REAL TIME GUN PROFILER FOR HIGH SPEED OBJECTS

By

Amna Irum (Ldr)

Wajeeha Kanwal

Hafiz Asif Raza

Capt. Faisal Javeed

A thesis submitted to the faculty of Computer Science Department Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of B.E in Computer Software Engineering.

April 2006

Declaration

No portion of the work presented in this study has been submitted in support of another award or qualification either at this institute or elsewhere.

Dedication

We would like to dedicate this project to our loving parents for their unwavering support and to the talented, energetic faculty of our college for their guidance and assistance.

Abstract

We address the issue of tracking speedy objects such as aircrafts in a situation where the target object and the camera tracking this target, are both in motion. Such a case would be when an aircraft is being detected and tracked through a video stream obtained from a camera mounted on an anti-aircraft machine. The target is tracked with respect to the gunner aiming at the target from his machine. The path information of both, the target movement and the gunner's aim is compared to provide a complete tracking profile of the gunner. Other motion parameters are analyzed to give hit or miss status of the target when the gunner fires at the aircraft. The horizontal, vertical and radial deviation of the target from the aim is used to assess the aiming stability of the gunner. The proposed system is used to train personnel for real time aircraft shooting without wasting ammo. It will help in evaluating the aiming capabilities of a gunner.

Acknowledgements

We would like to acknowledge our advisors Lt. Col Naveed Khattak (NUST) and Dr. Moeed Mufti (TELEMATIX) for their supervision and support and for keeping their faith in our abilities. We would also like to thank our esteemed instructors who helped make us who we are and nurtured our capabilities, giving us the skills and the confidence to take on mountains and emerge victorious.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1Introduction

This chapter introduces the Real Time Gun Profiler and the development of this project. It includes both the brief and detailed objectives. The benefits of the project as well as the background are also provided. This chapter gives a brief overview of the entire document.

1.2Aim

The cost of training personnel for anti aircraft machines by using live testing with real artillery is prohibitively handsome. There are many chances of human errors in such a testing system which lead not only to a waste of ammunition but also provides a difficult situation for evaluation of the gunner's shooting ability. To reduce the reliance on human expertise a need is felt to develop a system that provides a live anti aircraft machine testing environment for the trainer; and allows real time field testing without requiring real shells and bullets. The motive that drives this project is to create an aircraft shooting system that does not involve wastage of ammunition. Moreover in case of fast moving targets (aircrafts) it is impossible to determine the exact tracking path followed by a human as he aims for the target; nor is it possible to calculate the accurate deviation of the target from its aim when the target is shot. To avoid the cost and consumption of real missiles and to avoid the limitations of a shooting system that relies on human expertise, we offer an automated shooting system that allows real time or live shooting of aircrafts without wasting shells. The system has been based on visual tracking seeing that this field has emerged as an important component of defense systems and it has existed since the first time aircrafts were used in battle. Previously designed systems which used visually guided tracking as a tool to maintain gaze on moving objects with respect to the eyes did not provide any information regarding the tracking skills of a human gunner and for that reason they could not be used as a training tool. The aim of this project is to

address this specific issue of tracking speedy objects in a case where camera and target both are in motion in the context of visual input to provide a complete shooting profile of the trainer and help evaluate his stability and aiming aptitude for training purposes. The system will help in estimating and improving the aiming capabilities of a trainee (the gunner) without wasting ammo. This training tool can be used as the basic idea for developing further simulators or trainer systems of advanced weaponry and guns.

1.3 Background

Target detection and tracking moving objects in sequential images is an essential task in a number of applications, such as security, surveillance, vehicle navigation and robotics etc. One such application is to train personnel in using anti-aircraft machines for aiming and firing at high speed moving targets in the air; without wasting real ammunition. The processing of a video stream for characterizing events of interest such as the aiming at an object in the field view of a gunner relies on the detection, in each frame, of the objects involved, and the temporal integration of this frame based information. This high level description of a video stream relies on accurate detection and tracking of the moving objects, and on the relationship of their trajectories to the scene.

Most of the techniques used for tracking moving objects deal with a stationary camera [1],[2] or closed world representations [3],[4] which rely on a fixed background or a specific knowledge on the type of actions taking place. We deal with a more challenging type of video streams: the one obtained from a moving camera. Moreover the target tracking algorithms developed for tracking targets based on sonar and radar measurements could be used for tracking (also known as motion correspondence) but for a system to operate at video frame-rate (possibly even higher rates) limits the use of these well-established statistical and non-statistical tracking algorithms.

Therefore to meet the real time requirements, we propose to identify and track targets by their measured positions and motion parameters derived from video stream processing.

1.4 Organization of the Study

In the next chapter, there is a brief account of the block level overview of the system. Next we present each module of the project in detail and the techniques used for identifying and tracking target objects, along with the measuring of the target deviation from the aim. Chapter 9 includes some screenshots and system menu details and in chapter 10, we give the experimental results obtained by running the algorithm on actual hardware. The main focus of this document is on describing a real-time gun profiler system that tracks targets with respect to a human gunner and describing various techniques needed to accomplish this. Finally, we present the limitations of the system and the future work associated with it.

1.5 Detailed Objectives

The proposed system is a real time aircraft shooting system and the basic objective was to create a live aircraft tracking system that does not involve wastage of ammunition for training of personnel. It can be used as a training tool to teach personnel, the necessary skills for shooting aircrafts and it can also evaluate the shooting ability of a person who is tracking and aiming at a high speed moving object such as an aircraft. Other objectives include the generation of a complete shooting profile of the trainer to help evaluate his stability and aiming capabilities. The human gunner performs realistic aiming at the target from his machine while the system generates a graphical representation of this aiming path followed by the anti aircraft machine. The system not only determines the tracking path followed by a human trainee as he aims for the aircraft and fires at it but also calculates the deviation of the tracked aircraft from its aim when the target is shot.

This project presents a visual tracker which can keep track of the target with respect to the gunner's aim for the target. The system will help in estimating and improving the aiming abilities of a trainee (the gunner) without wasting ammo. The trainee will track the high speed target such as an aircraft with a camera mounted on his gun and press a trigger to shoot it just like real time shooting but without a shell. Using the information of

the camera a profile is made which includes deviation of the actual aircraft from the tracking movements of the trainee and gives the hit or missed status information of the target when it is shot. To accomplish this central challenge is to determine the image position of an object, or target region of an object, as the target moves through a camera's field of view. Once the image position is determined the distance from the target relative to its size is calculated. This information is used for realistic simulation of aircrafts. The system allows field testing with real machines but without real ammunition.

The end-users of the system will be the personnel who will use this software as a training tool to teach new gunners the art of aircraft aiming and firing. The trainee will be evaluated by the software for his gunning capabilities.

1.6 Long Term Objectives and Benefits

The work done so far in the visual target tracking field is related to only a limited set of applications such as surveillance [8],[9], autonomous vehicle navigation [9], robotic gaze control [11],[12], visual reconstruction [13] and missile guidance. Target detection and tracking moving objects in sequential images is an essential task in all such applications. One other application is to train personnel in using anti-aircraft machines for aiming and firing at high speed moving targets in the air; without wasting real ammunition. This issue has been addressed by our project however many long term benefits root out from this project such as the implementation of a similar visual tracking system for more advanced and heavy artillery. The system can be extended to provide exact details of the target object such as the percentage damage done to it when the aircraft is shot. As a training and evaluation tool this gun profiler system can be used for the realistic training of troops and for the evaluation of their target tracking and shooting skills. The wastage of real ammunition is not required in any scenario, whether the system is used or training or for evaluation.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The Microsoft Visual C++ platform has been used for the development of the project as many features have been built into this development environment to enable creation of very advanced applications for the Windows and NT platforms. Microsoft Visual C++ development environment and its set of tools allow developers to create Windows based applications with a great level of ease and speed. The DirectX support has been used for acquiring video stream data from the camera. In the literature review section a brief account of the Visual C++ development environment and the DirectX APIs is included. The DirectX component DirectShow which has been integrated into the main application to perform multimedia steaming is also discussed in this chapter.

2.2 Microsoft Visual C++

Microsoft Visual C++ provides the dynamic development environment for creating Microsoft Windows–based and Microsoft .NET–based applications, dynamic Web applications, and XML Web services using the C++ development language. Visual C++ .NET includes the industry-standard Active Template Library (ATL) and Microsoft Foundation Class (MFC) libraries, advanced language extensions, and powerful integrated development environment (IDE) features that enable developers to edit and debug source code efficiently.

It provides developers with a proven, object-oriented language for building powerful and performance-conscious applications. With advanced template features, low-level platform access, and an optimizing compiler, Visual C++ .NET delivers superior functionality for generating robust applications and components. The product enables developers to build a wide variety of solutions, including Web applications and Microsoft Windows-based applications. C++ is the world's most popular systems-level language,

and Visual C++ gives developers a world-class tool with which to build software. The Microsoft Visual C++ Framework is designed to fulfill the following objectives

- To provide a consistent object-oriented programming environment.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the C++ Framework can integrate with any other code.

2.3 The Component Object Model (COM)

COM is the architecture for defining interfaces and interaction among objects implemented by widely varying software applications. A COM object instantiates one or more interfaces, each of which exposes zero or more properties and zero or more methods. All COM interfaces are derived from the base class **IUnknown**. Technologies built on the COM foundation include ActiveX, MAPI, and OLE.

COM specifies an object model and programming requirements that enable COM objects (also called COM components, or sometimes simply *objects*) to interact with other objects. These objects can be within a single process, in other processes, and can even be on remote machines. They can have been written in other languages, and they may be structurally quite dissimilar, which is why COM is referred to as a *binary standard*—a standard that applies after a program has been translated to binary machine code. COM (and all COM-based technologies) are not object-oriented language and COM does not specifying how an application should be structured; the language, structure, and implementation details are left to the application programmer.

2.3.1 Purpose

COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents) and ActiveX® (Internet-enabled components) technologies, as well as others. COM specifies the basic binary object standard and also defines certain basic interfaces that provide functions common to all COM-based technologies, and it provides a small number of API functions that all components require. COM also defines how objects work together over a distributed environment and has added security features to help provide system and component integrity.

2.3.2 Application

COM objects can be created with a variety of programming languages. Object-oriented languages, such as C++, provide programming mechanisms that simplify the implementation of COM objects. These objects can be within a single process, in other processes, even on remote machines.

2.3.3 Language Requirement

COM is designed primarily for C++ and Microsoft Visual Basic® developers. The only language requirement for COM is that code is generated in a language that can create structures of pointers and, either explicitly or implicitly, call functions through pointers. Object-oriented languages such as Microsoft® Visual C++® and Smalltalk provide programming mechanisms that simplify the implementation of COM objects, but languages such as C, Pascal, Ada, Java, and even BASIC programming environments can create and use COM objects.

2.4 COM Objects

COM defines the essential nature of a COM object. In general, a software object is made up of a set of data and the functions that manipulate the data. A COM object is one in which access to an object's data is achieved exclusively through one or more sets of related functions. These function sets are called *interfaces*, and the functions of an interface are called *methods*. Further, COM requires that the only way to gain access to the methods of an interface is through a pointer to the interface. The fundamental concepts of COM objects are,

- **Interfaces** — the mechanism through which an object exposes its functionality.
- **Interface Implementation** — the code a programmer supplies to carry out the actions specified in an interface definition.
- **Interface Pointers** — an instance of an interface implementation is actually a pointer to an array of pointers to methods.
- **IUnknown Interface** — the basic interface on which all others are based. It implements the reference counting and interface querying mechanisms running through COM.
- **Reference counting** — the technique by which an object (or, strictly, an interface) decides when it is no longer being used and is therefore free to remove itself.
- **QueryInterface** — the method used to query an object for a given interface.
- **Marshaling** — the mechanism that enables objects to be used across thread, process, and network boundaries, allowing for location independence.
- **Aggregation** — a way in which one object can make use of another.

2.4.1 Interfaces

An interface is the way in which an object exposes its functionality to the outside world. In COM, an interface is a table of pointers (like a C++ vtable) to functions implemented

by the object. The table represents the interface, and the functions to which it points are the methods of that interface. An object can expose as many interfaces as it chooses.

COM uses the word *interface* in a sense different from that typically used in Visual C++ programming. A COM interface refers to a predefined group of related functions that a COM class implements, but a specific interface does not necessarily represent all the functions that the class supports. (Java defines interfaces in just the same way). Referring to an object implementing an interface means that the object uses code that implements each method of the interface and provides COM binary-compliant pointers to those functions to the COM library.

COM makes a fundamental distinction between interface definitions and their implementations. An interface is actually a contract that consists of a group of related function prototypes whose usage is defined but whose implementation is not. These function prototypes are equivalent to pure virtual base classes in C++ programming. An interface definition specifies the interface's member functions, called methods, their return types, the number and types of their parameters, and what they must do. There is no implementation associated with an interface. The COM interface can be summarized as,

- **A COM interface is not the same as a C++ class** — the pure virtual definition carries no implementation. An instance of an object that implements an interface must be created for the interface actually to exist.

- **A COM interface is not an object** — it is simply a related group of functions and is the binary standard through which clients and objects communicate. As long as it can provide pointers to interface methods, the object can be implemented in any language with any internal state representation.

- **COM interfaces are strongly typed** — every interface has its own interface identifier (a GUID), which eliminates the possibility of duplication that could occur with any other naming scheme.

• **COM interfaces are immutable** — a new version of an old interface with the same identifier cannot be defined. Adding or removing methods of an interface or changing semantics creates a new interface, not a new version of an old interface. Therefore, a new interface cannot conflict with an old interface. The interface ID (IID) defines the interface contract explicitly and uniquely.

2.4.2 Interface Implementation

An interface implementation is the code a programmer supplies to carry out the actions specified in an interface definition. Implementations of many of the interfaces a programmer can use in an object-based application are included in the COM libraries. However, programmers are free to ignore these implementations and write their own. An interface implementation is to be associated with an object when an instance of that object is created, and the implementation provides the services that the object offers.

Interfaces define a contract between an object and its clients. The contract specifies the methods that must be associated with each interface and what the behavior of each of the methods must be in terms of input and output. The contract generally does not define how to implement the methods in an interface. Another important aspect of the contract is that if an object supports an interface, it must support all of that interface's methods in some way. Not all of the methods in an implementation need to do something if an object does not support the function implied by a method, its implementation may be a simple return or perhaps the return of a meaningful error message but the methods must exist.

2.4.3 Interface Pointers

An instance of an interface implementation is actually a pointer to an array of pointers to methods—that is, a function table that refers to an implementation of all of the methods specified in the interface. Objects with multiple interfaces can provide pointers to more than one function table. Any code that has a pointer through which it can access the array can call the methods in that interface. With appropriate compiler support (which is

inherent in C and C++), a client can call an interface method through its name, not its position in the array. Because an interface is a type, the compiler, given the names of methods, can check the types of parameters and return values of each interface method call.

Each interface the immutable contract of a functional group of methods—is referred to at run time with a globally unique interface identifier (IID). This IID, which is a specific instance of a globally unique identifier (GUID) supported by COM, allows a client to ask an object precisely whether it supports the semantics of the interface, without unnecessary overhead and without the confusion that could arise in a system from having multiple versions of the same interface with the same name. However, objects can support multiple interfaces simultaneously and can expose interfaces that are successive revisions of an interface, with different identifiers.

2.4.4 IUnknown Interface

IUnknown is the base interface of every other COM interface. **IUnknown** defines three methods:

- QueryInerface.
- AddRef.
- Release.

QueryInterface allows an interface user to ask the object for a pointer to another of its interfaces. AddRef and Release implement reference counting on the interface. All COM objects must implement the IUnknown interface because it provides the means, using QueryInterface, to move freely between the different interfaces that an object supports as well as the means to manage its lifetime by using AddRef and Release. The methods of **IUnknown** allow navigation to other interfaces exposed by the object.

Interface inheritance does not mean code inheritance because no implementations are associated with interfaces. It means only that the contract associated with an interface is

inherited in a C++ pure-virtual base-class fashion and modified either by adding new methods or by further qualifying the allowed usage of methods. There is no selective inheritance in COM. If one interface inherits from another, it includes all the methods that the other interface defines.

While there are a few interfaces that inherit their definitions from a second interface in addition to IUnknown, the majority simply inherit the IUnknown interface methods along with their own already defined methods. This makes most interfaces relatively compact and easy to encapsulate.

Also, each interface is given a unique interface ID (IID). This uniqueness makes it easy to support interface versioning. A new version of an interface is simply a new interface, with a new IID. IIDs for the standard COM and OLE interfaces are predefined.

2.4.5 Reference Counting

COM itself does not automatically try to remove an object from memory when it thinks the object is no longer being used. Instead, the object programmer must remove the unused object. The programmer determines whether an object can be removed based on a reference count. COM uses the **IUnknown** methods, AddRef and Release, to manage the reference count of interfaces on an object. The general rules for calling these methods are:

- Whenever a client receives an interface pointer, **AddRef** must be called on the interface.
- Whenever the client has finished using the interface pointer, it must call **Release**.

In a simple implementation, each AddRef call increments and each Release call decrements a counter variable inside the object. When the count returns to zero, the interface no longer has any users and is free to remove itself from memory.

Reference counting can also be implemented so that each reference to the object (not to an individual interface) is counted. In this case, each AddRef and Release call delegates

to a central implementation on the object, and Release frees the entire object when its reference count reaches zero. When a CComObject-derived object is constructed using the new operator, the reference count is 0. Therefore, a call to AddRef must be made after successfully creating the CComObject-derived object.

2.4.6 QueryInterface

Although there are mechanisms by which an object can express the functionality it provides statically (before it is instantiated), the fundamental COM mechanism is to use the **IUnknown** method called QueryInterface.

Every interface is derived from IUnknown, so every interface has an implementation of QueryInterface. Regardless of implementation, this method queries an object using the IID of the interface to which the caller wants a pointer. If the object supports that interface, QueryInterface retrieves a pointer to the interface, while also calling AddRef. Otherwise, it returns the E_NOINTERFACE error code.

The Reference Counting rules must be obeyed at all times. If you call Release on an interface pointer to decrement the reference count to zero, you should not use that pointer again. Occasionally you may need to obtain a weak reference to an object (that is, you may wish to obtain a pointer to one of its interfaces without incrementing the reference count), but it is not acceptable to do this by calling QueryInterface followed by Release. The pointer obtained in such a manner is invalid and should not be used. This more readily becomes apparent when _ATL_DEBUG_INTERFACES is defined, so defining this macro is a useful way of finding reference counting bugs.

2.4.7 Marshalling

The COM technique of marshaling allows interfaces exposed by an object in one process to be used in another process. In marshaling, COM provides code (or uses code provided by the interface implementor) both to pack a method's parameters into a format that can be moved across processes (as well as, across the wire to processes running on other

machines) and to unpack those parameters at the other end. Likewise, COM must perform these same steps on the return from the call.

Marshaling is typically not necessary when an interface provided by an object is being used in the same process as the object. However, marshaling may be needed between threads.

2.4.8 Aggregation

There are times when an object's implementer would like to take advantage of the services offered by another, prebuilt object. Furthermore, it would like this second object to appear as a natural part of the first. COM achieves both of these goals through containment and aggregation.

Aggregation means that the containing (outer) object creates the contained (inner) object as part of its creation process and the interfaces of the inner object are exposed by the outer. An object allows itself to be aggregatable or not. If it is, then it must follow certain rules for aggregation to work properly.

Primarily, all **IUnknown** method calls on the contained object must delegate to the containing object.

In creating an object that supports aggregation, you would need to implement one set of IUnknown functions for all interfaces as well as a stand-alone IUnknown interface. In any case, any object implementor will implement IUnknown methods.

2.5 Microsoft DirectX 9.0

Microsoft® DirectX® is a set of low-level application programming interfaces (APIs) for creating games and other high-performance multimedia applications. It includes support for two-dimensional (2-D) and three-dimensional (3-D) graphics, sound effects and music, input devices, and networked applications such as multiplayer games.

2.5.1 Components of DirectX

DirectX has the following main components and each component has separate Application interfaces to serve their purpose. These components are discussed briefly.

- Direct3D Graphics
- DirectDraw
- DirectInput
- DirectPlay
- DirectSound
- DirectMusic
- DirectShow

2.5.1.1 Direct3D Graphics

Microsoft® Direct3D® is a low-level graphics application programming interface (API) that enables users to manipulate visual models of 3-dimensional objects and take advantage of hardware acceleration, such as video graphics cards

2.5.1.2 DirectDraw

The DirectDraw application programming interface (API) enables users to directly manipulate display memory, the hardware blitter, hardware overlay support, and flipping surface support.

2.5.1.3 DirectInput

Microsoft® DirectInput® (API) is used to process data from a keyboard, mouse, joystick, or other game controller.

2.5.1.4 DirectPlay

The DirectPlay application-programming interface (API) provides developers with the tools to develop multiplayer applications such as games or chat clients. DirectPlay provides many features that simplify the process of implementing many aspects of a multiplayer application, including:

- Creating and managing both peer-to-peer and client/server sessions.
- Managing users and groups within a session.
- Enabling applications to interact with lobbies.
- Enabling users to communicate with each other by voice.

2.5.1.5 DirectSound

Microsoft® DirectSound® provides a system to capture sounds from input devices and play sounds through various playback devices using advanced 3-dimensional positioning effects, and filters for echo, distortion, reverberation, and other effects.

2.5.1.6 DirectMusic

DirectMusic provides support for MIDI, downloadable sounds, and consistent playback on the Microsoft Windows platform.

2.6 Introduction to DirectShow

Microsoft® DirectShow® is architecture for streaming media on the Microsoft Windows® platform. DirectShow provides for high-quality capture and playback of multimedia streams. It supports a wide variety of formats, including Advanced Systems Format (ASF**)**, Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV sound files. It supports capture from digital and analog devices based on the Windows Driver Model (WDM) or Video for Windows. DirectShow is integrated with other DirectX technologies. It automatically detects and uses video and audio acceleration hardware when available, but also supports systems without acceleration hardware.

DirectShow simplifies media playback, format conversion, and capture tasks. At the same time, it provides access to the underlying stream control architecture for applications that require custom solutions.

DirectShow is based on the Component Object Model (COM). To write a DirectShow application or component, it is necessary to understand COM client programming.

2.6.1 Header Files and Library Files

All DirectShow applications use the header file shown in the following table. The DirectShow also uses the library file shown in table 2.2.

Header File	Required For
Dshow.h	All DirectShow applications.

Table 2.1 Header Files for Direct Show

Library File	Description
Strmiids.lib	Exports class identifiers (CLSIDs) and interface identifiers (IIDs). All DirectShow applications require this library.

Table 2.2 Library Files

2.6.2 DirectShow Filters

DirectShow uses a modular architecture, where each stage of processing is done by a COM object called a filter. DirectShow provides a set of standard filters for applications to use, and developers can write their own custom filters that extend the functionality of DirectShow.

For example, DirectShow filters can

- read files
- get video from a video capture device
- decode various stream formats, such as MPEG-1 video
- pass data to the graphics or sound card

Filters receive input and produce output. For example, if a filter decodes MPEG-1 video, the input is the MPEG-encoded stream and the output is a series of uncompressed video frames.

In DirectShow, an application performs any task by connecting chains of filters together, so that the output from one filter becomes the input for another. A set of connected filters is called a *filter graph*.

Filters have three possible states: running, stopped, and paused. When a filter is running, it processes media data. When it is stopped, it stops processing data.

Filters can be grouped into several broad categories:

- A ***source*** filter introduces data into the graph. The data might come from a file, a network, a camera, or anywhere else. Each source filter handles a different type of data source.
- A ***transform*** filter takes an input stream, processes the data, and creates an output stream. Encoders and decoders are examples of transform filters.
- ***Renderer*** filters sit at the end of the chain. They receive data and present it to the user. For example, a video renderer draws video frames on the display; an audio

renderer sends audio data to the sound card; and a file-writer filter writes data to a file.

- A ***splitter*** filter splits an input stream into two or more outputs, typically parsing the input stream along the way. For example, the AVI Splitter parses a byte stream into separate video and audio streams.
- A ***mux*** filter takes multiple inputs and combines them into a single stream. For example, the AVI Mux performs the inverse operation of the AVI Splitter. It takes audio and video streams and produces an AVI-formatted byte stream.

2.6.3 IGraphBuilder Interface

This interface provides methods that enable an application to build a filter graph. IGraphBuilder provides basic operations, such as adding a filter to the graph or connecting two pins. IGraphBuilder adds further methods that construct graphs from partial information. For example, the IGraphBuilder::RenderFile method builds a graph for file playback, given the name of the file. The IGraphBuilder::Render method renders data from an output pin by connecting new filters to the pin.

2.6.4 ICaptureGraphBuilder2 Interface

The ICaptureGraphBuilder2 interface builds capture graphs and other custom filter graphs. Since capture graph are often more complicated than file playback graphs, ICaptureGraphBuilder2 make it easier to build a capture graph. To make it easier for applications to build capture graphs, DirectShow provides a helper object called the Capture Graph Builder. The ICaptureGraphBuilder2 interface contains methods for building and controlling a capture graph.

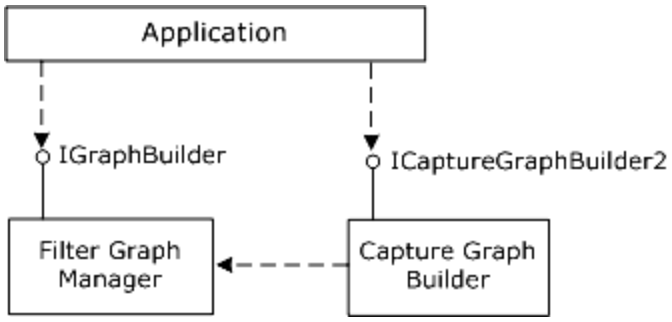


Figure 2.1 ICaptureGraphBuilder2 Interface

2.6.5 IMediaControl Interface

The IMediaControl interface provides methods for controlling the flow of data through the filter graph. It includes methods for running, pausing, and stopping the graph.

2.6.6 IMediaEvent Interface

The IMediaEvent interface contains methods for retrieving event notifications and for overriding the Filter Graph Manager's default handling of events. Applications can use it to respond to events that occur in the filter graph, such as the end of a stream or a rendering error.

2.6.7 IVideoWindow Interface

The IVideoWindow interface sets properties on the video window. Applications can use it to set the window owner, the position and dimensions of the window, and other properties. The Video Renderer filter and the Filter Graph Manager both expose this interface. The Filter Graph Manager forwards all method calls to the Video Renderer.

2.6.8 ICreateDevEnum Interface

The ICreateDevEnum interface creates an enumerator for devices within a particular category, such as video capture devices, audio capture devices, video compressors, and so forth.

Applications can use this interface to enumerate devices and to create the DirectShow filter that manages each device. The CreateClassEnumerator method returns an enumerator object for a specific device category. The enumerator object supports the IEnumMoniker interface and returns a list of monikers, where each moniker represents a different device. In some cases, the same DirectShow filter manages an entire category of devices.

2.6.9 IBaseFilter Interface

The IBaseFilter interface provides methods for controlling a filter. All DirectShow filters expose this interface. The Filter Graph Manager uses this interface to control filters. Applications can use this interface to enumerate pins and query for filter information, but should not use it to change the state of a filter. Instead, use the **IMediaControl** interface on the Filter Graph Manager.

2.6.10 ISampleGrabberCB Interface

The ISampleGrabberCB interface provides callback methods for the ISampleGrabber::SetCallback method. The Sample Grabber filter provides a way to retrieve samples as they pass through the filter graph. It is a transform filter with one input pin and one output pin. It passes all samples downstream unchanged, so you can insert it into a filter graph without altering the data stream. The application can then retrieve individual samples from the filter by calling methods on the ISampleGrabber interface.

2.6.11 Video Capturing

The term *video capture* describes any application where video is received from a hardware device. Video capture devices include not only cameras, but also TV tuner cards, video tape recorders (VTRs), and so forth. The captured video can be saved to disk or previewed live.

2.6.12 Pin Categories

A capture filter often has two or more output pins that deliver the same kind of data for example, a preview pin and a capture pin. Therefore, media types are not a good way to distinguish the pins. Instead, the pins are distinguished by their functionality, which is identified using a GUID, called the *pin category*.

Preview Pins and Capture Pins

Some video capture devices have separate output pins for preview and capture. The preview pin is used to render video to the screen, while the capture pin is used to write video to a file. The pin category for preview pins is PIN_CATEGORY_PREVIEW. The category for capture pins is PIN_CATEGORY_CAPTURE.

2.6.13 Selecting a Capture Device

To select a capture device, use the System Device Enumerator. This helper object returns a collection of device monikers, selected by filter category. (A moniker is a COM object that contains information about another object, which enables the application to get information about the object without creating the object itself. Later, the application can use the moniker to create the object)

For capture devices, the following categories are relevant.

Category GUID	Description
CLSID_AudioInputDeviceCategory	Audio capture devices
CLSID_VideoInputDeviceCategory	Video capture devices

Table 2.3 Categories of capture devices

2.6.14 Previewing Video

To build a video preview graph call the **ICaptureGraphBuilder2::RenderStream** method. The following diagram shows the simplest possible graph for previewing video.

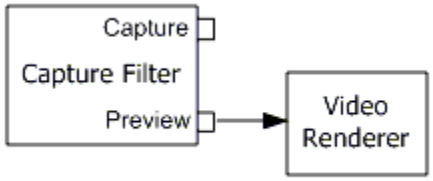


Figure 2.2 Previewing video

In this diagram, the capture filter has a preview pin, which connects directly to the video renderer.If the capture filter has only a capture pin, the Capture Graph Builder inserts a Smart Tee filter, which splits the stream into a capture stream and a preview stream. In some cases, the video stream must go through the Overlay Mixer filter. If so, the **RenderStream** method adds it to the graph automatically.

2.7 ADO (ActiveX Data Objects)

ADO is a COM component that provides Automation compatible interfaces used by high level programming languages to access data, normally using OLE DB providers. An OLE DB is a set of COM interfaces that define abstract access to data stored in diverse information sources. OLE DB providers provide implementations of the OLE DB interfaces. ADO is an object model layered upon OLE DB to provide access to OLE DB provides through oleautomation-compatible interfaces.

2.7.1 Purpose

ADO provides a COM-based application-level interface for OLE DB data providers. ADO supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects using live connections to data in relational databases and other stores. And, like ADO.NET, ADO can construct client-side recordsets, use loosely coupled recordsets, and handle OLE DB's data shaping rowsets.

ADO also supports some behaviors such as scrollable, server-side cursors. However, because server-side cursors require holding database resources, their use might have a significant negative impact on the performance and scalability of the application

ADO's ease of use, speed, and low memory overhead make it ideal for server-side scripting. In fact, ADO is the recommended technology for data access for ASP applications. ADO can be called directly from server-side scripts or from business components.

Unlike earlier data access methods, ADO does not require navigation through a hierarchy to create objects; most ADO objects can be created independently, which allows greater flexibility in reusing objects in different contexts and reduces memory consumption. ADO also takes advantage of ODBC 3.0 connection pooling for ODBC data sources, and session pooling for OLE DB providers. This eliminates the need to continuously create new **Connection** objects for each user, which is very resource intensive.

2.7.2 Limitations

What ADO cannot do, however, is provide remote data to the client. Once the data has been retrieved and sent to the browser, the user cannot easily manipulate it or make changes to it within the client application. Data operations-including filtering and record modifications-must take place on the server, where the actual data manipulation objects reside.

2.7.3 OLE DB

OLE DB, the foundation of the Universal Data Access model, is a set of COM interfaces that provides a standard way for programs to access data. OLE DB is the strategic system-level programming interface for accessing data, and is the underlying technology for ADO as well as a source of data for ADO.NET. The way applications use ADO functionality is partially determined by whether or not there is an OLE DB provider for the data.

OLE DB is an open standard for accessing all kinds of data — both relational and non-relational data including: mainframe ISAM/VSAM and hierarchical databases; e-mail and file system stores; text, graphical, and geographical data; and custom business objects.

OLE DB provides consistent, high-performance access to data and supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects using live connections to data in relational databases and other stores.

2.7.4 ADO Data Controls

A data control lets you define a query with which to access data. You then connect data-bound controls to a data source. The ADO controls have been used to provide access to the entire database records by specifying a single query. The following table shows the require support files for ADO Controls.

Control	Support files
Microsoft ADO Data Control (ADODC)	Msadodc.oxc; help file is Adodc98.chm.
Microsoft RemoteData Control (MSRDC)	Msrdc20.ocx; help file is RDO98.chm.

Table 2.4 ADO Controls and Support files

2.7.5 Using Data Controls with Data-Bound Controls

Visual C++ 6.0 and later provides a new set of data-bound controls based on ADO, which is a COM wrapper for OLE DB; it also provides a new data control (ADODC).

- Visual C++ 6.0 and later does not allow you to use the version 6.0 ADO data-bound controls with the older remote data control (MSRDC). You must use RDO data-bound controls with the remote data control (MSRDC).
- You also cannot use the older remote data-bound controls with the version 6.0 ADO data control (ADODC). You must use ADO data-bound controls with the ADO data control (ADODC).
- The exception is for simple bound data controls, which may be used interchangeably with the MSRDC and ADODC. The simple bound controls include the MS Masked Edit control and the MS Rich Text control.

Chapter 3

METHODOLOGY

3.1 Introduction

This chapter includes a brief account of the existing target tracking systems and the techniques used in such system. It also describes the proposed system and its benefits along with the functional and non functional requirements of the system. The methodology used for the development of the system is based on those techniques and procedures that may help to remove the problems and limitations faced by previous target tracking system.

3.2 Existing System

The problem of target tracking has stymied engineers, mathematicians, and computer scientists for years. During World War II American scientists tried, in the days before computers, to develop mathematical algorithms that would help to track enemy aircraft and automatically guide an anti-aircraft gun to shoot it down. The problem of tracking high speed targets has existed since the first time aircraft were used in battle. Improvements are constantly required in defense systems to make them impregnable and give our forces a realistic chance of defending our boundaries.

3.2.1 Previously used techniques for Target Tracking

Most of the techniques used for tracking moving objects deal with a stationary camera or closed world representations which rely on a fixed background or a specific knowledge on the type of actions taking place. Different methods may be chosen to track targets. “Radar based tracking” and “Visually guided tracking” are two such methods. Radar based tracking is an alternative solution to visually guided tracking. Visually guided

tracking of targets is a skill used in nearly all activities in order to maintain gaze on objects moving with respect to the eyes. But it becomes slightly more complicated when related to the battlefield.

Moreover the target tracking algorithms developed for tracking targets based on sonar and radar measurements could be used for tracking (also known as motion correspondence) but for a system to operate at video frame-rate (possibly even higher rates) limits the use of these well-established statistical and non-statistical tracking algorithms.

3.2.2 Drawbacks of Existing System

- Previous target tracking systems did not provide any information regarding the tracking skills of a human gunner.
- These systems cannot be used as a training tool for training personnel and for evaluation of their shooting skills.
- Visually guided tracking uses analog signal to preview and capture video and does not allow any processing on the video stream.
- None of these systems allow real time or live shooting of aircrafts without wasting shells.
- The reliance on human expertise, cost and consumption of real missiles are other drawbacks of such target tracking systems.

3.3 Proposed System

The proposed solution to the problems of existing system is to develop software to identify and track targets by their measured positions and motion parameters derived from video stream processing. The proposed system will break the video stream into frames and then will calculate motion in them. In addition the proposed system will generate tracking information and the distance of the target from the aim at all times.

Such a system can be used to provide the tracking information of the human gunner and evaluate his shooting skills. It also removes the limitations of using target tracking as a training tool. Real time and live shooting of aircrafts is allowed in this system since it digitizes the analog signals obtained from the camera and allows for processing on this data.

3.3.1 Target Tracking

The video input is acquired from a high speed camera mounted on the anti aircraft machine and it is read through a TV tuner card. These frames are saved in the form of gray scale images. Then noise is eliminated to obtain foreground objects. The target is identified by using Union Find algorithm in each frame. This algorithm identifies the target separately in each frame and does not rely on the information of location of the target in the previous frames. Target tracking is performed in each frame independently. The algorithm runs in two passes through the binary image to give the labeled image of connected components and it populates two data structures in each pass. In first pass union–find data structure is populated with values, it store a collection of disjoint sets to efficiently implement the operation of union to merge two set into one and find which set a particular element is in. This is accomplished with a vector array of PARENT. In second pass using the information in PARENT, *find* operation is performed.

3.4 Functional Requirements

The Real Time Testing Environment should be capable of providing live image acquisition area for generating user specific environment with real guns; the gun should be calibrated with the aim of the camera. The Gun profiler should be capable of Providing interface transparency for input (the input is fed by the Camera and the gun on which the camera is mounted), Removing noise from the input, Extracting designated target in each frame, Estimating the distance of the target from the aim and generating output in aspect of tracking graph.

3.5 Non Functional Requirements

The system was originally targeted to perform according to these non functional constraints: The generated system accepts frames at a rate of 30 FPS (Frames per Second). This is to present a smooth live view of the real environment to the user. The System should receive frames synchronously from the camera at a frame rate of 30 FPS. Transmission of images between the modules should be in sequence and within acceptable information loss (image should be clear enough for the target identification step). The decisions taken regarding the design and development of the system are elaborated below.

3.5.1 Challenges of Visually Guided Tracking

The first thing to keep in mind is that the computer can't see objects. It takes in video information as a grid of pixels. It can tell you a pixel's position and its color (if you are using a color camera). From these facts, other information can be determined; the brightest pixel can be determined by seeing which pixel has the highest color values, a "blob" of color can be determined by choosing a starting color, setting a range of variation, and checking the neighboring pixels of a selected pixel to see if they are in the range of variation, areas of change can be determined by comparing one frame of video with a previous frame, and seeing which pixels have the most significantly different color values, and areas of pattern can be followed by selecting an area to track, and continuing to search for areas that match the pattern of pixels selected.

Colors to be tracked need consistent lighting. The computer can't tell if an object is red, for example; it can tell that one pixel or a range of pixels contains the color value [255, 0, 0] perhaps, but if the lighting changes and the object appears gray because there is no red light for it to reflect, the computer will no longer "see" it as red.

Objects to be tracked need to stay somewhat consistent in shape. The computer doesn't have stereoscopic vision (two eyes that allow us to determine depth by comparing the difference in image that our two eyes receive), so it sees everything as flat. If an object turns sideways with respect to the camera, the pattern changes because the object appears thinner. So the computer may no longer recognize it as the object it "saw" before.

One simple way of getting consistent tracking of light (or lit objects) is to reduce the amount of information the computer has to track. For example, if the camera is equipped with an infrared filter, it will see only infrared light. This is very useful, since incandescent sources (light-bulbs with filaments) give off infrared, whereas fluorescent sources don't. Furthermore, the human body doesn't give off infrared light either. This is also useful for tracking in front of a projection, since the image from most LCD projectors contains no infrared light.

3.5.2 Challenges of Tracking High Speed Objects in Real Time

The additional challenges posed by the tracking of high speed objects are derived from the demand for increased sample rates, higher picture quality and better frame rates.

Real Time Tracking inevitably imposes a demand for increased computational power due to the high amount of computing involved. The need for quick response implies a need for employing efficient algorithms.

3.6 Benefits

The benefit of this project lies in many spheres, the first being the relatively less expense at which it will help to train personnel for our anti-aircraft systems.

This engineering project provides a pioneering effort into the visual tracking field in Pakistan, which other people can also build on. It ensures the real time training of Pakistan’s anti-aircraft artillery (AAA) most of which is obsolete in its present forms and is in serious need of upgrade and transformation. This project aims to achieve this transformation at very low cost, by reducing operator errors and increasing accuracy and

reliability of existing hardware (anti aircraft weapons). Not only does it reduce reliance on human expertise, but is also cost-effective.

The testing tool developed during this period provides an opportunity for any developing team to test their weapons and develop a similar tracker before taking it out into the field since it performs roughly the same in all weather conditions and stress environments. The cost benefits associated with this utility are immense. The exorbitant cost of live-testing cannot be overemphasized. Another benefit arises from its hardware independence. It is applicable across a wide variety of platforms.

To summarize: It gets automation into our anti aircraft systems most of which are obsolete now. It reduces chances of operator (human) errors, increases accuracy and reliability of existing hardware (anti aircraft weapons), reduces reliance on human expertise, performs roughly the same in all weather conditions and stress environments, and provides a pioneering effort into the visual tracking field in Pakistan which other people can also build on. It is Hardware Independent and the technology is very versatile. Most importantly it avoids the exorbitant cost of live-testing. The tracking skills of the gunner can be evaluated efficiently. Digitized video data is made available for processing. The consumption cost of shells and bullets can be saved. A real time training tool is available to train personnel aircraft shooting without wasting shells

Chapter 4

SYSTEM MODEL

4.1 Introduction

The proposed system is a five-stage process which is shown in block diagram of figure 1. The first stage is to acquire the image from the camera that is mounted on an anti-aircraft gun. Once the image is acquired it is processed further to detect the target and to track it in following frames. The position of the target object with respect to the aim is estimated so as to determine the deviation of the target from the aim in all frames. This information is used to generate the graphical representation of the aiming path. Each of these modules is discussed in detail.

4.1.1 Video Input

The process is the basic requirement of the software getting the streaming video from the video capturing device to user view.

Inputs: Video Stream

Processing: The process gets the stream of video from the video capturing device and renders to the output stream. The video input is acquired from a high speed camera that is mounted on the anti aircraft machine. A TV tuner card is used to read the input from the camera and store the frames on the hard disk.

Output: If a video capturing device receives a stream the stream is fetched and rendered. The rendered video frames are the output of this process.

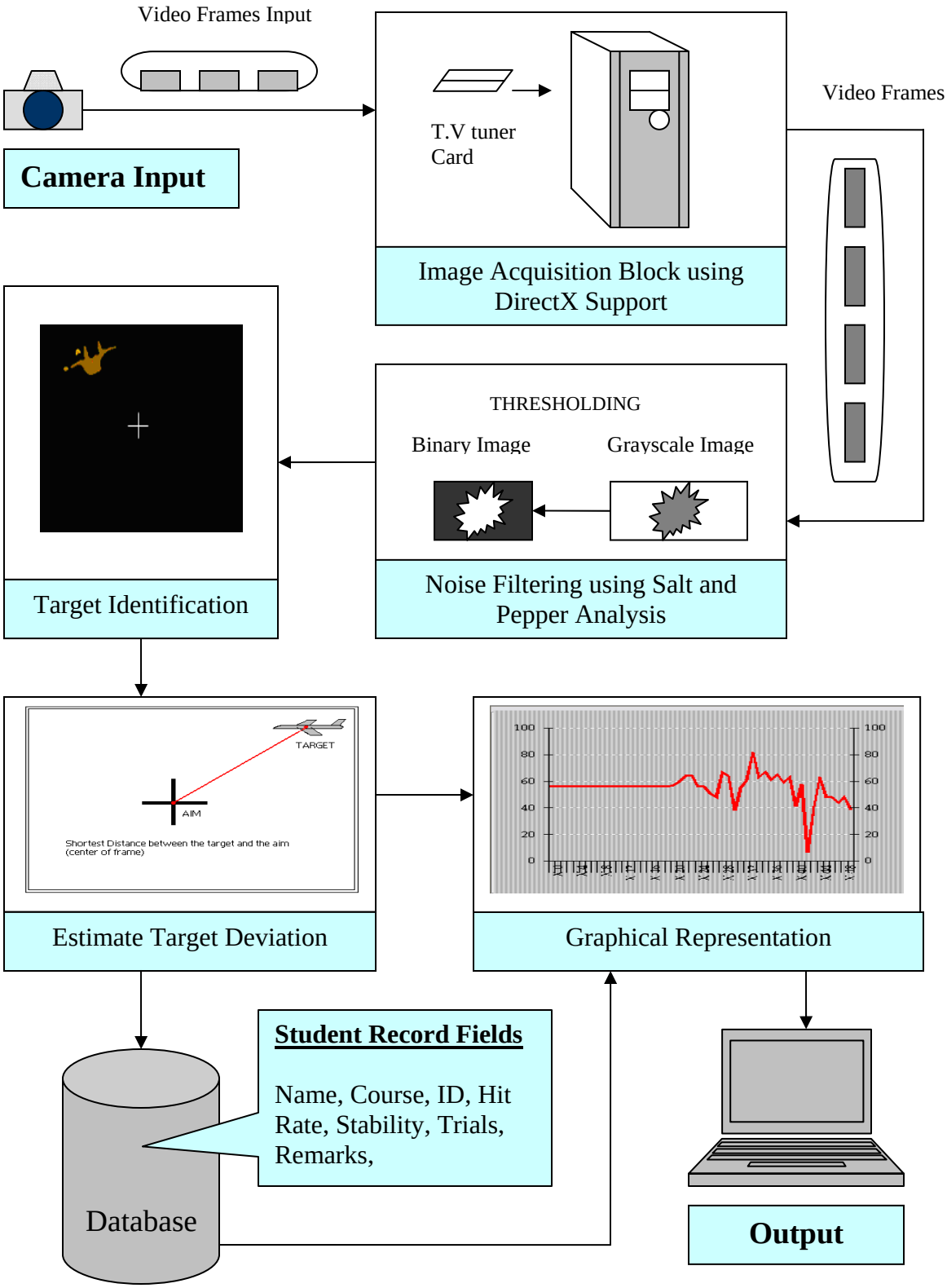


Figure 4.1 SYSTEM DETAILED BLOCK DIAGRAM

4.1.2 Image Acquisition

The Block includes the capturing of the rendered frames for processing, it allows the user to start, stop and preview the video image.

Inputs: Rendered Video Frames.

Processing: This is done by using the DirectX support for capturing multimedia streams. The image sequences are being acquired from a camera however the camera is mounted on a gun and the images are dependent upon the movement of the gun. While the gunner aims for a moving target from his gun, the system has to rely on the field view of the gunner to detect and track the moving target as the camera can not move independently to locate the target.

Output: Acquired real video images which can be further processed.

4.1.3 Noise Filtering

The next process is to eliminate noise from the image data obtained so that the foreground objects can be distinguished from the background.

Inputs: Acquired Video Frames.

Processing: This process converts real images into gray scale then applies thresholding on them to obtain binary images. Thresholding enables the selection of a range of pixel values in grayscale and color images to separates the target blob object under consideration from the background. The Salt and Pepper Analysis is used to remove noise introduced by thresholding.

Output: Clear Binary Images.

4.1.4 Target Identification

Next stage is a process to identify the moving target which is an object that moves independently of the observer.

Inputs: Thresholded Binary Image of the Current Frame.

Processing: The algorithm identifies this target as a blob by assuming that the background contains no objects. It only considers the objects that are present in the foreground. Since we are dealing with targets such as aircrafts in the air, our assumption is quite close to reality because there is significant variance in the shape or orientation of a high speed moving target as compared to any other object in the sky.

Output: Identified Target Object or Blob.

4.1.5 Target Deviation

Once the target is identified and under tracking, the next step is to relate the information of the target movement with the aim of the gunner.

Inputs: Identified Target Object, its Center, its Size and Distance of object from aim.

Processing: Once the gunner presses the trigger this information is used to capture the location of the target when the trigger is pressed. At the exact moment of the trigger pressing the location of the target with respect to the aim of the gunner and other motion parameters are estimated to determine the hit or missed condition of the target aircraft. The relative size of the target as it appears on the image is used as a motion parameter to estimate the distance location of the target from the gun. To eliminate any ambiguities in the size estimation of distance it is assumed that the aircraft is gliding side ways horizontally in front of the camera when it is shot. The size and orientation of the aircraft when moving towards or away from the gunner would violate the size

estimation that is based on the fact that distant objects appears smaller and closer objects appear bigger in the image.

Output: Distance of the target from the aim in each Binary frame image.

4.1.6 Database

The estimated distances from the aim are written down in a database for future reference. This process includes the Add, Update and Delete operations on the Database.

Inputs: Student Record fields as provided by the user and calculated target distance from aim.

Processing: Add a new record or Update the existing record by the information provided by user. It also allows for the deletion of previous records. The database can be navigated to find different records as well.

Output: The record of each student along with his tracking efficiency.

4.1.7 Graphical Representation

The final stage process is to generate a graphical representation of the tracking path followed by the gunner while aiming for the moving target.

Inputs: Target Distance from aim of database or from the target deviation block.

Processing: Three graphical representations are used for this purpose that show the horizontal, vertical and the radial deviation of the target from the aim.

Output: Horizontal, Vertical and Radial Deviation Graphs.

4.2 Software System Attributes

Software System Attributes discusses about the reliability, availability, security and maintainability of the system.

4.2.1 Reliability

The system must be able to provide all the basic functionality to the user without any delays. All the information must be available at the right time, when it is needed. System must take care of selecting the default devices.

4.2.2 Availability

Proposed system must be able to render the video from any of the capturing devices and should be able to accept and enumerate any new installed device.

4.2.3 Maintainability

Since the system can merge new devices into itself, so it is easier to maintain the system. There is also a proper communication interface defined for each device, so entry of a new device will not create any problems and system will upgrade itself.

4.2.4 Usability

System will fulfill all functional and non-functional requirements in efficient manner.

4.2.5 Economical

The system will be economical so that it may be easily affordable and must provide all its services.

CHAPTER 5

DESIGN

5.1 Introduction

This chapter discusses the design process of the software. Design is the process of building a model or representation of an entity that will be built, the design expands what was learned during analysis phase into a working implementation. Design is a series of decisions on which implementation of the software is carried out. This chapter discusses the four major area of concern for the system that is the data, architecture, interfaces, and components.

Figure 7.2 Camera Calibration

5.2 Software Design

Software design is formulized using the Object Oriented Design methodology. The main components of this are the following diagram,

- Use Case Diagram
- Data Flow Diagram
- Activity Diagram
- Class Diagram

5.3 Data Design

The data design transforms the information domain into the data structures that will be required to implement the software. The data objects and relationships defined in the entity relationship diagram and the detailed use case diagrams provide the basis for the data design activity.

A use case diagram illustrates a set of use cases for the software, the actors, and the relation between the actors and use cases. The purpose of this diagram is to represent a kind of context diagram by which one can quickly understand the external actors of software and the key ways in which they use it. The following are the software use-case diagram components which are actors, use cases and the relationship between them.

5.3.1 Actors

An actor is an entity who in some way participates in the story of use case. The actors involved in the proposed system are

- **Learner / Trainee:** The person who is tracking and aiming at the target from his gun.
- **User / Evaluator**: User is a person who is using the software.
- **PC**: Personal computer that is used to run the software.
- **Camera**: It's the camera that is mounted on the trainee's gun for previewing video.
- **Database**: The database holds the records of students

5.3.2 Use-Cases

A use-case describes a process. A use-case is the action performed by actors.

Uses cases involved in the system are:

- Tracking Movement of the gun.
- Receive the video frames.
- Preview the video.
- Evaluate the gunner ability to track.
- Preview the graphical representation of the aiming path.
- Manage and update the database.

Figure 5.1, 5.2 shows the UML icon for a use case actor and a use case.

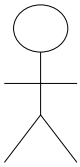


Figure 5.1 Actor icon in UML

Figure 5.2 Use case icon in UML

5.3.3 Use-case Diagram

The following is the software use-case diagram, which shows actors, use cases and the relationships between them.

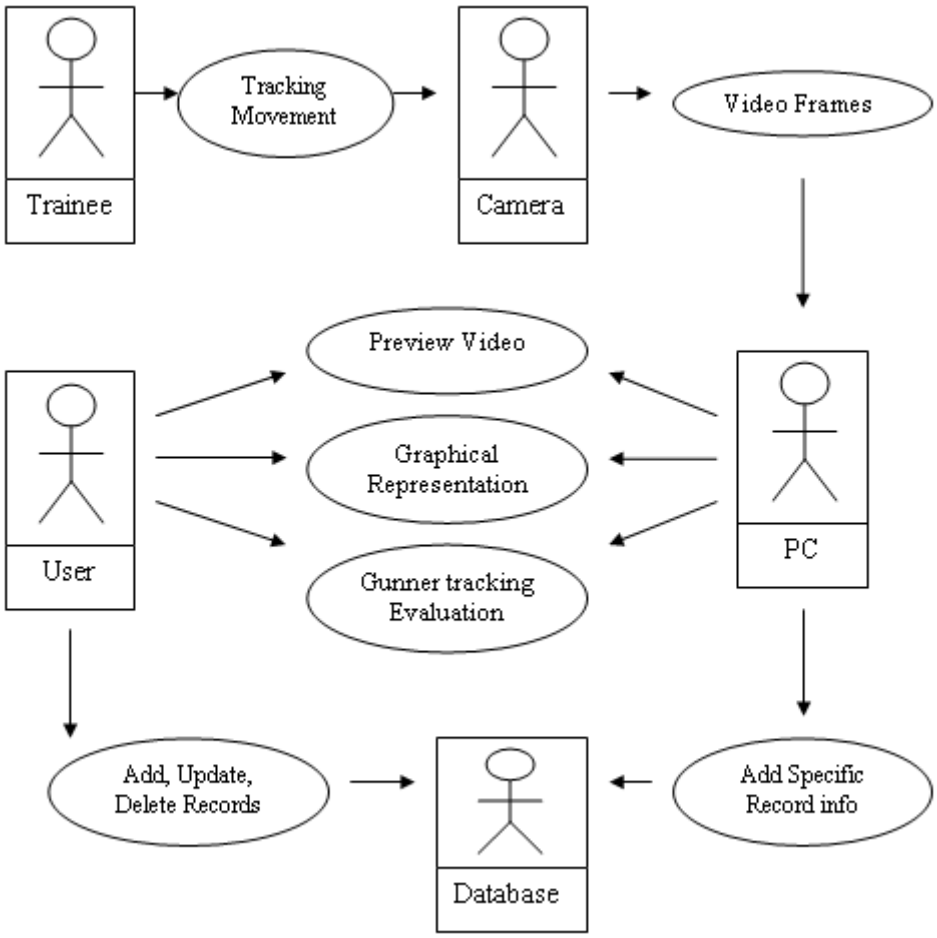


Figure 5.3 Use-Case Diagram

5.3.4 Uses-cases Description

UC-01: **Tracking Movement of the Gun**	
Actors	Trainee, Camera
Assumptions	The camera is mounted on the Gun.
Pre-conditions	The center of camera is calibrated with the gun aim.
Use Case Description	The user can record the tracking path of the gun with respect to the

	target in its view.
Use Case Initialization	This use case starts when camera starts capturing video data.
Use Case Termination	This use case terminates when the triggers is pressed or tracking time expires.
Post-conditions	Previewing of stream

UC-02: **Receive the video Frames**	
Actors	Camera, PC
Assumptions	No assumptions for this use case.
Pre-conditions	Audio Video Type signal at the T.V tuner Capturing card
Use Case Description	The capture device(camera) sends the video to PC
Use Case Initialization	This use case starts on demand.
Use Case Termination	This use case terminates when triggers is pressed or tracking time expires.
Post-conditions	Upon successful the system will start analyzing the video information

UC-03: **Preview the video**	
Actors	User, PC
Assumptions	No assumptions for this use case.
Pre-conditions	Audio Video Type signal at the T.V tuner Capturing card
Use Case Description	The software takes the video from video capturing device as a camera and previews it.
Use Case Initialization	This use case starts when training or trial begins.
Use Case Termination	This use case terminates when stop preview or trigger is pressed.
Post-conditions	Upon successful the system will start previewing the video

UC-04,05: **Gunner Tracking Evaluation and Graphical Representation**	
Actors	User, PC
Assumptions	A video is being rendered and target is identified.
Pre-conditions	The video stream is being saved on hard disk.
Use Case Description	This use case provides the user functionality of getting the tracking evaluation of the gunner from the video data obtained.
Use Case Initialization	This use case starts when user stops preview and selects stability.
Use Case Termination	This use case terminates when user exits the graphical representation menu.
Post-conditions	Video Preview

UC-06: **Add, Update, Delete Records**	
Actors	User, PC, Database
Assumptions	A video is being rendered, tracking evaluation performed
Pre-conditions	The video stream is being rendered, student is training for evaluation
Use Case Description	If the user wants database management the specific student record is managed by the software.
Use Case Initialization	This use case starts when user selects Training or database management.
Use Case Termination	This use case terminates when user exists database.
Post-conditions	Upon successful the database will reflect the updates made to it.

5.4 Architectural Design

The architectural design represents the structure of data and the program components that are required to build a computer-based system. It considers the architectural style that the system will follow and defines the relationship between major structural elements of the software that can be used to achieve the requirements that have been defined for the system. The activity diagram shows the dynamic behavior of the system and the relationships between system modules.

5.4.1 Activity Diagram

Activity diagram is an activity oriented diagram. An activity is a task that needs to be done by a human or by a computer. From a specification perspective or an implementation perspective, an activity is a method for a class. It represents a series of activities that need to be completed to do a particular function. It shows behavior with control structure, also shows many objects in single use case and implementation of methods.

Figure shows the UML icons for an activity diagram. Activity diagram for the Image Acquiring phase is shown in figure 5.5.

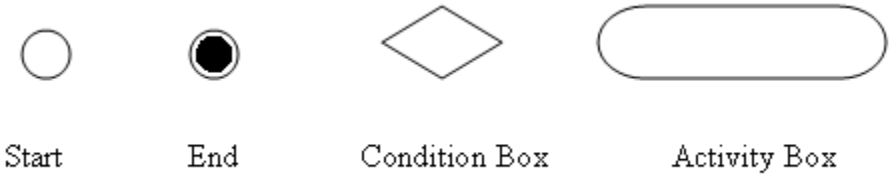


Figure 5.4 UML icons for Activity Diagram

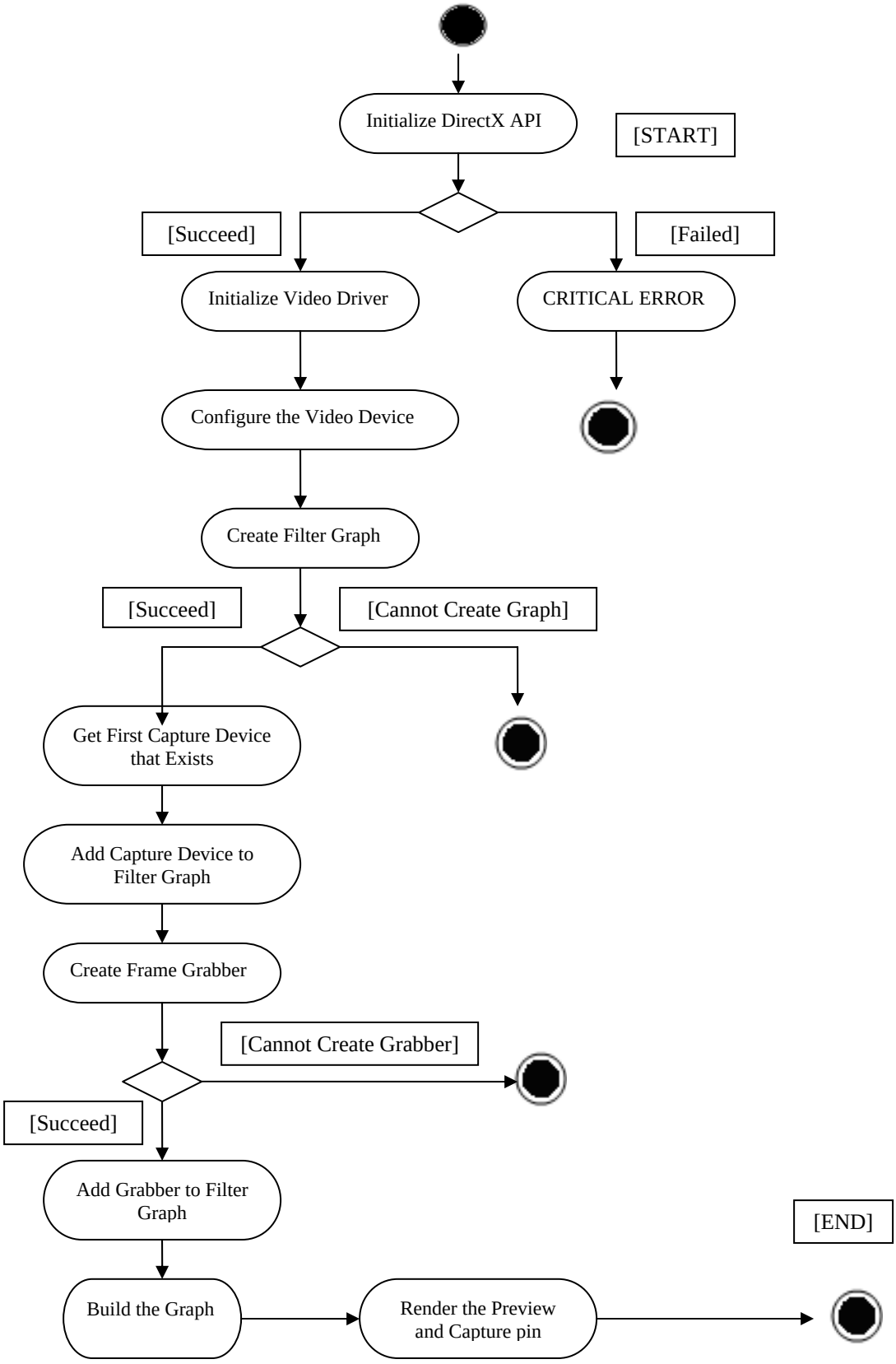


Figure 5.5 Activity diagram for the Image Acquiring phase

Activity diagram for the Target Identification module is given below.

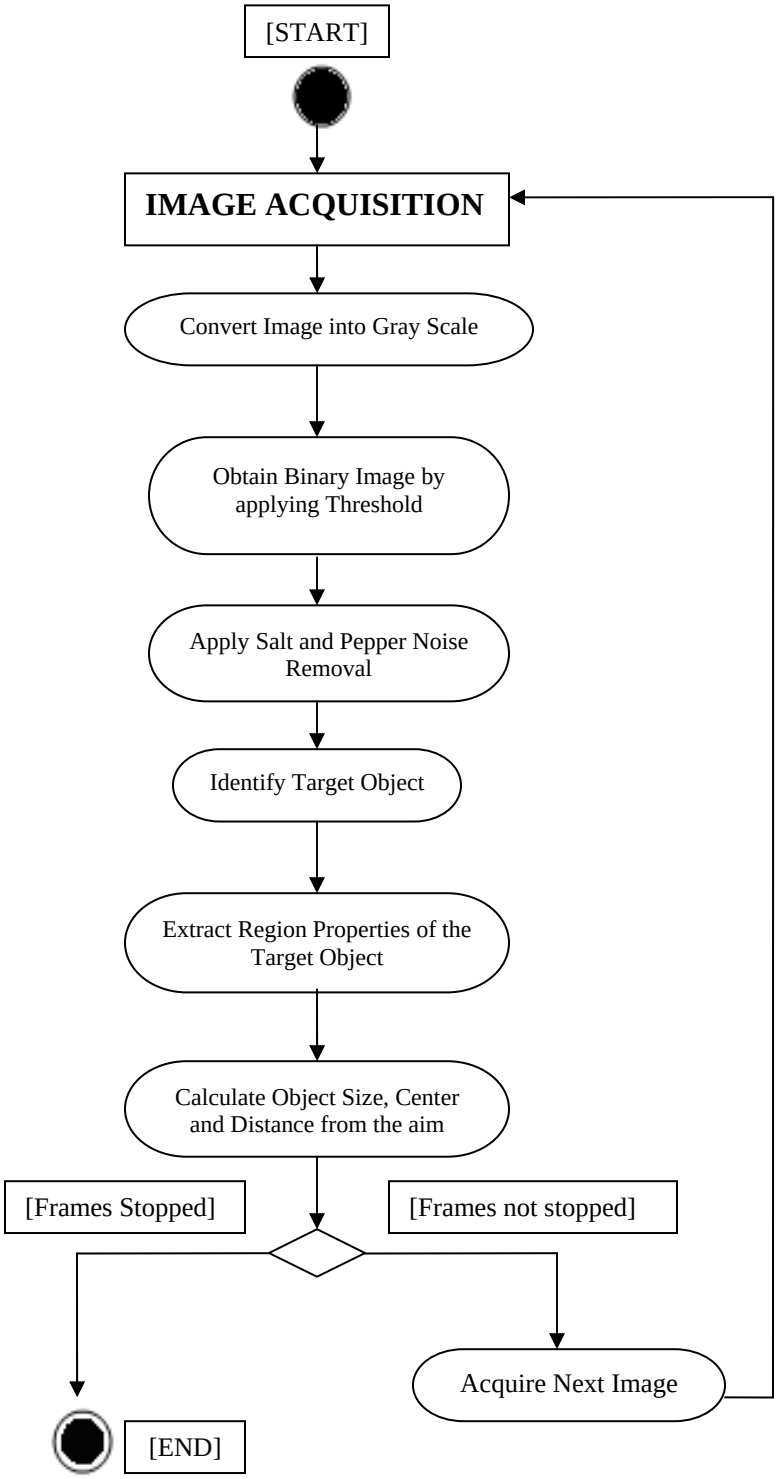


Figure 5.6 Activity diagram for the Target Identification Phase

5.5 Interface Design

The interface design describes how the software communicates within itself, with systems that interoperate with it, and with humans who use it. An interface implies a flow of information and a specific type of behavior. Therefore data and control flow diagrams provide much of the information required for interface design. The DFD provides an indication of how data are transformed as they move through the system and depict the functions that transform the data flow.

5.5.1 Data Flow Model

The data flow diagram model gives details such as the primary data objects to be processed by the system, the composition of each data object and the attributes that describe this object, the relationships between each object.

The UML notations for the Data Flow Diagram are shown below. Nodes are the data objects and the links are the transformations that occur to translate one data object into another.

Figure 5.7 UML notations for Data Flow Model

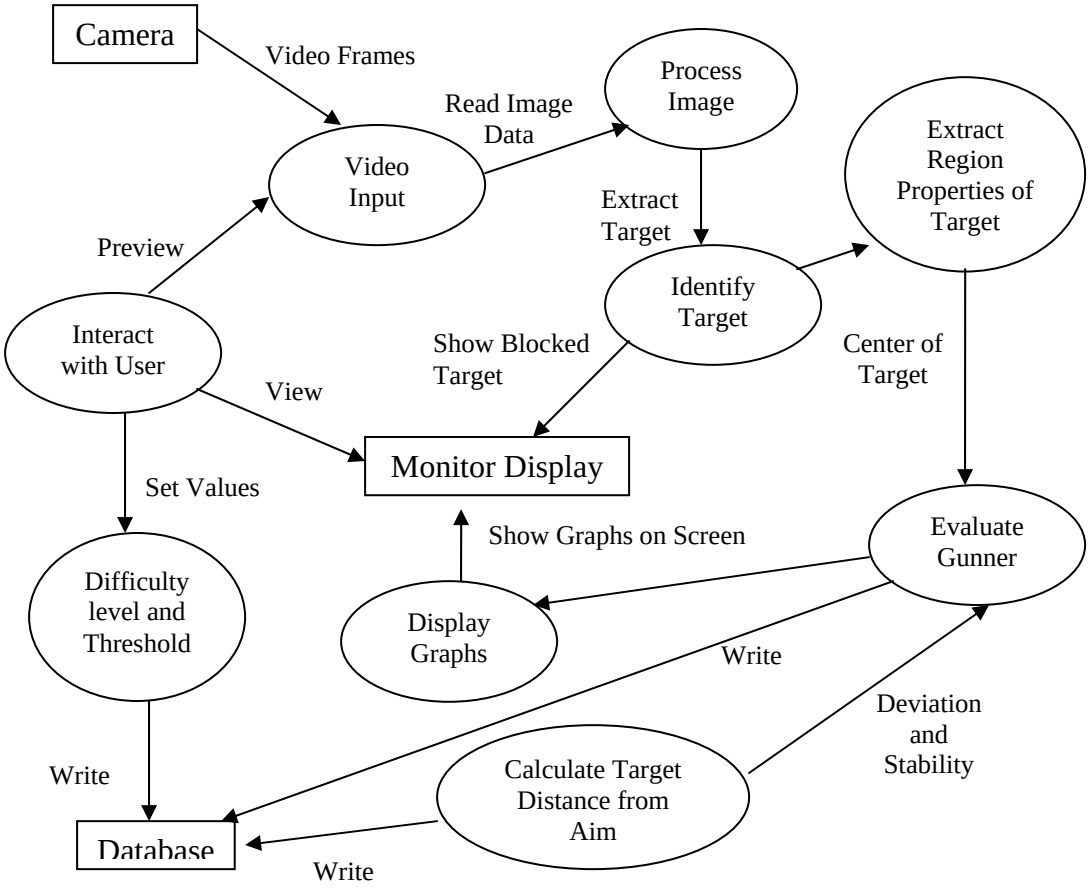


Figure 5.8 Data Flow Diagram

5.6 Component level Design

The Component level design transforms structural elements of the software architecture into a procedural description of software components. Class diagrams are used to show the most basic design of the software.

5.6.1 Class Diagrams

In a class diagram we represent some main classes and there relationships. A class diagram illustrates the specification for software classes and interfaces. Typical information it includes is

- Classes.
- Attributes
- Methods
- Associations

5.6.2 Software Class Diagrams

The following is the class diagram for the software

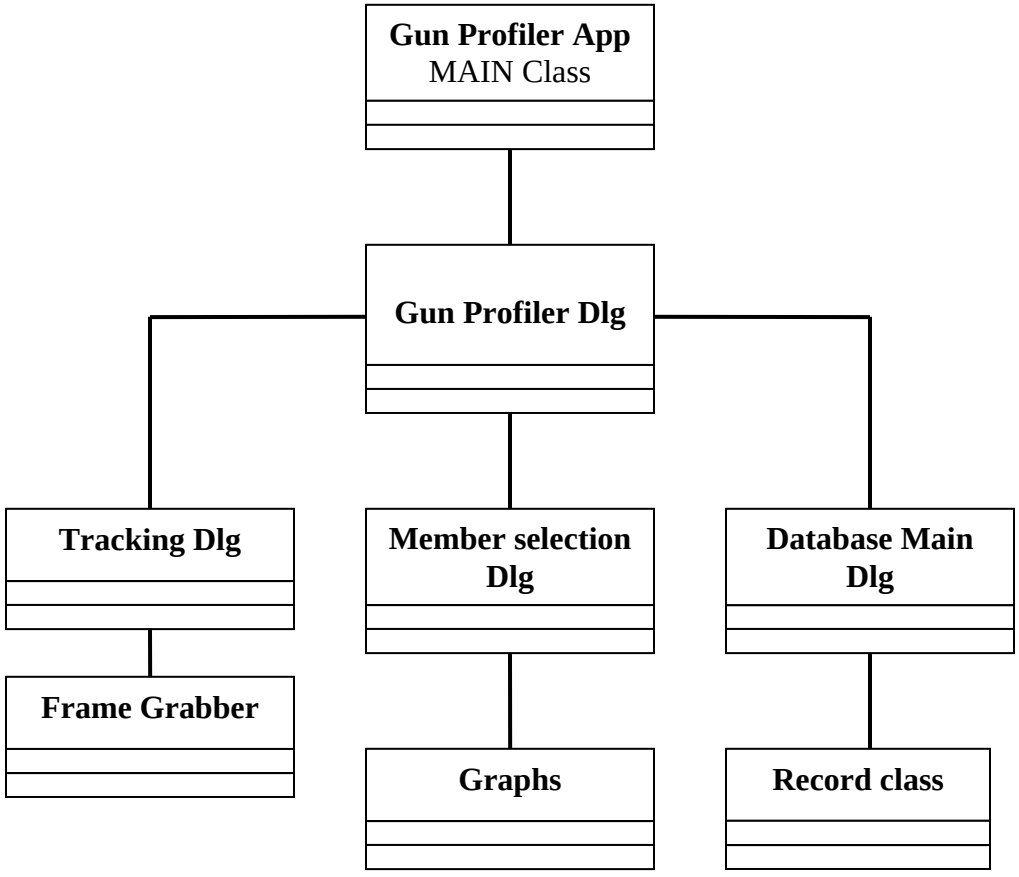


Figure 5.9 Software class diagram

5.6.3 Software Class Description

The software contains the following classes

- Gun Profiler Dlg
- Tracking Dlg
- Member selection Dlg
- Database Main Dlg
- Record class
- Graphs

5.6.4 Gun Profiler Dlg Class

This is the class that handles the main dialog of the application. Other dialogs such as the database menu and the training dialog follow from this class.

Members	Description
CGunProfilerDlg(CWnd* pParent)	Standard constructor
DoDataExchange	DDX/DDV support
OnInitDialog	Generated message map functions
OnPaint	If you add a minimize button to your dialog, you will need the code below to draw the icon. For MFC applications using the document/view model, this is automatically done for you by the framework.
OnQueryDragIcon	The system calls this function to obtain the cursor to display while the user drags the minimized window.

m_hIcon	Handle to icon
OnExit()	Exit the application
OnTraining()	Call to Tracking Dlg class
OnMember()	Call to Member Selection Dlg class
OnDataBase()	Call to Database Dlg class

5.6.5 Tracking Dlg Class

This class handles the training and tracking menu for the user. It previews the video input and handles the threshold along with other options such as the difficulty level. Once the tracking is performed the show graphs method is used to invoke the graphs class.

Members	**Description**
CTrackingDlg(CWnd* pParent)	Standard constructor
DoDataExchange	DDX/DDV support
OnInitDialog	Generated message map functions
DestroyWindow()	Call ClearGraphs() and exit
GetDefaultCapDevice(IBaseFilter ** ppCap)	Find first video capture device and bind it to filter graph.
InitStillGraph( )	Create a filter graph, add the capture device and sample grabber to the graph, then build the graph.
StopGraphs( )	To stop the filter graph.
RunGraphs( )	To start the filter graph.

ClearGraphs( )	Destroy capture graph and Remove filter graph from the running object table .
OnShowGraph()	Call StopGraphs( ) and initiate Graphs class.
OnRColor()	Display Real Video input
OnRThreshold()	Display Binary Image.
OnRAlgo()	Display Algorithmic Image.

5.6.6 Database Main Dlg

This class is used to provide access to the database for viewing and for update and add operations. The methods of this class are.

Members	**Description**
DataBaseShow(CWnd* pParent)	Standard constructor
DoDataExchange	DDX/DDV support
OnInitDialog	Generated message map functions
Connect()	Connect to the Database
GenerateError(HRESULT hr, PWSTR pwszDescription)	Format and Display the error message
GetRec()	return pointer to Record object
RefreshBoundData()	Take values from the database
OnAdd()	Add new record to database
OnUpdate()	Update existing record in database
OnDelete()	Delete existing record from database
OnFirst()	Move to the first record of database

OnLast()	Move to the last record of database
OnNext()	Move to the next record in database

5.6.7 Record class

This class is used to bind the application variables to the database. Each property of the records has a separate variable attached to it.

Members	Description
BEGIN_ADO_BINDING(CRecord)	Begins binding the variables of C Record to the database.
END_ADO_BINDING()	End the binding of variables
ADO_VARIABLE_LENGTH_ENTRY2()	Function Call to bind a variable given as function argument.

5.6.8 Graphs Class

This class is used to generate graphs of tracking path. It also shows the stability and other tracking information of the gunner.

Members	Description
Graph(CWnd* pParent = NULL);	Standard constructor
DoDataExchange	DDX/DDV support
OnInitDialog	Generated message map functions
AddOrUpdateChartData(int	Adds and Updates the graphs.

noOfRows)	
PopulateChartData(CString output)	Read point data from file and plot the graph accordingly
GetRec()	return pointer to Record object
UpdateBoundData()	Search for specific record then update the record in database
idValue(int value,int flag,int diff)	Copy the value of student ID
OnHorizantal()	Show the horizontal deviation graph.
OnVertical()	Show the vertical deviation graph
OnRadial()	Show the radial deviation graph
OnSize(UINT nType, int cx, int cy)	For resizing the window
OnStabiility()	Calculate the tracking efficiency of gunner

CHAPTER 6

IMPLEMENTATION

6.1 Introduction

Implementation is the next level of the design process. The Classes and code developed for the implementation of each system module will be discussed in this chapter. As mentioned in chapter 4 the Gun profiler system has six basic modules. The block diagram shown is shown in figure below.

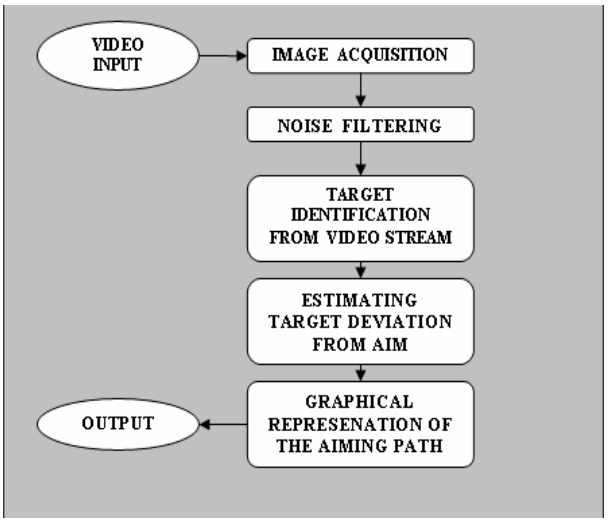


Figure 6.1 Block Diagram

6.2 Camera Calibration

Camera selection and calibration is of great importance as image grabbing and manipulation of the image grabbed are the main components of the software. The selection of a high resolution camera is a very sensitive issue. The software has to grab

the image, convert it into a binary image and then process the data available in this binary image to track the object. Better the resolution of the camera good will be the picture of the object and easier would be for the software to track the object; also since we are tracking fast moving aircrafts and very short time is left to track it The normal camera available in the market has a frame rate of 30 frames per second but for the project we require a camera with high resolution..

The main achievement is finding the distance of the aircraft from camera with the help of picture. Once the image is acquired and converted into binary image then the software counts the number of pixels covered by the target and compare it with a reference whose distance from the camera is known to us. If the pixels covered are less then the reference image pixels then it shows that the object is out of range and if the pixels of the object captured are more then the reference image then it means that the object is within range.

Calibration is one of the major process for setting the gun and camera at one point. The term calibration means that the camera and aim of the gunner are at the same point. This is done by placing a target with in the effective range of the gun and then making the gunner aim at it. Once the gunner aims at it then we place and adjust the camera in such a manner that the camera is also focused and aligned on the same point on which the gun is aiming. This complete process involves the hardware adjustment which includes the attachment of camera with the gun.

6.3 IMAGE ACQUISITION

Camera which is used to acquire images is taking thirty frames per second which is compatible with the software tool used. The software tool used for acquiring image data from a camera is the DirectX 9.0 SDK and the DirectShow API.

6.3.1 Drawbacks of Video for Windows

The simplest approach for acquiring Image data could have been the Video for Windows API. Although it provides a very convenient interface for linking with the source device however VFW falls short in some aspects.

- **No Direct Video Buffer Access :** The VFW API does not provide a direct access mechanism to its video buffer. Thus even if a captured frame is present in the video buffer it is not accessible. The only method by which the captured data can be accessed is to get the API to write the data onto the hard disk and then read it from there. This obviously involves Disk IO which is extremely slow for real-time applications.

- **Support for only Bitmap Image format :**VFW only supports saving of files in Bitmap format. Thus if you want to save the captured frame as a JPEG then you have to get it saved on the hard disk as a Bitmap image and then re-read it and convert it to JPEG format. This involves four Disk IO's which renders the application impossible for real-time processing.

- **Low Frame Rate :** Due to the amount of Disk IO involved VFW doesn't generate more than 5 FPS on most computers. This leaves the application unable to perform real-time computations.

6.3.2 DirectShow API

DirectShow is a component of DirectX that provides a set of low-level application programming interfaces (APIs) for creating games and other high-performance multimedia applications. It includes support for two-dimensional (2-D) and three-dimensional (3-D) graphics, sound effects and music, input devices, and networked applications such as multiplayer games. DirectShow is the architecture used for streaming media on the Microsoft Windows platform and it provides for high-quality capture and playback of multimedia streams.

6.4 NOISE FILTERING

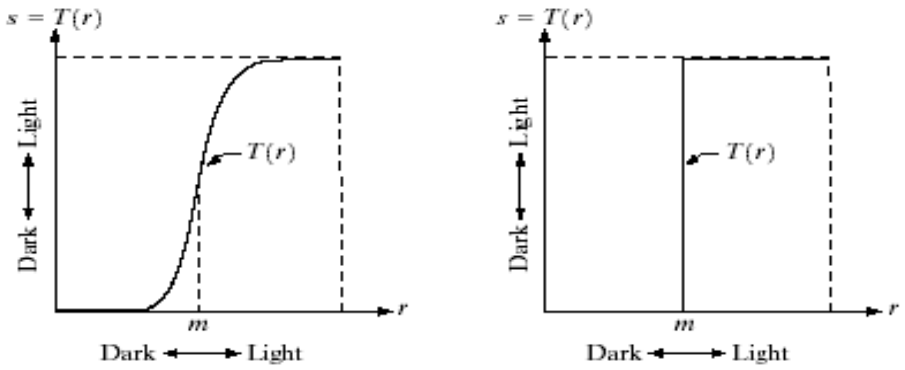
After acquiring video frames, they are converted into grayscale images. The grayscale images include all shades of gray from black to white and it is obtained by approximating the three primary color values red, green and blue of a pixel as shown in equation 1.

$$\text{Grayscale value} = \frac{\text{Red} + \text{Green} + \text{Blue}}{3} \qquad \text{(Equation 1)}$$

6.4.1 Image Enhancement Filters

The principal objective of enhancement is to process an image so that the result is more suitable than the original image for a specific application. Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term *spatial domain* refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image. *Frequency domain* processing techniques are based on modifying the Fourier transform of an image.

Gray-level transformation functions are among the simplest of all image-enhancement techniques. The values of pixels, before and after processing, will be denoted by *r* and *s*, respectively. As indicated in the previous section, these values are related by an expression of the form s=T(r), where T is a transformation that maps a pixel value *r* into a pixel value *s*. Since we are dealing with digital quantities, values of the transformation function typically are stored in a one-dimensional array and the mappings from *r* to *s* are implemented via table lookups. For an 8-bit environment, a lookup table containing the values of *T* will have 256 entries. See Figure 4.7



height levels before smoothing height levels after smoothing

Figure 6.2 Height levels after Thresholding

The effect of this transformation would be to produce an image of higher contrast than the original by darkening the levels below *m* and brightening the levels above *m* in the original image. A mapping of this form is called a *threshold* function. Some fairly simple, yet powerful, processing approaches can be formulated with gray-level transformations.

6.4.2 Thresh-holding

Thresh-holding is the basis for any image processing technique. It is the fundamental concept in image extraction and is used to track objects which differentiate a great deal from their environment. This technique though usually very simple and easy to implement fails in situations where the back ground is cluttered. Thresh-holding is basically a process in which we usually determine the average color over a set of pixels and compare it with already selected thresholds. Thresholding works by setting specific values to pixels that fall within a certain range called the threshold interval. All other pixel values in the image that do not fall in the threshold interval are set to 0. This helps to extract the object of desired intensity of color from the actual background. Thresh-holding always plays an important role in improving the efficiency of the system.

Thresholding is applied to the grayscale images to obtain binary images as it is fast and easy to work with binary images. A thresholding operation chooses one of the pixels as the foreground pixels that make up the object of interest and sets the rest as a

background. In present case *threshold below* operation is used which makes the value less then or equal to threshold value as foreground. The difference in a grayscale and thresholded image is shown in figure 2(i).

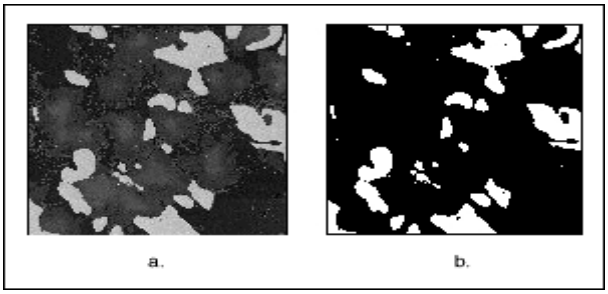


Figure 6.3 An Image Before and After Thresholding

Noise Filtering is required because the binary image contains the signal or the structure of interest along with unwanted variations which have to be suppressed. It is useful to remove these variations or very small regions from the current frame. One of these types of regions is called the salt n pepper noise that occurs as a natural result of creating a binary image via thresholding .Salt corresponds to pixels in the bright region that were below threshold. These are the classification error resulting from variation in the surface of material or illumination or noise in frame gabber. This is removed by checking the neighbors of every pixel if all the neighboring pixels values of a pixel under consideration are same, then that pixel is assign the same value as that of its neighbors. Other small components are remove by knowing their size and deleting them from reference frame. The result of noise filtering is a clear binary image as shown in figure 2(ii).

Figure 6.4 An Image Before and after noise removal

The following code has been used for the removal of noise

```
 void saltpepper_Removal()
 {
   int high=0,low=0;

   for(int i=1;i<239 ;i++)
    {
     for(int j=1;j<319;j++)
        {
          for(int k=i-1;k<=i+1 ;k++)
             {
              for(int m=j-1;m<=j+1;m++)
                   {
                    if(k!=i || m!=j)
                    {
                     if(pArray[m][k].r==255)
                          ++high;
                     if(pArray[m][k].r==0)
                          ++low;
                    }


                   }//end of third for
              }


      if(pArray[j][i].r==255)
       {

if(low>high){pArray[j][i].r=0;pArray[j][i].g=0;pArray[j][i].b=0;}
       } //end of if

      else
       {

if(high>low){pArray[j][i].r=255;pArray[j][i].g=255;pArray[j][i].b=255;}

       } //end of else

     low=0;
     high=0;

          }//end of second for

      }//end of first for

 }//end of function
```

6.5 TARGET IDENTIFICATION FROM VIDEO STREAM

6.5.1 Algorithm

Target is identified by labeling all the objects which are present in foreground and the object of interest is taken as the one which is relatively larger in size and close to the aim than other objects. For identification purpose classical union-find algorithm is used. The most effective characteristic of this algorithm is that it is very fast, and therefore it is appropriate in real time situation .The connected components labeling of a binary image generates a labeled image in which the value of each pixel is the label of its connected components. The algorithm runs in two passes through the binary image to give the labeled image of connected components. In first pass union–find data structure is populated with values, this structure is used to store a collection of disjoint sets to efficiently implement the operation of union (merging two set into one) and find (determining which set a particular element is in). This is accomplished with a vector array of PARENT whose subscripts are the set of possible labels and whose values are label of parent node. In second pass using the information in PARENT, *find* operation is performed to known the parent of each label as a result complete labeled image is acquired.

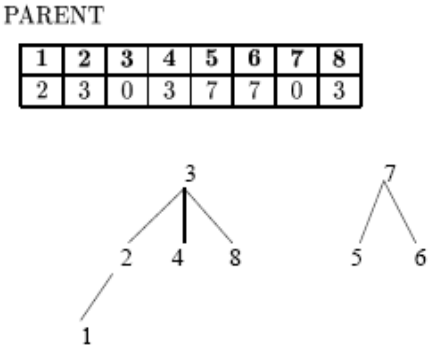


Figure 6.5 Union Find Data Structure for two set of labels**.**

6.5.2 Region Properties

Once a set of regions has been identified, the properties of the region become the input to higher-level procedures that perform other tasks such as recognition or inspection of the target object. These parameters are required in decision-making tasks to determine whether the target is hit or missed after a fire and to determine the deviation of the target from its aim to known tracking path form by the trainee. We have calculated the following properties of a region

- Center of Object
- Size
- Distance of object from aim

6.6 ESTIMATING TARGET DEVIATION FROM AIM

The pixel information of the target location is determined by calculating the centre of the target as describe in section 3.3.2 and centre of each frame is taken as the aim. The Euclidian Distance Measure is used to find the distance between two centers as shown in figure 4.

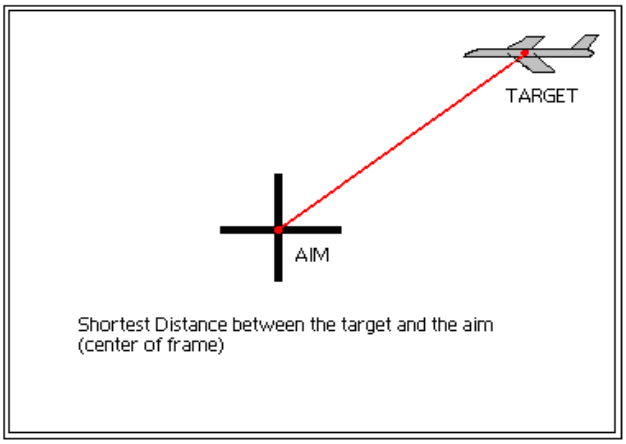


Figure 6.6 Pythagorean distance between Aim and Target.

Euclidian Distance Measure is the shortest distance between two points, and is basically the same as Pythagorus' equation when considered in 2 dimensions. Pythagorus' equation is given by Equation 2.

$$d_e = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

(Equation 2)

To calculate the distance of the target objects centeroid from the aim the above distance formula has been implemented as shown in the following code.

```
float distance_Formula(int x1,int y1,int x2,int y2)

{
 float x= float(x2-x1)* float(x2-x1);

 float y= float(y2-y1)* float(y2-y1);

 float d=x+y;
 return sqrtf(d);

}
```

6.6.1 TARGET HIT/MISSED INFORMATION

Target will be indicated hit only if trainee presses the trigger at the right time that is when the moving target object is exactly in front of the aim of the gun. The hit or missed information is calculated by considering all parameters of the real time shooting such as the inertia of gun, speed of bullet, speed of wind, parabolic motion of ammo and the firing range of the gun.

6.7 GRAPHICAL REPRESENATION OF THE AIMING PATH

The deviation of the target from the aim is characterized by three graphical representations, horizontal, vertical and the radial deviation graphs. The horizontal deviation graph shown in figure 4 denotes the right and left movement of the target from the center of frame (aim). The aim of the camera is taken as the mid point of the video frame that is obtained from it and as the aim of the camera is calibrated with the aim of the gun. Therefore both aims point to the same location. The deviation graphs are very

helpful in determining the aiming ability of a gunner so if a target is hit the deviation graphs can show clearly if it was tracked with concentration or if it was hit by chance.

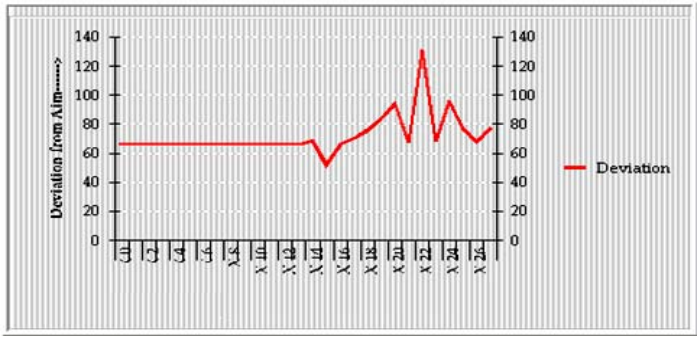


Figure 6.7 Tracking Graph

The horizontal deviation graph gives a clear idea of the swiftness and quickness in tracking of the target object.

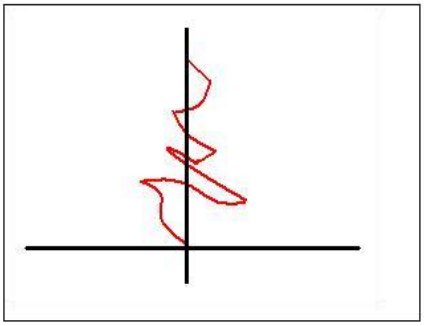


Figure 6.8 Horizontal Deviation Graph

In figure 5 the vertical deviation is shown as the movement of the target above and below the mean position (aim). The vertical deviation can be used to find the stability through the aiming.

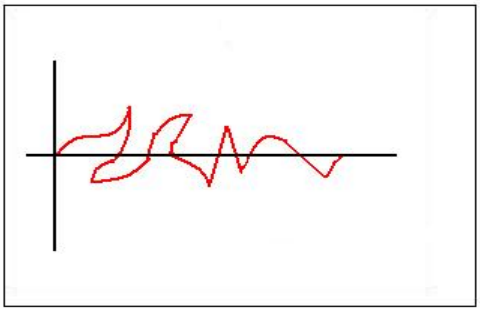


Figure 6.9 Vertical Deviation Graph

The radial deviation is shown in figure 6 as the complete trail of the target around the aim. The graph considers the moving target with respect to a stationary point that is the mean position.

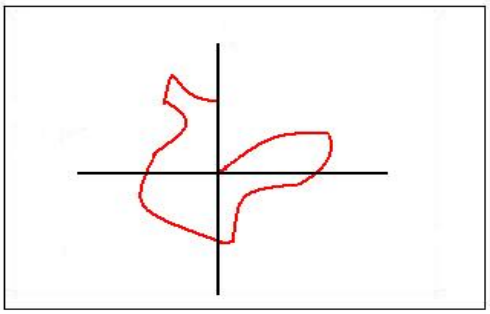


Figure 6.10 Radial Deviation Graph

CHAPTER 7

TESTING

7.1 Introduction

This chapter discusses the how testing is performed on this system. Different testing techniques are applied to find the reliability of the software. The results of applying different inputs to the gun profiler and the outputs are shown as screen shots. Different system Dialog windows are also provided.

7.2 Accuracy

The most important stage in the operation of the system is the recognition of the object of interest in the image and calculation of its region properties. If the object of interest is properly identified then the deviation estimation algorithm will have a solid input to work on. If the system fails to locate the correct object it will inherently fail to produce accurate results.

The factors obscuring the target include Noise in the image, Blurred focus of the camera, the Camera zoomed in too much or zoomed out too much, and non-uniform distribution

of light on the object. The camera is calibrated with the gun as shown in the figure below. A mount is created to hold the camera in line with the gun.

Figure 7.1 Camera Mount

For clear noiseless images generated in uniform brightness and light conditions for tests, the system was found to perform successfully approximately 87% of the time. With noisy images this success rate dropped with the increase in the original pixel to noise ratio. Field testing of the tracker has been performed by using an ordinary gun for tracking birds. The camera is calibrated with the gun aim by adjusting the camera mount until both the aims point to same location. In real time situations where the sky is clear and therefore the background does not have more than a single object of interest the accuracy of this system is nearly 100%. The figure shows a side view of the camera mounted on the testing gun.

Figure 7.2 Camera Calibration

The frame rates and target identification times depend on the Quality of the image, the Hardware being used such as the quality of the camera, Noise, and the Frame sizes.
The system on which these tests were performed is a standard Pentium4 (1.7GHz) with 256MB of RAM. For various test environments generated the system performed in the range of 20-24 Frames / second.

7.3 Limitations of the system

The idea of tracking motion on a computer using a video camera has been around for a couple of years and is still not perfect, because the construction of vision is far more complex than it would originally seem. The system has to operate in real time and therefore it should spend minimum time in processing each video frame however areas` of change need to be determined by comparing one frame of video with the previous frame, and seeing each pixel's value change, requires a lot of time for computation. In real time situations this time has to be minimized as much as possible.

The target is identified using thresholding and the most common problem with this technique is that it produces salt and pepper noise, the images taken under bright light or in the present case as images would be captured in sunlight it is quite possible that certain portions of the aircraft become a part of the background after thresholding. Although the noise has been removed by using filtering but the shape of the aircraft becomes distorted.

The system assumes that the target moves horizontally in front of the camera when the target is shot however if the aircraft was moving towards the camera or away from the camera at the time of trigger pressing then the result provided by our algorithm would not be exact. The size of the aircraft provides a measure of its distance from the gun and this assumption is violated when the object is not moving horizontally.

CHAPTER 8

FUTURE WORK AND CONCLUSION

8.1 Overview

The proposed system is a real time aircraft shooting system and the basic objective was to create a live aircraft tracking system that does not involve wastage of ammunition for

training of personnel. It can be used as a training tool to teach personnel, the necessary skills for shooting aircrafts and it can also evaluate the shooting ability of a person who is tracking and aiming at a high speed moving object such as an aircraft. Other objectives include the generation of a complete shooting profile of the trainer to help evaluate his stability and aiming capabilities. The human gunner performs realistic aiming at the target from his machine while the system generates a graphical representation of this aiming path followed by the anti aircraft machine. The system not only determines the tracking path followed by a human trainee as he aims for the aircraft and fires at it but also calculates the deviation of the tracked aircraft from its aim when the target is shot.

In real time scenario the aircraft is being tracked by a human gunner. A camera is mounted on his gun to record the movement of the aircraft with respect to the aim. The aim is considered at the center of the camera and is calibrated with the gun's aim. The video stream obtained from the camera is analyzed to detect and follow the moving aircraft through a sequence of images.

When the trigger is pressed to shoot the target, the position of the target as derived from the video stream are used to estimate the deviation of the aircraft from the actual aim of the machine. The hit or missed status of the target is also provided along with the path movement information of the target in reference to the aim.

The horizontal, vertical and radial deviation of the target from the aim is used to assess the aiming stability of the gunner. The tracking information of each trainee is stored in a database and the tracking efficiency of a gunner can be reviewed by accessing it.
The main objective of the project was to develop software that provides a better way to train personnel for aiming and firing at aircrafts. The objectives defined initially have been successfully achieved. This project provides the basis of DirectShow technology. A detail description of DirectX SDK is provided. In the project DirectShow which is component of DirectX is used. Many interfaces of DirectShow have been implemented to accomplish many tasks like enumerated devices and render video. Many shortcomings of the existing system have been removed by providing a better solution to save the

ammunition cost of training personnel. The proposed system also generates a complete shooting profile of the human gunner to help evaluate his shooting skills.

8.2 Future Work

Visual tracking is a complex field and there is always room for improvement in any application that using visual reconstruction. The future work that directly roots out from this project includes the extension of the system to by adding a module that identifies the complete boundary of the target object and does not eliminate and brighter portions of the object during thresholding. The limitation that a target should move only horizontally in front of the camera can be removed by considering the complete dimensions of the aircraft when estimating its relative distance from the gun, instead of taking the size as the only estimate to find the distance.

8.2.1 Distributed Tracking System

A distributed vision-based tracking system is envisaged. The system acquires and processes images through one or multiple Camera Units monitoring certain area(s) via a Local Area Network (LAN) and is capable of combining information from multiple Camera Units to obtain a consensus decision. The input from the camera should be distributed among the various sub units of the distributed system each of which carries out a specific task. Edge detection, segmentation & Template matching processes may be added and performed on nodes dedicated for this purpose with the central server synchronizing the activities and controlling the correction factors generated by the system. The server would act as the interface to the system successively receiving the inputs and generating the outputs. Nodes can be added to this system to boost its computational power and the complexity of the object extraction and prediction algorithms.

8.2.2 Accurate Shooting System

This project can be extended to implement an accurate ammo firing system. In such a system the camera input may determine whether the target is inline with the gun aim or not and depending upon this information the system may decide to either shoot the aircraft or not. Such a system may not allow any missed targets and nor would it fire any ammo that will not hit the target.

8.2.3 Night time operation

The current system has for its input true color cameras without night vision filters. Such a system has its operation limited to only daytime operations. Incorporation of night vision or thermal filters can increase the ability of the system to perform in night time operations as well.

8.3 Conclusion

We have presented a practical, widely applicable and extensible project for real time visual target detection, tracking and position estimation. This system can be used on any single camera and for any target tracking machine. The system can help train personnel for any artillery that requires efficient target tracking. Multimedia programming and strong hardware concepts were required for making such a cost effective and user friendly system. The project is very beneficial for all organizations who want to train their men for target tracking is a cost effective manner.

BIBLIOGRAPHY

[1] W.E.L. Grimson, L. Lee, R. Romano, and C. Stauffer. *Using adaptive tracking to classify and monitor activi-ties in a site*. In *CVPR9*8, pages 22–31, 1998.

[2] I. Haritaoglu, D. Harwood, and L.S. Davis. W4S: *A real-time system for detecting and tracking people in 2 1/2-d*. In *ECCV9*8, 1998.

[3] M. Isard and A. Blake. *A mixed-state condensation tracker with automatic model-switching.* In *ICCV9*8.

[4] S.S. Intille, J.W. Davis, and A.F. Bobick. *Real time closed world tracking. In CVPR97,* pages 697–703, 1997.

[5] R. Zabih and J. Woodfill. *Non-parametric local transforms for computing visual correspondence*. In *ECCV*, Stockholm, Sweden, May 1994.

[6] S. Blackman and R. Popoli, "*Design and Analysis of Modern Tracking Systems"*, Artech House, 1999.

[7] S. Blackman and R. Popoli, "*Design and Analysis of Modern Tracking Systems*", Artech House, 1999.

[8] K. Sethi and R. Jain, "*Finding trajectories of feature points in a monocular image sequence"*, *IEEE Trans. Pattern Analysis and Machine Intelligenc*e, Vol. 9., No. 1, pp.56-73, 1987.

[9] M. Irani, P. Anandan, and S. Hsu. *Mosaic based representation of video sequences and their applications*. In *ICCV*, pages 605–611, Cambridge, Massachusetts, June 1995. IEEE.

[11] R. Smith, M. Self, and P. Cheeseman. *Estimating Uncertain Spatial Relationships in Robotics.* In *Au-tonomous Robot Vehicle*s, I.J. Cox and G.T. Wilfon, Eds, New York: Springer Verlag, 1990, pages 167–193.

[12] P. Allen, B. Yoshimi, and A. Timcenko, "*Hand-eye coordination for robotics tracking and grasping*," in *Visual Servoing* (K. Hashimoto, ed.), pp. 33–70, World Scientific, 1994.

[13] D. Burschka and G. Hager. *Dynamic composition of tracking primitives for interactive vision-guided navigation*. In *Proc. of SPIE 2001, Mobile Robots XV*I, pages 114–125, November 2001.