of code that was originally used to output the maximum intensity value of the pixels in the image was commented out, and was used instead for outputting the location of the laser. The original code found in `processFrame` was commented out as shown below, for more details refer to Appendix D.1.

```
//------------------------------------------------------------
// sprintf(tmpbuf, "Max=%d", MaxValue);
// SetWindowText(hTextMaxVal, tmpbuf);
//------------------------------------------------------------
```

6.7.4 Outputting the Location of the Laser

A seperate function called detectLaser was added for the purpose of outputting the location of the laser. The inputs to the function is the current y position (pCurY), the current x position (pCurX), the last value of y (pLastY) and the intensity value. The arguments are passed onto the detectLaser function from the processFrame function. A complete listing of the function is shown below:

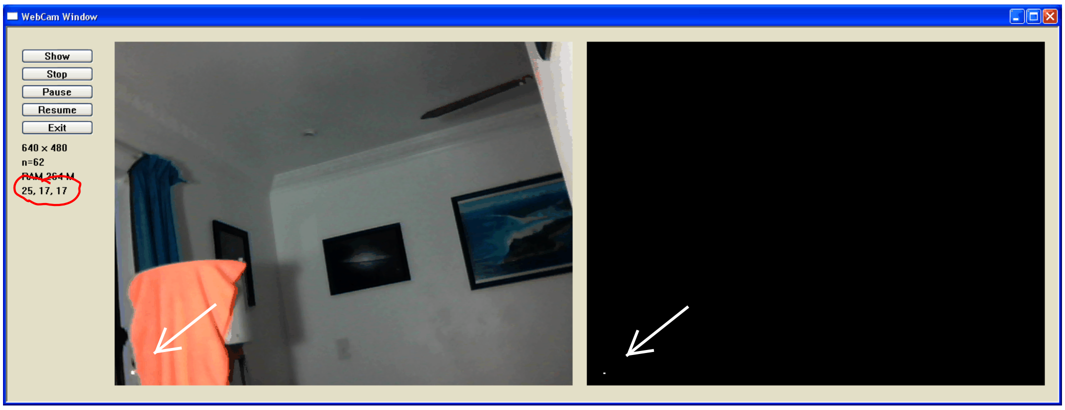
```
int  detectLaser(int pCurY, int pCurX, long *pLastY, int pIntensity) {
    if (pIntensity > LASER_DETECTION_THRESHOLD) {
        char tmpbuf[20];
        sprintf(tmpbuf, "%d, %d, %d", pCurX, pCurY, *pLastY);
        SetWindowText(hTextMaxVal, tmpbuf);
        *pLastY = pCurY;
    }
    return 0;
}
```

What this function does is it checks the pIntensity and makes sure that it is above the threshold, and if it is, then it reports the position of the laser. It does this by creating a string using sprintf into the string buffer tmpbuf which contains the location information of the laser.

After the string is created, the line of code that was used to output MaxValue (refer 6.7.3) is now used to output the location of the laser using the setWindowsText function.

6.7.5 Testing Laser Location Detection Approach

Now that the very basic location detection of the laser was implemented, the system could now be tested to see if the detection was successful. The approach that was used to do this was quite simple. The image is basically a Cartesian plane where the bottom



```
640 x 480
n=62
RAM 264 M
25, 17, 17
```

Figure 6.8: Laser pointed at the bottom left of the image, the result of (25,17) appears correct.

left corner of the image is (0,0) and the top right corner of the image is (640,480) which is the resolution of the image.

So what was done was the laser was pointed at the most obvious points and the location output on the left of the screen was examined for expected results, so if the laser was pointed at the bottom right of the screen, then the program would return values close to (0, 0), and if the laser was pointed near any edge of the screen, then either `x` or `y` would be either close to 0 or close to its maximum value, which for `x` is 640 and `y` is 480.

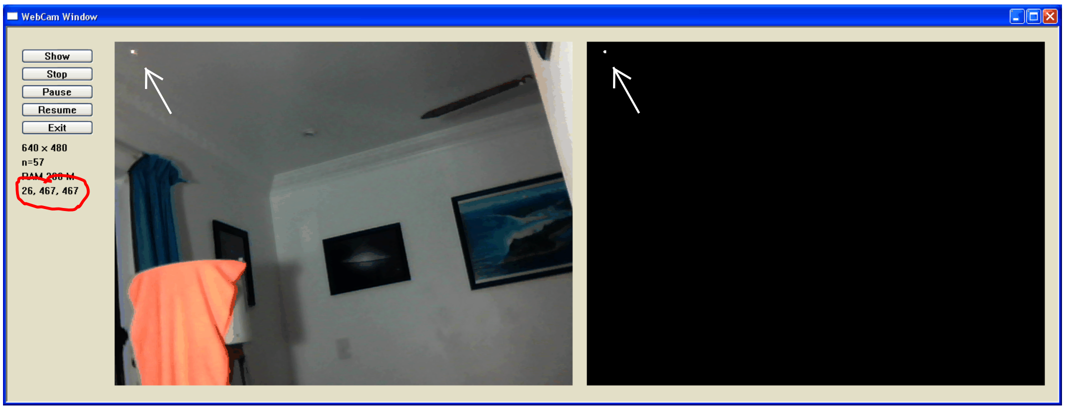
6.7.6 Testing Laser Location Detection

The project was compiled using `make`, and the laser was pointed in the areas of interest. Firstly the laser was pointed at the bottom right of the screen as shown in figure 6.8. Note that the result shows values near the origin.

The laser was then pointed at the top left corner of the image and the result of this is shown in figure 6.9, note how the value for `y` is close to its maximum of 480.

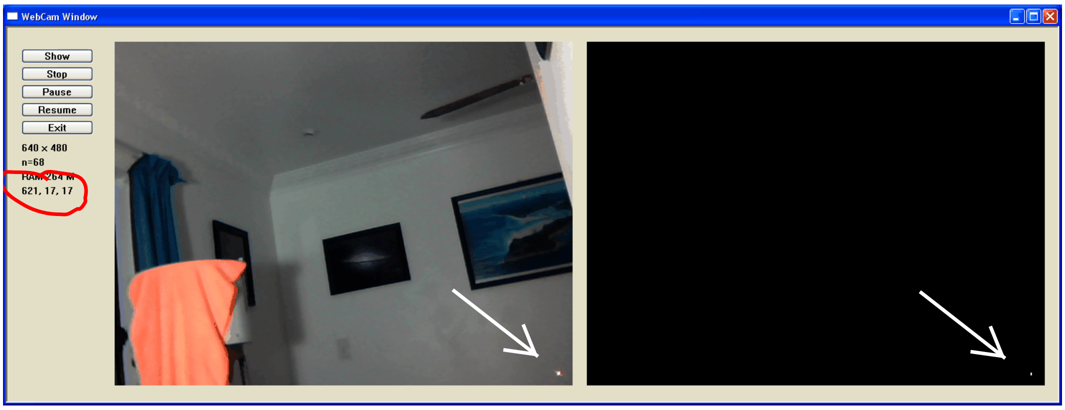
After this the laser was pointed at the bottom right corner of the image and the result of this is shown in figure 6.10, note how the value of `x` is close to its maximum of 640.

And finally, just to be sure, the laser was pointed at a location that was not in a corner.



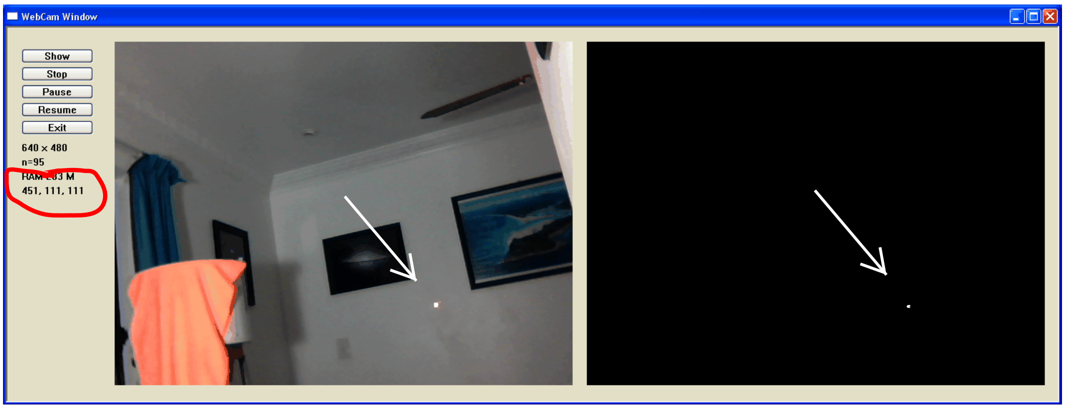
```
640 x 480
n=57
RAM 200 M
26, 467, 467
```

Figure 6.9: Laser pointed at the top left of the image, the result of (26,467) appears correct.



```
640 x 480
n=68
RAM 264 M
621, 17, 17
```

Figure 6.10: Laser pointed at the bottom right of the image, the result of (621,17) appears correct.



```
640 x 480
n=95
RAM 283 M
451, 111, 111
```

Figure 6.11: Laser pointed at a random spot , the result of (451,111) appears correct.

The result of this is shown in figure 6.11. Note the location values this time; `x` Is 451, which appears correct, and `y` is 111 which also appears just about right.

6.7.7 Summary

In this section the implementation of the initial basic laser detection is explained. The location of the laser is determined based on it being used in a perfect environment with no other sources of light, because this first inital laser detection system will return the location of the first pixel above the threshold.

So tests were done on this method and so far appeared quite correct, but more on this will be discussed in the next chapter.

6.8 Initial Laser Detection System Performance

6.8.1 Overview

This section discusses the implementation of a data collection system that could be used for testing.

6.8.2 Problems with Current System

Already some early problems with the laser detection system have been noticed, particularly with speed. The frame rate of the system doesn't seem to be fast enough, and setting the frame rate of capture of the camera to a higher value did not increase the actual capturing frame rate.

In order to perform tests on the performance of the system, it was necessary to write some more code that would enable this.

6.8.3 Measuring the Performance of the System

In information that was of interest in the performance measurement of the system, was the average frame rate as well as a list of the coordinates of the laser so that analysis could be done in the motion detection capabilities of the system.

6.8.4 Approach

The way that this performance measuring system was implemented was through outputting the data to a text file. One text file would contain all of the data collected in one minute, after which another text file is created for the next minute of detection. The reason for this is that the amount of data that could be collected could get to large for one file, so it was better to split them up.

The system starts recording the coordinates of the laser as soon as the laser is detected. Once the laser is no longer detected, the system determines the amount of time that the laser was on for and prints this at the end of the list as well as the calculation of how many frames per second was detected. So there is a new list of coordinates for each period that the laser was visible. This implementation made it easy to find the results of certain motion tests carried out.

6.8.5 Implementation of the Code

In order to implement the proposed data collection system, a few more functions were written. A function was written to print the position of the laser to the file, and a function was written to create the filename, and another function was written to print the detection time to the file, and the other function that was written was a function to calculate the detection rate.

The function that prints the laser position to file was called, `laserPosToFile()` and its implementation is shown below;

```
int  laserPosToFile() {
    FILE *fp;
    char * pfilename;

    fp = fopen(currentFileName, "a");
    if (fp == NULL) {return 1;}
    fprintf(fp, "X Min: %d, X Max: %d, Y Min: %d,
            Y Max: %d \n", xMin, xMax, yMin, yMax);
    fclose(fp);
    return 0;
}
```

Obviously the system would open the file with the `currentFileName` which is a global variable created by the function that creates the filename, `createFileName()`. The current laser position data, using `fprintf`, is printed onto the end of the text file that was opened. Currently only `X Min` and `Y Min` are used and they actually contain the actual position of the laser, not the minimum and maximum values.

The function that creates the filename is called `createFileName()`, and its purpose is to create a unique file name automatically for the output of the data. The source code of this function is shown below;

```
char * createFileName() {
        char * filename = "";
        // time_t timeVal;
        struct tm * timeVals;
        int test = 9;
        timeVals = localtime(&startTime);
        sprintf(filename, "Output_%i%i%i_%i%i.txt", timeVals->tm_mday,
                timeVals->tm_mon, timeVals->tm_year,
                timeVals->tm_hour, timeVals->tm_min);
        return filename;
    }
```

This function returns a `char` pointer (which is effectively a string in terms of more modern programming languages) to the name of the file. This function gets the current time from the computer and puts it into a structure which contains the various fields of date and time. The function then uses `sprintf` to assemble a `char` string containing the filename and returns it for use.

The detection rate and time is printed to the file using a function called `timeToFile()`. This function is presented below;

```
    int timeToFile() {
        FILE *fp;
        fp = fopen(currentFileName, "a");
        if (fp == NULL) {return 1;}
        fprintf(fp, "%s\n", calcDetectionRate());
        fclose(fp);
        return 0;
    }
```

This function is a simple function that prints the output of `calcDetectionRate()` into the end of the data output file. `calcDetectionRate()` Performs the calculation to determine how many frames per second the system is performing at. The implementation of this function is given below;

```
char * calcDetectionRate() {
    char * lResult = "";
    double timedif;
    double detectionRate;
    time_t endTime;

    // initialise the end time
    endTime = time(NULL);

    // difference between times
    timedif = difftime(endTime, startTime);

    // number of detections / number of seconds
    detectionRate = pixelCount / timedif;

    // format into string
    sprintf(lResult, "%i pixels counted. Detection Rate is %f pixels
                      per second. Total time was %f.\n",
                      pixelCount, detectionRate, timedif);

    return lResult;
}
```

This function sets the end time and then determines the difference between the start time and the end time. The start time, which is stored in the `startTime` variable was set in `processFrame` at the beginning of the detection period. The difference between the start time and the end time is stored in `timedif`. The detection rate is the number of pixels divided by the amount of time that passed. This calculation is stored in `detectionRate`.

All of this information is then partitioned into a string called `lResult` and used in the `timeToFile()` function.

All of the above functions are driven by code in the `processFrame` function that starts the data output as soon as the presence of the laser is detected. Firstly it stores the start time and then creates the filename for the data file.

After this it prints the position of the laser to the file using the `laserPosToFile()` function and then increments the count of pixels since first detected. If no pixels were detected, the total detection time would then be written to the file and the pixel count is then reset to 0.

The implementation of this code is shown below;

```
if (xMin > NOTSET &&  yMin > NOTSET) {
    if (countStarted == 0) {
        startTime = time(NULL);
        countStarted = 1;
        currentFileName = createFileName();
    }
    laserPosToFile();
    if (pixelCount > 0) {
        pixelCount ++;
    } else {
        pixelCount = 1;
    }

} else {
    timeToFile();
    countStarted = 0;
    pixelCount = 0;
}
```

For more details about the implementation of this data output system, refer to `WebCam.c` in Appendix D.1

6.8.6 Program Output

Samples of the output of this system can be found in Appendix **??**

6.8.7 Summary

For the ability to test the performance of this laser detection system it was necessary to set up a basic data output system that could print out information such as the positions of the laser during each detection as well as the detection time and detection frame rate of the system.

In this section the implementation of this system was explained.

6.9 Chapter Summary

In this chapter it was found that the high pass filter method was the best for the purposes of this project and that colour filtering didn't work so well. The initial laser detection system was developed and some initial tests were conducted on it. The results of these tests will be discussed in the next chapter.

After this the implementation of the data collection system was discussed and the output from this as well as some results of some tests that were performed with this system will be covered in the next chapter.

Chapter 7

Results and Discussions

7.1 Chapter Overview

In this chapter the findings from the results produced during implementation will be discussed. Firstly the problems with the colour filters will be discussed, which will then be followed by the discussion of the successful high pass filter approach.

After this the laser detection performance will be discussed from the data collected which is available in Appendix **??**. This chapter then concludes with the proposed improvements to the laser detection algorithm

7.2 Colour Filters

As it was explained in the previous chapter, filtering out just red from the webcam image did not make the laser point stand out and easier to detect. This was because mostly everything that reflects light will reflect a certain amount of each of the primary colours, which explains why the image was clearly visible even with just the red filtered out.

It was also evident from these tests that the laser was white and not red due to the web cam's tendency to overexpose bright areas of light. So it was clear that colour filtering would not work.

7.3 The High Pass Filter

The fact that the laser point was white bought rise to a new approach of just filtering out the bright light above a certain threshold. This enabled the laser to stand out and very easy to detect. As a result of this theory, the project proceeded further with the implementation of code that extracts the position of the laser from the image by looking at the first pixel that has an intensity value above a certain threshold.

7.4 Laser Detection Performance

As shown in the previous chapter in figures 6.8, 6.9, 6.10 and 6.11, the locations returned by the initial laser detection system were accurate.

The system was enhanced even further to record the positions of the laser in a text file along with the detection rate and detection times. The results of these test will now be discussed. After setting up the environment to become a controlled environment where there were no other sources of light appearing in the image, two types of motion tests were performed; tracking of the laser traveling from the top of the screen to the bottom of the screen, and tracking of the laser when tracing circles.

7.4.1 Laser Movement

In this test, the laser was traced from the top of the screen to the bottom in order to see if the positions that would be recorded in the text file were correct. Unfortunately the camera was sitting at an angle showing both the wall and the roof which meant that simple moving of the laser in a straight line from the top to the bottom of the image was a bit awkward.

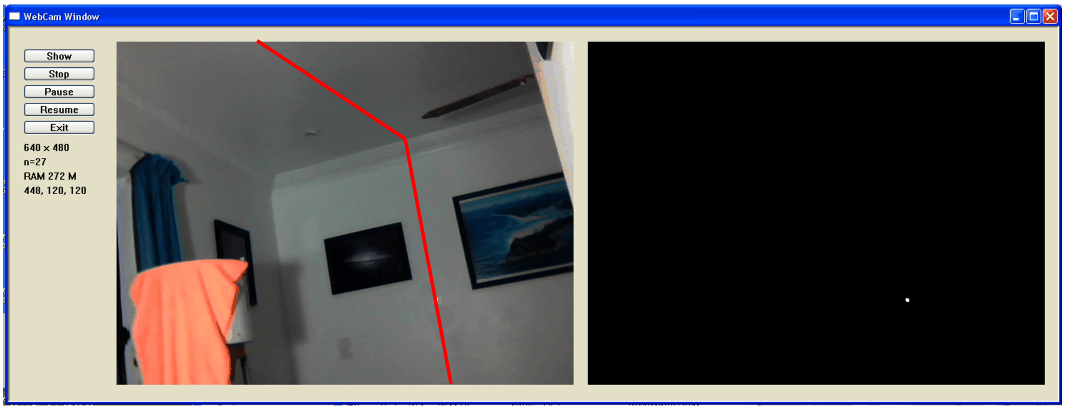


Figure 7.1: The motion of the laser is indicated by the red line

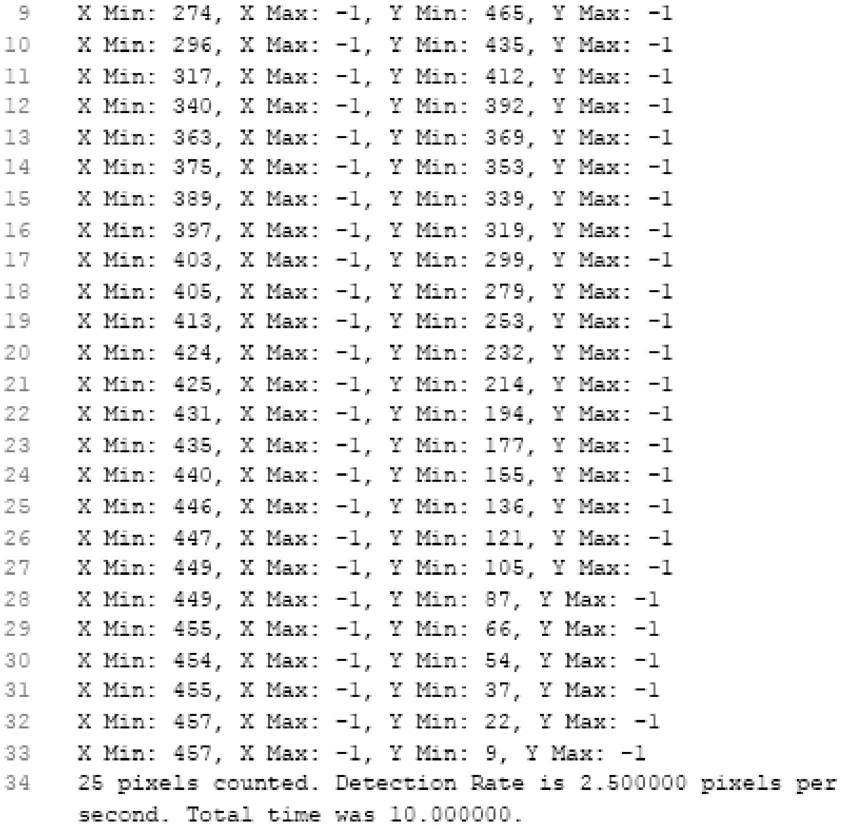
In real life the laser was simply traced straight down from the roof to the floor, but in captured image the laser did not travel in a straight line at all. The motion of the laser is illustrated in figure 7.1.

All of the resulting data sheets collected from the tests are given in Appendix **??**, but a sample of the output from the laser motion test described above is given in figure 7.2 and it can be referred to in Appendix refsec:lasermotiondata.

In order to determine the success of this laser detection test, the rate of change of x will be compared with the rate of change of y between the data and the image. The image is not an accurate reproduction of the path that the laser traveled, but just an estimate.

When looking at figure 7.1, starting from the top, the change in x is around the same as the change in y, but as the laser travels down the image, the rate of change between the x values become significantly less and the rate of change in y becomes slightly more.

Comparing these expectations with the actual data collected, it can be seen that the results corresponds with the expectations based on figure 7.1. From line 9 to 25 the rates of change between the values of x are quite fast, and then from 25 to 34 the rates of change are significantly less. When looking at the x values, the x values start at 274 and end at 457, and the range of x values are from 0 to 640, so it is evident that the laser did not travel across the whole range of x values, which is expected when looking at figure 7.1 as the laser was traveling mostly in the negative y direction.



```
X Min: 274, X Max: -1, Y Min: 465, Y Max: -1
X Min: 296, X Max: -1, Y Min: 435, Y Max: -1
X Min: 317, X Max: -1, Y Min: 412, Y Max: -1
X Min: 340, X Max: -1, Y Min: 392, Y Max: -1
X Min: 363, X Max: -1, Y Min: 369, Y Max: -1
X Min: 375, X Max: -1, Y Min: 353, Y Max: -1
X Min: 389, X Max: -1, Y Min: 339, Y Max: -1
X Min: 397, X Max: -1, Y Min: 319, Y Max: -1
X Min: 403, X Max: -1, Y Min: 299, Y Max: -1
X Min: 405, X Max: -1, Y Min: 279, Y Max: -1
X Min: 413, X Max: -1, Y Min: 253, Y Max: -1
X Min: 424, X Max: -1, Y Min: 232, Y Max: -1
X Min: 425, X Max: -1, Y Min: 214, Y Max: -1
X Min: 431, X Max: -1, Y Min: 194, Y Max: -1
X Min: 435, X Max: -1, Y Min: 177, Y Max: -1
X Min: 440, X Max: -1, Y Min: 155, Y Max: -1
X Min: 446, X Max: -1, Y Min: 136, Y Max: -1
X Min: 447, X Max: -1, Y Min: 121, Y Max: -1
X Min: 449, X Max: -1, Y Min: 105, Y Max: -1
X Min: 449, X Max: -1, Y Min: 87, Y Max: -1
X Min: 455, X Max: -1, Y Min: 66, Y Max: -1
X Min: 454, X Max: -1, Y Min: 54, Y Max: -1
X Min: 455, X Max: -1, Y Min: 37, Y Max: -1
X Min: 457, X Max: -1, Y Min: 22, Y Max: -1
X Min: 457, X Max: -1, Y Min: 9, Y Max: -1
25 pixels counted. Detection Rate is 2.500000 pixels per
second. Total time was 10.000000.
```

Figure 7.2: Data collected from the laser motion test

When looking at the y values, the rates of change are fairly similar right through the motion of the laser. It is also evident that the y values start at 465 and end at 9 and the range of y values are 0 to 480. This shows that the laser traveled across the whole range of y which is exactly what is expected. So it is clear from this test that the motion tracking did in fact work.

7.4.2 Circle Tracing

A brief test was done on the circle tracking capabilities of this system by tracing a circle against the wall and seeing the output results. The output of this test is in Appendix **??** lines 52 to 88.

The features in the data that has been looked for were periodic fluctuations in the values of x and y at the same time. If a circle was being traced, one would expect that the values of x and y would remain in the same area, and that they would increase and decrease in cycles and if this data was plotted it would look like a sine wave.

When looking at the circle tracing data in Appendix **??** lines 52 to 88 these features are clear in that the values of x stays between 460 and 420 and increases and decreases over time like a sine wave. The same occurs with the values of y, ranging between 65 and 110 also increasing and decreasing. The difference between these ranges between both x and y is about 45 meaning that the circle being traced was not oval in shape.

So the data given in Appendix **??** between lines 52 to 88 definitely shows evidence of circular tracing.

7.4.3 Detection Rate

From all of the tests results shown in Appendix **??**, it is clear that the detection rate of this system is currently at around 2.5 pixels a second which is quite slow and will cause problems when developing the circular clicking requirements of this project.

7.5 Problems With the Current Approach

The main problem with the current approach is that the system will register any bright light source as the laser and return it's coordinate even if it's not the laser. As soon as an area of bright light appears before the laser point, the system will be stuck on the same coordinates, that being the first pixel detected of this area of light. So an algorithm needs to be developed to make the laser detection system more intelligent and look for certain features of the laser and filter out anything that do not meed these features.

The other problem was with the detection rate of this system, which is way too slow, meaning that the user would need to trace the circle very slowly an object for the system to be able to register this as a click. More work needs to be done on this to improve the speed of detection so that more laser coordinates can be detected in a second so that quicker circle tracing can be performed.

-

7.6 Proposed Improvements

In this section the improvements that were designed for this laser detection system will be explained. These improvements were to enable for more intelligent laser detection so that bright areas of light are simply ignored. In order for this system to be designed it was important to know what the certain characteristics of the laser and the laser's possible behaviors are.

A possible set of steps for more intelligent laser detection is explained in the next section

7.6.1 Characteristics of the Laser Spot

The characteristics of the laser spot have been listed and a system was proposed to look for these characteristics;

1. There will be a number of pixels that will all have intensity values greater than the desired threshold and adjoining each other.
2. Width and height count of pixels will be very roughly the same as each other

   1 This forms a circle

   2 May also form a square or close to a square

      i. This is known as the bounding box

3. The width and height will not be larger than a certain amount

   1 If this is not the case then the spot is not a spot but an area of bright light that is not the characteristic of a laser

   2 Due to the nature of different resolutions, the width and height limit is determined by a certain percentage of the total width and the total height of the image taken by the camera

   3 This percentage generally tends to be larger with brighter lasers as per the example image that was taken and given in this document somewhere yet to be determined.

4. The number of pixels between the max and min of each x and y dimension should be above 50
5. The detected pixels will not be fewer than a certain amount, say 3 pixels

7.6.2 Method of Detection

1. When the first pixel is found its x and y positions are recorded as the first minimum and maximum values of x and y, and these coordinates are stored in the variables xMin, xMax, yMin, yMax, and readjusted as adjacent pixels are found.
2. At this point it can be assumed that suitable laser point has been found
3. For this assumption to stay true the next pixel will also be above the threshold.

   1 This essentially means that the difference between the x position of the last pixel and the current pixel is 1 and only 1. This meets requirement 1.

2 In order to perform the above, another variable is needed, hence the need for lastX. Which records the x position of the last time a pixel above the threshold has been detected.

4. As soon as a pixel is detected that is not above the threshold, it is assumed that the end of laser reflection for this particular row (this particular value of y) has been detected, and therefore, for the time being we are not interested in scanning any further because we think that we have found the laser.

   1 This is because it would be a waste of CPU time to scan further and should make the detection a bit faster.

5. The system would now start scanning down the next line of the pixels with the value of y incremented.

6. If a pixel is found to be over the threshold and the difference between its location and the location of the first x pixel (lastXMin) found in the previous row is no more than 2 then this pixel is part of the laser spot.

   1 This is because refer figure, if the difference was more than 2 then the resulting shape would not be a circle because it would result in a shape that is too flat to be a circle belonging to the laser.

   2 This would also result in the value for lastXmin to be updated with this value found

7. Each time a new pixel is found belonging to the laser, the values of xMin and xMax is checked and updated. But the value for yMax will still stay the same but yMin would be updated.

8. So the system continues to scan for more pixels until a pixel is found that is not above the threshold, then it starts scanning the next line by incrementing the value for y again.

9. During each row of pixels being scanned, a count of x values is kept (xCount). If this count exceeds the value for the maximum allowed diameter of the laser spot, and the count of y values (yCount) is also above the maximum allowed diameter (MAX_LASER_SPOT_DIAMETER). Then the light source being detected is not the laser, and could be something else.

10. If yCount is less than or equal to MAX_LASER_SPOT_DIAMETER then the system will continue to scan x values until the x value count reaches a certain length MAX_LASER_TRAIL_LENGTH, which is the maximum length the resulting trail of the laser is allowed to be. If this length is too long, then the user is not controlling the system but just playing with the laser.

11. Each time a new row is started and the condition 6 is met, then the value for yCount is incremented.

    If the value for xCount is greater than the MAX_LASER_SPOT_DIAMETER, then the detection is no longer the laser spot.

7.6.3 Chapter Summary

It this chapter is was seen that a good start to this laser pointer mouse project has been made by some promising results regarding the laser detection capabilities of the system. It has been found that the high pass filter method works the best with the detection of the laser, but will need some more improvements to introduce more intelligent detection algorithms. This chapter concluded with some proposals as to how this intelligence could be implemented.

Chapter 8

Conclusions

8.1 Introduction

The aim of this project was to develop software that can enable a user to control the computer of a projected presentation using only a basic laser pointer and webcam exclusively. Unfortionately the completed package that can do this have not been achieved in this project. But what has been achieved was the initial detection of the laser using an ordinary webcam and laser pointer, which lays the foundation for future work, due to this project still being in its very early stages.

8.2 Existing devices

A lot of time and effort (in fact too much time and effort!) has been spent at the start of this project to find examples of work done by other people towards the laser pointer mouse concept, and although there weren't that many out there that could satisfy the requirements of this project, there were some that came close.

There were varying approaches towards the detection of the laser, and the three main ones were:

1. Looking for pixels with intensity values above a certain threshold
2. Applying physical and code driven colour filters to make laser detection more obvious, and;
3. Detecting changes in the frames of images captured from the camera in order to detect the moving laser dot

In this project looking for pixels with high intensity values was referred to as a high pass filter, and this approach was used in this project.

8.3 Laser Detection

The detection of the laser and the return of the location of the laser was the only real successfull achievement during the course of this project. But it has been determined that the high pass filter approach works reliably and it quite robust, and as a result, this achievement is a good start to further development in the project.

The colour filtering methods described in some of the other works was found to be insuffiently reliable, and as a result was not taken any further. One of the successfull colour filter approaches required a physical red filter to be put in front of the camera, and since the requirements of this project was to use basic equipment with no add-ons, this idea of colour filtering was not persued any further.

Some problems were found with the laser detection rate in that the system currently is only able to detect two locations per second which is not good enough for reliable motion tracking requirements, especially for circle detection. More investigation on this needs to be done in future work.

8.4 Motion Tracking

In order to be able to detect circles being traced by the user, the system needs to be able to keep track of the motion of the laser. Prototype code has been developed to detect the motion of the laser and output the collected coordinates into a text file for further analysis.

It was found that the system did corretly find the position of the laser when several tests were performed such as moving the laser from the top of the screen to the bottom. Tests were also performed with circle tracing and it has been found that the coordinates returned possed some of the geometric characteristics of a circle.

8.5 Work Not Completed

Due to the bad project and time management of the author, and other professional commitments and a limited amount of high energy time at the end of the day, a lot of work has not yet been completed and sadly quite a few of the objectives as outlined in Appendix A has not been achieved.

As time was starting to run out some work was started on the more reliable detection of the laser, but the project has not taken off as well as planned and as a result no work was done on calibration, circle tracing, navigation features, auto calibration and system integration.

A lot of time was also wasted at the start of this project doing too much research on the existence of such devices and alternative devices, and as a result ended up with a lot of analysis of the existing devices but limited achievements for this project.

8.6 Further Work

The next step with this project is to continue from the steps explained in section 7.6.1 and develop and test the code that would enable for more robust detection of the laser no matter what the lighting conditions are.

After this, a calibration system needs to be developed so that the system can return the true coordinates of the laser. This calibration system may take some real effort and time to get right which may be a reason for the next phase of this work to end here.

The next step would be to implement circle detection and then the final integration into the system

8.7 Final Conclusion

It has been shown through the achievements of this project that it is possible to detect a laser using a basic webcam and basic laser pointer. There may be a few performance and reliability issues, but with a bit more work this system can become a reality!

References

Armenio, C. (n.d.), Design of a laser controlled mouse using opencv, Technical report.

Atul Chowdhary, Vivek Agrawal, S. K. S. S. (n.d.), Laser actuated presentation system, Technical report, Saha Institute of Nuclear Physics.

Carsten Kirstein, H. M. (2002), 'Interaction with a projection screen using a camera-tracked laser pointer', *Multimedia Modeling, 1998. MMM '98. Proceedings. 1998* .

Cheng, K. & Pulo, K. (2003), 'Direct interaction with large-scale display systems using infrared laser tracking devices', *In APVis '03: Proceedings of the Asia-Pacifc symposium on Information visualisation* pp. 67–74.

Code::Blocks (2010), Code::blocks. `http://www.codeblocks.org/`.

CodeProject (2010), Simultaneous previewing & video capture using directshow. `http://www.codeproject.com/KB/audio-video/DXCapture.aspx`.

de Bruijn, R. (2008), Camera Supported User Interface for Presentation Software, PhD thesis, Technische Universiteit Eindhoven.

Dekel, A. & Kirkpatrick, S. (n.d.), 'Room user interface (rui) design for laser based interaction'.

Derhgawen, A. (2008), Laser tracking with webcam for human-computer interaction. /urlhttp://ashishrd.blogspot.com/2008/03/laser-tracking-with-webcam-for-human.html.

Eyo (2010), Panasonic ub-t760 interactive panaboard 64" ub-t760. `http://www.retailing.com.au/prod_185890_proddesc_PANASONIC_UB-T760_INTERACTIVE_PANABOARD_64_UB-T760.html`.

Finley, M. C. (2003), 'Laser and digital camera computer pointer device system'.

Hancock, A. P. N. (2010), *Research Project Reference Book*, University of Southern Queensland.

HO, D. (2010), Notepad++. `http://notepad-plus-plus.org/`.

Kelvin Cheng, M. T. (2006), 'Real-time monocular tracking of view frustum for large screen human-computer interaction'.

Kitware (2010), Cmake. `http://www.cmake.org/`.

Lee, J. C. (2008), 'Hacking the nintendo wii remote', pp. 42–43.

Logitech (2010*a*), Logitech hd pro webcam c910. `http://www.logitech.com/en-au/webcam-communications/webcams/devices/6816section=specs`.

Logitech (2010*b*), Logitech webcam c200. `http://www.logitech.com/en-au/webcam-communications/webcams/devices/5865`.

Logitech (2010*c*), Professional presenter r800. `http://www.logitech.com/en-us/mice_pointers/presentation_remote/devices/5873`.

Lukauskas, S. (2010), Laser pointer tracking. /urlhttp://dev.saulius.me/2010/03/20/laser-pointer-tracking/.

Microsoft (2010*a*), Visual c++ developer center. `http://msdn.microsoft.com/en-us/visualc/default.aspx`.

Microsoft (2010*b*), Wireless notebook presenter mouse 8000. `http://www.microsoft.com/hardware/Presenter/productdetails.aspx?pid=085`.

Nan Wodarz, E. (October 2005), 'Electronic whiteboards for schools: An effective instructional tool or just another trend?', *School Business Affairs* **Volume 71, Number 9**, 43–44.

Navia, J. (2010), lcc-win32: A compiler system for windows by jacob navia. http://www.cs.virginia.edu/~lcc-win32/.

of New South Wales, T. U. (2010), Legal and compliance - commercial activities - risk assessment template. www.legal.unsw.edu.au/commercial/Risk_Assessment_Template.doc.

Olsen, D. R. & Nielsen, T. (2001), 'Laser pointer interaction. in proceedings of acm conference on human factors in computing systems (chi'01)', *ACM Press* pp. pp 17–22.

Popovich, E. (n.d.), Presentermouse laser-pointer tracking system, Technical report, Technion I.I.T.

Raynor, Jeffrey (Ambridge Hall17 West Catherine Place, E. E. H. G. (2007), 'Device and system for presenting information'. http://www.freepatentsonline.com/EP1830246A1.html.

skypointer (2010), The skyfinder. http://www.skypointer.net/brktfr.html.

Targus (2010), Wireless presenter with cursor control. http://www.targus.com/sg/product_details.asp?sku=AMP17AP.

TechBuy (2010), Targus basic presenter - wireless, suits notebooks. http://www.techbuy.com.au/p/126154/POINTINGDEVICES_PRESENTATION_DEVICES/Targus/AMP18US.asp.

Toshiharu Wada, Masanobu Takahashi, K. K. J. O. (2007), Laser pointer as a mouse, *in* 'SICE Annual Conference 2007'.

T.V.Thati, M. K. (2005), Using webcam for human computer interaction, project report, University of Hyderabad, India.

Wikipedia (2010*a*), Cmake. http://en.wikipedia.org/wiki/CMake.

Wikipedia (2010*b*), Directshow. http://en.wikipedia.org/wiki/DirectShow.

Wikipedia (2010*c*), Interactive whiteboard, wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Interactive_whiteboard.

Wikipedia (2010*d*), Laser safety. http://en.wikipedia.org/wiki/Laser_safety.

Wikipedia (2010*e*), Lcc (compiler). http://en.wikipedia.org/wiki/LCC_%28compiler%29.

Wikipedia (2010*f*), make (software). http://en.wikipedia.org/wiki/Make_%28software%29.

Wikipedia (2010*g*), Notepad++. http://en.wikipedia.org/wiki/Notepad%2B%2B.

Wikipedia (2010*h*), Opencv. http://en.wikipedia.org/wiki/OpenCV.

Wilson, T. V. & Crawford, S. (2010), How the ipod touch works. http://electronics.howstuffworks.com/ipod-touch.htm/printable.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

FOR: **ANTHONY GIBBONS**

TOPIC: LASER POINTER MOUSE

SUPERVISORS: Dr. John Leis

ENROLMENT: ENG 4111 – S1, X, 2010
ENG 4112 – S2, X, 2010

PROJECT AIM: Develop software that can enable a user to control the computer of a projected presentation using only a basic laser pointer and webcam exclusively.

SPONSORSHIP: The University of Southern Queensland

PROGRAMME: **Issue A2, 30 March 2010**

1. Investigate the concept of a laser-pointer mouse for use with data projectors by investigating literature on the topic, patents, and possible existing commercial devices.

2. Investigate user interface technology and the need for such a device in the commercial market or specialized markets such as to aid persons with certain disabilities.

3. Investigate color point detecting algorithms using a webcam-based vision system, and determine if the recognition of a laser spot can be performed in real-time.

4. Investigate motion-tracking algorithms for the laser spot which account for the variability of the pointer motion, including circle-tracing to emulate mouse-clicking.

5. Develop prototype code for testing the above algorithms, and assess the feasibility of the concept.

6. Investigate the requirement for calibration of the pointer system, and implement prototype test code.

As time permits:

7. Integrate the code into a system which can interface with the Windows API in order to inject mouse events into the system.

8. Integrate into a complete standalone application which can use the laser as a mouse.

AGREED:

______________________ (Student) ______________________ (Supervisor)

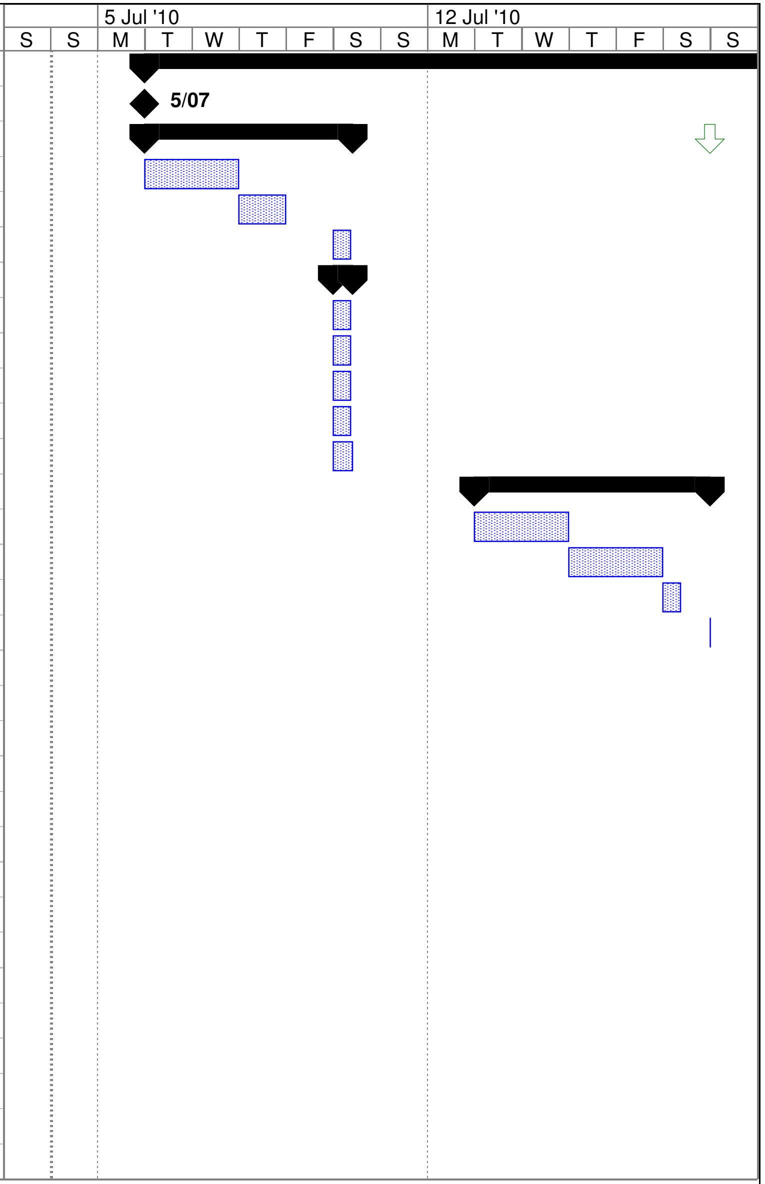
____/____/____ ____/____/____

Examiner/Co-examiner: ____

Appendix B

Project Timeline

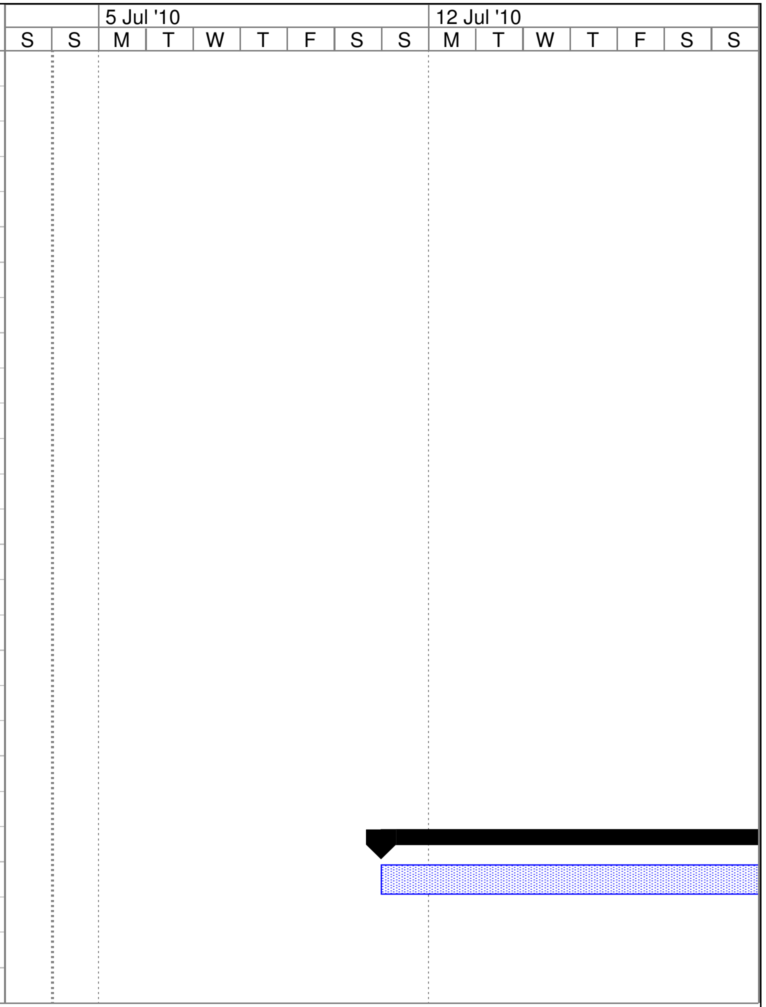
ID	ℹ	Task Name	Duration	Start	Finish
1		**Completing Project**	**246 hrs?**	**Mon 5/07/10**	**Thu 28/10/10**
2	▦	Implementation Start	0 hrs	Mon 5/07/10	Mon 5/07/10
3		**Setting up and preparation**	**7 hrs?**	**Mon 5/07/10**	**Sat 10/07/10**
4	▦	Find and Test Graphics Processing API	3 hrs?	Mon 5/07/10	Wed 7/07/10
5	▦	Find and Test Web Cam API	2 hrs?	Wed 7/07/10	Thu 8/07/10
6	▦	Find and Test System Mouse API	2 hrs?	Fri 9/07/10	Sat 10/07/10
7		**Preparing Equipment**	**2 hrs**	**Sat 10/07/10**	**Sat 10/07/10**
8	▦	Determining and Setting Up IDE	1 hr	Sat 10/07/10	Sat 10/07/10
9	▦	Preparing a Laser Pointer	1 hr	Sat 10/07/10	Sat 10/07/10
10	▦	Choosing and Purchasing a Webcam	1 hr	Sat 10/07/10	Sat 10/07/10
11	▦	Arranging Access to Equipment	1 hr	Sat 10/07/10	Sat 10/07/10
12	▦	Experimenting with Laser Photos	2 hrs	Sat 10/07/10	Sat 10/07/10
13		**Interfacing the camera**	**10 hrs?**	**Mon 12/07/10**	**Sat 17/07/10**
14	▦	System Design	2.13 hrs?	Mon 12/07/10	Wed 14/07/10
15	▦	Picture Request	2.13 hrs?	Wed 14/07/10	Fri 16/07/10
16	▦	Camera Settings	2.13 hrs?	Fri 16/07/10	Sat 17/07/10
17	▦	Finalising and Testing Code	2 hrs?	Sat 17/07/10	Sat 17/07/10
18		**Finding the position of the laser point**	**37.13 hrs?**	**Mon 19/07/10**	**Sat 7/08/10**
19	▦	Choosing a Source of Information	5 hrs	Mon 19/07/10	Fri 23/07/10
20	▦	Source Reverse Engineering and Testing	5 hrs	Sat 24/07/10	Sat 24/07/10
21	▦	Improvements	3 hrs	Sun 25/07/10	Sun 25/07/10
22	▦	Developing Original Algoritm and System	10 hrs	Mon 26/07/10	Sat 31/07/10
23	▦	Testing and Improving	4 hrs	Sun 1/08/10	Sun 1/08/10
24	▦	Performance Analysis	5 hrs	Mon 2/08/10	Fri 6/08/10
25	▦	Finalising Code	2.13 hrs?	Sat 7/08/10	Sat 7/08/10
26		**Calibrating**	**29 hrs?**	**Mon 9/08/10**	**Sun 22/08/10**
27	▦	Choosing a Source of Information	5 hrs	Mon 9/08/10	Fri 13/08/10
28	▦	Source Reverse Engineering and Testing	5 hrs	Fri 13/08/10	Sat 14/08/10
29	▦	Improvements	3 hrs	Sun 15/08/10	Sun 15/08/10
30	▦	Developing Original Algoritm and System	10 hrs	Mon 16/08/10	Sat 21/08/10
31	▦	Testing and Improving	4 hrs	Sun 22/08/10	Sun 22/08/10
32	▦	Performance Analysis	5 hrs	Sat 21/08/10	Sat 21/08/10



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▬
Split	┆┆┆┆┆┆┆	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

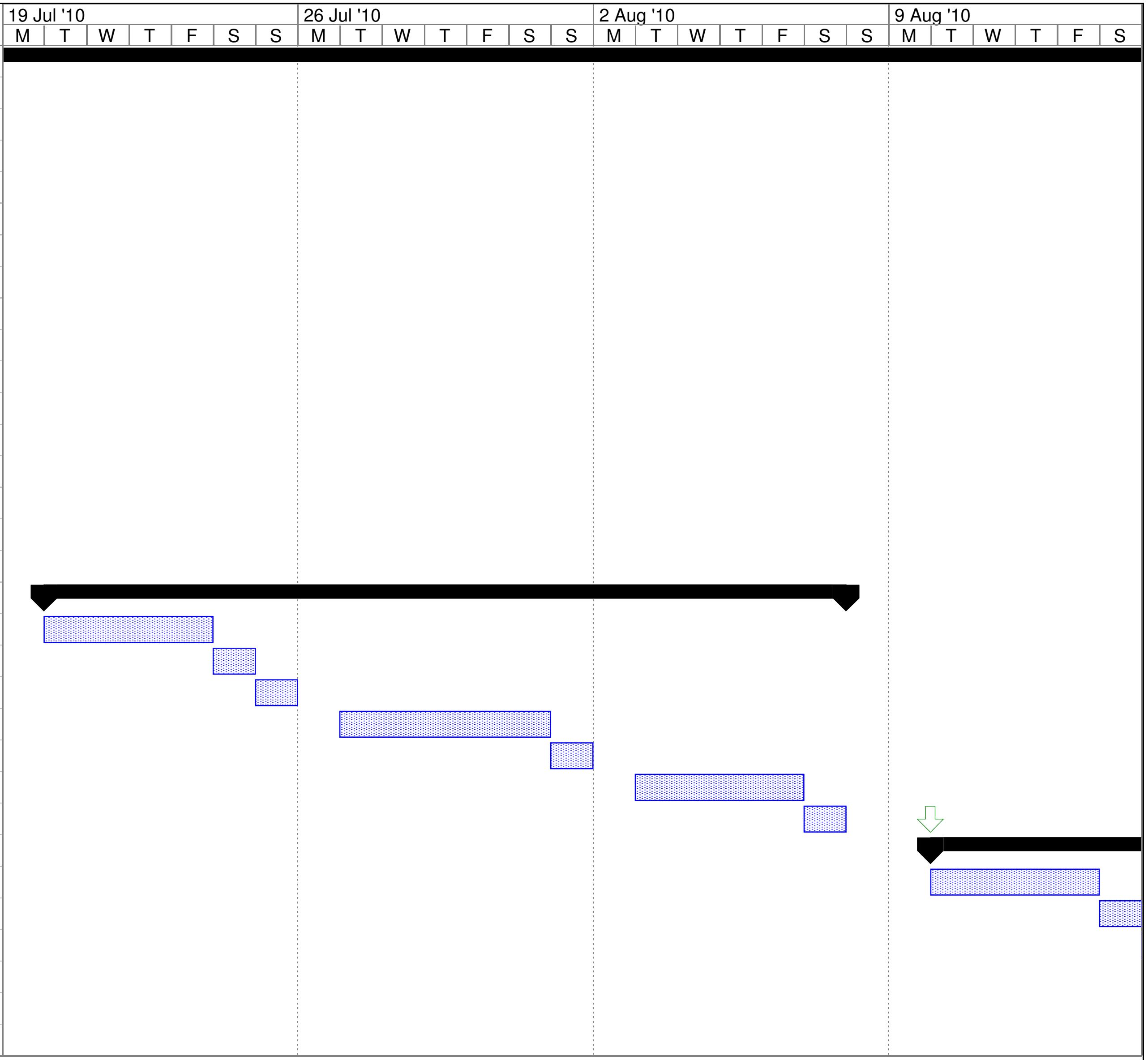
ID	ℹ	Task Name	Duration	Start	Finish
33	📅	Finalising Code	2.13 hrs?	Sat 21/08/10	Sat 21/08/10
34		**Circle Clicking**	**37.13 hrs?**	**Mon 23/08/10**	**Sat 11/09/10**
35	📅	Choosing a Source of Information	5 hrs	Mon 23/08/10	Fri 27/08/10
36	📅	John Leis source examination	5 hrs?	Mon 23/08/10	Fri 27/08/10
37	📅	Source Reverse Engineering and Testing	5 hrs	Fri 27/08/10	Sat 28/08/10
38	📅	Improvements	3 hrs	Sun 29/08/10	Sun 29/08/10
39	📅	Developing Original Algoritm and System	10 hrs	Mon 30/08/10	Sat 4/09/10
40	📅	Testing and Improving	4 hrs	Sun 5/09/10	Sun 5/09/10
41	📅	Performance Analysis	5 hrs	Mon 6/09/10	Fri 10/09/10
42	📅❗	Finalising Code	2.13 hrs?	Sat 11/09/10	Sat 11/09/10
43		**Automatic Calibration**	**30 hrs**	**Thu 9/09/10**	**Wed 22/09/10**
44	📅	Research some edge Detection and Related Algoritms	10 hrs	Thu 9/09/10	Sun 12/09/10
45	📅	Experiment with these on some Exdisting Infrastructure	10 hrs	Sun 12/09/10	Fri 17/09/10
46	📅	Develop the Algorithm	3 hrs	Sat 18/09/10	Sat 18/09/10
47	📅	Implement the Algorithm	5 hrs	Sun 19/09/10	Sun 19/09/10
48	📅	Test the Algorithm	3 hrs	Mon 20/09/10	Wed 22/09/10
49		**Slide Navigation**	**5 hrs**	**Tue 21/09/10**	**Sat 25/09/10**
50	📅	Develop Algoritm	3 hrs	Tue 21/09/10	Thu 23/09/10
51	📅	Perform Detection of Laser	1 hr	Thu 23/09/10	Thu 23/09/10
52	📅	Determine where the laser is	2 hrs	Fri 24/09/10	Sat 25/09/10
53	📅	Execute API Command for Slide Navigation	1 hr	Sat 25/09/10	Sat 25/09/10
54	📅	System Integration into a standalone package	30 hrs	Mon 27/09/10	Sun 10/10/10
55		**Dissertation Production**	**236 hrs?**	**Sun 11/07/10**	**Wed 27/10/10**
56	📅✓	Weekly work on Dissertation	65 hrs?	Sun 11/07/10	Sun 3/10/10
57	📅	Dissertation Production	40 hrs	Sun 3/10/10	Fri 22/10/10
58	📅	Collating and Publishing	16 hrs?	Sat 23/10/10	Wed 27/10/10
59	📅	Project Performance	0 days?	Thu 28/10/10	Thu 28/10/10



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▬
Split	⋯	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

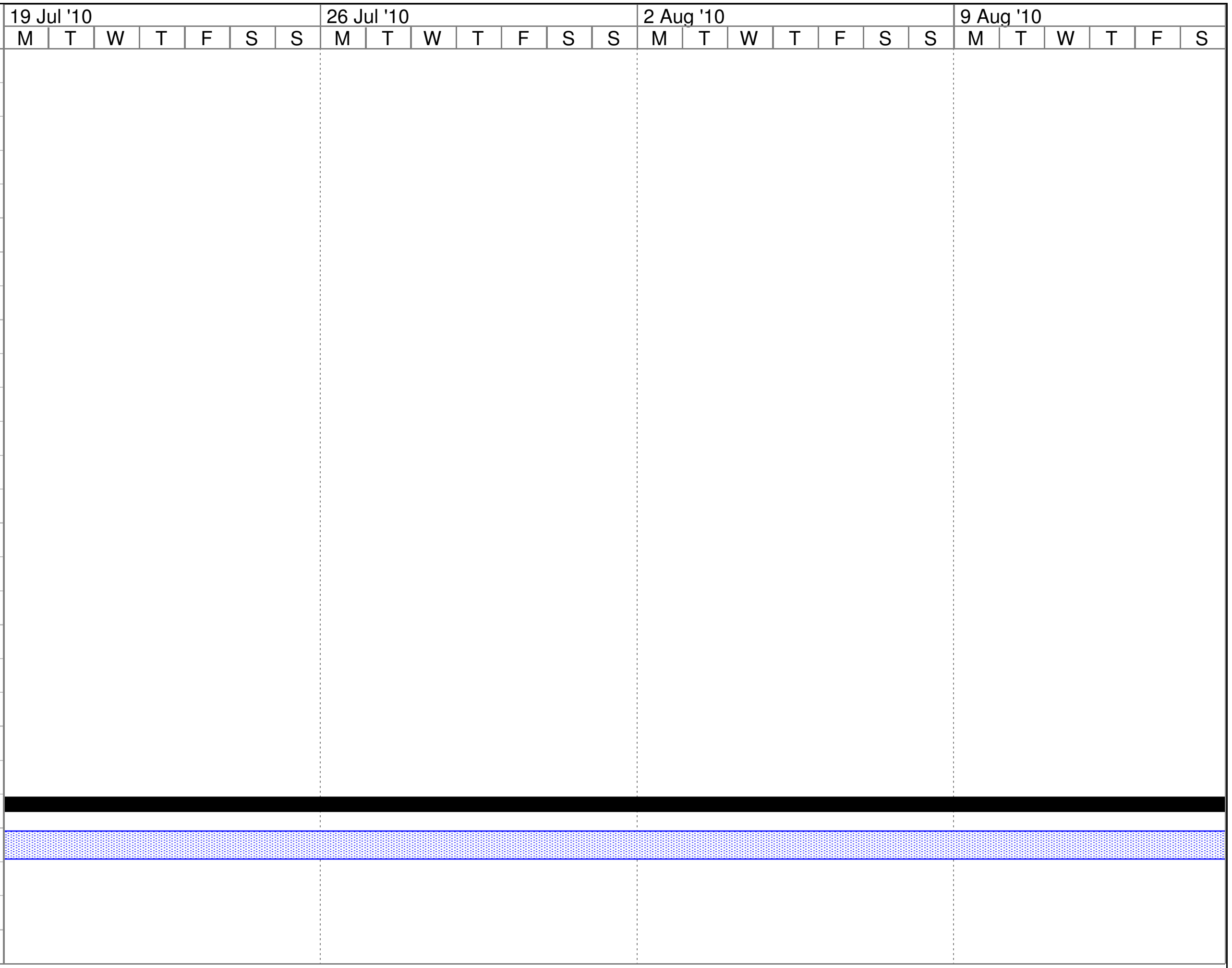
ID	ℹ	Task Name
1		**Completing Project**
2	▦	Implementation Start
3		**Setting up and preparation**
4	▦	Find and Test Graphics Processing API
5	▦	Find and Test Web Cam API
6	▦	Find and Test System Mouse API
7		**Preparing Equipment**
8	▦	Determining and Setting Up IDE
9	▦	Preparing a Laser Pointer
10	▦	Choosing and Purchasing a Webcam
11	▦	Arranging Access to Equipment
12	▦	Experimenting with Laser Photos
13		**Interfacing the camera**
14	▦	System Design
15	▦	Picture Request
16	▦	Camera Settings
17	▦	Finalising and Testing Code
18		**Finding the position of the laser point**
19	▦	Choosing a Source of Information
20	▦	Source Reverse Engineering and Testing
21	▦	Improvements
22	▦	Developing Original Algoritm and System
23	▦	Testing and Improving
24	▦	Performance Analysis
25	▦	Finalising Code
26		**Calibrating**
27	▦	Choosing a Source of Information
28	▦	Source Reverse Engineering and Testing
29	▦	Improvements
30	▦	Developing Original Algoritm and System
31	▦	Testing and Improving
32	▦	Performance Analysis



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▭
Split	┆┆┆┆┆┆┆┆	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

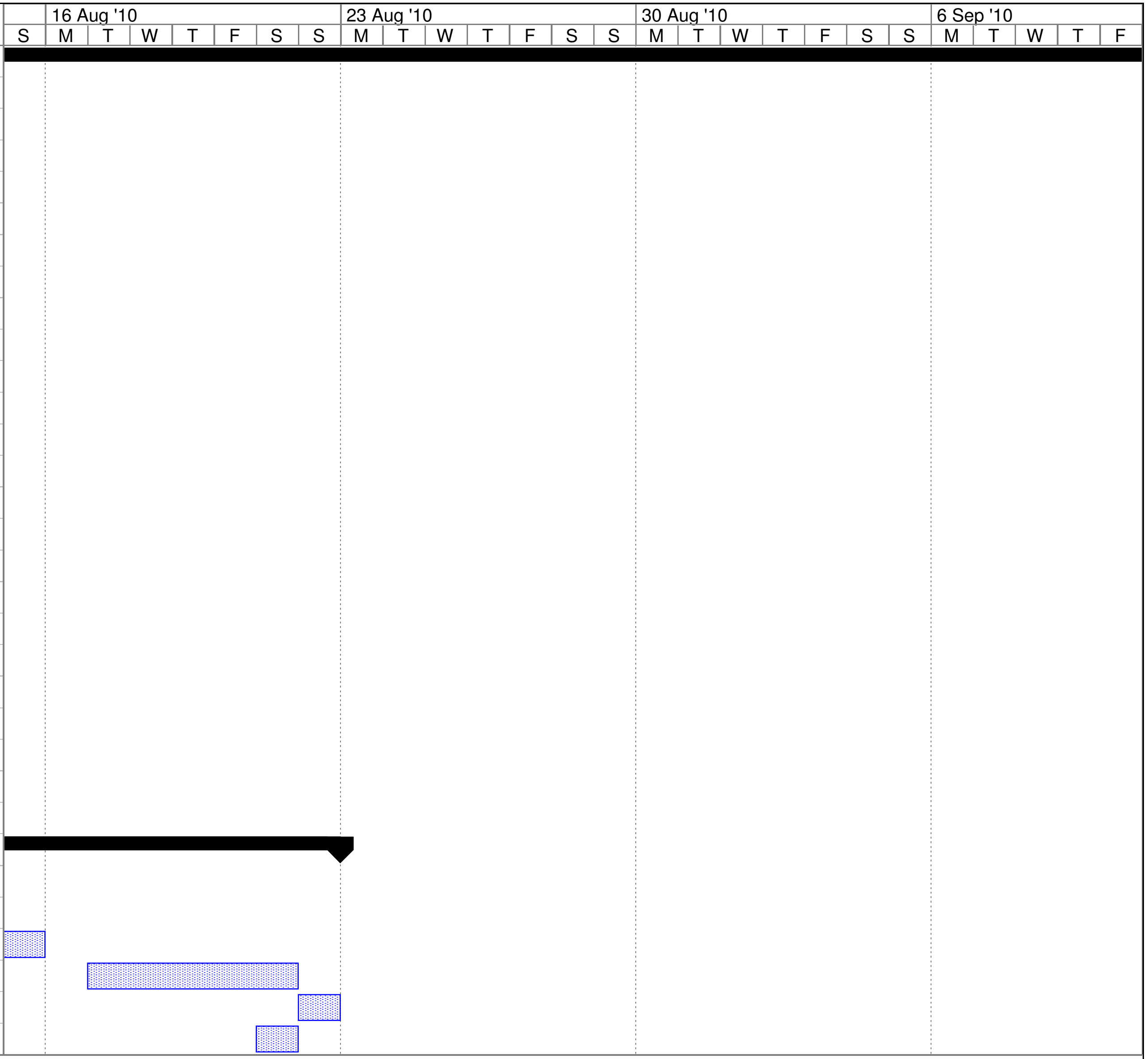
ID	ℹ	Task Name
33	▦	Finalising Code
34		**Circle Clicking**
35	▦	Choosing a Source of Information
36	▦	John Leis source examination
37	▦	Source Reverse Engineering and Testing
38	▦	Improvements
39	▦	Developing Original Algoritm and System
40	▦	Testing and Improving
41	▦	Performance Analysis
42	▦ ❗	Finalising Code
43		**Automatic Calibration**
44	▦	Research some edge Detection and Related Algoritms
45	▦	Experiment with these on some Exdisting Infrastructure
46	▦	Develop the Algorithm
47	▦	Implement the Algorithm
48	▦	Test the Algorithm
49		**Slide Navigation**
50	▦	Develop Algoritm
51	▦	Perform Detection of Laser
52	▦	Determine where the laser is
53	▦	Execute API Command for Slide Navigation
54	▦	System Integration into a standalone package
55		**Dissertation Production**
56	▦ ✓	Weekly work on Dissertation
57	▦	Dissertation Production
58	▦	Collating and Publishing
59	▦	Project Performance



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

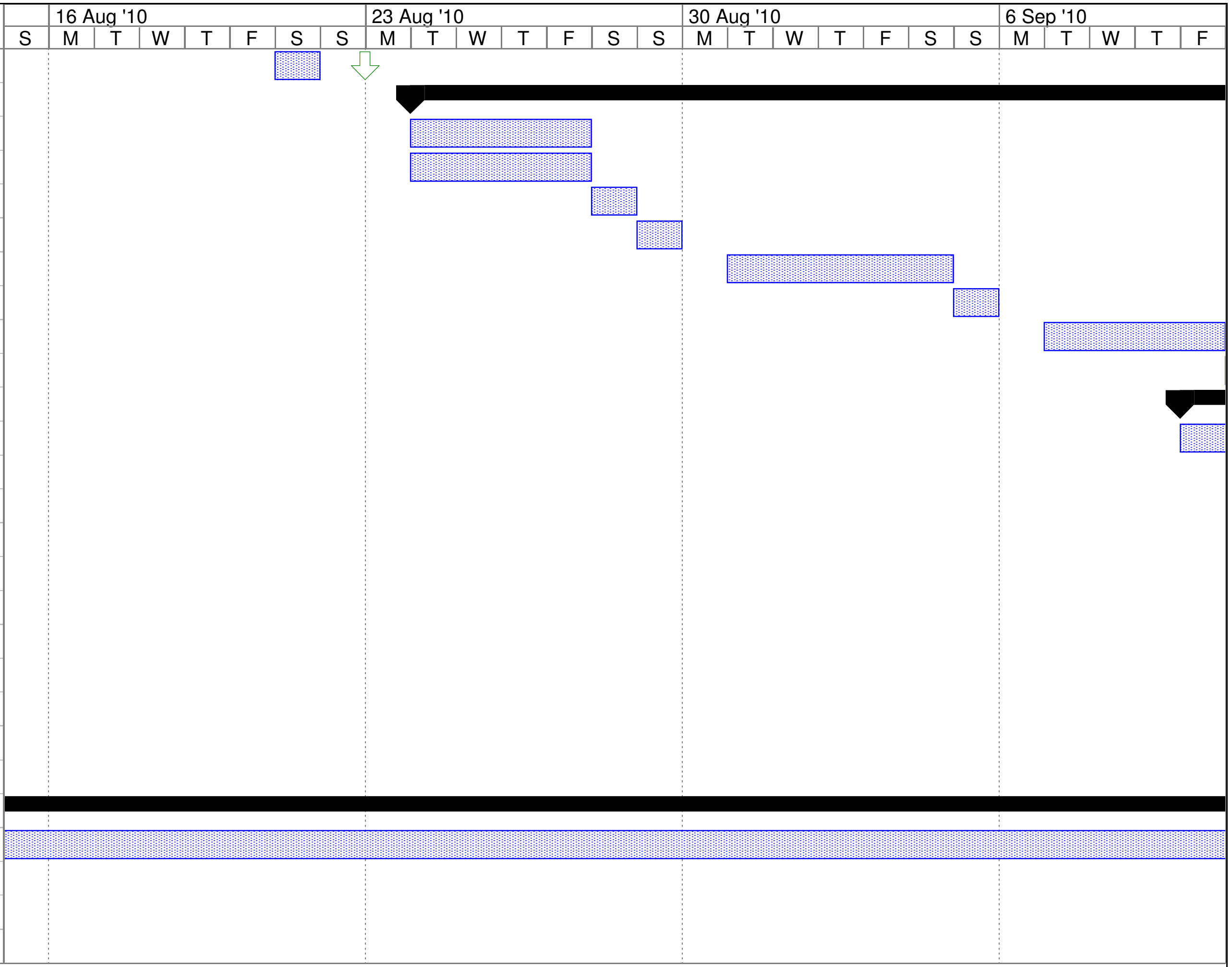
Task	▨	Milestone	◆	External Tasks	▬
Split	┆┆┆┆┆┆	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

ID	Task Name
1	**Completing Project**
2	Implementation Start
3	**Setting up and preparation**
4	Find and Test Graphics Processing API
5	Find and Test Web Cam API
6	Find and Test System Mouse API
7	**Preparing Equipment**
8	Determining and Setting Up IDE
9	Preparing a Laser Pointer
10	Choosing and Purchasing a Webcam
11	Arranging Access to Equipment
12	Experimenting with Laser Photos
13	**Interfacing the camera**
14	System Design
15	Picture Request
16	Camera Settings
17	Finalising and Testing Code
18	**Finding the position of the laser point**
19	Choosing a Source of Information
20	Source Reverse Engineering and Testing
21	Improvements
22	Developing Original Algoritm and System
23	Testing and Improving
24	Performance Analysis
25	Finalising Code
26	**Calibrating**
27	Choosing a Source of Information
28	Source Reverse Engineering and Testing
29	Improvements
30	Developing Original Algoritm and System
31	Testing and Improving
32	Performance Analysis



Task	▨	Milestone	◆	External Tasks	▬
Split	┅	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

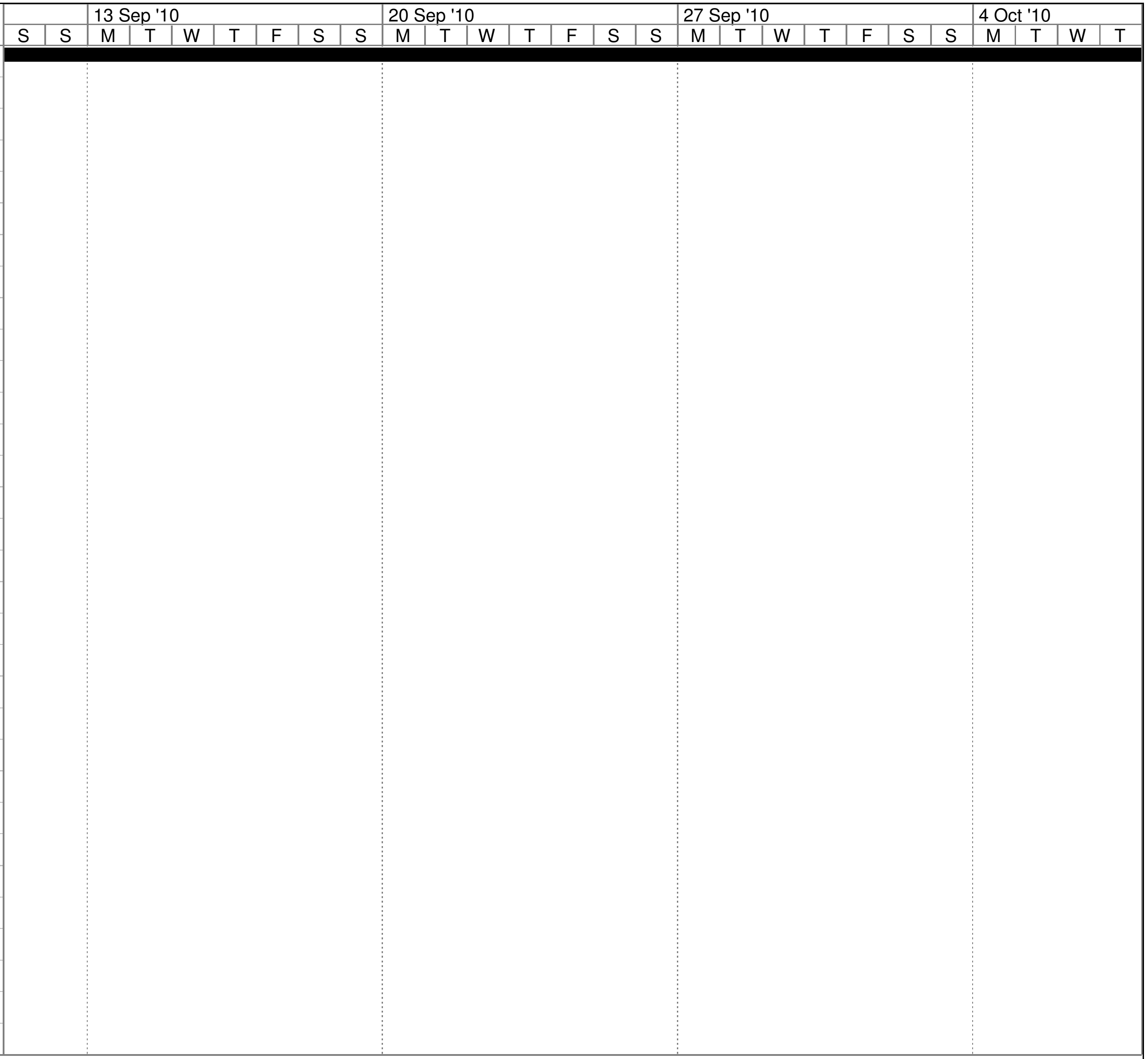
ID	ℹ	Task Name
33	📅	Finalising Code
34		**Circle Clicking**
35	📅	Choosing a Source of Information
36	📅	John Leis source examination
37	📅	Source Reverse Engineering and Testing
38	📅	Improvements
39	📅	Developing Original Algoritm and System
40	📅	Testing and Improving
41	📅	Performance Analysis
42	📅 ❗	Finalising Code
43		**Automatic Calibration**
44	📅	Research some edge Detection and Related Algoritms
45	📅	Experiment with these on some Exdisting Infrastructure
46	📅	Develop the Algorithm
47	📅	Implement the Algorithm
48	📅	Test the Algorithm
49		**Slide Navigation**
50	📅	Develop Algoritm
51	📅	Perform Detection of Laser
52	📅	Determine where the laser is
53	📅	Execute API Command for Slide Navigation
54	📅	System Integration into a standalone package
55		**Dissertation Production**
56	📅 📋	Weekly work on Dissertation
57	📅	Dissertation Production
58	📅	Collating and Publishing
59	📅	Project Performance



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▨
Split	┆	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

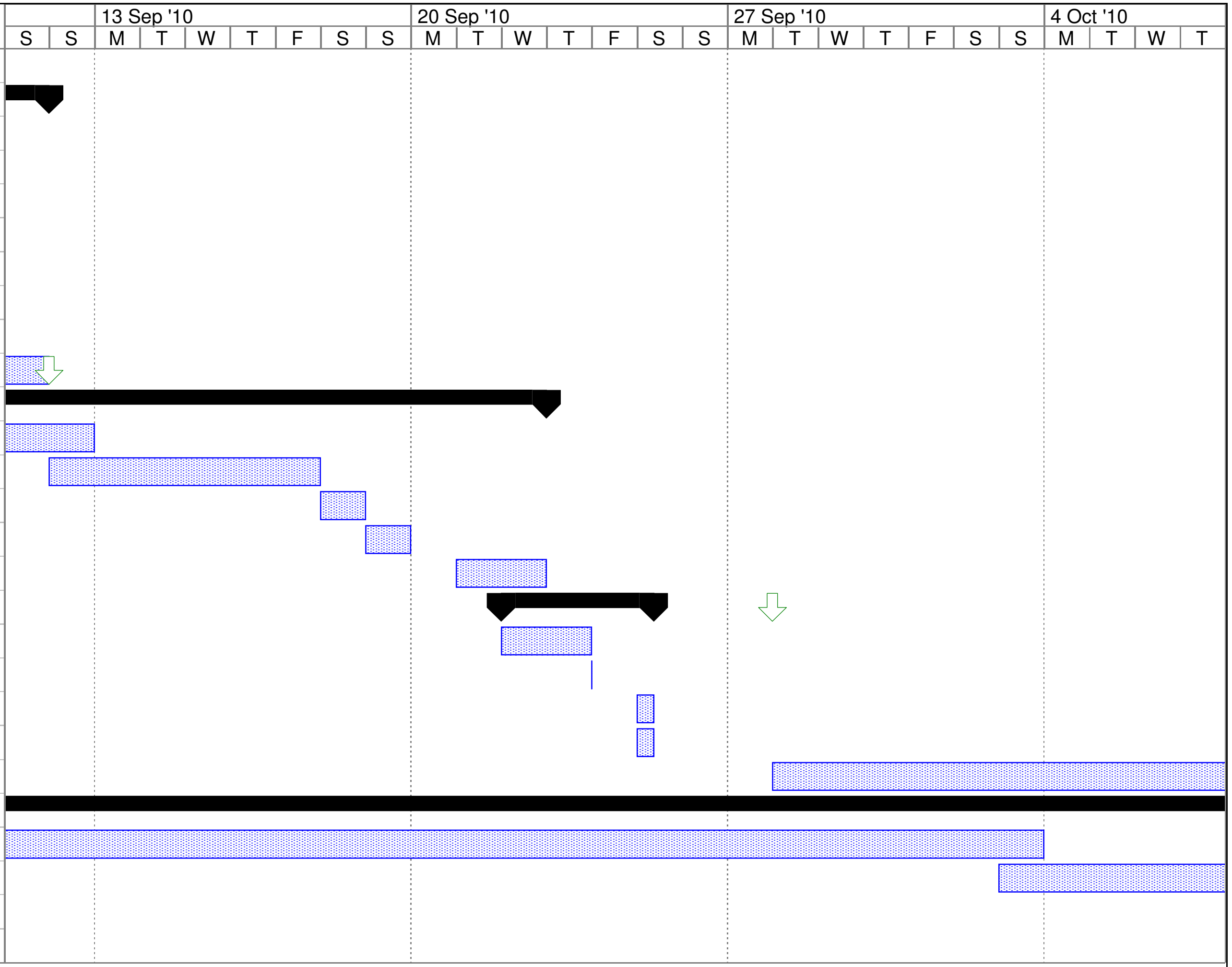
ID	ℹ	Task Name
1		**Completing Project**
2	▦	Implementation Start
3		**Setting up and preparation**
4	▦	Find and Test Graphics Processing API
5	▦	Find and Test Web Cam API
6	▦	Find and Test System Mouse API
7		**Preparing Equipment**
8	▦	Determining and Setting Up IDE
9	▦	Preparing a Laser Pointer
10	▦	Choosing and Purchasing a Webcam
11	▦	Arranging Access to Equipment
12	▦	Experimenting with Laser Photos
13		**Interfacing the camera**
14	▦	System Design
15	▦	Picture Request
16	▦	Camera Settings
17	▦	Finalising and Testing Code
18		**Finding the position of the laser point**
19	▦	Choosing a Source of Information
20	▦	Source Reverse Engineering and Testing
21	▦	Improvements
22	▦	Developing Original Algoritm and System
23	▦	Testing and Improving
24	▦	Performance Analysis
25	▦	Finalising Code
26		**Calibrating**
27	▦	Choosing a Source of Information
28	▦	Source Reverse Engineering and Testing
29	▦	Improvements
30	▦	Developing Original Algoritm and System
31	▦	Testing and Improving
32	▦	Performance Analysis



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▭
Split	⋯	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

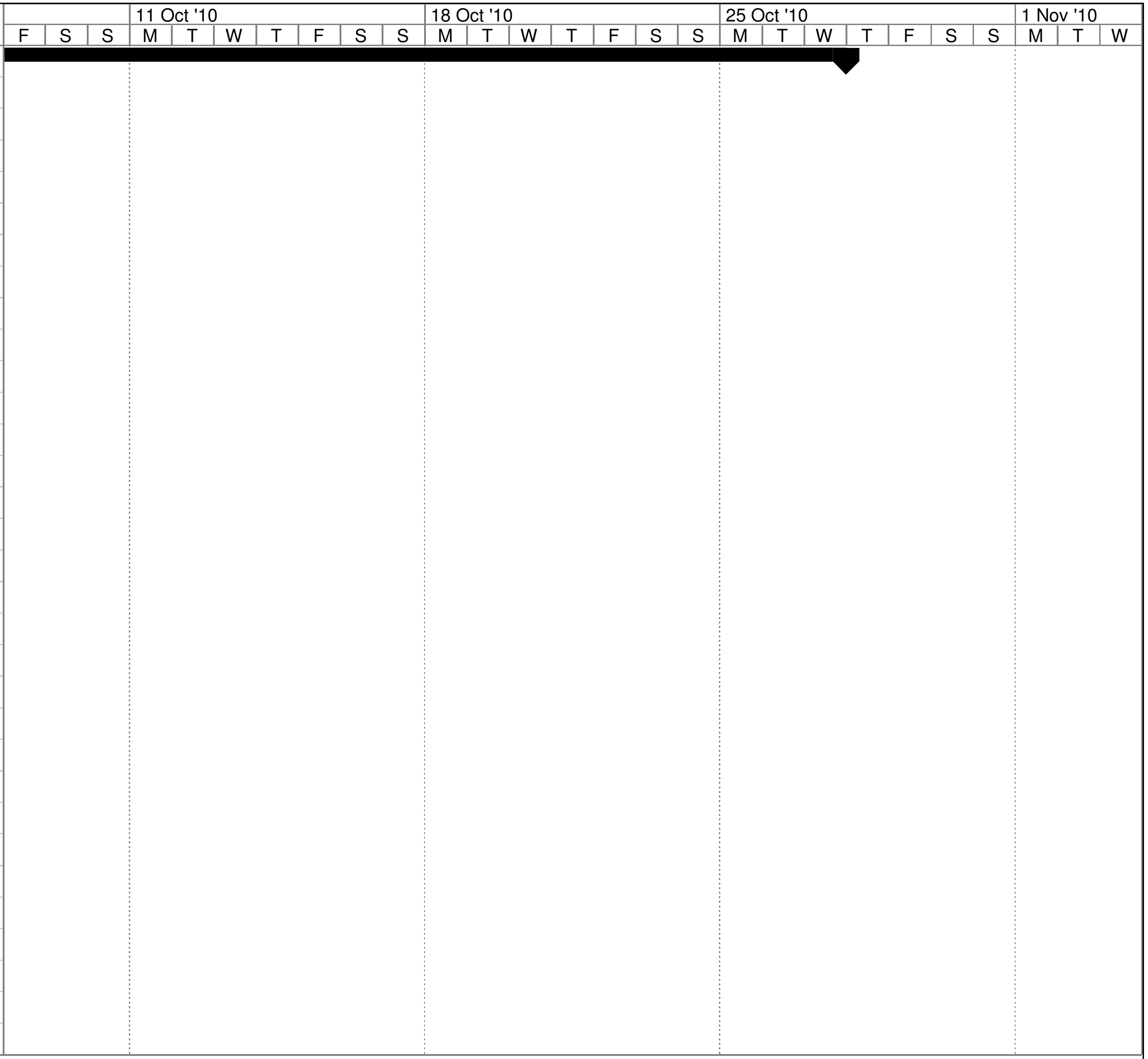
ID	ℹ	Task Name
33	▦	Finalising Code
34		**Circle Clicking**
35	▦	Choosing a Source of Information
36	▦	John Leis source examination
37	▦	Source Reverse Engineering and Testing
38	▦	Improvements
39	▦	Developing Original Algoritm and System
40	▦	Testing and Improving
41	▦	Performance Analysis
42	▦ ❗	Finalising Code
43		**Automatic Calibration**
44	▦	Research some edge Detection and Related Algoritms
45	▦	Experiment with these on some Exdisting Infrastructure
46	▦	Develop the Algorithm
47	▦	Implement the Algorithm
48	▦	Test the Algorithm
49		**Slide Navigation**
50	▦	Develop Algoritm
51	▦	Perform Detection of Laser
52	▦	Determine where the laser is
53	▦	Execute API Command for Slide Navigation
54	▦	System Integration into a standalone package
55		**Dissertation Production**
56	▦ ✓	Weekly work on Dissertation
57	▦	Dissertation Production
58	▦	Collating and Publishing
59	▦	Project Performance



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▭
Split	⋯	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

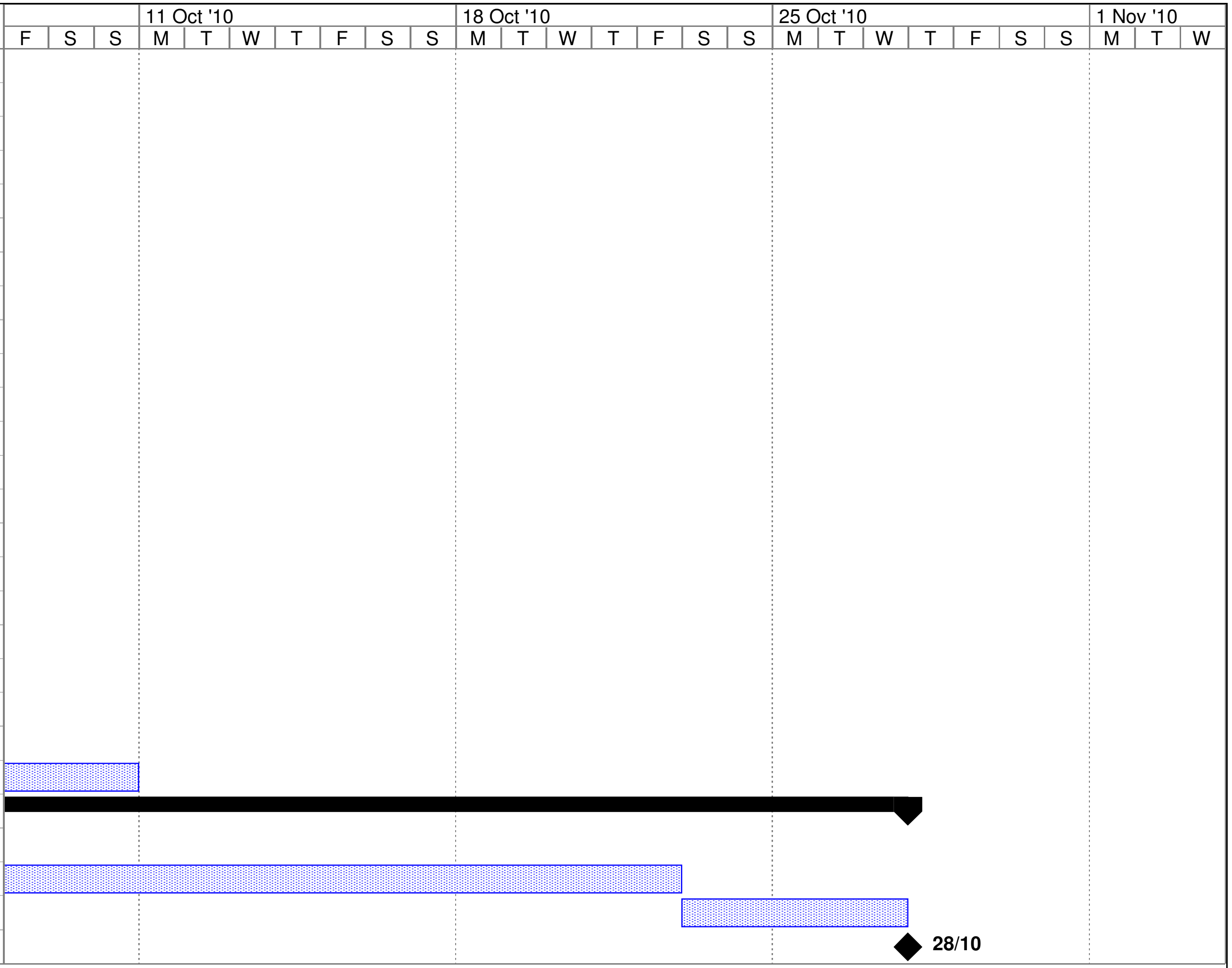
ID	ℹ	Task Name
1		**Completing Project**
2	▦	Implementation Start
3		**Setting up and preparation**
4	▦	Find and Test Graphics Processing API
5	▦	Find and Test Web Cam API
6	▦	Find and Test System Mouse API
7		**Preparing Equipment**
8	▦	Determining and Setting Up IDE
9	▦	Preparing a Laser Pointer
10	▦	Choosing and Purchasing a Webcam
11	▦	Arranging Access to Equipment
12	▦	Experimenting with Laser Photos
13		**Interfacing the camera**
14	▦	System Design
15	▦	Picture Request
16	▦	Camera Settings
17	▦	Finalising and Testing Code
18		**Finding the position of the laser point**
19	▦	Choosing a Source of Information
20	▦	Source Reverse Engineering and Testing
21	▦	Improvements
22	▦	Developing Original Algoritm and System
23	▦	Testing and Improving
24	▦	Performance Analysis
25	▦	Finalising Code
26		**Calibrating**
27	▦	Choosing a Source of Information
28	▦	Source Reverse Engineering and Testing
29	▦	Improvements
30	▦	Developing Original Algoritm and System
31	▦	Testing and Improving
32	▦	Performance Analysis



Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Task	▨	Milestone	◆	External Tasks	▭
Split	┆┆┆┆┆┆┆┆	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

ID	ℹ	Task Name
33	▦	Finalising Code
34		**Circle Clicking**
35	▦	Choosing a Source of Information
36	▦	John Leis source examination
37	▦	Source Reverse Engineering and Testing
38	▦	Improvements
39	▦	Developing Original Algoritm and System
40	▦	Testing and Improving
41	▦	Performance Analysis
42	▦ ◆	Finalising Code
43		**Automatic Calibration**
44	▦	Research some edge Detection and Related Algoritms
45	▦	Experiment with these on some Exdisting Infrastructure
46	▦	Develop the Algorithm
47	▦	Implement the Algorithm
48	▦	Test the Algorithm
49		**Slide Navigation**
50	▦	Develop Algoritm
51	▦	Perform Detection of Laser
52	▦	Determine where the laser is
53	▦	Execute API Command for Slide Navigation
54	▦	System Integration into a standalone package
55		**Dissertation Production**
56	▦ ✓	Weekly work on Dissertation
57	▦	Dissertation Production
58	▦	Collating and Publishing
59	▦	Project Performance



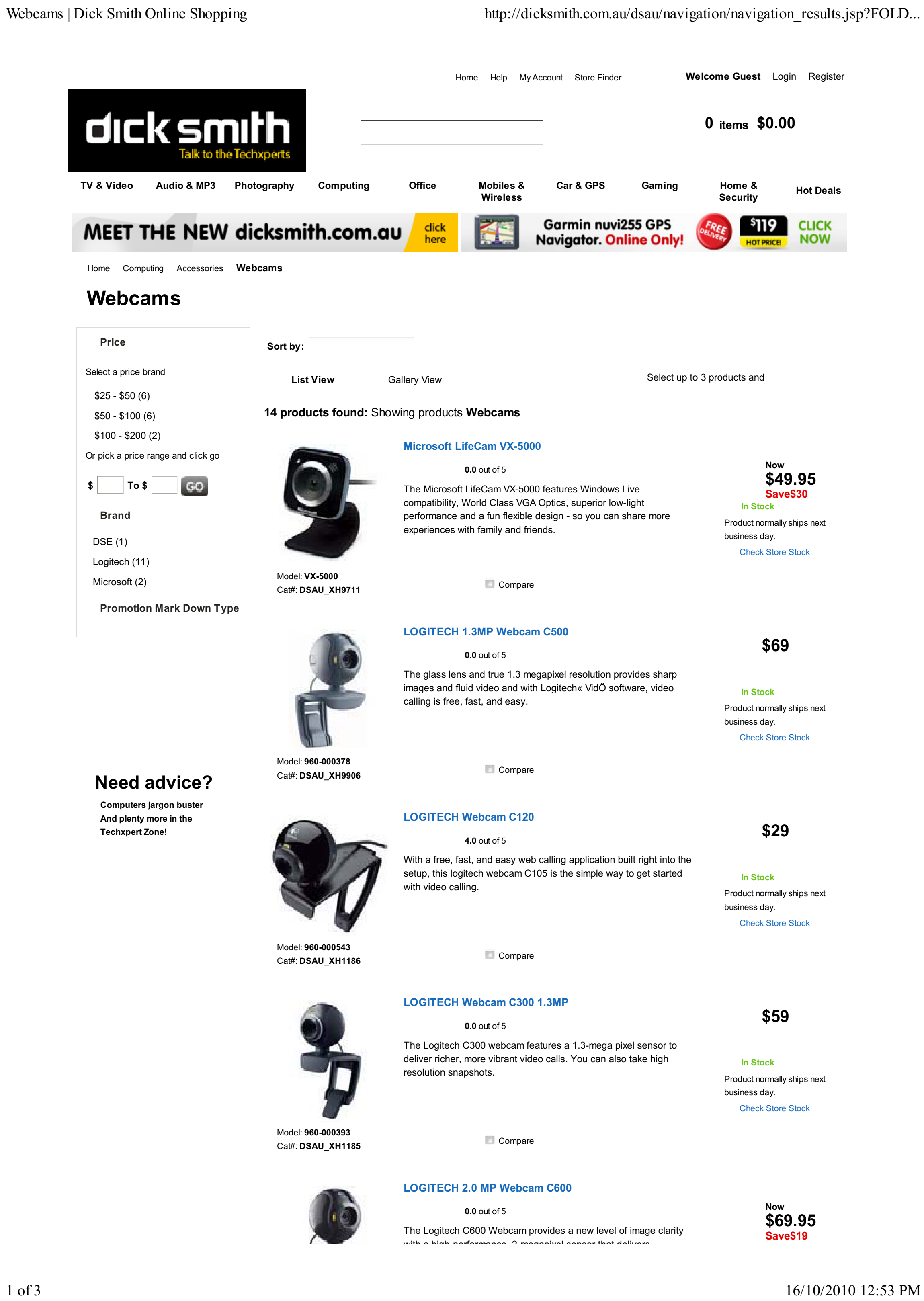
Task	▨	Milestone	◆	External Tasks	▬
Split	⋯	Summary	▬	External Milestone	◆
Progress	▬	Project Summary	▬	Deadline	⇩

Project: ProjectPlanMay4.mpp
Date: Sun 4/07/10

Appendix C

Data and Specifications

C.1 DSE Prices on Webcams



Webcams | Dick Smith Online Shopping http://dicksmith.com.au/dsau/navigation/navigation_results.jsp?FOLD...

Home Help My Account Store Finder Welcome Guest Login Register

dick smith Talk to the Techxperts

0 items $0.00

TV & Video | Audio & MP3 | Photography | Computing | Office | Mobiles & Wireless | Car & GPS | Gaming | Home & Security | Hot Deals

Home Computing Accessories **Webcams**

Webcams

Price

Select a price brand

$25 - $50 (6)

$50 - $100 (6)

$100 - $200 (2)

Or pick a price range and click go

$ ☐ To $ ☐ GO

Brand

DSE (1)

Logitech (11)

Microsoft (2)

Promotion Mark Down Type

Sort by:

List View Gallery View Select up to 3 products and

14 products found: Showing products **Webcams**

Microsoft LifeCam VX-5000

0.0 out of 5

The Microsoft LifeCam VX-5000 features Windows Live compatibility, World Class VGA Optics, superior low-light performance and a fun flexible design - so you can share more experiences with family and friends.

Model: **VX-5000**
Cat#: **DSAU_XH9711**

☐ Compare

Now **$49.95** Save$30

In Stock

Product normally ships next business day.

Check Store Stock

LOGITECH 1.3MP Webcam C500

0.0 out of 5

The glass lens and true 1.3 megapixel resolution provides sharp images and fluid video and with Logitech« VidÖ software, video calling is free, fast, and easy.

Model: **960-000378**
Cat#: **DSAU_XH9906**

☐ Compare

$69

In Stock

Product normally ships next business day.

Check Store Stock

Need advice?

Computers jargon buster
And plenty more in the Techxpert Zone!

LOGITECH Webcam C120

4.0 out of 5

With a free, fast, and easy web calling application built right into the setup, this logitech webcam C105 is the simple way to get started with video calling.

Model: **960-000543**
Cat#: **DSAU_XH1186**

☐ Compare

$29

In Stock

Product normally ships next business day.

Check Store Stock

LOGITECH Webcam C300 1.3MP

0.0 out of 5

The Logitech C300 webcam features a 1.3-mega pixel sensor to deliver richer, more vibrant video calls. You can also take high resolution snapshots.

Model: **960-000393**
Cat#: **DSAU_XH1185**

☐ Compare

$59

In Stock

Product normally ships next business day.

Check Store Stock

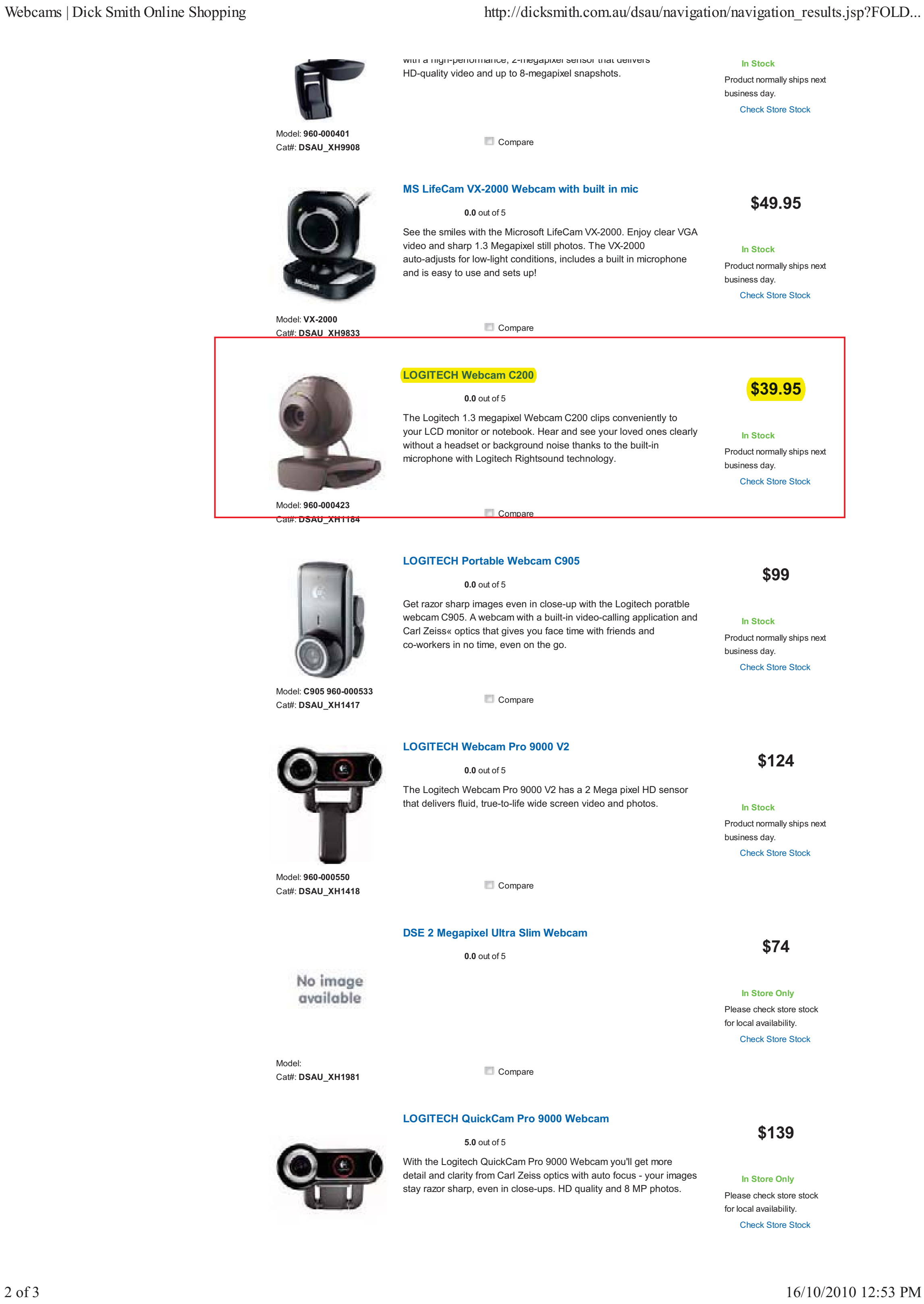
LOGITECH 2.0 MP Webcam C600

0.0 out of 5

The Logitech C600 Webcam provides a new level of image clarity

Now **$69.95** Save$19

1 of 3 16/10/2010 12:53 PM



Webcams | Dick Smith Online Shopping http://dicksmith.com.au/dsau/navigation/navigation_results.jsp?FOLD...

with a high-performance, 2-megapixel sensor that delivers HD-quality video and up to 8-megapixel snapshots.

In Stock

Product normally ships next business day.

Check Store Stock

Model: **960-000401**
Cat#: **DSAU_XH9908**

□ Compare

MS LifeCam VX-2000 Webcam with built in mic

0.0 out of 5

See the smiles with the Microsoft LifeCam VX-2000. Enjoy clear VGA video and sharp 1.3 Megapixel still photos. The VX-2000 auto-adjusts for low-light conditions, includes a built in microphone and is easy to use and sets up!

$49.95

In Stock

Product normally ships next business day.

Check Store Stock

Model: **VX-2000**
Cat#: **DSAU_XH9833**

□ Compare

LOGITECH Webcam C200

0.0 out of 5

The Logitech 1.3 megapixel Webcam C200 clips conveniently to your LCD monitor or notebook. Hear and see your loved ones clearly without a headset or background noise thanks to the built-in microphone with Logitech Rightsound technology.

$39.95

In Stock

Product normally ships next business day.

Check Store Stock

Model: **960-000423**
Cat#: **DSAU_XH1184**

□ Compare

LOGITECH Portable Webcam C905

0.0 out of 5

Get razor sharp images even in close-up with the Logitech poratble webcam C905. A webcam with a built-in video-calling application and Carl Zeiss« optics that gives you face time with friends and co-workers in no time, even on the go.

$99

In Stock

Product normally ships next business day.

Check Store Stock

Model: **C905 960-000533**
Cat#: **DSAU_XH1417**

□ Compare

LOGITECH Webcam Pro 9000 V2

0.0 out of 5

The Logitech Webcam Pro 9000 V2 has a 2 Mega pixel HD sensor that delivers fluid, true-to-life wide screen video and photos.

$124

In Stock

Product normally ships next business day.

Check Store Stock

Model: **960-000550**
Cat#: **DSAU_XH1418**

□ Compare

DSE 2 Megapixel Ultra Slim Webcam

0.0 out of 5

No image available

$74

In Store Only

Please check store stock for local availability.

Check Store Stock

Model:
Cat#: **DSAU_XH1981**

□ Compare

LOGITECH QuickCam Pro 9000 Webcam

5.0 out of 5

With the Logitech QuickCam Pro 9000 Webcam you'll get more detail and clarity from Carl Zeiss optics with auto focus - your images stay razor sharp, even in close-ups. HD quality and 8 MP photos.

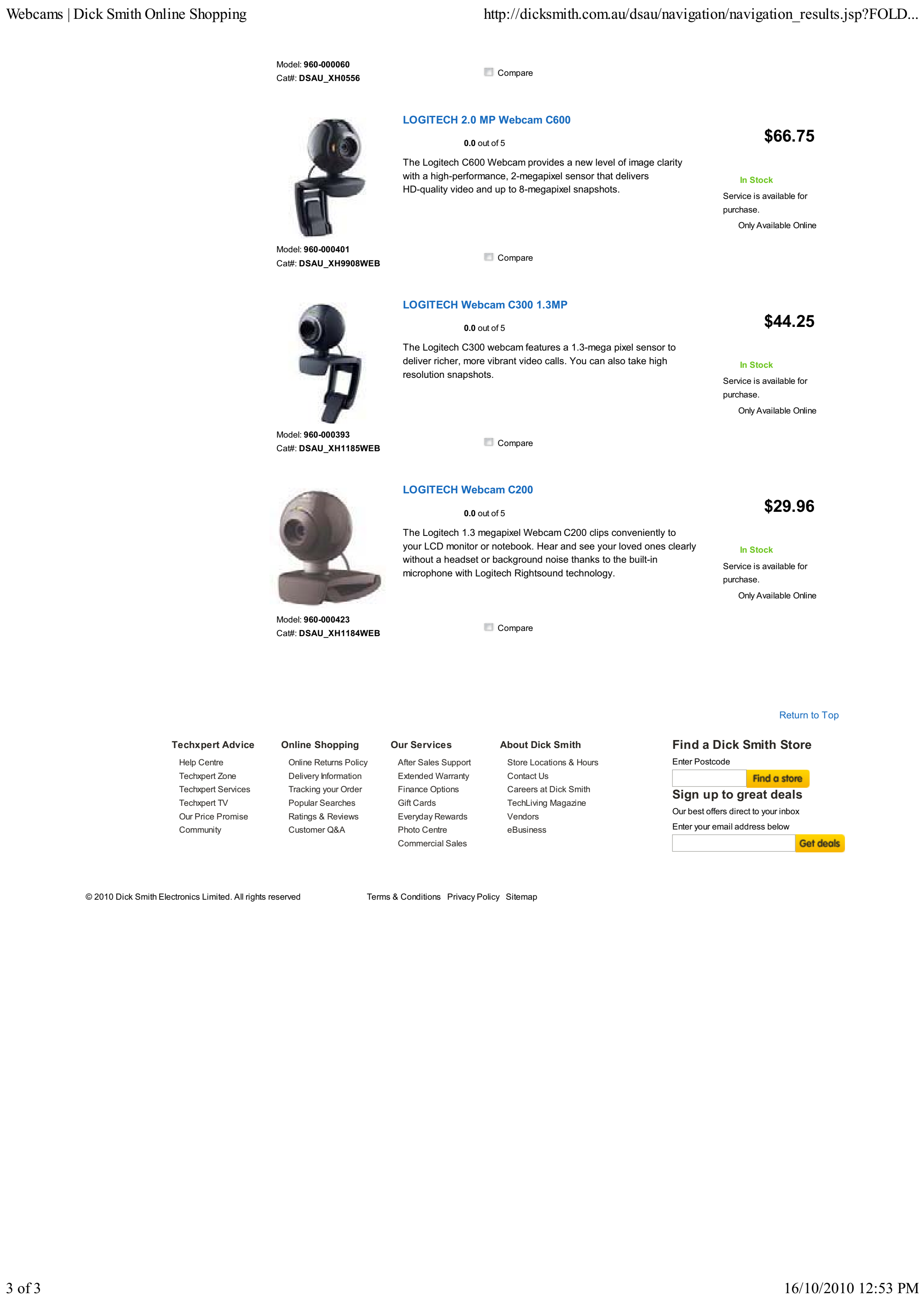
$139

In Store Only

Please check store stock for local availability.

Check Store Stock

2 of 3 16/10/2010 12:53 PM



Webcams | Dick Smith Online Shopping http://dicksmith.com.au/dsau/navigation/navigation_results.jsp?FOLD...

Model: **960-000060**
Cat#: **DSAU_XH0556**

□ Compare

LOGITECH 2.0 MP Webcam C600

0.0 out of 5

The Logitech C600 Webcam provides a new level of image clarity with a high-performance, 2-megapixel sensor that delivers HD-quality video and up to 8-megapixel snapshots.

$66.75

In Stock

Service is available for purchase.

Only Available Online

Model: **960-000401**
Cat#: **DSAU_XH9908WEB**

□ Compare

LOGITECH Webcam C300 1.3MP

0.0 out of 5

The Logitech C300 webcam features a 1.3-mega pixel sensor to deliver richer, more vibrant video calls. You can also take high resolution snapshots.

$44.25

In Stock

Service is available for purchase.

Only Available Online

Model: **960-000393**
Cat#: **DSAU_XH1185WEB**

□ Compare

LOGITECH Webcam C200

0.0 out of 5

The Logitech 1.3 megapixel Webcam C200 clips conveniently to your LCD monitor or notebook. Hear and see your loved ones clearly without a headset or background noise thanks to the built-in microphone with Logitech Rightsound technology.

$29.96

In Stock

Service is available for purchase.

Only Available Online

Model: **960-000423**
Cat#: **DSAU_XH1184WEB**

□ Compare

Return to Top

Techxpert Advice
Help Centre
Techxpert Zone
Techxpert Services
Techxpert TV
Our Price Promise
Community

Online Shopping
Online Returns Policy
Delivery Information
Tracking your Order
Popular Searches
Ratings & Reviews
Customer Q&A

Our Services
After Sales Support
Extended Warranty
Finance Options
Gift Cards
Everyday Rewards
Photo Centre
Commercial Sales

About Dick Smith
Store Locations & Hours
Contact Us
Careers at Dick Smith
TechLiving Magazine
Vendors
eBusiness

Find a Dick Smith Store
Enter Postcode
Find a store

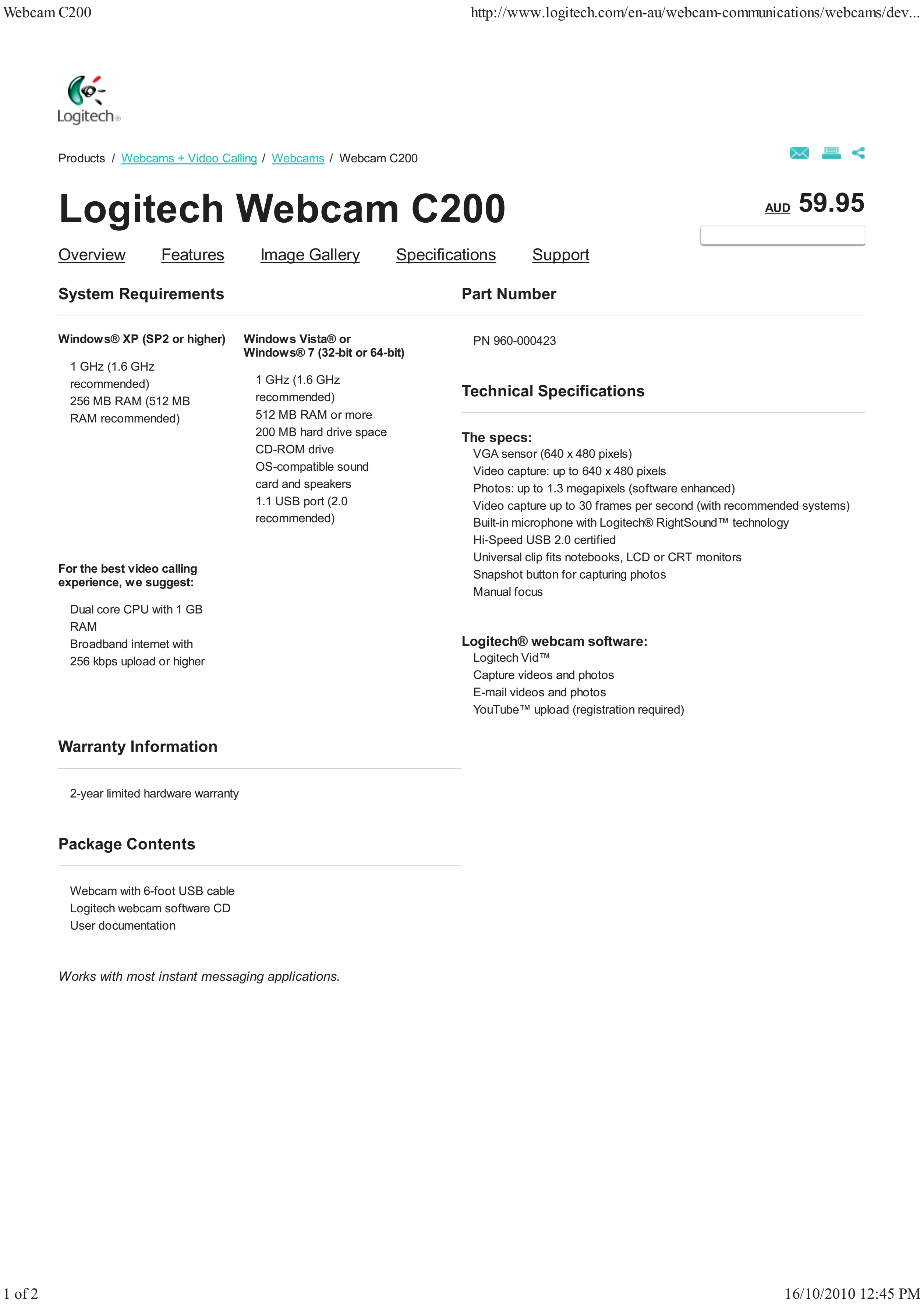
Sign up to great deals
Our best offers direct to your inbox
Enter your email address below
Get deals

© 2010 Dick Smith Electronics Limited. All rights reserved

Terms & Conditions Privacy Policy Sitemap

3 of 3 16/10/2010 12:53 PM

C.2 Logitech Webcam C200 Specifications



Webcam C200 http://www.logitech.com/en-au/webcam-communications/webcams/dev...

Logitech

Products / Webcams + Video Calling / Webcams / Webcam C200

Logitech Webcam C200

AUD 59.95

Overview Features Image Gallery Specifications Support

System Requirements

Windows® XP (SP2 or higher)

- 1 GHz (1.6 GHz recommended)
- 256 MB RAM (512 MB RAM recommended)

Windows Vista® or Windows® 7 (32-bit or 64-bit)

- 1 GHz (1.6 GHz recommended)
- 512 MB RAM or more
- 200 MB hard drive space
- CD-ROM drive
- OS-compatible sound card and speakers
- 1.1 USB port (2.0 recommended)

For the best video calling experience, we suggest:

- Dual core CPU with 1 GB RAM
- Broadband internet with 256 kbps upload or higher

Warranty Information

2-year limited hardware warranty

Package Contents

- Webcam with 6-foot USB cable
- Logitech webcam software CD
- User documentation

Works with most instant messaging applications.

Part Number

PN 960-000423

Technical Specifications

The specs:

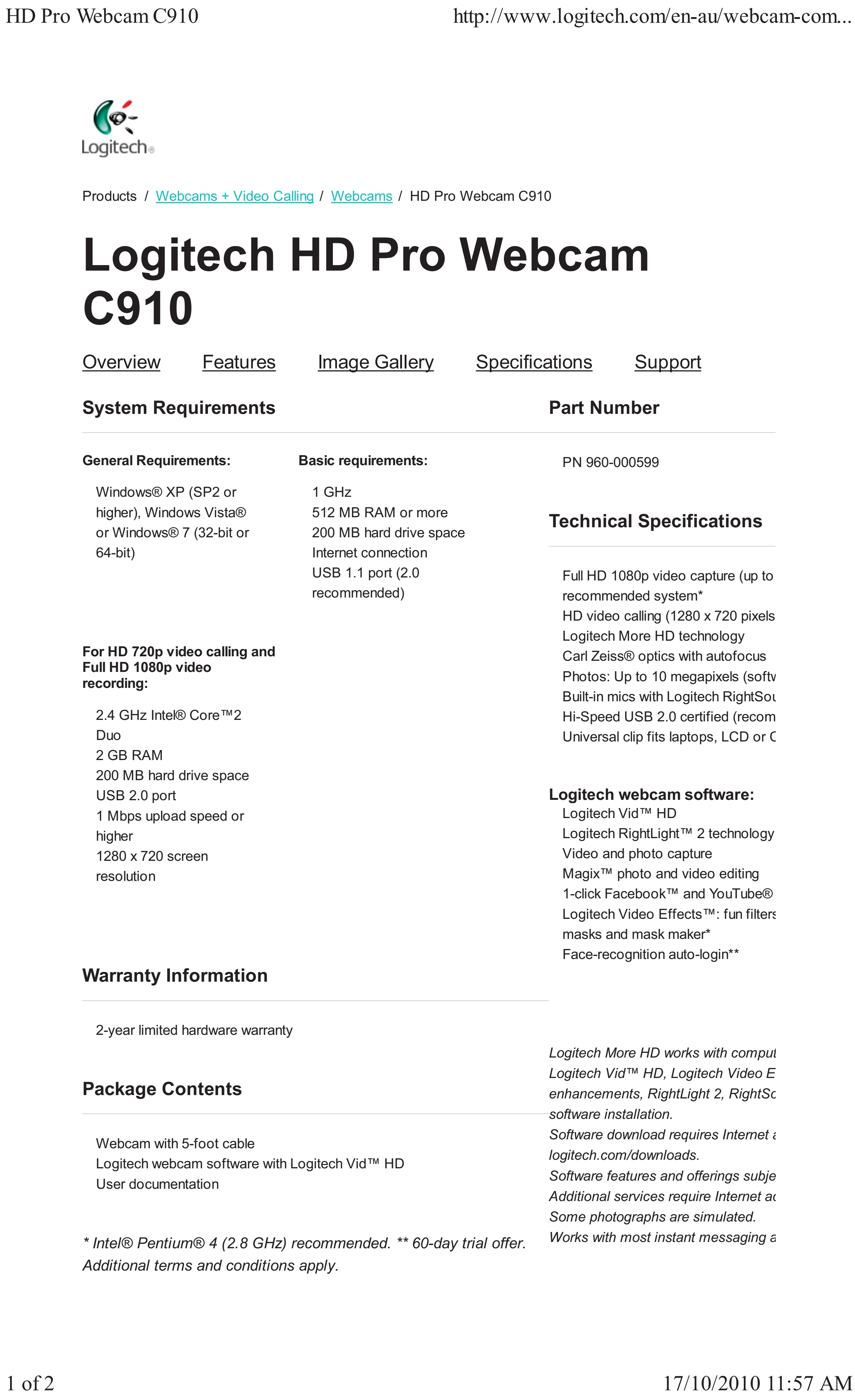
- VGA sensor (640 x 480 pixels)
- Video capture: up to 640 x 480 pixels
- Photos: up to 1.3 megapixels (software enhanced)
- Video capture up to 30 frames per second (with recommended systems)
- Built-in microphone with Logitech® RightSound™ technology
- Hi-Speed USB 2.0 certified
- Universal clip fits notebooks, LCD or CRT monitors
- Snapshot button for capturing photos
- Manual focus

Logitech® webcam software:

- Logitech Vid™
- Capture videos and photos
- E-mail videos and photos
- YouTube™ upload (registration required)

1 of 2 16/10/2010 12:45 PM

C.3 Logitech Webcam C910 Specifications



HD Pro Webcam C910 http://www.logitech.com/en-au/webcam-com...

Logitech

Products / Webcams + Video Calling / Webcams / HD Pro Webcam C910

Logitech HD Pro Webcam C910

Overview Features Image Gallery Specifications Support

System Requirements

General Requirements:

Windows® XP (SP2 or higher), Windows Vista® or Windows® 7 (32-bit or 64-bit)

Basic requirements:

1 GHz
512 MB RAM or more
200 MB hard drive space
Internet connection
USB 1.1 port (2.0 recommended)

For HD 720p video calling and Full HD 1080p video recording:

2.4 GHz Intel® Core™2 Duo
2 GB RAM
200 MB hard drive space
USB 2.0 port
1 Mbps upload speed or higher
1280 x 720 screen resolution

Warranty Information

2-year limited hardware warranty

Package Contents

Webcam with 5-foot cable
Logitech webcam software with Logitech Vid™ HD
User documentation

** Intel® Pentium® 4 (2.8 GHz) recommended. ** 60-day trial offer. Additional terms and conditions apply.*

Part Number

PN 960-000599

Technical Specifications

Full HD 1080p video capture (up to
recommended system*
HD video calling (1280 x 720 pixels
Logitech More HD technology
Carl Zeiss® optics with autofocus
Photos: Up to 10 megapixels (softv
Built-in mics with Logitech RightSou
Hi-Speed USB 2.0 certified (recom
Universal clip fits laptops, LCD or C

Logitech webcam software:

Logitech Vid™ HD
Logitech RightLight™ 2 technology
Video and photo capture
Magix™ photo and video editing
1-click Facebook™ and YouTube®
Logitech Video Effects™: fun filters
masks and mask maker*
Face-recognition auto-login**

Logitech More HD works with comput
Logitech Vid™ HD, Logitech Video E
enhancements, RightLight 2, RightSc
software installation.
Software download requires Internet a
logitech.com/downloads.
Software features and offerings subje
Additional services require Internet ac
Some photographs are simulated.
Works with most instant messaging a

1 of 2 17/10/2010 11:57 AM

University of Southern Queensland

Faculty of Engineering & Surveying

Development of a Laser Pointer Mouse
(Volume 2)

A dissertation submitted by

Mr Anthony Gibbons

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Software Engineering)

Submitted: October, 2010

Appendix D

Source Code Files

D.1 Webcam.c Source

```c
/* WebCam.c
 *
 * Simple webcam capture using COM
   - displays full capture from webcam
   - displays on second window, optionally processed image

 * adapted to C from
   http://secure.codeproject.com/KB/audio-video/DXCapture.aspx
 *
 * Note: may need to create uuid.lib first. To do this
    cd    C:\lcc\bin
    buildguid
 *
 */


//--------------------------------------------------------------------
#include <windows.h>
//#include <objbase.h>
#include <shlobj.h>
//--------------------------------------------------------------------

//-------------------------------------------------------------
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//-------------------------------------------------------------

//-------------------------------------------------------------
#include "WebCamMin.h"
//-------------------------------------------------------------

//-------------------------------------------------------------
// in WebCamLib.c
extern int      InitWebCamCapture(HWND hWnd, int iDeviceID, int *pWidth, int *pHeight);
extern int      GrabWebCamFrame(unsigned char *pFrameOut, unsigned long FrameBufferLen, int *pWidth, int *pHeight);
extern int      StopWebCamCapture(void);
extern int      PauseWebCamCapture(void);
extern int      ResumeWebCamCapture(void);
extern int      CloseWebCamCapture(void);
//--------------------------------------------------------------------

//-------------------------------------------------------------
//          For debugging purposes
int     laserPosToFile(void);
```

```
char     *createFileName(void);
int      timeToFile(void);
//-------------------------------------------------------------


void     resetAllLaserData(void);

//-------------------------------------------------------------
int      ProcessFrame(unsigned char *pFrameIn, unsigned char *pFrameOut, int Width, int Height);
//-------------------------------------------------------------


//-------------------------------------------------------------
int      detectLaser(int pCurY, int pCurX, long *pLastY, int pIntensity);
char * calcDetectionRate(void);
//-------------------------------------------------------------





// Flag used in  int detectLaser to limit messages displayed to one
int gblMessageStatus;


//-------------------------------------------------------------
// buttons
#define IDB_SHOWBUTTON      100
#define IDB_STOPBUTTON      101
#define IDB_PAUSEBUTTON     102
#define IDB_RESUMEBUTTON    103
#define IDB_EXITBUTTON      110
//-------------------------------------------------------------

//-------------------------------------------------------------
#define IDE_IMAGEDIMS       200
#define IDE_NUMFRAMES       201
#define IDE_RAMFREE         202

#define IDE_MAXVAL          203
#define IDE_MAXRUNLEN       204
//-------------------------------------------------------------

//-------------------------------------------------------------
#define IDW_CAPWIN          1000
//-------------------------------------------------------------
```

```

//---------------------------------------------------------------
#define ID_TIMER        350 //350

// frame rate in ms
#define FRAMERATE       400 //100
//---------------------------------------------------------------


//---------------------------------------------------------------
#define WEBCAM_X    150
#define WEBCAM_Y    20

#define DEFAULT_CANVAS_WIDTH    200
#define DEFAULT_CANVAS_HEIGHT   200
//---------------------------------------------------------------

//---------------------------------------------------------------
#define MAXIMAGE_WIDTH    600 //400
#define MAXIMAGE_HEIGHT   600 //400
//---------------------------------------------------------------

//---------------------------------------------------------------
#define PIXELDIFF_THRESHOLD     10
//---------------------------------------------------------------

//---------------------------------------------------------------
#define     KBYTES  1024
#define     MBYTES  (1024*1024)
#define     GBYTES  (1024*1024*1024)
//---------------------------------------------------------------


#define NOTSET                      -1
#define LASER_DETECTION_THRESHOLD   250
#define GAP_AMOUNT                  1


//---------------------------------------------------------------
char szAppName[] = "WebCamApp";
//---------------------------------------------------------------

//---------------------------------------------------------------
int  WebCamRunning = 0;
int  WebCamPaused = 0;
//---------------------------------------------------------------


```

```
//-----------------------------------------------------------
HWND    CreateMainWindow(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nWinMode);
HWND    CreateCaptureWindow(HWND hParentWnd);
HWND    CreateCanvasWindow(HWND hParentWnd);
//-----------------------------------------------------------

//-----------------------------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndProcCaptureWindow(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK WndProcCanvasWindow(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam);
//-----------------------------------------------------------

//-----------------------------------------------------------
static long nFrames = 0L;
//-----------------------------------------------------------

//-----------------------------------------------------------
HWND        hMainWnd;
HWND        hCapWnd;
HWND        hCanvasWnd;

HWND        hTextImageDims;
HWND        hTextNumFrames;
HWND        hTextRAMFree;
HWND        hTextMaxVal, hTextMaxRunLen;

HINSTANCE   hInstSave;
//-----------------------------------------------------------

//-----------------------------------------------------------
static int WebcamImageWidth = DEFAULT_CANVAS_WIDTH;
static int WebcamImageHeight = DEFAULT_CANVAS_HEIGHT;
//-----------------------------------------------------------


//-----------------------------------------------------------
unsigned long ImageBufferLen;
static BYTE *ImageBuffer = (BYTE *)NULL;
static BYTE *ImageBuffer1 = (BYTE *)NULL;
//-----------------------------------------------------------



// Laser Detection global Variables
```

```
-------------------------------------------------------------------------------------------------------------------------

int lastX = -1;             // The last X coordinate
int lastXmin = 0;           // The last min value of Y coordinate
int yMin = -1;              // The smallest value of the y coordinate detecteds
int xMin = -1;              // The smallest value of the x coordinate detected
int yMax = -1;              // The largest value of the y coordinate detected
int xMax = -1;              // The largest value of the x coordinate detected
int pixelCount = 0;         // The amount of pixels detected so far
int gapCount = 0;           // The amount of pixel gaps found in the laser point - too many of this gap means that it cannot be the laser pointer
time_t startTime;           // The time  that the detection starts
char * currentFileName = "";        // The name of the file the laser detection outputs go into
int countStarted = 0;           // flag to tell whether the counter has started or not


// ------------------------------------------------------------------------------------------------------------------------------------------------------





//---------------------------------------------------------------
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nWinMode)
{
    MSG     msg;

    //---------------------------------------------------------
    // required for manifest.xml
    InitCommonControls();
    //---------------------------------------------------------

    //--------------------------------------------------------------------
    if( CoInitialize(NULL) != S_OK )
    {
        printf("CoInitialize() failed\n");
        return 0;
    }
    //--------------------------------------------------------------------

    //--------------------------------------------------------------------
    WebCamRunning = 0;
    WebCamPaused = 0;
    //-----------------------------------------------------------------

    //------------------------------------------------------
    hMainWnd = CreateMainWindow(hInstance, hPrevInstance,
lpszCmdLine, nWinMode);
    if( ! hMainWnd )
        return 0;
    //------------------------------------------------------

    //----------------------------------------------------------
    hCapWnd = CreateCaptureWindow(hMainWnd);
    //----------------------------------------------------------

    //----------------------------------------------------------
    hCanvasWnd = CreateCanvasWindow(hMainWnd);
    //----------------------------------------------------------

    //----------------------------------------------------------
    ImageBufferLen = MAXIMAGE_WIDTH * MAXIMAGE_HEIGHT * 3L;
    //----------------------------------------------------------

    //----------------------------------------------------------
    ImageBuffer = (BYTE *)malloc(ImageBufferLen);
    if( ! ImageBuffer )
    {
        MessageBox(NULL, "Image buffer alloc failed",
szAppName, MB_OK);
    }
    //----------------------------------------------------------

    //----------------------------------------------------------
    ImageBuffer1 = (BYTE *)malloc(ImageBufferLen);
    if( ! ImageBuffer1 )
    {
        MessageBox(NULL, "Image buffer alloc failed",
szAppName, MB_OK);
    }
    //----------------------------------------------------------

    //------------------------------------------------------
    while( GetMessage(&msg, NULL, 0, 0) )
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    //------------------------------------------------------

    return msg.wParam;
}
//------------------------------------------------------------

//------------------------------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT ps;
    //HDC         hdc;
    //int      Width, Height;
    RECT        rect;
    int         ExtraWidth, ExtraHeight;
    char        tmpbuf[1000];


    switch( message )
    {
        case WM_COMMAND:                    // menu & commands
            switch( LOWORD(wParam) )
            {
                //--------------------------------------------
                case IDB_SHOWBUTTON:
                    //--------------------------------------------



                    if( WebCamRunning )
                    {
                        // already running
                        return 0;
                    }
                    //--------------------------------------------

                    //--------------------------------------------
                    // assume device 0
                    if( ! InitWebCamCapture(hCapWnd, 0, &WebcamImageWidth, &WebcamImageHeight) )
                    {
                        MessageBox(hwnd, "InitWebCamCapture() failed", "WebCam", MB_OK);
                        return 0;
                    }
                    //--------------------------------------------

                    //--------------------------------------------
                    WebCamRunning = 1;
                    WebCamPaused = 0;
                    //-------------------------------------------

                    gblMessageStatus = 0;



                    //-------------------------------------------
                    sprintf(tmpbuf, "%d x %d", WebcamImageWidth, WebcamImageHeight);
                    SetWindowText(hTextImageDims, tmpbuf);
                    //-------------------------------------------



                    //sprintf(tmpbuf, "w=%d h=%d", WebcamImageWidth, WebcamImageHeight);
                    //MessageBox(hwnd, tmpbuf, "WebCam", MB_OK);

                    //MessageBox(hwnd, "InitWebCamCapture() OK", "WebCam", MB_OK);
#if 0
                    //-------------------------------------------
                    if( GrabWebCamFrame(ImageBuffer, ImageBufferLen, &WebcamImageWidth, &WebcamImageHeight ) )
                    {
                        MessageBox(hwnd, "GrabFrame OK", "WebCam", MB_OK);
                        //sprintf(tmpbuf, "w=%d h=%d", WebcamImageWidth, WebcamImageHeight);
                        //MessageBox(hwnd, tmpbuf, "WebCam", MB_OK);
                    }
                    else
                    {
                        MessageBox(hwnd, "GrabFrame not OK", "WebCam", MB_OK);
                    }
                    //-------------------------------------------
#endif

                    //-------------------------------------------
                    // to resize image to actual driver size
                    //-------------------------------------------

                    //-------------------------------------------
                    //GetWindowRect(hMainWnd, &rect);
                    //sprintf(tmpbuf, "w=%d h=%d", rect.right, rect.bottom);
                    //sprintf(tmpbuf, "l=%d t=%d", rect.left, rect.top);
                    //MessageBox(hwnd, tmpbuf, "WebCam", MB_OK);
```

```
                //ExtraWidth = DEFAULT_CANVAS_WIDTH - rect.right;
                //ExtraHeight = DEFAULT_CANVAS_HEIGHT - rect.bottom;
                //--------------------------------------------

                //--------------------------------------------
                // resize to capture default
                //--------------------------------------------

                //--------------------------------------------
                GetWindowRect(hMainWnd, &rect);
                MoveWindow(hMainWnd,
                                rect.left, rect.top,
                                2*WebcamImageWidth + 200,
                                WebcamImageHeight + 80,
                                TRUE);

                MoveWindow(hCapWnd,
                                WEBCAM_X, WEBCAM_Y,
                                WebcamImageWidth, WebcamImageHeight,
                                TRUE);

                MoveWindow(hCanvasWnd,
                                WEBCAM_X + WebcamImageWidth + 20,
                                WEBCAM_Y,
                                WebcamImageWidth, WebcamImageHeight,
                                //CANVAS_WIDTH, CANVAS_HEIGHT,
                                TRUE);
                //--------------------------------------------

                //----------------------------------------------------------
                SetTimer(hMainWnd, ID_TIMER, FRAMERATE, NULL);
                //----------------------------------------------------------

                //----------------------------------------------------------
                nFrames = 0L;
                //----------------------------------------------------------

                break;
            //--------------------------------------------

            //--------------------------------------------
            case IDB_STOPBUTTON:
```

```
            //-------------------------------------------
            if( ! WebCamRunning )
            {
                // already running
                return 0;
            }
            CloseWebCamCapture();

            WebCamRunning = 0;
            //-------------------------------------------
            break;

        //-------------------------------------------
        case IDB_PAUSEBUTTON:

            //-------------------------------------------
            if( ! WebCamRunning )
                return 0;

            if( WebCamPaused )
                return 0;
            //-------------------------------------------

            //----------------------------------------------------------------
            PauseWebCamCapture();
            WebCamPaused = 1;
            //----------------------------------------------------------------

            //----------------------------------------------------------------
            break;
        //-------------------------------------------

        //-------------------------------------------
        case IDB_RESUMEBUTTON:
            //-------------------------------------------
            if( ! WebCamRunning )
                return 0;

            if( ! WebCamPaused )
                return 0;
            //-------------------------------------------

            //----------------------------------------------------------------
            ResumeWebCamCapture();
            WebCamPaused = 0;
            //----------------------------------------------------------------

            //----------------------------------------------------------------
```

```
                    break;
                //--------------------------------------------

                //--------------------------------------------
                case IDB_EXITBUTTON:
                    //-----------------------------------------------------------------
                    PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
                    break;
                //--------------------------------------------
            }
            return 0;         // WM_COMMAND handled

        case WM_CREATE:
            return 0;

        case WM_PAINT:
            //------------------------------------------------------------
            //hdc = BeginPaint(hwnd, &ps);
            BeginPaint(hwnd, &ps);

            // paint window here

            EndPaint(hwnd, &ps);
            //------------------------------------------------------------

            return 0;

        case WM_TIMER:
            //------------------------------------------------------------
            if( ! WebCamRunning )
            {
                return 0;
            }
            //------------------------------------------------------------

            //------------------------------------------------------------
            if( WebCamPaused )
            {
                return 0;
            }
            //------------------------------------------------------------

            // debug ------------------------------------------------------------

                xMin = 0;
                yMin = 0;


            //------------------------------------------------------------


            //------------------------------------------------------------
            if( GrabWebCamFrame(ImageBuffer, ImageBufferLen, &WebcamImageWidth, &WebcamImageHeight ) )
            {
                // MessageBox(hwnd, "GrabFrame OK", "WebCam", MB_OK);
                ProcessFrame(ImageBuffer, ImageBuffer1, WebcamImageWidth, WebcamImageHeight);
            }
            else
            {
                // MessageBox(hwnd, "GrabFrame not OK", "WebCam", MB_OK);
            }
            //------------------------------------------------------------

            return 0;

        case WM_CLOSE:
            //MessageBox(hwnd, "close", szAppName, MB_OK);
            PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            //MessageBox(hwnd, "destroy", szAppName, MB_OK);

            //---------------------------------------------
            KillTimer(hwnd, ID_TIMER);
            //---------------------------------------------

            //---------------------------------------------
            if( ImageBuffer )
            {
                free((void *)ImageBuffer);
            }
            //---------------------------------------------

            //---------------------------------------------
            if( ImageBuffer1 )
            {
                free((void *)ImageBuffer1);
            }
            //---------------------------------------------

            //---------------------------------------------
            CloseWebCamCapture();
            //--------------------------------------------
            //Sleep(2000);

            //--------------------------------------------
            CoUninitialize();
            //--------------------------------------------

            //--------------------------------------------
            PostQuitMessage(0);
            //--------------------------------------------

            return 0;

        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
}
//------------------------------------------------------------


//------------------------------------------------------------
HWND    CreateMainWindow(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nWinMode)
{
    WNDCLASSEX  wndclass;
    HWND        hwnd;


    //-------------------------------------------------------
    // main window
    wndclass.cbSize      = sizeof(WNDCLASSEX);
    wndclass.style       = 0; //CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProc;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance   = hInstance;

    // no icons
    wndclass.hIcon       = NULL;   // desktop icon 32x32
    wndclass.hIconSm     = NULL;   // minimize/menu icon 16x16

    wndclass.hCursor     = LoadCursor(NULL, (LPTSTR) IDC_ARROW);

    // window background
    //wndclass.hbrBackground =
    //          (HBRUSH)GetStockObject(LTGRAY_BRUSH);
    //wndclass.hbrBackground = (HBRUSH)(COLOR_BACKGROUND + 1);
    wndclass.hbrBackground = GetSysColorBrush(COLOR_BTNFACE);

    wndclass.lpszMenuName  = NULL;      // register menu name
    wndclass.lpszClassName =  szAppName;
    //--------------------------------------------------------------

    //--------------------------------------------------------------
    if( ! RegisterClassEx(&wndclass) )
        return 0;
    //--------------------------------------------------------------


    //--------------------------------------------------------------
    hwnd = CreateWindowEx(WS_EX_CLIENTEDGE,
                          szAppName, "WebCam Window",
                          WS_OVERLAPPEDWINDOW,        // normal
                          CW_USEDEFAULT, CW_USEDEFAULT, // x, y
                          DEFAULT_CANVAS_WIDTH + DEFAULT_CANVAS_WIDTH + 200,
                          DEFAULT_CANVAS_HEIGHT + 80,
                          HWND_DESKTOP,       // no parent window
                          NULL,
                          hInstance, NULL);
    //--------------------------------------------------------------

    //--------------------------------------------------------------
    // save main window handle
    hMainWnd = hwnd;
    hInstSave = hInstance;

    ShowWindow(hwnd, nWinMode);
    UpdateWindow(hwnd);
    //--------------------------------------------------------------


    //----------------- show button ------------------------
    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Show",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 30, 100, 20,
                    hwnd, (HMENU)IDB_SHOWBUTTON,
                    hInstSave, (LPVOID)NULL);
    //--------------------------------------------------------------

    //------------------ stop button -------------------------
    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Stop",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 55, 100, 20,
                    hwnd, (HMENU)IDB_STOPBUTTON,
                    hInstSave, (LPVOID)NULL);
    //--------------------------------------------------------


    //------------------ pause button -------------------------
    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Pause",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 80, 100, 20,
                    hwnd, (HMENU)IDB_PAUSEBUTTON,
                    hInstSave, (LPVOID)NULL);
    //--------------------------------------------------------

    //------------------ resume button -------------------------
    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Resume",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 105, 100, 20,
                    hwnd, (HMENU)IDB_RESUMEBUTTON,
                    hInstSave, (LPVOID)NULL);
    //--------------------------------------------------------

    //------------------ exit button -------------------------
    CreateWindowEx(BS_PUSHBUTTON,
                    "button",       // window class name
                    "Exit",
                    WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                    20, 130, 100, 20,
                    hwnd, (HMENU)IDB_EXITBUTTON,
                    hInstSave, (LPVOID)NULL);
    //--------------------------------------------------------

    //----------------------------------------------------------------------
    // image dimension
    hTextImageDims = CreateWindowEx(BS_PUSHBUTTON,
                    "edit",       // window class name
                    "",
                    //WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT |
ES_READONLY,
                    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY,
                    20, 160, 90, 16,
                    hwnd, (HMENU)IDE_IMAGEDIMS,
                    hInstSave, (LPVOID)NULL);
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    // number of frames
    hTextNumFrames = CreateWindowEx(BS_PUSHBUTTON,
                    "edit",      // window class name
                    "",
                    //WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT | ES_READONLY,
                    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY,
                    20, 180, 90, 16,
                    hwnd, (HMENU)IDE_NUMFRAMES,
                    hInstSave, (LPVOID)NULL);
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    // RAM free, MBytes
    hTextRAMFree = CreateWindowEx(BS_PUSHBUTTON,
                    "edit",      // window class name
                    "",
                    //WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT | ES_READONLY,
                    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY,
                    20, 200, 90, 16,
                    hwnd, (HMENU)IDE_RAMFREE,
                    hInstSave, (LPVOID)NULL);
    //----------------------------------------------------------------------


    //----------------------------------------------------------------------
    // max B&W value
    hTextMaxVal = CreateWindowEx(BS_PUSHBUTTON,
                    "edit",      // window class name
                    "",
                    //WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT | ES_READONLY,
                    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY,
                    20, 220, 90, 16,
                    hwnd, (HMENU)IDE_MAXVAL,
                    hInstSave, (LPVOID)NULL);
    //-----------------------------------------------------------------------

    //-----------------------------------------------------------------------
    // max run len value
    hTextMaxRunLen = CreateWindowEx(BS_PUSHBUTTON,
                    "edit",      // window class name
                    "",
                    //WS_CHILD | WS_VISIBLE | WS_BORDER | ES_LEFT | ES_READONLY,
                    WS_CHILD | WS_VISIBLE | ES_LEFT | ES_READONLY,
                    20, 240, 90, 16,
                    hwnd, (HMENU)IDE_MAXRUNLEN,
                    hInstSave, (LPVOID)NULL);
    //-----------------------------------------------------------------------


    return hwnd;
}
//--------------------------------------------------------------

//--------------------------------------------------------------
HWND    CreateCanvasWindow(HWND hParentWnd)
{
    WNDCLASSEX  wndclass;
    HWND        hwnd;


    //--------------------------------------------------------
    // canvas window
    wndclass.cbSize      = sizeof(WNDCLASSEX);
    wndclass.style       = 0; //CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProcCanvasWindow;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance   = hInstSave;

    // no icons
    wndclass.hIcon       = NULL;    // desktop icon 32x32
    wndclass.hIconSm     = NULL;    // minimize/menu icon 16x16

    wndclass.hCursor     = LoadCursor(NULL, (LPTSTR) IDC_ARROW);

    // window background
    //wndclass.hbrBackground =
    //         (HBRUSH)GetStockObject(LTGRAY_BRUSH);
    //wndclass.hbrBackground = (HBRUSH)(COLOR_BACKGROUND + 1);
    wndclass.hbrBackground = GetSysColorBrush(WHITE_BRUSH);

    wndclass.lpszMenuName  = NULL;       // register menu name
    wndclass.lpszClassName =  "CanvasWindowClass";
    //--------------------------------------------------------

    //--------------------------------------------------------
    if( ! RegisterClassEx(&wndclass) )
        return 0;
    //--------------------------------------------------------


    //--------------------------------------------------------
    hwnd = CreateWindowEx(0, //WS_EX_CLIENTEDGE,
                          "CanvasWindowClass", "WebCam Canvas Window",
                          //WS_CHILD | WS_VISIBLE | WS_DLGFRAME,  // frame, no controls
                          WS_CHILD | WS_VISIBLE,
                          //WS_BORDER,
                          //WS_POPUP,
                          WEBCAM_X + WebcamImageWidth + 20,
                          WEBCAM_Y,
                          DEFAULT_CANVAS_WIDTH, DEFAULT_CANVAS_HEIGHT,
                          hParentWnd,
                          NULL,
                          hInstSave, NULL);
    //--------------------------------------------------------

    //--------------------------------------------------------
    SetMapMode(hwnd, MM_TEXT);
    //--------------------------------------------------------

    //--------------------------------------------------------
    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);
    //--------------------------------------------------------
    return hwnd;
}
//------------------------------------------------------------


//------------------------------------------------------------
LRESULT CALLBACK WndProcCanvasWindow(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT     ps;
    //HDC        hdc;
    //char       tmpbuf[1000];


    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            //hdc = BeginPaint(hwnd, &ps);
            BeginPaint(hwnd, &ps);

            // paint window here

            EndPaint(hwnd, &ps);
            return 0;

        case WM_CLOSE:
            PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
}
//------------------------------------------------------------


//------------------------------------------------------------
HWND    CreateCaptureWindow(HWND hParentWnd)
{
    WNDCLASSEX  wndclass;
    HWND        hwnd;
    char        tmpbuf[1000];


    //----------------------------------------------------------
    // canvas window
    wndclass.cbSize      = sizeof(WNDCLASSEX);
    wndclass.style       = 0; //CS_HREDRAW | CS_VREDRAW;
    wndclass.lpfnWndProc = WndProcCaptureWindow;
    wndclass.cbClsExtra  = 0;
    wndclass.cbWndExtra  = 0;
    wndclass.hInstance   = hInstSave;

    // no icons
    wndclass.hIcon       = NULL;   // desktop icon 32x32
    wndclass.hIconSm     = NULL;   // minimize/menu icon 16x16

    wndclass.hCursor     = LoadCursor(NULL, (LPTSTR) IDC_ARROW);

    // window background
    //wndclass.hbrBackground =
    //         (HBRUSH)GetStockObject(LTGRAY_BRUSH);
    //wndclass.hbrBackground = (HBRUSH)(COLOR_BACKGROUND + 1);
    wndclass.hbrBackground = GetSysColorBrush(WHITE_BRUSH);

    wndclass.lpszMenuName  = NULL;      // register menu name
    wndclass.lpszClassName =  "CaptureWindowClass";
    //-------------------------------------------------------

    //-------------------------------------------------------
    if( ! RegisterClassEx(&wndclass) )
        return 0;
    //-------------------------------------------------------


    //-------------------------------------------------------
    hwnd = CreateWindowEx(0, //WS_EX_CLIENTEDGE,
                          "CaptureWindowClass", "WebCam Capture Window",
                          //WS_CHILD | WS_VISIBLE | WS_DLGFRAME,  // frame, no controls
                          WS_CHILD | WS_VISIBLE,
                          //WS_BORDER,
                          //WS_POPUP,
                          WEBCAM_X, WEBCAM_Y,
                          DEFAULT_CANVAS_WIDTH, DEFAULT_CANVAS_HEIGHT,
                          hParentWnd,
                          NULL,
                          hInstSave, NULL);
    //-------------------------------------------------------

    //-------------------------------------------------------
    SetMapMode(hwnd, MM_TEXT);
    //-------------------------------------------------------

    //-------------------------------------------------------
    ShowWindow(hwnd, SW_SHOW);
    UpdateWindow(hwnd);
    //-------------------------------------------------------
    return hwnd;
}
//------------------------------------------------------------


//------------------------------------------------------------
LRESULT CALLBACK WndProcCaptureWindow(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    PAINTSTRUCT     ps;
    //HDC         hdc;
    //char        tmpbuf[1000];


    switch( message )
    {
        case WM_CREATE:
            return 0;

        case WM_PAINT:
            //hdc = BeginPaint(hwnd, &ps);
            BeginPaint(hwnd, &ps);

            // paint window here

            EndPaint(hwnd, &ps);
            return 0;

        case WM_CLOSE:
            PostMessage(hwnd, WM_DESTROY, (WPARAM)0, (LPARAM)0);
            return 0;

        case WM_DESTROY:
            PostQuitMessage(0);
            return 0;

        default:
            return DefWindowProc(hwnd, message, wParam, lParam);
    }
}
//------------------------------------------------------------

```

```

//---------------------------------------------------------
// per-frame processing
int     ProcessFrame(unsigned char *pFrameIn, unsigned char *pFrameOut, int Width, int Height)
{
    unsigned long   DisplayWidth, DisplayHeight;
    unsigned long   ImageSizeBytes;

    DWORD           ByteNum;
    LPBYTE          pByteIn, pByteOut;
    BYTE            ByteOut;
    BYTE            *pRGBImage;
    RECT            ClientRect;

    HDC             hCanvasDC;
    HBITMAP         hCanvasBitmap;
    HDC             hdcMem;
    HBITMAP         hMemBitmap;

    BITMAPINFOHEADER    bmihdr;
    MEMORYSTATUS        MemStatus;
    double              RAMFreeMBytes;

    int             x, y;
    BYTE            RedByte, GreenByte, BlueByte;
    int             PixelValue;
    int             MaxValue; //, MaxRunLen, CurrRunLen;

    int             PrevPixelValue;
    int             PixelIndex;

    char            tmpbuf[1000];

    int detectLaserOutput;
    long lastY;

    // laser detection algorithm variables ---------------------------------------------------------------------------------------------

    // The x minimum and maximum values of the current row
    int             curXmin =   NOTSET;
    int             curXmax =   NOTSET;

    // The difference between the current and last x positions
    int             xDiff   =   NOTSET;

    // The minimum x value of the previous line that was scanned
    int             lastXmin =  NOTSET;
    int             lastXmax =  NOTSET;


    //-----------------------------------------------------------------------------------------------------------------------------------

    //---------------------------------------------------------
    // update number of frames processed
    nFrames++;

    sprintf(tmpbuf, "n=%ld", nFrames);
    SetWindowText(hTextNumFrames, tmpbuf);
    //---------------------------------------------------------

    //---------------------------------------------------------
    // update memory free
    MemStatus.dwLength = sizeof(MEMORYSTATUS);
    GlobalMemoryStatus(&MemStatus);

    RAMFreeMBytes = (double)MemStatus.dwAvailPhys;
    RAMFreeMBytes /= (double)MBYTES;

    sprintf(tmpbuf, "RAM %.0lf M", RAMFreeMBytes);
    SetWindowText(hTextRAMFree, tmpbuf);
    //---------------------------------------------------------

    //---------------------------------------------------------
    ImageSizeBytes = (unsigned long)Width*(unsigned long)Height*3L;
    //---------------------------------------------------------

    //---------------------------------------------------------
    if( (! pFrameIn) || (! pFrameOut) )
    {
        // image buffer malloc failed at startup
        return 0;
    }
    //---------------------------------------------------------

    //---------------------------------------------------------
    if( ImageSizeBytes > (MAXIMAGE_WIDTH * MAXIMAGE_HEIGHT * 3L) )
    {
        // need larger image size params
        return 0;
    }
    //-----------------------------------------------------------

#if 0
    //-----------------------------------------------------------
    // copy or invert pixels
    pByteIn = pFrameIn;
    pByteOut = pFrameOut;
    for(ByteNum = 0; ByteNum < ImageSizeBytes; ByteNum++)
    {
        CurrByte = *pByteIn++;

        //CurrByte = ~CurrByte;

        *pByteOut++ = CurrByte;
    }
    //-----------------------------------------------------------
#endif

#if 1
    //-----------------------------------------------------------
    // convert to grayscale
    //-----------------------------------------------------------

    //-----------------------------------------------------------
    pByteIn = pFrameIn;
    pByteOut = pFrameOut;
    //-----------------------------------------------------------

    //-----------------------------------------------------------
    MaxValue = 0;
    //MaxRunLen = 0;
    //CurrRunLen = 0;
    //-----------------------------------------------------------





    resetAllLaserData();
    lastY = 0;

    //-----------------------------------------------------------
    for(y = 0; y < Height; y++)
    {

        // Reset the x min
        curXmin = NOTSET;

```

```
            //-----------------------------------------------------------
            for(x = 0; x < Width; x++)
            {
                //-----------------------------------------------------------
                // retrieve image bytes
                BlueByte  = *(pByteIn+0);
                GreenByte = *(pByteIn+1);
                RedByte   = *(pByteIn+2);
                //-----------------------------------------------------------

                //-----------------------------------------------------------
                // calculate new pixel value
                PixelValue = (int)RedByte + (int)BlueByte + (int)GreenByte;
                PixelValue /= 3;
                //-----------------------------------------------------------






//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

                        // if the pixel is bright enough, it could be part of the laser point, but it can not be certain as this is only
                        // one pixel therfore more data collection will be required when the next pixel is encountered.

                if (PixelValue > LASER_DETECTION_THRESHOLD) {

                    detectLaser(y, x, &lastY, PixelValue);
                    // debug

                    xMin = x;
                    yMin = y;

                    // end debug


                    // record the minimum value of x and y
                    // if the current minimum value of x is greater than the current x value, then it is no longer the
                    // minimum value of x and therefore the new minimum value of x and y needs to be recorded.
                    // OR
                    // If this is the first pixel detected above the laser detection threshold then the value of xmin is likely to be
                    // NOTSET, therefore the minimum value needs to be applied.
                    //
                    // the minimum values of x and y is one of the boundaries of the area of pixels of the laser point, and is used in the
                    // calculation of the coordinate of the laser.

                    // printf("Xmin: %i, YMin: %i\n", xMin, yMin);

                    // if (xMin > x || xMin == NOTSET) {
                            // xMin = x;
                    // }
                    /*
                    if (xMin == NOTSET) {
                            xMin = x;
                    }
                    if (xMin > x) {
                            xMin = x;
                    }
                    */

                    /*

                    if (yMin > y || yMin == NOTSET){
                            yMin = y;
                    }

                    if (yMin == NOTSET) {
                            yMin = y;
                    }

                    if (yMin > y) {
                            yMin = y;
                    }
                    */
                    /*
                    // set the current rows x minimum
                    if (curXmin == NOTSET) {
                            curXmin = x;
                            xDiff = 1;
                    } else {
                            // The difference between the last positive value of x and this current value of x
                            xDiff = x - curXmax;
                    }
                    */

```

```
/*
if (xDiff > 0 && xDiff < 2) {
    if (xDiff < 2) { // (GAP_AMOUNT+1)
        // set the new current row's x maximum
        curXmax = x;
        // set the new maximum value of x
        // set the count of positive pixels
        // set the count of x positive pixels
    }
} else {

    if (yMax != NOTSET) {

        if (curXmin > lastXmin && curXmin < lastXmax) {
            // x = Width;
            yMax = y;
            if (x > xMax) {
                xMax = x;
            }
        } else if (curXmin > lastXmin && curXmax < lastXmax) {
            // x = Width;
            yMax = y;
            if (x > xMax) {
                xMax = x;
            }
        } else if (curXmin < lastXmin && curXmax > lastXmax) {
            // x = Width;
            yMax = y;
            if (x > xMax) {
                xMax = x;
            }
        } else {

            if (x > (lastXmax + 1)) {
                // x = Width;
                // y = Height;
            }


                curXmin = NOTSET;
        }

    } else {
        yMax = y;
        // x = Width;
    }

}
```

```
        */







      }



//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


      // keep this for debugging purposes in order to see the laser
      // "High pass" filter
      if (PixelValue < LASER_DETECTION_THRESHOLD) {
        PixelValue = 0;

      }

      //------------------------------------------------------
      // update image max greyscale value
      if( PixelValue >= MaxValue )
      {
        // MaxValue = PixelValue;
      }
      //------------------------------------------------------

      //------------------------------------------------------
#if 0
      if( (RedByte >= 100) &&  (RedByte >= MaxValue) && (GreenByte < 100)&& (BlueByte < 100) )
      //if( (RedByte >= MaxValue) )
      {
        //MaxValue = PixelValue;
        MaxValue = RedByte;

        CurrRunLen++ ;
      }
      else
      {
        CurrRunLen = 0;
      }

```

```
            if( CurrRunLen > MaxRunLen )
            {
                MaxRunLen = CurrRunLen;
            }
#endif
            //------------------------------------------------------------

            //------------------------------------------------------------
            // restore pixel value
            BlueByte  = (BYTE)PixelValue;
            GreenByte = (BYTE)PixelValue;
            RedByte   = (BYTE)PixelValue;
            //------------------------------------------------------------

            //------------------------------------------------------------
            *(pByteOut+0) = BlueByte;
            *(pByteOut+1) = GreenByte;
            *(pByteOut+2) = RedByte;
            //------------------------------------------------------------

            //------------------------------------------------------------
            // point to next pixel
            pByteIn += 3;
            pByteOut += 3;
            //------------------------------------------------------------
        }
        //------------------------------------------------------------
    }
    //------------------------------------------------------------

    // check the laser data

    // verifyLaser
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------

    // the most obvious detection




    //
--------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------

    printf("After: Xmin: %i, Ymin: %i\n", xMin, yMin);




    // debug ------------------------------------------------------------------------------------------
            if (xMin > NOTSET &&  yMin > NOTSET) {

                if (countStarted == 0) {
                    startTime = time(NULL);
                    countStarted = 1;
                    currentFileName = createFileName();
                }

                laserPosToFile();
                if (pixelCount > 0) {
                    pixelCount ++;
                } else {
                    pixelCount = 1;
                }

            } else {
                timeToFile();
                countStarted = 0;
                pixelCount = 0;
            }
            //
------------------------------------------------------------------------------------------------------





    //-------------------------------------------------------------------
    // sprintf(tmpbuf, "Max=%d", MaxValue);
    // SetWindowText(hTextMaxVal, tmpbuf);
    //-------------------------------------------------------------------


    //-------------------------------------------------------------------
    //sprintf(tmpbuf, "Run=%d", MaxRunLen);
    //SetWindowText(hTextMaxRunLen, tmpbuf);
    //-------------------------------------------------------------------

#endif

#if 0
    //-------------------------------------------------------------------
    // operate on copied buffer, (x,y) mode, to output to processed buffer
    // horizontal edge detection
    for(y = 0; y < Height; y++)
    {
        PrevPixelValue = 0;
        for(x = 0; x < Width; x++)
        {
            PixelIndex = 3*(y*Width + x);
            PixelValue = ImageBuffer[PixelIndex];

            ByteOut = 0;
            if( abs(PixelValue - PrevPixelValue) >
PIXELDIFF_THRESHOLD )
            {
                ByteOut = 255;
            }

            pByteOut = &ImageBuffer1[PixelIndex];
            *(pByteOut+0) = ByteOut;
            *(pByteOut+1) = ByteOut;
            *(pByteOut+2) = ByteOut;

            PrevPixelValue = PixelValue;
        }
    }
    //------------------------------------------------------------
#endif

#if 0
    //------------------------------------------------------------
    // operate on coped buffer, (x,y) mode, to output to processed buffer
    // vertical edge detection
    for(x = 0; x < Width; x++)
    {
        PrevPixelValue = 0;
        for(y = 0; y < Height; y++)
        {
            PixelIndex = 3*(y*Width + x);
            PixelValue = ImageBuffer[PixelIndex];

            ByteOut = 0;
            if( abs(PixelValue - PrevPixelValue) >
PIXELDIFF_THRESHOLD )
            {
                ByteOut = 255;
            }

            pByteOut = &ImageBuffer1[PixelIndex];
            *(pByteOut+0) = ByteOut;
            *(pByteOut+1) = ByteOut;
            *(pByteOut+2) = ByteOut;

            PrevPixelValue = PixelValue;
        }
    }
    //----------------------------------------------------------
#endif

    //----------------------------------------------------------
    // update display window
    //----------------------------------------------------------

    //----------------------------------------------------------
    pRGBImage = pFrameOut;
    //----------------------------------------------------------

    //----------------------------------------------------------
    GetClientRect(hCanvasWnd, &ClientRect);
    DisplayWidth = ClientRect.right;
    DisplayHeight = ClientRect.bottom;
    //----------------------------------------------------------

    //----------------------------------------------------------
    hCanvasDC = GetDC(hCanvasWnd);
    hCanvasBitmap = CreateCompatibleBitmap(hCanvasDC, DisplayWidth, DisplayHeight);
    SelectObject(hCanvasDC, hCanvasBitmap);
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    hdcMem = CreateCompatibleDC(hCanvasDC);
    hMemBitmap = CreateCompatibleBitmap(hCanvasDC, Width, Height);
    SelectObject(hdcMem, hMemBitmap);
    //----------------------------------------------------------


    //----------------------------------------------------------------------
    // clear data structure, set structure size
    memset((void *)&bmihdr, 0, sizeof(BITMAPINFOHEADER));

    bmihdr.biSize = (DWORD)sizeof(BITMAPINFOHEADER);
    bmihdr.biWidth = (DWORD)Width;
    bmihdr.biHeight = (DWORD)Height;
    bmihdr.biPlanes = (DWORD)1;
    bmihdr.biBitCount = (DWORD)24;                  // 24 bits/pixel RGB mode
    bmihdr.biCompression = (DWORD)BI_RGB;           // no compression
    bmihdr.biSizeImage = (DWORD)ImageSizeBytes;     // total size in bytes
    //----------------------------------------------------------------------

#if 0
    //----------------------------------------------------------------------
    // use the raw data to form a bitmap
    SetDIBits(hdcMem, hMemBitmap, (UINT)0, (UINT)CaptureHeight, (void *)pRGBImage, (BITMAPINFO *)&bmihdr, DIB_RGB_COLORS);
    //----------------------------------------------------------------------
#endif

#if 0
    //----------------------------------------------------------------------
    // use the copied data to form a bitmap
    SetDIBits(hdcMem, hMemBitmap, (UINT)0, (UINT)CaptureHeight, (void *)ImageBuffer, (BITMAPINFO *)&bmihdr, DIB_RGB_COLORS);
    //----------------------------------------------------------------------
#endif

#if 1
    //----------------------------------------------------------------------
    // use processed buffer
    SetDIBits(hdcMem, hMemBitmap, (UINT)0, (UINT)Height, (void *)ImageBuffer1, (BITMAPINFO *)&bmihdr, DIB_RGB_COLORS);
    //----------------------------------------------------------------------
#endif

    //----------------------------------------------------------------------
    // this is from the bitmap created from ImageBuffer using SetDIBits
    SetStretchBltMode(hdcMem, COLORONCOLOR);
    StretchBlt(hCanvasDC, 0, 0, DisplayWidth, DisplayHeight ,
                   hdcMem, 0, 0, Width, Height, SRCCOPY);
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    ReleaseDC(hCanvasDC, hCanvasWnd);
    DeleteDC(hCanvasDC);
    DeleteObject(hCanvasBitmap);
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    ReleaseDC(hCanvasWnd, hdcMem);
    DeleteDC(hdcMem);
    DeleteObject(hMemBitmap);
    //----------------------------------------------------------------------

    return 0;
}
//--------------------------------------------------------------


int     detectLaser(int pCurY, int pCurX, long *pLastY, int pIntensity) {



    if (pIntensity > LASER_DETECTION_THRESHOLD) {

        char tmpbuf[20];



            sprintf(tmpbuf, "%d, %d, %d", pCurX, pCurY, *pLastY);
            SetWindowText(hTextMaxVal, tmpbuf);

            *pLastY = pCurY;




    }
    return 0;

}


int     laserPosToFile() {

    FILE *fp;
    char * pfilename;



    fp = fopen(currentFileName, "a");

    if (fp == NULL) {

        return 1;

    }


    fprintf(fp, "X Min: %d, X Max: %d, Y Min: %d, Y Max: %d \n", xMin, xMax, yMin, yMax);

    fclose(fp);

    return 0;

}

char * createFileName() {


    char * filename = "";
    // time_t timeVal;
    struct tm * timeVals;
    int test = 9;

    // timeVal = time(NULL);

    timeVals = localtime(&startTime);

    sprintf(filename, "Output_%i%i%i_%i%i.txt", timeVals->tm_mday, timeVals->tm_mon, timeVals->tm_year, timeVals->tm_hour, timeVals->tm_min);

    return filename;

}

int timeToFile() {

    FILE *fp;


    fp = fopen(currentFileName, "a");

    if (fp == NULL) {

        return 1;

    }



    fprintf(fp, "%s\n", calcDetectionRate());

    fclose(fp);

    return 0;

}

char * calcDetectionRate() {

    char * lResult = "";
    double timedif;
    double detectionRate;
    time_t endTime;

    // initialise the end time
    endTime = time(NULL);


    // difference between times
    timedif = difftime(endTime, startTime);

    // number of detections / number of seconds
    detectionRate = pixelCount / timedif;

    // format into string
    sprintf(lResult, "%i pixels counted. Detection Rate is %f pixels per second. Total time was %f.\n", pixelCount, detectionRate, timedif);

    // printf("printf text being displayed here: %s\n", lResult);

    return lResult;

}

void resetAllLaserData() {

    lastX = NOTSET;         // The last X coordinate
    lastXmin = NOTSET;      // The last min value of Y coordinate
    yMin = NOTSET;          // The smallest value of the y coordinate detecteds
    xMin = NOTSET;          // The smallest value of the x coordinate detected
    yMax = NOTSET;          // The largest value of the y coordinate detected
    xMax = NOTSET;          // The largest value of the x coordinate detected

}
```

D.2 WebCamLib.c Source

```c
/* WebCamLib.c
 * see also wmlib.c
 * adapted to C from
 *  http://secure.codeproject.com/KB/audio-video/DXCapture.aspx
 */

//-------------------------------------------------------------------
#include <windows.h>
//#include <objbase.h>
#include <shlobj.h>
//-------------------------------------------------------------------


//-------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
//-------------------------------------------------------------------

//-------------------------------------------------------------------
#include "WebCamMin.h"
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// from uuids.h
GUID CLSID_FilterGraph = { 0xe436ebb3, 0x524f, 0x11ce, 0x9f, 0x53, 0x00, 0x20, 0xaf, 0x0b, 0xa7, 0x70 };
GUID CLSID_VideoMixingRenderer = { 0xB87BEB7B, 0x8D29, 0x423f, 0xAE, 0x4D, 0x65, 0x82, 0xC1, 0x01, 0x75, 0xAC };
GUID CLSID_SystemDeviceEnum = { 0x62BE5D10,0x60EB,0x11d0, 0xBD,0x3B,0x00,0xA0,0xC9,0x11,0xCE,0x86 };
GUID CLSID_VideoInputDeviceCategory = { 0x860BB310,0x5D01, 0x11d0,0xBD,0x3B,0x00,0xA0,0xC9,0x11,0xCE,0x86 };
GUID FORMAT_VideoInfo = { 0x05589f80, 0xc356, 0x11ce, 0xbf, 0x01, 0x00, 0xaa, 0x00, 0x55, 0x59, 0x5a };
//-------------------------------------------------------------------

//-------------------------------------------------------------------
// from strmif.h
GUID IID_IBaseFilter = { 0x56a86895, 0x0ad4, 0x11ce, 0xb0, 0x3a, 0x00, 0x20, 0xaf, 0x0b, 0xa7, 0x70 };
GUID IID_IGraphBuilder= { 0x56a868a9, 0x0ad4, 0x11ce, 0xb0, 0x3a, 0x00, 0x20, 0xaf, 0x0b, 0xa7, 0x70 };
GUID IID_IVMRFilterConfig = { 0x9e5530c5, 0x7034, 0x48b4, 0xbb, 0x46, 0x0b, 0x8a, 0x6e, 0xfc, 0x8e, 0x36 };
GUID IID_IVMRWindowlessControl = { 0x0eb1088c, 0x4dcd, 0x46f0, 0x87, 0x8f, 0x39, 0xda, 0xe8, 0x6a, 0x51, 0xb7 };
GUID IID_ICreateDevEnum = { 0x29840822, 0x5B84, 0x11D0, 0xBD, 0x3B, 0x00, 0xA0, 0xC9, 0x11, 0xCE, 0x86 };
GUID IID_IAMStreamConfig = { 0xC6E13340, 0x30AC, 0x11d0, 0xA1, 0x8C, 0x00, 0xA0, 0xC9, 0x11, 0x89, 0x56 };

//------------------------------------------------------------------

//------------------------------------------------------------------
// from oaldl.h
GUID IID_IPropertyBag = { 0x55272A00, 0x42CB, 0x11CE, 0x81, 0x35, 0x00, 0xAA, 0x00, 0x4B, 0xB8, 0x51 };
//------------------------------------------------------------------

//------------------------------------------------------------------
// control.h
GUID IID_IMediaControl = { 0x56a868b1,0x0ad4,0x11ce,0xb0,0x3a,0x00,0x20,0xaf,0x0b,0xa7,0x70 };
GUID IID_IMediaEventEx = { 0x56a868c0,0x0ad4,0x11ce,0xb0,0x3a,0x00,0x20,0xaf,0x0b,0xa7,0x70 };
//------------------------------------------------------------------



//------------------------------------------------------------------
// global interface pointers
IVMRWindowlessControl *m_pWC = NULL;
IMediaControl         *m_pMC = NULL;
IPin                  *m_pCamOutPin = NULL;
IGraphBuilder         *pIGraphBuilder = NULL;
IBaseFilter           *pIBaseFilter = NULL;
IMediaEventEx         *m_pME = NULL;
//------------------------------------------------------------------

//------------------------------------------------------------------
static int WebCamInitialized = 0;
static int WebCamPaused = 0;
//------------------------------------------------------------------

//------------------------------------------------------------------
int     InitWebCamCapture(HWND hWnd, int iDeviceID, int *pWidth, int *pHeight);
int     GrabWebCamFrame(unsigned char *pFrameOut, unsigned long FrameBufferLen, int *pWidth, int *pHeight);
int     StopWebCamCapture(void);
int     PauseWebCamCapture(void);
int     ResumeWebCamCapture(void);
int     CloseWebCamCapture(void);

int     Convert24Image(BYTE *p32Img, BYTE *p24Img, DWORD
dwSize32);

int     InitializeWindowlessVMR(HWND hWnd);
int     BindFilter(int deviceId, IBaseFilter **pFilter);
int     InitVideoWindow(HWND hWnd,int *pWidth, int *pHeight);
//--------------------------------------------------------------------

//--------------------------------------------------------------------
//BYTE *pFrame;
//long nFramelen;
//--------------------------------------------------------------------


//--------------------------------------------------------------------
int     InitWebCamCapture(HWND hWnd, int iDeviceID, int *pWidth, int *pHeight)
{
    IEnumPins      *pEnum;
    //IMediaEventEx *m_pME;
    HRESULT         hr;


    //--------------------------------------------------------------------
    if( WebCamInitialized )
    {
        // already running
        return 0;
    }
    //--------------------------------------------------------------------

    //--------------------------------------------------------------------
    // Get the interface for DirectShow's GraphBuilder
    hr = CoCreateInstance(&CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
                            &IID_IGraphBuilder, (void **)&pIGraphBuilder);

    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }
    //--------------------------------------------------------------------

    //--------------------------------------------------------------------
    if( ! InitializeWindowlessVMR(hWnd) )
    {
        return 0;
    }

    if( ! BindFilter(iDeviceID, &pIBaseFilter) )
    {
        return 0;
    }

    hr = pIGraphBuilder->lpVtbl->AddFilter(pIGraphBuilder,
pIBaseFilter, L"Video Capture");
    if (FAILED(hr))
    {
        return 0;
    }

    pIBaseFilter->lpVtbl->EnumPins(pIBaseFilter, &pEnum);

    hr = pEnum->lpVtbl->Reset(pEnum);
    hr = pEnum->lpVtbl->Next(pEnum, 1, &m_pCamOutPin, NULL
);

    // QueryInterface for DirectShow interfaces
    hr = pIGraphBuilder->lpVtbl->QueryInterface(
pIGraphBuilder, &IID_IMediaControl, (void **)&m_pMC);
    if (FAILED(hr))
    {
        return 0;
    }

    hr = pIGraphBuilder->lpVtbl->QueryInterface(
pIGraphBuilder, &IID_IMediaEventEx, (void **)&m_pME);
    if (FAILED(hr))
    {
        return 0;
    }

    if( ! InitVideoWindow(hWnd, pWidth, pHeight) )
    {
        return 0;
    }

    //nFramelen = (*pWidth)*(*pHeight)*3;
    //pFrame = (BYTE*)malloc(nFramelen);

    hr = pIGraphBuilder->lpVtbl->Render(pIGraphBuilder,
m_pCamOutPin);
    if (FAILED(hr))
    {
        return 0;
    }

    hr = m_pMC->lpVtbl->Run(m_pMC);
    if (FAILED(hr))
    {
        return 0;
    }
    //----------------------------------------------------------------------

    //----------------------------------------------------------------------
    WebCamInitialized = 1;
    WebCamPaused = 0;
    //----------------------------------------------------------------------

    return 1;
}
//----------------------------------------------------------------------

#if 0
HRESULT CVMR_Capture::Init(int iDeviceID,HWND hWnd, int iWidth, int iHeight)
{
    HRESULT hr;
    // Get the interface for DirectShow's GraphBuilder
    hr=CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
                         IID_IGraphBuilder, (void **)&m_pGB);

    if(SUCCEEDED(hr))
    {
        // Create the Video Mixing Renderer and add it to the graph
        InitializeWindowlessVMR(hWnd);
        // Bind Device Filter.  We know the device because the id was passed in
        if(!BindFilter(iDeviceID, &m_pDF))
            return S_FALSE;

        hr=m_pGB->AddFilter(m_pDF, L"Video Capture");
        if (FAILED(hr))
        return hr;

        CComPtr<IEnumPins> pEnum;
        m_pDF->EnumPins(&pEnum);

        hr = pEnum->Reset();
        hr = pEnum->Next(1, &m_pCamOutPin, NULL);


        // QueryInterface for DirectShow interfaces
        hr = m_pGB->QueryInterface(IID_IMediaControl, (void
 **)&m_pMC);

        hr = m_pGB->QueryInterface(IID_IMediaEventEx, (void
 **)&m_pME);

        // Have the graph signal event via window callbacks for performance
        //hr = pME->SetNotifyWindow((OAHWND)hWnd, WM_GRAPHNOTIFY, 0);


        hr = InitVideoWindow(hWnd,iWidth, iHeight,
m_pCamOutPin);

        m_nFramelen=iWidth*iHeight*3;
        m_pFrame=(BYTE*) new BYTE[m_nFramelen];


        // Run the graph to play the media file

        m_psCurrent=Stopped;

        hr = m_pGB->Render(m_pCamOutPin);
        hr = m_pMC->Run();
        m_psCurrent=Running;
    }
    return hr;

}
#endif


//------------------------------------------------------------------
int     InitializeWindowlessVMR(HWND hWnd)
{
    IBaseFilter      *pVmr = NULL;
    IVMRFilterConfig *pConfig;
    //IVMRWindowlessControl *m_pWC;

    HRESULT hr;

    // Create the VMR and add it to the filter graph.
    hr = CoCreateInstance(&CLSID_VideoMixingRenderer, NULL,
                                  CLSCTX_INPROC, &
IID_IBaseFilter, (void**)&pVmr);

    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    hr = pGraphBuilder->lpVtbl->AddFilter(pGraphBuilder,
pVmr, L"Video Mixing Renderer");
    if( ! SUCCEEDED(hr))
    {
        return 0;
    }

    // Set the rendering mode and number of streams.

    hr = pVmr->lpVtbl->QueryInterface(pVmr, &
IID_IVMRFilterConfig, (void**)&pConfig);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    pConfig->lpVtbl->SetRenderingMode(pConfig,
VMRMode_Windowless);
    pConfig->lpVtbl->Release(pConfig);

    hr = pVmr->lpVtbl->QueryInterface(pVmr, &
IID_IVMRWindowlessControl, (void**)&m_pWC);
    if( ! SUCCEEDED(hr))
    {
        return 0;
    }

    m_pWC->lpVtbl->SetVideoClippingWindow(m_pWC, hWnd);

    pVmr->lpVtbl->Release(pVmr);

    return 1;
}
//-----------------------------------------------------------------

#if 0
HRESULT CVMR_Capture::InitializeWindowlessVMR(HWND hWnd)
{
    IBaseFilter* pVmr = NULL;

    // Create the VMR and add it to the filter graph.
    HRESULT hr = CoCreateInstance(CLSID_VideoMixingRenderer
, NULL,
                                  CLSCTX_INPROC,
IID_IBaseFilter, (void**)&pVmr);
```

```
    if (SUCCEEDED(hr))
    {
        hr = m_pGB->AddFilter(pVmr, L"Video Mixing
Renderer");
        if (SUCCEEDED(hr))
        {
            // Set the rendering mode and number of streams.
            IVMRFilterConfig* pConfig;

            hr = pVmr->QueryInterface(IID_IVMRFilterConfig,
 (void**)&pConfig);
            if( SUCCEEDED(hr))
            {
                pConfig->SetRenderingMode(
VMRMode_Windowless);
                pConfig->Release();
            }

            hr = pVmr->QueryInterface(
IID_IVMRWindowlessControl, (void**)&m_pWC);
            if( SUCCEEDED(hr))
            {
                m_pWC->SetVideoClippingWindow(hWnd);

            }
        }
        pVmr->Release();
    }

    return hr;
}
#endif


//----------------------------------------------------------------
int    BindFilter(int deviceId, IBaseFilter **pFilter)
{
    // enumerate all video capture devices
    ICreateDevEnum *pCreateDevEnum;
    IEnumMoniker   *pEm;
    HRESULT        hr;
    ULONG cFetched;
    IMoniker *pM;
    IPropertyBag *pBag;
    int index = 0;

    hr = CoCreateInstance(&CLSID_SystemDeviceEnum, NULL,
CLSCTX_INPROC_SERVER,
                            &IID_ICreateDevEnum, (void**)&pCreateDevEnum);
    if (hr != NOERROR)
    {
        return 0;
    }

    hr = pCreateDevEnum->lpVtbl->CreateClassEnumerator(pCreateDevEnum, &CLSID_VideoInputDeviceCategory, &pEm, 0);
    if( hr != NOERROR)
    {
        return 0;
    }

    pEm->Reset();
    index = 0;

    while(hr = pEm->lpVtbl->Next(pEm, 1, &pM, &cFetched), hr==S_OK, index <= deviceId)
    {
        hr = pM->lpVtbl->BindToStorage(pM, 0, 0, &IID_IPropertyBag, (void **)&pBag);
        if(SUCCEEDED(hr))
        {
            VARIANT var;
            var.vt = VT_BSTR;
            hr = pBag->lpVtbl->Read(pBag,  L"FriendlyName", &var, NULL);
            if (hr == NOERROR)
            {
                if (index == deviceId)
                {
                    pM->lpVtbl->BindToObject(pM, 0, 0, &IID_IBaseFilter, (void**)pFilter);
                }
                //SysFreeString(var.bstrVal);
            }
            pBag->lpVtbl->Release(pBag);
        }
        pM->lpVtbl->Release(pM);
        index++;
    }

    return 1;
}
//---------------------------------------------------------------

#if 0
```

```
bool CVMR_Capture::BindFilter(int deviceId, IBaseFilter **
pFilter)
{
    if (deviceId < 0)
        return false;

    // enumerate all video capture devices
    CComPtr<ICreateDevEnum> pCreateDevEnum;
    HRESULT hr = CoCreateInstance(CLSID_SystemDeviceEnum,
NULL, CLSCTX_INPROC_SERVER,
              IID_ICreateDevEnum, (void**)&pCreateDevEnum);
    if (hr != NOERROR)
    {

        return false;
    }

    CComPtr<IEnumMoniker> pEm;
    hr = pCreateDevEnum->CreateClassEnumerator(
CLSID_VideoInputDeviceCategory,
                                &pEm, 0);
    if (hr != NOERROR)
    {
        return false;
    }

    pEm->Reset();
    ULONG cFetched;
    IMoniker *pM;
    int index = 0;
    while(hr = pEm->Next(1, &pM, &cFetched), hr==S_OK,
index <= deviceId)
    {
        IPropertyBag *pBag;
        hr = pM->BindToStorage(0, 0, IID_IPropertyBag, (
void **)&pBag);
        if(SUCCEEDED(hr))
        {
            VARIANT var;
            var.vt = VT_BSTR;
            hr = pBag->Read(L"FriendlyName", &var, NULL);
            if (hr == NOERROR)
            {
                if (index == deviceId)
                {
                    pM->BindToObject(0, 0, IID_IBaseFilter,
 (void**)pFilter);
                }
                SysFreeString(var.bstrVal);
            }
            pBag->Release();
        }
        pM->Release();
        index++;
    }
    return true;
}
#endif


//-------------------------------------------------------------------
int     InitVideoWindow(HWND hWnd, int *pWidth, int * pHeight)
{
    HRESULT hr;
    RECT rcDest;
    IAMStreamConfig  *pConfig;
    IEnumMediaTypes *pMedia;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;
    VIDEOINFOHEADER *vih;
    char    tmpbuf[1000];


    hr = m_pCamOutPin->lpVtbl->EnumMediaTypes(m_pCamOutPin, &pMedia);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    while(pMedia->lpVtbl->Next(pMedia, 1, &pmt, 0) == S_OK)
    {
        //if( pmt->formattype == FORMAT_VideoInfo )
        if( memcmp((void *)& pmt->formattype, (void *)& FORMAT_VideoInfo, sizeof(GUID)) == 0 )
        {

            //  MessageBox(NULL, "here", "debug", MB_OK);

            vih = (VIDEOINFOHEADER *)pmt->pbFormat;
            //if( vih->bmiHeader.biWidth == width && vih->bmiHeader.biHeight == height )
            {
                pfnt = pmt;

                //sprintf(tmpbuf, "found mediatype with %i %i\n",
vih->bmiHeader.biWidth, vih->bmiHeader.biHeight );
                //MessageBox(NULL, tmpbuf, "debug", MB_OK);
                //char test[100];
                //sprintf(test,"Width=%d\nHeight=%d",vih->bmiHeader.biWidth, vih->bmiHeader.biHeight);
                //MessageBox(test);
                break;
            }
            //DeleteMediaType( pmt );
            //CoTaskMemFree((void *)pmt);
        }
    }

    pMedia->lpVtbl->Release(pMedia);

    hr = m_pCamOutPin->lpVtbl->QueryInterface(m_pCamOutPin, &IID_IAMStreamConfig, (void **)&pConfig);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    if( ! pfnt )
    {
        return 0;
    }

    hr = pConfig->lpVtbl->SetFormat(pConfig, pfnt);

    //if(SUCCEEDED(hr))
    //MessageBox("OK");

    //DeleteMediaType(pfnt);
    CoTaskMemFree((void *)pfnt);

    hr = pConfig->lpVtbl->GetFormat(pConfig, &pfnt);
    if( ! SUCCEEDED(hr) )
    {
        return 0;
    }

    *pWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biWidth;

    *pHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat)->bmiHeader.biHeight;

    //DeleteMediaType( pfnt );
    CoTaskMemFree((void *)pfnt);

    // sets to window dims
    //GetClientRect(hWnd, &rcDest);
    //hr = m_pWC->lpVtbl->SetVideoPosition(m_pWC, NULL, &rcDest);

    // sets to webcam dims
    rcDest.left = 0;
    rcDest.top = 0;
    rcDest.right = *pWidth;
    rcDest.bottom = *pHeight;
    hr = m_pWC->lpVtbl->SetVideoPosition(m_pWC, NULL, & rcDest);

    return 1;
}
//-----------------------------------------------------------------


#if 0
HRESULT CVMR_Capture::InitVideoWindow(HWND hWnd,int width, int height)
{

    // Set the grabbing size
    // First we iterate through the available media types and
    // store the first one that fits the requested size.
    // If we have found one, we set it.
    // In any case we query the size of the current media type
    // to have this information for clients of this class.
    //     Gerhard Reitmayr <reitmayr@i ..............>

    HRESULT hr;
    RECT rcDest;

    CComPtr<IAMStreamConfig> pConfig;
    IEnumMediaTypes *pMedia;
    AM_MEDIA_TYPE *pmt = NULL, *pfnt = NULL;

    hr = m_pCamOutPin->EnumMediaTypes( &pMedia );
    if(SUCCEEDED(hr))
    {

        while(pMedia->Next(1, &pmt, 0) == S_OK)
        {
            if( pmt->formattype == FORMAT_VideoInfo )
            {
                VIDEOINFOHEADER *vih = (VIDEOINFOHEADER *)
pmt->pbFormat;
                // printf("Size %i  %i\n", vih->bmiHeader.biWidth,
vih->bmiHeader.biHeight );
                if( vih->bmiHeader.biWidth == width && vih
->bmiHeader.biHeight == height )
                {
                    pfnt = pmt;

                    // printf("found mediatype with %i %i\n",
vih->bmiHeader.biWidth, vih->bmiHeader.biHeight );
                    //char test[100];

//sprintf(test,"Width=%d\nHeight=%d",vih->bmiHeader.biWidth, vih->bmiHeader.biHeight);
                    //MessageBox(test);
                    break;
                }
                DeleteMediaType( pmt );
            }
        }
        pMedia->Release();
    }
    hr = m_pCamOutPin->QueryInterface( IID_IAMStreamConfig,
 (void **) &pConfig );
    if(SUCCEEDED(hr))
    {
        if( pfnt != NULL )
        {
            hr=pConfig->SetFormat( pfnt );

            //if(SUCCEEDED(hr))
            //MessageBox("OK");

            DeleteMediaType( pfnt );
        }
        hr = pConfig->GetFormat( &pfnt );
        if(SUCCEEDED(hr))
        {

            m_nWidth = ((VIDEOINFOHEADER *)pfnt->pbFormat
)->bmiHeader.biWidth;
            m_nHeight = ((VIDEOINFOHEADER *)pfnt->pbFormat
)->bmiHeader.biHeight;

            DeleteMediaType( pfnt );
        }
    }
    ::GetClientRect (hWnd,&rcDest);
    hr = m_pWC->SetVideoPosition(NULL, &rcDest);
    return hr;
}
#endif


//----------------------------------------------------------------
int     GrabWebCamFrame(unsigned char *pFrameOut, unsigned long FrameBufferLen, int *pWidth, int *pHeight)
{
    BYTE *lpCurrImage = NULL;
    LPBITMAPINFOHEADER  pdib;
    BYTE *pTemp32;
    unsigned long FrameSize24;
    unsigned long Height, Width;
    char    tmpbuf[1000];

    if( ! m_pWC )
    {
        MessageBox(NULL, "m_pWC failed", "E", MB_OK);
        return 0;
    }

    if( m_pWC->lpVtbl->GetCurrentImage(m_pWC, &lpCurrImage) != S_OK )
    {
        MessageBox(NULL, "Get current image failed OK", "E", MB_OK);
        return 0;
    }

    pdib = (LPBITMAPINFOHEADER)lpCurrImage;

    //m_nFramelen=pdib->biHeight * pdib->biWidth * 3;
    //m_pFrame=new BYTE [pdib->biHeight * pdib->biWidth * 3] ;
    //if(pdib->biBitCount ==32)
    //MessageBox(NULL, "OK", "E", MB_OK);

    pTemp32 = lpCurrImage + sizeof(BITMAPINFOHEADER);

    //change from 32 to 24 bit /pixel
    //this->Convert24Image(pTemp32, m_pFrame, pdib->biSizeImage);

    Height = pdib->biHeight;
    Width = pdib->biWidth;
    FrameSize24 = Width*Height*3L;

    *pWidth = Width;
    *pHeight = Height;

    // pdib->biSizeImage is 4 bytes/pixel
    //if( pdib->biSizeImage > FrameBufferLen )
    if( FrameSize24 > FrameBufferLen )
    {
        MessageBox(NULL, "Frame size failed failed OK", "E"
, MB_OK);
        CoTaskMemFree(lpCurrImage); //free the image
        return 0;
    }

    //memcpy((void *)pFrameOut, pTemp32, pdib->biSizeImage);

    Convert24Image(pTemp32, pFrameOut, pdib->biSizeImage);

    CoTaskMemFree(lpCurrImage); //free the image

    //sprintf(tmpbuf, "grab frame, %d x %d, size %ld", *pWidth, *pHeight,
pdib->biSizeImage);
    //MessageBox(NULL, tmpbuf, "debug", MB_OK);

    return 1;
}
//------------------------------------------------------------------


#if 0
DWORD CVMR_Capture::GrabFrame()
{
    long lOut=-1;
    if(m_pWC )
    {
       BYTE* lpCurrImage = NULL;

        // Read the current video frame into a byte buffer.  The information
        // will be returned in a packed Windows DIB and will be allocated
        // by the VMR.
        if(m_pWC->GetCurrentImage(&lpCurrImage) == S_OK)
        {

            LPBITMAPINFOHEADER  pdib = (LPBITMAPINFOHEADER)
 lpCurrImage;

            if(m_pFrame==NULL || (pdib->biHeight * pdib->
biWidth * 3) !=m_nFramelen )
            {
                if(m_pFrame!=NULL)
                delete []m_pFrame;

                m_nFramelen=pdib->biHeight * pdib->biWidth * 3;
                m_pFrame=new BYTE [pdib->biHeight * pdib->biWidth * 3] ;


            }

            if(pdib->biBitCount ==32)
            {
                DWORD  dwSize=0, dwWritten=0;

                BYTE *pTemp32;
                pTemp32=lpCurrImage + sizeof(BITMAPINFOHEADER);

                //change from 32 to 24 bit /pixel
                this->Convert24Image(pTemp32, m_pFrame, pdib->biSizeImage);
            }

            CoTaskMemFree(lpCurrImage); //free the image
        }
        else
        {
            return lOut;
        }

    }
    else
    {
        return lOut;
    }




    return lOut=m_nFramelen;

}
#endif
//----------------------------------------------------------------------

//----------------------------------------------------------------------
#if 0
int     Convert24Image(BYTE *p32Img, BYTE *p24Img, DWORD dwSize32)
```

```
{
    DWORD dwSize24;
    BYTE *pTemp,*ptr;
    DWORD index;
    unsigned char r, g, b;

    if( (! p32Img) || (! p24Img) || (dwSize32 == 0) )
    {
        return 0;
    }

    dwSize24=(dwSize32 * 3)/4;

    //pTemp=p32Img + sizeof(BITMAPINFOHEADER); ;
    pTemp=p32Img;

    ptr=p24Img + dwSize24-1 ;

    //int ival=0;
    for (index = 0; index < dwSize32/4 ; index++)
    {
        r = *(pTemp++);
        g = *(pTemp++);
        b = *(pTemp++);
        (pTemp++);//skip alpha

        *(ptr--) = b;
        *(ptr--) = g;
        *(ptr--) = r;
    }
    return 1;
}
#endif
//--------------------------------------------------------------------

//--------------------------------------------------------------------
// JL fixed up
int     Convert24Image(BYTE *p32Img, BYTE *p24Img, DWORD dwSize32)
{
    DWORD dwSize24;
    BYTE *pTemp,*ptr;
    DWORD index;
    unsigned char r, g, b;

    if( (! p32Img) || (! p24Img) || (dwSize32 == 0) )
    {
        return 0;
    }

    dwSize24=(dwSize32 * 3)/4;

    //pTemp=p32Img + sizeof(BITMAPINFOHEADER); ;
    pTemp=p32Img;

    //ptr=p24Img + dwSize24-1 ;
    ptr=p24Img;

    //int ival=0;
    for (index = 0; index < dwSize32/4 ; index++)
    {
        *ptr++ = *pTemp++;
        *ptr++ = *pTemp++;
        *ptr++ = *pTemp++;
        pTemp++;

        /**
   r = *(pTemp++);
   g = *(pTemp++);
   b = *(pTemp++);
   (pTemp++);//skip alpha

   *(ptr++) = b;
   *(ptr++) = g;
   *(ptr++) = r;
   **/
    }
    return 1;
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
#if 0
bool CVMR_Capture::Convert24Image(BYTE *p32Img, BYTE *p24Img,DWORD dwSize32)
{

    if(p32Img != NULL && p24Img != NULL && dwSize32>0)
    {

        DWORD dwSize24;

        dwSize24=(dwSize32 * 3)/4;

        BYTE *pTemp,*ptr;
        //pTemp=p32Img + sizeof(BITMAPINFOHEADER); ;
        pTemp=p32Img;

        ptr=p24Img + dwSize24-1 ;

        int ival=0;
        for (DWORD index = 0; index < dwSize32/4 ; index++)
        {
            unsigned char r = *(pTemp++);
            unsigned char g = *(pTemp++);
            unsigned char b = *(pTemp++);
            (pTemp++);//skip alpha

            *(ptr--) = b;
            *(ptr--) = g;
            *(ptr--) = r;

        }
    }
    else
    {
        return false;
    }

return true;
}
#endif
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
int     StopWebCamCapture(void)
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    hr = m_pMC->lpVtbl->Stop(m_pMC);
    if( FAILED(hr) )
    {
        return 0;
    }

    return 0;
}
//---------------------------------------------------------------------------
```

```

#if 0
void CVMR_Capture::StopCapture()
{
    HRESULT hr;

    if((m_psCurrent == Paused) || (m_psCurrent == Running))
    {
        LONGLONG pos = 0;
        hr = m_pMC->Stop();
        m_psCurrent = Stopped;
        // Display the first frame to indicate the reset condition
        hr = m_pMC->Pause();
    }
}
#endif

//----------------------------------------------------------------
int     PauseWebCamCapture(void)
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    if( WebCamPaused )
    {
        return 0;
    }

    hr = m_pMC->lpVtbl->Pause(m_pMC);
    if( FAILED(hr) )
    {
        return 0;
    }

    //----------------------------------------------------------------
    WebCamPaused = 1;
    //----------------------------------------------------------------

    return 0;
}
//----------------------------------------------------------------

//----------------------------------------------------------------
int     ResumeWebCamCapture(void)
```

```
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    if( ! WebCamPaused )
    {
        return 0;
    }

    hr = m_pMC->lpVtbl->Run(m_pMC);
    if( FAILED(hr) )
    {
        return 0;
    }

    //-------------------------------------------------------------------
    WebCamPaused = 0;
    //-------------------------------------------------------------------

    return 0;
}
//-------------------------------------------------------------------

//-------------------------------------------------------------------
int     CloseWebCamCapture(void)
{
    HRESULT hr;

    if( ! WebCamInitialized )
    {
        return 0;
    }

    hr = m_pMC->lpVtbl->Stop(m_pMC);

    if( FAILED(hr) )
    {
        //MessageBox(NULL, "stop failed", "debug", MB_OK);
        //return 0;
    }

    //if (m_pME)
    //    hr = m_pME->SetNotifyWindow((OAHWND)NULL, 0, 0);

    if(m_pCamOutPin)
    {
        hr = m_pCamOutPin->lpVtbl->Disconnect(m_pCamOutPin);
        if( FAILED(hr) )
        {
            //MessageBox(NULL, "disconnect failed", "debug", MB_OK);
            //return 0;
        }
    }

    SAFE_RELEASE(m_pCamOutPin);
    SAFE_RELEASE(pIGraphBuilder);
    /*
  SAFE_RELEASE(m_pGB);
  */
    SAFE_RELEASE(pIBaseFilter);
    SAFE_RELEASE(m_pWC);
    SAFE_RELEASE(m_pMC);

    WebCamInitialized = 0;

    //MessageBox(NULL, "stop", "debug", MB_OK);

    return 1;
}
//------------------------------------------------------------------

#if 0
void CVMR_Capture::CloseInterfaces(void)
{
    HRESULT hr;


    // Stop media playback
    if(m_pMC)
        hr = m_pMC->Stop();
    m_psCurrent = Stopped;

    // Disable event callbacks
/*
  if (pME)
     hr = m_pME->SetNotifyWindow((OAHWND)NULL, 0, 0);
*/

//  SAFE_RELEASE(pME);



```

```
        // Release and zero DirectShow interfaces
        if(m_pCamOutPin)
        m_pCamOutPin->Disconnect ();


        SAFE_RELEASE(m_pCamOutPin);
        SAFE_RELEASE(m_pMC);
        SAFE_RELEASE(m_pGB);
        SAFE_RELEASE(m_pWC);
        SAFE_RELEASE(m_pDF);
#endif

```

D.3 WebCamMin.h Source

```
// WebCamMin.h
// see windmo.h


//------------------------------------------------------------------
// windows defs from SDK
//------------------------------------------------------------------


//------------------------------------------------------------------
#pragma pack(1)
//------------------------------------------------------------------

//------------------------------------------------------------------
#define SAFE_RELEASE(x) { if (x) x->Release(); x = NULL; }
//------------------------------------------------------------------

//------------------------------------------------------------------
// guiddef.h
/*****
#pragma pack(1)
typedef struct _GUID
{
    unsigned long  Data1;
    unsigned short Data2;
    unsigned short Data3;
    unsigned char  Data4[8];
} GUID;
*****/
//------------------------------------------------------------------

//------------------------------------------------------------------
typedef LONGLONG REFERENCE_TIME;
typedef DWORD_PTR HEVENT;
typedef DWORD_PTR HSEMAPHORE;
//------------------------------------------------------------------

//------------------------------------------------------------------
typedef DWORD *LPDIRECTDRAWSURFACE7;
typedef DWORD *LPDDPIXELFORMAT;
//typedef DWORD *LPBITMAPINFOHEADER;
//------------------------------------------------------------------

//------------------------------------------------------------------
typedef  enum _FilterState
{   State_Stopped   = 0,
    State_Paused    = ( State_Stopped + 1 ) ,
    State_Running   = ( State_Paused + 1 )
} FILTER_STATE;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
typedef enum
    {   VMRMode_Windowed    = 0x1,
    VMRMode_Windowless  = 0x2,
    VMRMode_Renderless  = 0x4,
    VMRMode_Mask    = 0x7
    }   VMRMode;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
typedef struct _AMMediaType
    {
    GUID majortype;
    GUID subtype;
    BOOL bFixedSizeSamples;
    BOOL bTemporalCompression;
    ULONG lSampleSize;
    GUID formattype;
    IUnknown *pUnk;
    ULONG cbFormat;
    BYTE *pbFormat;
    }   AM_MEDIA_TYPE;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
typedef enum _PinDirection
{   PINDIR_INPUT    = 0,
    PINDIR_OUTPUT   = ( PINDIR_INPUT + 1 )
}   PIN_DIRECTION;
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
#define __RPC__in
#define __RPC__out
#define __RPC__deref_out
#define __RPC__in_opt
#define __RPC__deref_out_opt
//#define __RPC__inout_ecount_full(cOutputBufferCount)

#define __out
#define __out_opt
#define __in_opt
#define __deref_out_opt
#define __in
//---------------------------------------------------------------------------
```

```


//-----------------------------------------------------------------------
// from amvideo.h
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// The BITMAPINFOHEADER contains all the details about the video stream such
// as the actual image dimensions and their pixel depth. A source filter may
// also request that the sink take only a section of the video by providing a
// clipping rectangle in rcSource. In the worst case where the sink filter
// forgets to check this on connection it will simply render the whole thing
// which isn't a disaster. Ideally a sink filter will check the rcSource and
// if it doesn't support image extraction and the rectangle is not empty then
// it will reject the connection. A filter should use SetRectEmpty to reset a
// rectangle to all zeroes (and IsRectEmpty to later check the rectangle).
// The rcTarget specifies the destination rectangle for the video, for most
// source filters they will set this to all zeroes, a downstream filter may
// request that the video be placed in a particular area of the buffers it
// supplies in which case it will call QueryAccept with a non empty target

typedef struct tagVIDEOINFOHEADER {

    RECT                rcSource;          // The bit we really want to use
    RECT                rcTarget;          // Where the video should go
    DWORD               dwBitRate;         // Approximate bit data rate
    DWORD               dwBitErrorRate;    // Bit error rate for this stream
    REFERENCE_TIME      AvgTimePerFrame;   // Average time per frame
(100ns units)

    BITMAPINFOHEADER bmiHeader;

} VIDEOINFOHEADER;
//-----------------------------------------------------------------------


//-----------------------------------------------------------------------
// from ocidl.h
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IErrorLogVtbl *lpVtbl;
}IErrorLog ;

typedef struct IErrorLogVtbl {
 BEGIN_INTERFACE
 HRESULT (_stdcall *QueryInterface)(IErrorLog *,REFIID,void * *);
 ULONG (_stdcall *AddRef)(IErrorLog *);
 ULONG (_stdcall *Release)(IErrorLog *);
 HRESULT (_stdcall *AddError)(IErrorLog *,LPCOLESTR, EXCEPINFO *);
END_INTERFACE
} IErrorLogVtbl;

//----------------------------------------------------------------------


//----------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IPropertyBagVtbl *lpVtbl;
}  IPropertyBag ;

typedef struct IPropertyBagVtbl {
 BEGIN_INTERFACE
 HRESULT (_stdcall *QueryInterface)(IPropertyBag *,REFIID, void * *);
 ULONG (_stdcall *AddRef)(IPropertyBag *);
 ULONG (_stdcall *Release)(IPropertyBag *);
 HRESULT (_stdcall *Read)(IPropertyBag *, LPCOLESTR, VARIANT *,IErrorLog *);
 HRESULT (_stdcall *Write)(IPropertyBag *,LPCOLESTR,VARIANT *);
END_INTERFACE
} IPropertyBagVtbl;
//----------------------------------------------------------------------



//----------------------------------------------------------------------
// from strmif.h
//----------------------------------------------------------------------

//----------------------------------------------------------------------
typedef /* [public][public][public][public][public][public] */ struct __MIDL___MIDL_itf_strmif_0000_0123_0001
    {
    DWORD dw1;
    DWORD dw2;
    }   DDCOLORKEY;

typedef DDCOLORKEY *LPDDCOLORKEY;
//----------------------------------------------------------------------
```

```

//---------------------------------------------------------------------
typedef struct _NORMALIZEDRECT
    {
    float left;
    float top;
    float right;
    float bottom;
    }   NORMALIZEDRECT;

typedef struct _NORMALIZEDRECT *PNORMALIZEDRECT;
//---------------------------------------------------------------------

//---------------------------------------------------------------------
typedef struct _VMRVIDEOSTREAMINFO
    {
    LPDIRECTDRAWSURFACE7 pddsVideoSurface;
    DWORD dwWidth;
    DWORD dwHeight;
    DWORD dwStrmID;
    FLOAT fAlpha;
    DDCOLORKEY ddClrKey;
    NORMALIZEDRECT rNormal;
    }   VMRVIDEOSTREAMINFO;
//---------------------------------------------------------------------



//---------------------------------------------------------------------
// forward declarations
typedef struct _IFilterGraph IFilterGraph;
typedef struct _PinInfo PIN_INFO;
//---------------------------------------------------------------------

//---------------------------------------------------------------------
typedef struct _FilterInfo
{
    WCHAR achName[ 128 ];
    IFilterGraph *pGraph;
}   FILTER_INFO;
//---------------------------------------------------------------------


//---------------------------------------------------------------------
//interface IReferenceClock
typedef struct
{
    CONST_VTBL struct IReferenceClockVtbl *lpVtbl;
} IReferenceClock;
//----------------------------------------------------------------------

//----------------------------------------------------------------------
typedef struct IReferenceClockVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IReferenceClock * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IReferenceClock * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IReferenceClock * This);

    HRESULT ( STDMETHODCALLTYPE *GetTime )(
        IReferenceClock * This,
        /* [out] */
        __out  REFERENCE_TIME *pTime);

    HRESULT ( STDMETHODCALLTYPE *AdviseTime )(
        IReferenceClock * This,
        /* [in] */ REFERENCE_TIME baseTime,
        /* [in] */ REFERENCE_TIME streamTime,
        /* [in] */ HEVENT hEvent,
        /* [out] */
        __out  DWORD_PTR *pdwAdviseCookie);

    HRESULT ( STDMETHODCALLTYPE *AdvisePeriodic )(
        IReferenceClock * This,
        /* [in] */ REFERENCE_TIME startTime,
        /* [in] */ REFERENCE_TIME periodTime,
        /* [in] */ HSEMAPHORE hSemaphore,
        /* [out] */
        __out  DWORD_PTR *pdwAdviseCookie);

    HRESULT ( STDMETHODCALLTYPE *Unadvise )(
        IReferenceClock * This,
        /* [in] */ DWORD_PTR dwAdviseCookie);

    END_INTERFACE
} IReferenceClockVtbl;
//----------------------------------------------------------------------
```

```


//-------------------------------------------------------------------
//interface IEnumMediaTypes
typedef struct IEnumMediaTypes
{
    CONST_VTBL struct IEnumMediaTypesVtbl *lpVtbl;
} IEnumMediaTypes;
//-------------------------------------------------------------------

//-------------------------------------------------------------------
typedef struct IEnumMediaTypesVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IEnumMediaTypes * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IEnumMediaTypes * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IEnumMediaTypes * This);

    HRESULT ( STDMETHODCALLTYPE *Next )(
        IEnumMediaTypes * This,
        /* [in] */ ULONG cMediaTypes,
        /* [size_is][out] */
        //__out_ecount_part(cMediaTypes, *pcFetched)  AM_MEDIA_TYPE
**ppMediaTypes,
        AM_MEDIA_TYPE **ppMediaTypes,
        /* [out] */
        __out_opt  ULONG *pcFetched);

    HRESULT ( STDMETHODCALLTYPE *Skip )(
        IEnumMediaTypes * This,
        /* [in] */ ULONG cMediaTypes);

    HRESULT ( STDMETHODCALLTYPE *Reset )(
        IEnumMediaTypes * This);

    HRESULT ( STDMETHODCALLTYPE *Clone )(
        IEnumMediaTypes * This,
        /* [out] */
        __out  IEnumMediaTypes **ppEnum);

    END_INTERFACE
} IEnumMediaTypesVtbl;
//------------------------------------------------------------------------


//------------------------------------------------------------------------
//interface IPin
typedef struct
{
    CONST_VTBL struct IPinVtbl *lpVtbl;
} IPin;
//------------------------------------------------------------------------

//------------------------------------------------------------------------
typedef struct IPinVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IPin * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IPin * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IPin * This);

    HRESULT ( STDMETHODCALLTYPE *Connect )(
        IPin * This,
        /* [in] */ IPin *pReceivePin,
        /* [in] */
        __in_opt  const AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *ReceiveConnection )(
        IPin * This,
        /* [in] */ IPin *pConnector,
        /* [in] */ const AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *Disconnect )(
        IPin * This);

    HRESULT ( STDMETHODCALLTYPE *ConnectedTo )(
        IPin * This,
        /* [out] */
        __out  IPin **pPin);

    HRESULT ( STDMETHODCALLTYPE *ConnectionMediaType )(
        IPin * This,
        /* [out] */
        __out  AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *QueryPinInfo )(
        IPin * This,
        /* [out] */
        __out  PIN_INFO *pInfo);

    HRESULT ( STDMETHODCALLTYPE *QueryDirection )(
        IPin * This,
        /* [out] */
        __out  PIN_DIRECTION *pPinDir);

    HRESULT ( STDMETHODCALLTYPE *QueryId )(
        IPin * This,
        /* [out] */
        __out  LPWSTR *Id);

    HRESULT ( STDMETHODCALLTYPE *QueryAccept )(
        IPin * This,
        /* [in] */ const AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *EnumMediaTypes )(
        IPin * This,
        /* [out] */
        __out  IEnumMediaTypes **ppEnum);

    HRESULT ( STDMETHODCALLTYPE *QueryInternalConnections )(
        IPin * This,
        /* [out] */
        //__out_ecount_part_opt(*nPin, *nPin)  IPin **apPin,
        IPin **apPin,
        /* [out][in] */ ULONG *nPin);

    HRESULT ( STDMETHODCALLTYPE *EndOfStream )(
        IPin * This);

    HRESULT ( STDMETHODCALLTYPE *BeginFlush )(
        IPin * This);

    HRESULT ( STDMETHODCALLTYPE *EndFlush )(
        IPin * This);

    HRESULT ( STDMETHODCALLTYPE *NewSegment )(
        IPin * This,
        /* [in] */ REFERENCE_TIME tStart,
        /* [in] */ REFERENCE_TIME tStop,
        /* [in] */ double dRate);

    END_INTERFACE
} IPinVtbl;
//--------------------------------------------------------------------

//--------------------------------------------------------------------
//interface IEnumPins
typedef struct
{
    CONST_VTBL struct IEnumPinsVtbl *lpVtbl;
} IEnumPins;
//--------------------------------------------------------------------

//--------------------------------------------------------------------
typedef struct IEnumPinsVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IEnumPins * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IEnumPins * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IEnumPins * This);

    HRESULT ( STDMETHODCALLTYPE *Next )(
        IEnumPins * This,
        /* [in] */ ULONG cPins,
        /* [size_is][out] */
        //__out_ecount_part(cPins, *pcFetched)  IPin **ppPins,
        IPin **ppPins,
        /* [out] */
        __out_opt  ULONG *pcFetched);

    HRESULT ( STDMETHODCALLTYPE *Skip )(
        IEnumPins * This,
        /* [in] */ ULONG cPins);

    HRESULT ( STDMETHODCALLTYPE *Reset )(
        IEnumPins * This);

    HRESULT ( STDMETHODCALLTYPE *Clone )(
        IEnumPins * This,
        /* [out] */
        __out  IEnumPins **ppEnum);

    END_INTERFACE
} IEnumPinsVtbl;
//---------------------------------------------------------------------

//---------------------------------------------------------------------
//interface IBaseFilter
typedef struct
{
    CONST_VTBL struct IBaseFilterVtbl *lpVtbl;
} IBaseFilter;


typedef struct IBaseFilterVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IBaseFilter * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IBaseFilter * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IBaseFilter * This);

    HRESULT ( STDMETHODCALLTYPE *GetClassID )(
        IBaseFilter * This,
        /* [out] */ CLSID *pClassID);

    HRESULT ( STDMETHODCALLTYPE *Stop )(
        IBaseFilter * This);

    HRESULT ( STDMETHODCALLTYPE *Pause )(
        IBaseFilter * This);

    HRESULT ( STDMETHODCALLTYPE *Run )(
        IBaseFilter * This,
        REFERENCE_TIME tStart);

    HRESULT ( STDMETHODCALLTYPE *GetState )(
        IBaseFilter * This,
        /* [in] */ DWORD dwMilliSecsTimeout,
        /* [out] */
        __out  FILTER_STATE *State);

    HRESULT ( STDMETHODCALLTYPE *SetSyncSource )(
        IBaseFilter * This,
        /* [in] */
        __in_opt  IReferenceClock *pClock);

    HRESULT ( STDMETHODCALLTYPE *GetSyncSource )(
        IBaseFilter * This,
        /* [out] */
        __deref_out_opt  IReferenceClock **pClock);

    HRESULT ( STDMETHODCALLTYPE *EnumPins )(
        IBaseFilter * This,
        /* [out] */
        __out  IEnumPins **ppEnum);

    HRESULT ( STDMETHODCALLTYPE *FindPin )(
        IBaseFilter * This,
        /* [string][in] */ LPCWSTR Id,
        /* [out] */
        __out  IPin **ppPin);

    HRESULT ( STDMETHODCALLTYPE *QueryFilterInfo )(
        IBaseFilter * This,
        /* [out] */
        __out  FILTER_INFO *pInfo);

    HRESULT ( STDMETHODCALLTYPE *JoinFilterGraph )(
        IBaseFilter * This,
        /* [in] */
        __in_opt  IFilterGraph *pGraph,
        /* [string][in] */
        __in_opt  LPCWSTR pName);

    HRESULT ( STDMETHODCALLTYPE *QueryVendorInfo )(
        IBaseFilter * This,
        /* [string][out] */
        __out  LPWSTR *pVendorInfo);

    END_INTERFACE
} IBaseFilterVtbl;
```

```
//------------------------------------------------------------------

//------------------------------------------------------------------
typedef struct _PinInfo
{
    IBaseFilter *pFilter;
    PIN_DIRECTION dir;
    WCHAR achName[ 128 ];
}   PIN_INFO;
//------------------------------------------------------------------

//------------------------------------------------------------------
//interface IEnumFilters
typedef struct
{
    CONST_VTBL struct IEnumFiltersVtbl *lpVtbl;
}IEnumFilters;
//------------------------------------------------------------------

//------------------------------------------------------------------
typedef struct IEnumFiltersVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IEnumFilters * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IEnumFilters * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IEnumFilters * This);

    HRESULT ( STDMETHODCALLTYPE *Next )(
        IEnumFilters * This,
        /* [in] */ ULONG cFilters,
        /* [out] */
        //__out_ecount_part(cFilters, *pcFetched)  IBaseFilter **ppFilter,
        IBaseFilter **ppFilter,
        /* [out] */
        __out  ULONG *pcFetched);

    HRESULT ( STDMETHODCALLTYPE *Skip )(
        IEnumFilters * This,
        /* [in] */ ULONG cFilters);

    HRESULT ( STDMETHODCALLTYPE *Reset )(
        IEnumFilters * This);

    HRESULT ( STDMETHODCALLTYPE *Clone )(
        IEnumFilters * This,
        /* [out] */
        __out  IEnumFilters **ppEnum);

    END_INTERFACE
} IEnumFiltersVtbl;
//-----------------------------------------------------------------



//-----------------------------------------------------------------
typedef struct IFilterGraphVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IFilterGraph * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IFilterGraph * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IFilterGraph * This);

    HRESULT ( STDMETHODCALLTYPE *AddFilter )(
        IFilterGraph * This,
        /* [in] */ IBaseFilter *pFilter,
        /* [string][in] */ LPCWSTR pName);

    HRESULT ( STDMETHODCALLTYPE *RemoveFilter )(
        IFilterGraph * This,
        /* [in] */ IBaseFilter *pFilter);

    HRESULT ( STDMETHODCALLTYPE *EnumFilters )(
        IFilterGraph * This,
        /* [out] */
        __out  IEnumFilters **ppEnum);

    HRESULT ( STDMETHODCALLTYPE *FindFilterByName )(
        IFilterGraph * This,
        /* [string][in] */ LPCWSTR pName,
        /* [out] */
        __out  IBaseFilter **ppFilter);

    HRESULT ( STDMETHODCALLTYPE *ConnectDirect )(
        IFilterGraph * This,
        /* [in] */ IPin *ppinOut,
        /* [in] */ IPin *ppinIn,
        /* [unique][in] */
        __in_opt  const AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *Reconnect )(
        IFilterGraph * This,
        /* [in] */ IPin *ppin);

    HRESULT ( STDMETHODCALLTYPE *Disconnect )(
        IFilterGraph * This,
        /* [in] */ IPin *ppin);

    HRESULT ( STDMETHODCALLTYPE *SetDefaultSyncSource )(
        IFilterGraph * This);

    END_INTERFACE
} IFilterGraphVtbl;


//interface IFilterGraph
typedef struct _IFilterGraph
{
    CONST_VTBL struct IFilterGraphVtbl *lpVtbl;
} IFilterGraph;
//-----------------------------------------------------------------


//-----------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IGraphBuilderVtbl *lpVtbl;
}IGraphBuilder;


typedef struct IGraphBuilderVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IGraphBuilder *This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */ 
        __RPC__deref_out  void **ppvObject);
        
    ULONG ( STDMETHODCALLTYPE *AddRef )( 
        IGraphBuilder * This);
        
    ULONG ( STDMETHODCALLTYPE *Release )( 
        IGraphBuilder * This);
        
    HRESULT ( STDMETHODCALLTYPE *AddFilter )( 
        IGraphBuilder * This,
        /* [in] */ IBaseFilter *pFilter,
        /* [string][in] */ LPCWSTR pName);
        
    HRESULT ( STDMETHODCALLTYPE *RemoveFilter )( 
        IGraphBuilder * This,
        /* [in] */ IBaseFilter *pFilter);
        
    HRESULT ( STDMETHODCALLTYPE *EnumFilters )( 
        IGraphBuilder * This,
        /* [out] */ 
        __out  IEnumFilters **ppEnum);
        
    HRESULT ( STDMETHODCALLTYPE *FindFilterByName )( 
        IGraphBuilder * This,
        /* [string][in] */ LPCWSTR pName,
        /* [out] */ 
        __out  IBaseFilter **ppFilter);
        
    HRESULT ( STDMETHODCALLTYPE *ConnectDirect )( 
        IGraphBuilder * This,
        /* [in] */ IPin *ppinOut,
        /* [in] */ IPin *ppinIn,
        /* [unique][in] */ 
        __in_opt  const AM_MEDIA_TYPE *pmt);
        
    HRESULT ( STDMETHODCALLTYPE *Reconnect )( 
        IGraphBuilder * This,
        /* [in] */ IPin *ppin);
        
    HRESULT ( STDMETHODCALLTYPE *Disconnect )( 
        IGraphBuilder * This,
        /* [in] */ IPin *ppin);
        
    HRESULT ( STDMETHODCALLTYPE *SetDefaultSyncSource )( 
        IGraphBuilder * This);
        
    HRESULT ( STDMETHODCALLTYPE *Connect )( 
        IGraphBuilder * This,
        /* [in] */ IPin *ppinOut,
        /* [in] */ IPin *ppinIn);

    HRESULT ( STDMETHODCALLTYPE *Render )(
        IGraphBuilder * This,
        /* [in] */ IPin *ppinOut);

    HRESULT ( STDMETHODCALLTYPE *RenderFile )(
        IGraphBuilder * This,
        /* [in] */ LPCWSTR lpcwstrFile,
        /* [unique][in] */
        __in_opt  LPCWSTR lpcwstrPlayList);

    HRESULT ( STDMETHODCALLTYPE *AddSourceFilter )(
        IGraphBuilder * This,
        /* [in] */ LPCWSTR lpcwstrFileName,
        /* [unique][in] */
        __in_opt  LPCWSTR lpcwstrFilterName,
        /* [out] */
        __out  IBaseFilter **ppFilter);

    HRESULT ( STDMETHODCALLTYPE *SetLogFile )(
        IGraphBuilder * This,
        /* [in] */ DWORD_PTR hFile);

    HRESULT ( STDMETHODCALLTYPE *Abort )(
        IGraphBuilder * This);

    HRESULT ( STDMETHODCALLTYPE *ShouldOperationContinue )(
        IGraphBuilder * This);

    END_INTERFACE
} IGraphBuilderVtbl;
//----------------------------------------------------------------------------------


//----------------------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IVMRImageCompositorVtbl *lpVtbl;
}IVMRImageCompositor;

typedef struct IVMRImageCompositorVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IVMRImageCompositor * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IVMRImageCompositor * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IVMRImageCompositor * This);

    HRESULT ( STDMETHODCALLTYPE *InitCompositionTarget )(
        IVMRImageCompositor * This,
        /* [in] */ IUnknown *pD3DDevice,
        /* [in] */ LPDIRECTDRAWSURFACE7 pddsRenderTarget);

    HRESULT ( STDMETHODCALLTYPE *TermCompositionTarget )(
        IVMRImageCompositor * This,
        /* [in] */ IUnknown *pD3DDevice,
        /* [in] */ LPDIRECTDRAWSURFACE7 pddsRenderTarget);

    HRESULT ( STDMETHODCALLTYPE *SetStreamMediaType )(
        IVMRImageCompositor * This,
        /* [in] */ DWORD dwStrmID,
        /* [in] */ AM_MEDIA_TYPE *pmt,
        /* [in] */ BOOL fTexture);

    HRESULT ( STDMETHODCALLTYPE *CompositeImage )(
        IVMRImageCompositor * This,
        /* [in] */ IUnknown *pD3DDevice,
        /* [in] */ LPDIRECTDRAWSURFACE7 pddsRenderTarget,
        /* [in] */ AM_MEDIA_TYPE *pmtRenderTarget,
        /* [in] */ REFERENCE_TIME rtStart,
        /* [in] */ REFERENCE_TIME rtEnd,
        /* [in] */ DWORD dwClrBkGnd,
        /* [in] */ VMRVIDEOSTREAMINFO *pVideoStreamInfo,
        /* [in] */ UINT cStreams);

    END_INTERFACE
} IVMRImageCompositorVtbl;
//------------------------------------------------------------------------


//------------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IVMRFilterConfigVtbl *lpVtbl;
}IVMRFilterConfig;
```

```

typedef struct IVMRFilterConfigVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IVMRFilterConfig * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IVMRFilterConfig * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IVMRFilterConfig * This);

    HRESULT ( STDMETHODCALLTYPE *SetImageCompositor )(
        IVMRFilterConfig * This,
        /* [in] */ IVMRImageCompositor *lpVMRImgCompositor);

    HRESULT ( STDMETHODCALLTYPE *SetNumberOfStreams )(
        IVMRFilterConfig * This,
        /* [in] */ DWORD dwMaxStreams);

    HRESULT ( STDMETHODCALLTYPE *GetNumberOfStreams )(
        IVMRFilterConfig * This,
        /* [out] */ DWORD *pdwMaxStreams);

    HRESULT ( STDMETHODCALLTYPE *SetRenderingPrefs )(
        IVMRFilterConfig * This,
        /* [in] */ DWORD dwRenderFlags);

    HRESULT ( STDMETHODCALLTYPE *GetRenderingPrefs )(
        IVMRFilterConfig * This,
        /* [out] */ DWORD *pdwRenderFlags);

    HRESULT ( STDMETHODCALLTYPE *SetRenderingMode )(
        IVMRFilterConfig * This,
        /* [in] */ DWORD Mode);

    HRESULT ( STDMETHODCALLTYPE *GetRenderingMode )(
        IVMRFilterConfig * This,
        /* [out] */ DWORD *pMode);

    END_INTERFACE
} IVMRFilterConfigVtbl;
//-------------------------------------------------------------
```

```

//---------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IVMRWindowlessControlVtbl *lpVtbl;
}IVMRWindowlessControl;

typedef struct IVMRWindowlessControlVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IVMRWindowlessControl * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IVMRWindowlessControl * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IVMRWindowlessControl * This);

    HRESULT ( STDMETHODCALLTYPE *GetNativeVideoSize )(
        IVMRWindowlessControl * This,
        /* [out] */ LONG *lpWidth,
        /* [out] */ LONG *lpHeight,
        /* [out] */ LONG *lpARWidth,
        /* [out] */ LONG *lpARHeight);

    HRESULT ( STDMETHODCALLTYPE *GetMinIdealVideoSize )(
        IVMRWindowlessControl * This,
        /* [out] */ LONG *lpWidth,
        /* [out] */ LONG *lpHeight);

    HRESULT ( STDMETHODCALLTYPE *GetMaxIdealVideoSize )(
        IVMRWindowlessControl * This,
        /* [out] */ LONG *lpWidth,
        /* [out] */ LONG *lpHeight);

    HRESULT ( STDMETHODCALLTYPE *SetVideoPosition )(
        IVMRWindowlessControl * This,
        /* [in] */ const LPRECT lpSRCRect,
        /* [in] */ const LPRECT lpDSTRect);

    HRESULT ( STDMETHODCALLTYPE *GetVideoPosition )(
        IVMRWindowlessControl * This,
        /* [out] */ LPRECT lpSRCRect,
        /* [out] */ LPRECT lpDSTRect);

    HRESULT ( STDMETHODCALLTYPE *GetAspectRatioMode )(
        IVMRWindowlessControl * This,
        /* [out] */ DWORD *lpAspectRatioMode);

    HRESULT ( STDMETHODCALLTYPE *SetAspectRatioMode )(
        IVMRWindowlessControl * This,
        /* [in] */ DWORD AspectRatioMode);

    HRESULT ( STDMETHODCALLTYPE *SetVideoClippingWindow )(
        IVMRWindowlessControl * This,
        /* [in] */ HWND hwnd);

    HRESULT ( STDMETHODCALLTYPE *RepaintVideo )(
        IVMRWindowlessControl * This,
        /* [in] */ HWND hwnd,
        /* [in] */ HDC hdc);

    HRESULT ( STDMETHODCALLTYPE *DisplayModeChanged )(
        IVMRWindowlessControl * This);

    HRESULT ( STDMETHODCALLTYPE *GetCurrentImage )(
        IVMRWindowlessControl * This,
        /* [out] */ BYTE **lpDib);

    HRESULT ( STDMETHODCALLTYPE *SetBorderColor )(
        IVMRWindowlessControl * This,
        /* [in] */ COLORREF Clr);

    HRESULT ( STDMETHODCALLTYPE *GetBorderColor )(
        IVMRWindowlessControl * This,
        /* [out] */ COLORREF *lpClr);

    HRESULT ( STDMETHODCALLTYPE *SetColorKey )(
        IVMRWindowlessControl * This,
        /* [in] */ COLORREF Clr);

    HRESULT ( STDMETHODCALLTYPE *GetColorKey )(
        IVMRWindowlessControl * This,
        /* [out] */ COLORREF *lpClr);

    END_INTERFACE
} IVMRWindowlessControlVtbl;

//-----------------------------------------------------------------


```

```
//-----------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct ICreateDevEnumVtbl *lpVtbl;
} ICreateDevEnum;

typedef struct ICreateDevEnumVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        ICreateDevEnum * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        ICreateDevEnum * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        ICreateDevEnum * This);

    HRESULT ( STDMETHODCALLTYPE *CreateClassEnumerator )(
        ICreateDevEnum * This,
        /* [in] */ REFCLSID clsidDeviceClass,
        /* [out] */
        __out  IEnumMoniker **ppEnumMoniker,
        /* [in] */ DWORD dwFlags);

    END_INTERFACE
} ICreateDevEnumVtbl;
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IAMStreamConfigVtbl *lpVtbl;
} IAMStreamConfig;


typedef struct IAMStreamConfigVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IAMStreamConfig * This,
        /* [in] */ REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IAMStreamConfig * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IAMStreamConfig * This);

    HRESULT ( STDMETHODCALLTYPE *SetFormat )(
        IAMStreamConfig * This,
        /* [in] */ AM_MEDIA_TYPE *pmt);

    HRESULT ( STDMETHODCALLTYPE *GetFormat )(
        IAMStreamConfig * This,
        /* [out] */
        __out  AM_MEDIA_TYPE **ppmt);

    HRESULT ( STDMETHODCALLTYPE *GetNumberOfCapabilities )(
        IAMStreamConfig * This,
        /* [out] */
        __out  int *piCount,
        /* [out] */
        __out  int *piSize);

    HRESULT ( STDMETHODCALLTYPE *GetStreamCaps )(
        IAMStreamConfig * This,
        /* [in] */ int iIndex,
        /* [out] */
        __out  AM_MEDIA_TYPE **ppmt,
        /* [out] */
        __out  BYTE *pSCC);

    END_INTERFACE
} IAMStreamConfigVtbl;
//-----------------------------------------------------------------

//-----------------------------------------------------------------
// objidl.h
#if 0
typedef struct
{
    CONST_VTBL struct IEnumMonikerVtbl *lpVtbl;
} IEnumMoniker;

typedef struct
{
    BEGIN_INTERFACE

```

```
    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IEnumMoniker * This,
        /* [in] */ __RPC__in REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IEnumMoniker * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IEnumMoniker * This);

    /* [local] */ HRESULT ( STDMETHODCALLTYPE *Next )(
        IEnumMoniker * This,
        /* [in] */ ULONG celt,
        /* [length_is][size_is][out] */ IMoniker **rgelt,
        /* [out] */ ULONG *pceltFetched);

    HRESULT ( STDMETHODCALLTYPE *Skip )(
        IEnumMoniker * This,
        /* [in] */ ULONG celt);

    HRESULT ( STDMETHODCALLTYPE *Reset )(
        IEnumMoniker * This);

    HRESULT ( STDMETHODCALLTYPE *Clone )(
        IEnumMoniker * This,
        /* [out] */ __RPC__deref_out_opt IEnumMoniker **ppenum);

    END_INTERFACE
} IEnumMonikerVtbl;
//----------------------------------------------------------------------
#endif

//----------------------------------------------------------------------
// control.h
//----------------------------------------------------------------------

//----------------------------------------------------------------------
typedef long OAFilterState;
typedef LONG_PTR OAEVENT;
typedef LONG_PTR OAHWND;
//----------------------------------------------------------------------

//----------------------------------------------------------------------
typedef struct IMediaControl
{
    CONST_VTBL struct IMediaControlVtbl *lpVtbl;
```

```
}IMediaControl;

typedef struct IMediaControlVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IMediaControl * This,
        /* [in] */ __RPC__in REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IMediaControl * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IMediaControl * This);

    HRESULT ( STDMETHODCALLTYPE *GetTypeInfoCount )(
        IMediaControl * This,
        /* [out] */ __RPC__out UINT *pctinfo);

    HRESULT ( STDMETHODCALLTYPE *GetTypeInfo )(
        IMediaControl * This,
        /* [in] */ UINT iTInfo,
        /* [in] */ LCID lcid,
        /* [out] */ __RPC__deref_out_opt ITypeInfo **ppTInfo);

    HRESULT ( STDMETHODCALLTYPE *GetIDsOfNames )(
        IMediaControl * This,
        /* [in] */ __RPC__in REFIID riid,
//      /* [size_is][in] */ __RPC__in_ecount_full(cNames) LPOLESTR *rgszNames,
        /* [size_is][in] */  LPOLESTR *rgszNames,
        /* [range][in] */ UINT cNames,
        /* [in] */ LCID lcid,
//      /* [size_is][out] */ __RPC__out_ecount_full(cNames) DISPID *rgDispId);
        /* [size_is][out] */ DISPID *rgDispId);

    /* [local] */ HRESULT ( STDMETHODCALLTYPE *Invoke )(
        IMediaControl * This,
        /* [in] */ DISPID dispIdMember,
        /* [in] */ REFIID riid,
        /* [in] */ LCID lcid,
        /* [in] */ WORD wFlags,
        /* [out][in] */ DISPPARAMS *pDispParams,
        /* [out] */ VARIANT *pVarResult,
        /* [out] */ EXCEPINFO *pExcepInfo,
        /* [out] */ UINT *puArgErr);

    HRESULT ( STDMETHODCALLTYPE *Run )(
        IMediaControl * This);

    HRESULT ( STDMETHODCALLTYPE *Pause )(
        IMediaControl * This);

    HRESULT ( STDMETHODCALLTYPE *Stop )(
        IMediaControl * This);

    HRESULT ( STDMETHODCALLTYPE *GetState )(
        IMediaControl * This,
        /* [in] */ LONG msTimeout,
        /* [out] */ __RPC__out OAFilterState *pfs);

    HRESULT ( STDMETHODCALLTYPE *RenderFile )(
        IMediaControl * This,
        /* [in] */ __RPC__in BSTR strFilename);

    HRESULT ( STDMETHODCALLTYPE *AddSourceFilter )(
        IMediaControl * This,
        /* [in] */ __RPC__in BSTR strFilename,
        /* [out] */ __RPC__deref_out_opt IDispatch **ppUnk);

    /* [propget] */ HRESULT ( STDMETHODCALLTYPE *
get_FilterCollection )(
        IMediaControl * This,
        /* [retval][out] */ __RPC__deref_out_opt IDispatch **ppUnk);

    /* [propget] */ HRESULT ( STDMETHODCALLTYPE *
get_RegFilterCollection )(
        IMediaControl * This,
        /* [retval][out] */ __RPC__deref_out_opt IDispatch **ppUnk);

    HRESULT ( STDMETHODCALLTYPE *StopWhenReady )(
        IMediaControl * This);

    END_INTERFACE
} IMediaControlVtbl;
//---------------------------------------------------------------------------------


//---------------------------------------------------------------------------------
typedef struct
{
    CONST_VTBL struct IMediaEventExVtbl *lpVtbl;
}IMediaEventEx;

```

```

typedef struct IMediaEventExVtbl
{
    BEGIN_INTERFACE

    HRESULT ( STDMETHODCALLTYPE *QueryInterface )(
        IMediaEventEx * This,
        /* [in] */ __RPC__in REFIID riid,
        /* [iid_is][out] */
        __RPC__deref_out  void **ppvObject);

    ULONG ( STDMETHODCALLTYPE *AddRef )(
        IMediaEventEx * This);

    ULONG ( STDMETHODCALLTYPE *Release )(
        IMediaEventEx * This);

    HRESULT ( STDMETHODCALLTYPE *GetTypeInfoCount )(
        IMediaEventEx * This,
        /* [out] */ __RPC__out UINT *pctinfo);

    HRESULT ( STDMETHODCALLTYPE *GetTypeInfo )(
        IMediaEventEx * This,
        /* [in] */ UINT iTInfo,
        /* [in] */ LCID lcid,
        /* [out] */ __RPC__deref_out_opt ITypeInfo **ppTInfo);

    HRESULT ( STDMETHODCALLTYPE *GetIDsOfNames )(
        IMediaEventEx * This,
        /* [in] */ __RPC__in REFIID riid,
//      /* [size_is][in] */ __RPC__in_ecount_full(cNames) LPOLESTR *rgszNames,
        /* [size_is][in] */ LPOLESTR *rgszNames,
        /* [range][in] */ UINT cNames,
        /* [in] */ LCID lcid,
//      /* [size_is][out] */ __RPC__out_ecount_full(cNames) DISPID *rgDispId);
        /* [size_is][out] */  DISPID *rgDispId);

    /* [local] */ HRESULT ( STDMETHODCALLTYPE *Invoke )(
        IMediaEventEx * This,
        /* [in] */ DISPID dispIdMember,
        /* [in] */ REFIID riid,
        /* [in] */ LCID lcid,
        /* [in] */ WORD wFlags,
        /* [out][in] */ DISPPARAMS *pDispParams,
        /* [out] */ VARIANT *pVarResult,
        /* [out] */ EXCEPINFO *pExcepInfo,
        /* [out] */ UINT *puArgErr);

    HRESULT ( STDMETHODCALLTYPE *GetEventHandle )(
        IMediaEventEx * This,
        /* [out] */ __RPC__out OAEVENT *hEvent);

    HRESULT ( STDMETHODCALLTYPE *GetEvent )(
        IMediaEventEx * This,
        /* [out] */ __RPC__out long *lEventCode,
        /* [out] */ __RPC__out LONG_PTR *lParam1,
        /* [out] */ __RPC__out LONG_PTR *lParam2,
        /* [in] */ long msTimeout);

    HRESULT ( STDMETHODCALLTYPE *WaitForCompletion )(
        IMediaEventEx * This,
        /* [in] */ long msTimeout,
        /* [out] */ __RPC__out long *pEvCode);

    HRESULT ( STDMETHODCALLTYPE *CancelDefaultHandling )(
        IMediaEventEx * This,
        /* [in] */ long lEvCode);

    HRESULT ( STDMETHODCALLTYPE *RestoreDefaultHandling )(
        IMediaEventEx * This,
        /* [in] */ long lEvCode);

    HRESULT ( STDMETHODCALLTYPE *FreeEventParams )(
        IMediaEventEx * This,
        /* [in] */ long lEvCode,
        /* [in] */ LONG_PTR lParam1,
        /* [in] */ LONG_PTR lParam2);

    HRESULT ( STDMETHODCALLTYPE *SetNotifyWindow )(
        IMediaEventEx * This,
        /* [in] */ OAHWND hwnd,
        /* [in] */ long lMsg,
        /* [in] */ LONG_PTR lInstanceData);

    HRESULT ( STDMETHODCALLTYPE *SetNotifyFlags )(
        IMediaEventEx * This,
        /* [in] */ long lNoNotifyFlags);

    HRESULT ( STDMETHODCALLTYPE *GetNotifyFlags )(
        IMediaEventEx * This,
        /* [out] */ __RPC__out long *lplNoNotifyFlags);

    END_INTERFACE
} IMediaEventExVtbl;
//----------------------------------------------------------------------

```

D.4 Makefile

```
# lcc makefile to build webcam
#
# compile windows media (wmv) window application using lcc and COM library
# requires wmvcore.lib from windows sdk
# Also requires uuid.lib for lcc. To make this,
#  cd \lcc\bin
#  buildguid
#
# -A in lcc for warnings
#
# To build import library for lcc:
#     See manual.doc, Q12 How do I use an MSVC import library with lcc?
# pedump /exp wmvcore.lib > wmvcore.exp
# buildlib wmvcore.exp wmvcore-lcc.lib
#
# note that lcc uses .obj not .o
#
# type "make" or "make basic" or "make -f makefile"
# Makefile reference:
#   "Topics in C Programming", S. Kochan & P. Wood,
#   Hayden Books, Chapter 7
#
# Notes:
#   The command after dependency lines must be on
#       the next line, and *must* be a tab character.
#   Use / not \ as path separator in makefiles.
#
# If using a text editor, be sure to set 'display tabs'
# and not 'expand tabs' (to spaces), as this will cause
# the makefile to be interpreted incorrectly.
#
#
# John Leis

# compiler
# default include dir is \lcc\include
# lcc  -c -Ic:\lcc\include -g2 %1.c
# -O is optimize
# -A for warnings
CC              = lcc
#CCOPTS         = -g2
#CCOPTS         = -O -A
CCOPTS          = -O
BASEDIR         = \tmp
#INCDIR         = -I\lcc\include
INCDIR          = -I.

# linker
# default lib dir is \lcc\lib
# lcclnk -subsystem console %1.obj -o %1.exe
# lcclnk -subsystem windows %1.obj -o %1.exe
# lcclnk -subsystem windows %1.obj winmm.lib -o %1.exe
# lcclnk -subsystem windows %1.obj %1.res -o %1.exe

LD          = lcclnk
WMLIBLCC    = wmvcore-lcc.lib
WMLIBMS     = wmvcore.lib
COMLIBS     = ole32.lib uuid.lib
LDFLAGS     = -lm

# for windows or console apps
# -s to strip debug symbol info
LDOPTS      = -subsystem windows -s
#LDOPTS      = -subsystem console

# resource compiler
# lrc %1.rc -o %1.res
RC          = lrc
RCOPTS      =

# dependencies
OBJS     = WebCam.obj WebCamLib.obj
OBJSMIN  = WebCamMin.obj WebCamLib.obj
OBJSEDGE = WebCamEdge.obj WebCamLib.obj

WINRES  = WebCam.res

# executable target
TARGET  = WebCam.exe
TARGETMIN  = WebCamMin.exe
TARGETEDGE  = WebCamEdge.exe

# suffix rules
.c.obj:
    $(CC) -c $(CCOPTS) $(INCDIR) $*.c -o $*.obj

.rc.res:
    $(RC) $(RCOPTS) $*.rc -o $*.res


# make targets - note tab on rule line

# rebuild
all: $(TARGET) $(TARGETMIN)
    @echo ALL

```

```
# executable
#$(TARGET): $(OBJS)
#  $(LD)  $(LDOPTS) $(OBJS) $(WINRES) $(LIBS) $(WEBCAMLIBS) $(COMLIBS) $(LDFLAGS) -o $(TARGET)

$(TARGET): $(OBJS)  $(WINRES) $(WMLIBLCC)
    $(LD)  $(LDOPTS) $(OBJS)  $(WINRES) $(WMLIBLCC) $(COMLIBS) $(LDFLAGS) -o $(TARGET)

$(TARGETMIN): $(OBJSMIN) $(WMLIBLCC)
    $(LD)  $(LDOPTS) $(OBJSMIN) $(WMLIBLCC) $(COMLIBS) $(LDFLAGS) -o $(TARGETMIN)

$(TARGETEDGE): $(OBJSEDGE) $(WMLIBLCC)
    $(LD)  $(LDOPTS) $(OBJSEDGE) $(WMLIBLCC) $(COMLIBS) $(LDFLAGS) -o $(TARGETEDGE)

# Windows media library suitable for lcc
# Note: do not delete wmvcore.lib (comes from MS SDK)
$(WMLIBLCC):
    pedump /exp wmvcore.lib > wmvcore.exp
    buildlib wmvcore.exp wmvcore-lcc.lib

# cleanup - also delete capture files *.cap
clean:
    del *.obj
    del *.res
    del *.exe
#   del $(WMLIBLCC)
#   del *.lib
    del *.exp

noexe:
    del *.exe

```

Appendix E

Program Output

E.1 Laser Motion Data Program Output

```
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
X Min: 639, X Max: -1, Y Min: 479, Y Max: -1
6 pixels counted. Detection Rate is 2.000000 pixels per
second. Total time was 3.000000.

X Min: 274, X Max: -1, Y Min: 465, Y Max: -1
X Min: 296, X Max: -1, Y Min: 435, Y Max: -1
X Min: 317, X Max: -1, Y Min: 412, Y Max: -1
X Min: 340, X Max: -1, Y Min: 392, Y Max: -1
X Min: 363, X Max: -1, Y Min: 369, Y Max: -1
X Min: 375, X Max: -1, Y Min: 353, Y Max: -1
X Min: 389, X Max: -1, Y Min: 339, Y Max: -1
X Min: 397, X Max: -1, Y Min: 319, Y Max: -1
X Min: 403, X Max: -1, Y Min: 299, Y Max: -1
X Min: 405, X Max: -1, Y Min: 279, Y Max: -1
X Min: 413, X Max: -1, Y Min: 253, Y Max: -1
X Min: 424, X Max: -1, Y Min: 232, Y Max: -1
X Min: 425, X Max: -1, Y Min: 214, Y Max: -1
X Min: 431, X Max: -1, Y Min: 194, Y Max: -1
X Min: 435, X Max: -1, Y Min: 177, Y Max: -1
X Min: 440, X Max: -1, Y Min: 155, Y Max: -1
X Min: 446, X Max: -1, Y Min: 136, Y Max: -1
X Min: 447, X Max: -1, Y Min: 121, Y Max: -1
X Min: 449, X Max: -1, Y Min: 105, Y Max: -1
X Min: 449, X Max: -1, Y Min: 87, Y Max: -1
X Min: 455, X Max: -1, Y Min: 66, Y Max: -1
X Min: 454, X Max: -1, Y Min: 54, Y Max: -1
X Min: 455, X Max: -1, Y Min: 37, Y Max: -1
X Min: 457, X Max: -1, Y Min: 22, Y Max: -1
X Min: 457, X Max: -1, Y Min: 9, Y Max: -1
25 pixels counted. Detection Rate is 2.500000 pixels per
second. Total time was 10.000000.

X Min: 589, X Max: -1, Y Min: 363, Y Max: -1
X Min: 581, X Max: -1, Y Min: 351, Y Max: -1
X Min: 552, X Max: -1, Y Min: 326, Y Max: -1
X Min: 535, X Max: -1, Y Min: 311, Y Max: -1
X Min: 513, X Max: -1, Y Min: 297, Y Max: -1
X Min: 491, X Max: -1, Y Min: 279, Y Max: -1
X Min: 465, X Max: -1, Y Min: 259, Y Max: -1
X Min: 443, X Max: -1, Y Min: 243, Y Max: -1
X Min: 418, X Max: -1, Y Min: 226, Y Max: -1
X Min: 137, X Max: -1, Y Min: 81, Y Max: -1
10 pixels counted. Detection Rate is 2.500000 pixels per
```

```
second. Total time was 4.000000.


```

E.2 Laser Circle Output

```
X Min: 73, X Max: -1, Y Min: 44, Y Max: -1
1 pixels counted. Detection Rate is 1.000000 pixels per second. Total time was 1.000000.

X Min: 53, X Max: -1, Y Min: 31, Y Max: -1
X Min: 37, X Max: -1, Y Min: 17, Y Max: -1
2 pixels counted. Detection Rate is 2.000000 pixels per second. Total time was 1.000000.

X Min: 37, X Max: -1, Y Min: 472, Y Max: -1
1 pixels counted. Detection Rate is inf pixels per second. Total time was 0.000000.

X Min: 53, X Max: -1, Y Min: 450, Y Max: -1
1 pixels counted. Detection Rate is inf pixels per second. Total time was 0.000000.

X Min: 25, X Max: -1, Y Min: 453, Y Max: -1
1 pixels counted. Detection Rate is 1.000000 pixels per second. Total time was 1.000000.

X Min: 17, X Max: -1, Y Min: 465, Y Max: -1
X Min: 15, X Max: -1, Y Min: 463, Y Max: -1
X Min: 17, X Max: -1, Y Min: 461, Y Max: -1
X Min: 35, X Max: -1, Y Min: 459, Y Max: -1
X Min: 58, X Max: -1, Y Min: 451, Y Max: -1
X Min: 89, X Max: -1, Y Min: 439, Y Max: -1
X Min: 107, X Max: -1, Y Min: 427, Y Max: -1
X Min: 128, X Max: -1, Y Min: 405, Y Max: -1
X Min: 144, X Max: -1, Y Min: 399, Y Max: -1
X Min: 171, X Max: -1, Y Min: 383, Y Max: -1
X Min: 203, X Max: -1, Y Min: 361, Y Max: -1
X Min: 210, X Max: -1, Y Min: 347, Y Max: -1
X Min: 234, X Max: -1, Y Min: 338, Y Max: -1
X Min: 254, X Max: -1, Y Min: 326, Y Max: -1
X Min: 279, X Max: -1, Y Min: 301, Y Max: -1
X Min: 287, X Max: -1, Y Min: 285, Y Max: -1
X Min: 304, X Max: -1, Y Min: 280, Y Max: -1
X Min: 328, X Max: -1, Y Min: 266, Y Max: -1
X Min: 345, X Max: -1, Y Min: 249, Y Max: -1
X Min: 361, X Max: -1, Y Min: 236, Y Max: -1
X Min: 380, X Max: -1, Y Min: 219, Y Max: -1
21 pixels counted. Detection Rate is 2.333333 pixels per second. Total time was 9.000000.

X Min: 410, X Max: -1, Y Min: 202, Y Max: -1
X Min: 422, X Max: -1, Y Min: 188, Y Max: -1
X Min: 437, X Max: -1, Y Min: 173, Y Max: -1
```

```
X Min: 455, X Max: -1, Y Min: 159, Y Max: -1
X Min: 480, X Max: -1, Y Min: 146, Y Max: -1
X Min: 496, X Max: -1, Y Min: 133, Y Max: -1
X Min: 519, X Max: -1, Y Min: 115, Y Max: -1
X Min: 553, X Max: -1, Y Min: 83, Y Max: -1
X Min: 565, X Max: -1, Y Min: 69, Y Max: -1
X Min: 595, X Max: -1, Y Min: 39, Y Max: -1
10 pixels counted. Detection Rate is 2.500000 pixels per
second. Total time was 4.000000.

X Min: 425, X Max: -1, Y Min: 100, Y Max: -1
X Min: 422, X Max: -1, Y Min: 99, Y Max: -1
X Min: 426, X Max: -1, Y Min: 94, Y Max: -1
X Min: 424, X Max: -1, Y Min: 94, Y Max: -1
X Min: 427, X Max: -1, Y Min: 97, Y Max: -1
X Min: 420, X Max: -1, Y Min: 96, Y Max: -1
X Min: 412, X Max: -1, Y Min: 87, Y Max: -1
X Min: 414, X Max: -1, Y Min: 74, Y Max: -1
X Min: 427, X Max: -1, Y Min: 65, Y Max: -1
X Min: 442, X Max: -1, Y Min: 68, Y Max: -1
X Min: 453, X Max: -1, Y Min: 75, Y Max: -1
X Min: 459, X Max: -1, Y Min: 85, Y Max: -1
X Min: 460, X Max: -1, Y Min: 96, Y Max: -1
X Min: 453, X Max: -1, Y Min: 105, Y Max: -1
X Min: 442, X Max: -1, Y Min: 112, Y Max: -1
X Min: 431, X Max: -1, Y Min: 110, Y Max: -1
X Min: 424, X Max: -1, Y Min: 106, Y Max: -1
X Min: 421, X Max: -1, Y Min: 99, Y Max: -1
X Min: 420, X Max: -1, Y Min: 90, Y Max: -1
X Min: 425, X Max: -1, Y Min: 81, Y Max: -1
X Min: 436, X Max: -1, Y Min: 78, Y Max: -1
X Min: 447, X Max: -1, Y Min: 81, Y Max: -1
X Min: 451, X Max: -1, Y Min: 87, Y Max: -1
X Min: 450, X Max: -1, Y Min: 99, Y Max: -1
X Min: 443, X Max: -1, Y Min: 107, Y Max: -1
X Min: 435, X Max: -1, Y Min: 109, Y Max: -1
X Min: 425, X Max: -1, Y Min: 105, Y Max: -1
X Min: 421, X Max: -1, Y Min: 97, Y Max: -1
X Min: 422, X Max: -1, Y Min: 90, Y Max: -1
X Min: 427, X Max: -1, Y Min: 83, Y Max: -1
X Min: 435, X Max: -1, Y Min: 77, Y Max: -1
X Min: 446, X Max: -1, Y Min: 73, Y Max: -1
X Min: 458, X Max: -1, Y Min: 78, Y Max: -1
X Min: 461, X Max: -1, Y Min: 91, Y Max: -1
X Min: 463, X Max: -1, Y Min: 107, Y Max: -1
X Min: 448, X Max: -1, Y Min: 114, Y Max: -1
36 pixels counted. Detection Rate is 2.571429 pixels per
second. Total time was 14.000000.
```

```

X Min: 429, X Max: -1, Y Min: 101, Y Max: -1
X Min: 421, X Max: -1, Y Min: 67, Y Max: -1
X Min: 421, X Max: -1, Y Min: 95, Y Max: -1
X Min: 429, X Max: -1, Y Min: 92, Y Max: -1
X Min: 445, X Max: -1, Y Min: 79, Y Max: -1
X Min: 446, X Max: -1, Y Min: 62, Y Max: -1
X Min: 420, X Max: -1, Y Min: 81, Y Max: -1
X Min: 419, X Max: -1, Y Min: 88, Y Max: -1
X Min: 433, X Max: -1, Y Min: 93, Y Max: -1
X Min: 439, X Max: -1, Y Min: 91, Y Max: -1
X Min: 441, X Max: -1, Y Min: 93, Y Max: -1
X Min: 430, X Max: -1, Y Min: 89, Y Max: -1
X Min: 450, X Max: -1, Y Min: 84, Y Max: -1
X Min: 457, X Max: -1, Y Min: 57, Y Max: -1
X Min: 455, X Max: -1, Y Min: 92, Y Max: -1
X Min: 447, X Max: -1, Y Min: 95, Y Max: -1
X Min: 445, X Max: -1, Y Min: 91, Y Max: -1
X Min: 440, X Max: -1, Y Min: 90, Y Max: -1
X Min: 437, X Max: -1, Y Min: 91, Y Max: -1
19 pixels counted. Detection Rate is 2.375000 pixels per second. Total time was 8.000000.


```