Summer 2009

A state machine representation of pilot eye movements

Artistee Shayna Harris
University of Iowa

A STATE MACHINE REPRESENTATION OF PILOT EYE MOVEMENTS

by

Artistee Shayna Harris

A thesis submitted in partial fulfillment of the
requirements for the Master of Science degree
in Electrical and Computer Engineering
in the Graduate College of
The University of Iowa

July 2009

Thesis Supervisor: Associate Professor Tom Schnell

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

_____

MASTER'S THESIS

_____

This is to certify that the Master's thesis of

Artistee Shayna Harris

has been approved by the Examining Committee
for the thesis requirement for the Master of Science
degree in Electrical and Computer Engineering at
the July 2009 graduation.

Thesis Committee: _____
Tom Schnell, Thesis Supervisor

_____
Jon Kuhl

_____
Gary E. Christensen

To my Mother and Father, Artistee D. & James W. Harris and Brother, James W. Harris II. My family was always there to take my hand to encourage me and show me the way. Your guidance has helped me become the person I am. You will always have my love, respect, and gratitude

ACKNOWLEDGMENTS

Special thanks to the distinguished faculty members who served on my committee: Professor Tom “MACH” Schnell (chair), Professor Jon Kuhl and Professor Gary E. Christensen. Lastly, I am thankful to all colleagues and friends who made my stay at the University of Iowa, a memorable and valuable experience. The guidance and informational assistance of Mathew Cover is gratefully acknowledged.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

ACT-R	Adaptive Control of Thought-Rational
ADS-B	Automatic Dependent Surveillance-Broadcast
Air MIDAS	Air Man Machine Integrated Design and Analysis System
AOI	Area of Interest
A-SA	Attention-Situation Awareness
ASI	Airspeed Indicator
ATC	Air Traffic Control
D-OMAR	Distributed Operator Model Architecture
EFIS	Electronic Flight Instrument System
FAA	Federal Aviation Administration
FPM	Flight Per Minute
GPS	Global Positioning Satellite
HCI	Human Computer Interaction
HITL	Human-In-The Loop
HPM	Human Performance Model
IDE	Integrated Development Environment
NASA	National Aeronautics and Space Administration
NAS	National Air System
NextGen	Next Generation Air Transportation System
OPL	Operator Performance Laboratory

OTW	Out-The-Window
PFD	Primary Flight Display
SVS	Synthetic Vision System
UDP	User Datagram Protocol
UTC	Unified Theory of Cognition
VSI	Vertical Speed Indicator
XML	Extensible Markup Language

CHAPTER 1: INTRODUCTION

In virtual computer-simulated model worlds, we are not bound to physical conditions of reality.

-Dr. Michael Klein, 1999

1.1. Problem Statement

With the development of new interfaces, such as the Next Generation Air Transportation System (NextGen), as well as the advancement of the United States National Air System (NAS) from Air Traffic Control (ATC) to satellite-based systems in air traffic management (FAA, 2009), new evaluations for efficiency and safety are required. Advanced tools have been developed to test the new interfaces and technologies. These tools are used to assemble data for analysis to determine user performance within the NextGen system. NextGen will enhance the capacity and safety of technological innovation such as weather forecasting, data networking and digital communications to reduce gridlock in the sky and at airport terminals. The development of the aviation-specific system for Global Positioning Satellites (GPS) called the Automatic Dependent Surveillance-Broadcast (ADS-B) will shift certain decision making responsibilities from the ground to the cockpit (FAA, 2009).

Evaluation of new technology for commercial aircraft cockpits is a time-consuming and expensive process. The designs of modern aircraft cockpit technologies are concentrated in the area of component functionality and technological performance instead of pilot usage and operability (Foyle, Hooey and Byrne, 2004). The cockpits of many aircraft are composed of diverse automation, displays and controls. Therefore, creating tests that imitate the pilot's visual interaction with the displays and controls have become a difficult and complex process (Crowe and Narayanan, 2000). According to Foyle (2004), the majority of aircraft mishaps can be linked to errors in judgment. Procedures must be developed to provide training so that errors in pilot

judgment can be decreased and subsequently improve the safety of flight. Gaining clear knowledge as it relates to human error and the utilization of modern complex systems is crucial. Advanced tools are needed for simulating pilot performance in different operational environments.

1.2. Proposed Solution Approach

The purpose of this project is to design a state machine that can be used to simulate specific phases of a pilot's performance while flying an aircraft. The goal is to develop state-machine representations of straight-and-level flight, turns, climbs and descents within the Pilot Eye Flight Deck Application to simulate pilots' eye movement.

Scientists have spent decades utilizing eye movement data to evaluate various characteristics of human instrument-scanning behavior (Jones, 1985). One area of study is collecting information in-flight while pilots are monitoring and controlling an aircraft's attitude, location, and rate of movement in three-dimensional space (Fitts, Jones, and Milton, 1954). Substituting a simulation of various phases of empirical evaluation for pilot performance, such as executing a specific maneuver like an aircraft landing, will improve the design of interfaces and new systems. In fact, engineering disciplines rely heavily on mathematical or computational simulation models as a routine part of design (Byrne, Kirlik, and Foyle, 2004). These tasks require visual behaviors such as search, fixation, tracking, and grouping. The design and implementation of a virtual eye movement application provides gaze and action visualizations, which can supply detailed data in reference to the allocation of visual attention across various interface entities (Crowe and Narayanan, 2000).

The following chapters provide needed background and in depth information regarding the Pilot Eye Flight Deck Application. Chapter 2: Literature Review and Background, provides a brief discussion on the anatomy of the human eye and how vision is created as it relates to complex operations that occur during a pilot's

visualization of the indicators and displays in the cockpit. In addition, Chapter 2 provides a detailed discussion of pilot eye tracking research and tools. Chapter 3: Design and Development, provides a detailed discussion of the design, development and functionality of the Pilot Eye Flight Deck Application. Chapter 4: Analysis and Results, provides a detailed discussion of the four basic flight maneuvers that are performed within the Pilot Eye Flight Deck Application. Furthermore, it also provides a detailed analysis of data collected during a simulated climb maneuver. Chapter 5: Conclusion and Discussion, contains a brief discussion of the conclusion and recommendations for future research.

CHAPTER 2: LITERATURE REVIEW AND BACKGROUND

The eye of a human being is a microscope, which makes the world seem bigger than it really is.

- Kahlil Gibran

2.1. Anatomy of the Human Eye

The sense of sight allows us to learn more about the surrounding world (Montgomery, 2009). The human eye is a complex system consisting of individual subsystems such as the lens, iris and others. The eye can view a range of angles up to 200 degrees and has the ability to see 10 million colors (AAJ, 2009). The individual components of the eye function similar to a camera. For example, the same principles apply to the camera-lens as they do to the lens of the human eye. The following section provides a brief overview of the eye's anatomy and structure.

2.1.1. The Eye

The eye socket, or orbit, is a cone shaped cavity in the skull where the eye is positioned. Layers of soft fatty tissue inside the orbit provide eye protection and allow movement (Editure, 2008). A section of the human eye is presented in Figure 1.

The **cornea** is a curved, highly transparent tissue that allows light to pass through it without distortion. The lens of the eye focuses the images transmitted through the cornea to the retina. These images are then transferred through the optic nerve to the brain.

The **lens** is a firm gel-like transparent tissue nearly eight millimeters (one third inch) in diameter and biconvex in shape. The lens focuses images onto the retina which acts as the film that records the picture to be transmitted by the optic nerve to the brain.

The **iris** is located in front of the lens and gives the eye its color. The iris acts like the diaphragm of a camera by adjusting the amount of light that enters the eye through the hole in its center which is called the pupil.

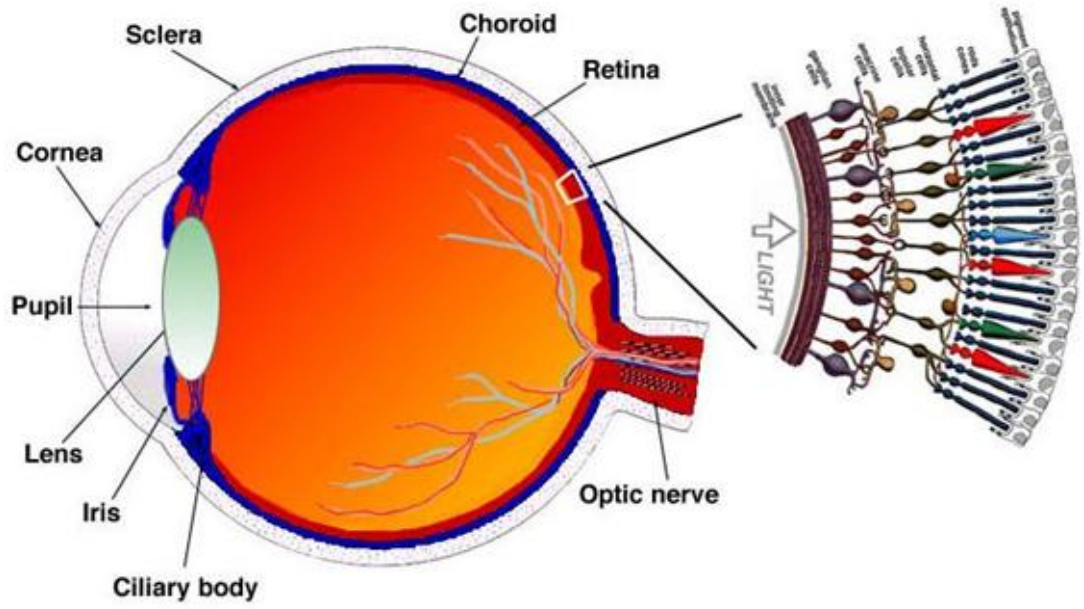


Figure 1: A drawing of a section through the human eye with a schematic enlargement of the retina.

Source: Encyclopedia Britannica, 1994.

The **retina** is tissue lining inside of the eye which is composed of special nerve cells that are sensitive to light.

The **optic nerve** is the pathway that light travels through the eye to the brain.

The **pupil** is an adjustable opening in the center of the iris through which light enters the eye.

Photoreceptors are the unique cell type that converts light energy into electrical signals. There are two types of photoreceptors (Editure, 2008), as shown in Figure 2.

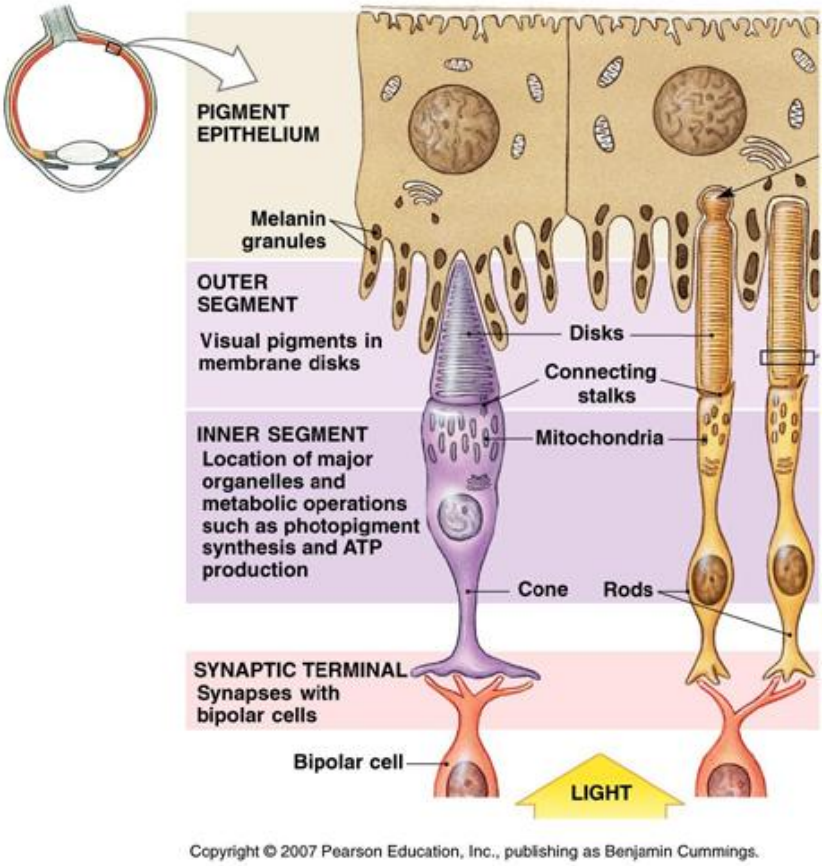


Figure 2: Structure and function of rods and cones.

Source: Pearson Education, 2007

The **cones** are light-sensitive nerve cells of the retina packed together in the fovea, which functions in bright light or daylight and are sensitive to color.

The **rods** are light-sensitive nerve cells of the retina, which lie outside the fovea in the peripheral parts of the retina and function entirely in dim light or night vision.

2.1.2. Vision

In order for sight or vision to occur, the following set of processes must be executed. Light must pass through the eye to the clear, watery fluid around the cornea to

the iris. The iris increases or decreases the amount of light entering by contracting or dilating (Nabatilan, 2007). As light exits the iris it continues to the pupil. To limit the amount of light, the size of the pupil is controlled by muscles in the iris. The lens is used to focus the light and provide clarity for near or distant objects by changing shape. The light reaches the sclera, which is the opaque protective outer layer of the eye.

When light enters the retina, the rod and cone cells starts chemical reactions, which convert the light into electrical signals that are sent through photoreceptors to the optic nerve (Nabatilan, 2007). The electrical signals are transported to the optic nerve to the primary visual cortex section of the brain. Vision is impossible, if the brain is unable to decode the electrical signals transmitted by the retina (Haines, 2005).

2.1.3. Visual Attention

Attention provides a way of limiting perceptual processes by focusing on specific visual regions and shifting the focus from one region to another. Furthermore, information is extracted from within a limited perceptual span to enable a decision for an action to achieve a goal. Selective processing of the information is used for visual tasks, such as searching and monitoring. Visual information is processed in pre-attentive and attentive stages. Pre-attentive is the stage that acknowledges the presence of five basic stimuli features: color, size, orientation, presence and direction of motion. Pre-attentive processing includes the units or objects toward which attention is subsequently directed. Attention is described as the system for controlling the way information is routed and for controlling processing priorities. To maintain a broad awareness of the environment, the brain must simultaneously take in a large amount of irrelevant information, and control selective attention (Hill, 1999).

2.2. Eye Tracking Research

The study of eye movements began approximately 100 years ago. Since then, many different approaches have been used to track eye movements. Usability engineering

is one of the earliest known studies of eye tracking applications (Jacob and Karn, 2003). Cameras were used by Fitts, Jones and Milton to track pilot eye movements during pilot visualization of cockpit controls while performing aircraft landings. The contributions of Fitts, Jones and Milton (1954) to capture eye movements provided valuable techniques of eye tracking that are still used today. Eye tracking applications can show errors in judgment or decisions about usability issues on interfaces, similar to operation of a vehicle in use prior to the development of computer interfaces.

The Russian psychologist Alfred Yarbus (1967) observed that subjects were utilizing rapid eye movements when viewing complex images. Yarbus discovered the more complex an object is the longer the eyes will gaze at it. The visual gaze on a single object in an image changes according to what an observer wants to see (Yarbus, 1967). Furthermore, information given to the observer is determined by thoughts and the analysis of information in the external environment.

Eye tracking in human-computer interaction has exhibited moderate growth, both as a way of studying the usability of computer interfaces and interacting with the computer (Jacob and Karn, 2003). Due to the study of *Human Computer Interaction* (HCI), psychology and neuroscience, as well as advertising and design, eye tracking research has become a useful tool in understanding human behavioral eye patterns. Eye tracking research relates to cognitive processes such as language comprehension, memory, mental imagery and decision making (Jacob and Karn, 2003). Since eye movements are tracked by modern technology with great speed and precision, they are used to provide practical applications in human-computer interactions (Jacob and Karn, 2003).

2.2.1. Classification of Eye Movement and Eye Tracking Methods

The primary focus on the accuracy and precision of eye movement measurement is based on fixations and saccades. Fixations are low velocity eye movements which correspond to the eye staring at a point of interest and making very small, randomly drifting, eye movements, in order to keep the target centered (Nabatilan, 2007). Saccades are eye movements that jump from point to point in the stimulus or image. There are also other eye movement measurements, which are briefly described in Table 1.

Table 1: Eye Movement Measurement

Metric	Definition	Quantitative Measure	Function	Influences
Fixation	Period of time after the eye acquires a new target and ceases movemen	Average: 200-300ms Range: < 100 ms - > 1s, movement < 1 visual degree	Time that visual information is exacted and relayed to the brain	Target Size, Cluster with information dense areas, Shifts in Gazing behavior
Saccade	Movement that occurs as the eye moves to the next area of fixation. Spatial change in fixations within a glance/dwell	275-900 deg/s rotational velocity, Typically < 1 deg	Links fixations together. Gathers additional detail of the same target. Short distance movements, such as looking at various parts of an attitude indicator	
Transition	A movement that is used for new target acquisition. A much longer saccade to a new location	Noticeable angular movement in eye vector angle. Typically > 1 deg	Used to move from one Area Of Interest (AOI) to the next	Peripheral object saliencies
Glance / Dwell	An area where fixations cluster together. A glance occurs when saccades and fixations are in a specific area or AOI	None defined	When quantitatively defined, identifies the difference between saccades and translations. Also useful in AOI analysis	Saccade and translation quantitative definition, fixations
Scan Path	Links all fixations, saccades, transitions and glances together	Trend analysis, standard deviation	Allows for link analysis; combines saccade, transition, and fixation metrics into one metric	Situational dependent, not necessarily driven by workload except for the same scenario
Gaze	Fixations within a specific AOI	Time, frequency	Shows subjects interest or disinterest in a specific AOI (i.e. Altimeter tape)	Subjectively defined areas, scenario dependent

Source: With permission from OPL

2.3. Research of Eye Movement on Piloting an Aircraft

Time and accuracy is measured based on how fast a person completes a task and how well the task is performed, eye movements provide insight into the visual, cognitive, and attention aspects of human performance.

Eye tracking can play an important analytical and experimental role in present and future applications of aviation (Duchowski, 2003). Eye tracking technology can reveal what source of information the subject is using to affect their decisions. Recently, eye tracking technology was used for sophisticated flight displays such as the *Synthetic Vision System* (SVS) that provides a clear sky view under all operating conditions (NASA, 2009). Researchers Foyle, Byrne, Hooey, and Kirlik (2008) used analysis of eye movements to illustrate the importance of the SVS as the primary source of information during poor visibility conditions that can substantially affect flight safety. These studies show the potential of eye movements for judgment of pilot performance, and also provide better information than *Out-The-Window* (OTW) viewing. Eye movements of professional pilots were recorded under realistic flight conditions in an analysis of human-machine interaction behavior for situation and mode awareness in a modern glass cockpit (Duchowski, 2003).

In the training environment, eye tracking can be used to provide feedback on the user's eye scanning pattern and as a teaching tool for improving training by showing effective eye movement behavior (Wetzel and Poprik, 1996). This study can be used to illustrate visual behavior differences between experienced pilots and novice pilots. The amount of time that a pilot allocates to the specific cockpit instrument may be determined with the use of eye tracking technology.

2.3.1. Workload Measurement

Modern aircraft models, such as Boeing's 787 Dreamliner, are now equipped with a large number of computerized systems. Automation is implemented in order to reduce

operational costs and workload, to provide precise flight paths, and to improve safety. On the other hand, increasing automation in the cockpit requires extra demand on pilots' limited capacity to process the multitudes of displays found in modern glass cockpits (Hancock and Desmond, 2000). In fact, several human performance problems have emerged which are related to the user interaction with automation technology. These problems are: (a) a reduction in the pilot system awareness, (b) an increase in pilot workload, and (c) degradation in manual flying abilities (Hancock and Desmond, 2000).

There are a number of laboratory and field studies to investigate the effects of automation on workload. These studies evaluate the potential significant changes in cockpit procedures and instrumentation. Workload measurement studies will improve the allocation of attention to more appropriate areas of responsibility as pilots aviate, communicate, navigate, and monitor (Veltman and Gaillard, 1993).

One aspect of workload consists of the effort required in executing the task. For example, a beginner and expert identical workload task will execute at different levels. When some people are given an increased workload task demand, they will not increase the level of effort (Farmer, Brownson, and QinetiQ, 2003). Most workload studies are concerned with a task level of performance, even though the level of performance alone does not provide an adequate measure of workload. There are other types of workload measurements that have been published, but the most common are performance, subjective, and physiological (Farmer, Brownson, and QinetiQ, 2003). The workload measurements by Farmer are described further in Table 2 and Figure 3.

2.3.2. Visual Demand and Allocation of Resources

The pilot has a demanding and complex visual system. Many cockpit display instruments provide purely visual information that require pilot actions. This information may consist of navigation, status check, or the monitoring of the displays for warning and caution messages. For example, the pilot must demonstrate good hand-eye coordination

to operate the aircraft flight controls and displays. Pilot vision plays an important role in determining a pilot's capabilities within a demanding and complex visual system within the flight deck (Jarrett, 2005).

2.4. Human-In-The-Loop (HITL) and Human Performance Model (HPM)

With the development of new avionics and glass cockpit displays, as well as various other technologies developed by the aviation industry, *Human-In-The-Loop* (HITL) studies are needed to enhance the safety and efficiency of commercial aviation. The documentation of equipment and procedures which do not fully support the operational needs of pilots provide for error reduction and improvements in flight safety (Leiden, Keller and French, 2001). Therefore, to understand human error with the deployment of complex systems, tools, methods, and evaluation techniques are needed for predicting pilot performance in real-world operational environments (Foyle and Hooey, 2008). Off-nominal and nominal are two evaluation techniques that can be used with the HITL. An off-nominal scenario highlights the unexpected conditions which may occur. In contrast, a nominal scenario follows normal, expected conditions (Leveson, 2001a, 2001b). The off-nominal unexpected conditions can range from minor deviations to the catastrophic failure of the system. Leveson (2001) advocates using off-nominal scenarios because we can determine system failures, usage, and procedure issues to eliminate errors early in the design of the system.

System models provide flexibility that cannot be achieved through observations and full systems. Scenarios derived from HITL data could be substituted in placed of models (Corker, 2002). The *Human Performance Model* (HPM) is a model that measures the performance of a human interfacing with avionics systems during numerous flight scenarios. HPMs are computer-based simulations that have human characteristics embedded within the computer software to represent the interaction between the human

and the computer (Gore and Corker, 2002). In addition, HPMs are flexible and economical because real-time simulations provide a very large number of observations from within a limited set of events. This can eliminate errors early in the design phases without creating expensive prototype hardware. Further, HPMs are supported by empirical data derived from HITL simulations and field studies. The model can be manipulated to explore procedures and operating modes that otherwise would never happen in real time environments (Corker, 2002). Finally, the HITL data is sometimes used to populate and develop models or to validate a model's output; however, to produce valid and informative results, we must have an understanding of the task and the environment.

As described above, HITL and HPMs prove to be useful tools to better understand the potential for human error in complex aviation systems and displays. These advanced tools and methods are needed for predicting pilot performance in operational environments (Foyle and Hooey, 2008). Therefore, HITL testing and HPM techniques are powerful when used together in the development of an error free system. During the designing phases of an aviation system these techniques will produce systems that are safer and more efficiently operational (Foyle and Hooey, 2008).

Piloting an aircraft is a highly complex task that involves the execution of multiple critical subtasks. The pilot's key responsibility in a cockpit consisting of electronic displays is to monitor the display imagery and to react to warning and advisories. The visual scan pattern is important when a pilot is monitoring various instruments in the glass cockpit (Verma and Corker, 2004). Pilots use eye movement techniques – for example, the T-Scan pattern which consists of an electromechanical airspeed indicator, altitude indicator, and vertical speed indicator (Hewett, 2006**) -** to seek information on the display screen.

A variety of models have been developed to simulate operating aircraft and provide information on how pilots perform complex tasks. Many simulators are

conceptual models that only provide representational and procedural elements of an operating aircraft. More complicated models have been developed that simulate and predict different characteristics of pilot behavior. Cognitive models have been successful in capturing lower level performance and higher level decision making in complex tasks (Salvucci, 2001, 2002, 2005, 2006). In particular, no single modeling architecture can fully address the range of interacting pilots' actions in the cockpit. As result, several HPM tools were developed: (1) *Attention-Situation Awareness* (A-SA), (Wickens and McCarley, 2001) (2) *Air Man Machine Integrated Design and Analysis System* (Air-MIDAS), (Corker, et al., 1985) (3) *Distributed Operator Model Architecture* (D-OMAR), (Deutschand Pew, 1998) (4) *Adaptive Control of Thought-Rational* (ACT-R), (Anderson, 1998) (5) *Soar* (Laird, Newell and Rosenbloom, 1983). Newell proposed to unify the theories of cognition, such as decision making, memory, perception and others. He argued for a unification of all separate theories in cognitive psychology. Therefore, a theory of cognition is a *Unified Theory of Cognition* (UTC), which is a set of cognitive mechanisms that covers the whole field of cognition. The most advanced computational theories of cognition are ACT-R and Soar (Laird, Newell and Rosenbloom, 1983). However, predicting human performance by computer modeling, although valuable, is a difficult process.

1. Performance: This type of workload can be divided into primary-task and secondary-task measures. The primary task is the task whose workload is being investigated, whereas a secondary task is used to determine the operator's spare capacity.
2. Subjective: This type includes both uni-dimensional and multi-dimensional scales.
3. Physiological: This type of workload determines the person's level of awareness.

Figure 3: Level of performance workload define

Source: Farmer, Brownson, and QinetiQ, 2003

Table 2: Workload measurements

Workload Measure	Advantages	Disadvantages	Disadvantages
Performance	• Primary-task measures provide a direct indication of performance on the task of interest • Secondary-task performance provides a useful index of spare capacity	• Performance on the primary task may be insensitive to workload change if operators compensate by increased effort • Some secondary tasks may be insensitive to the demands of the primary task	• Reaction time • Accuracy • Interval production • Time estimation • Random number generation • Probe reaction time • Memory
Subjective	• High 'face validity' and hence acceptance by operators • Most operators find it fairly easy to assign ratings	• Operators can rate changing demands of a given task, but find it difficult to compare workload on qualitatively different types of task • No unanimous agreement on the nature of the components of workload, and hence the set of scales that should be used	• Modified Cooper- • Harper scale • Instantaneou s Self Assessment • NASA Task Load Index
Physiological	• Can often be recorded continuously, with little intrusion on work activities • Are not affected by biases that sometimes contaminate subjective measures	• Often a large volume of data is collected, requiring sophisticated analysis • There may not be a simple relationship between physiological change and performance	• Event-Related Potentials (ERPs) • Heart rate • Heart rate variability (HRV) • Pupillary response • Eye Blink • Cortisol • DC Shift

Source: Farmer & Brownson, Qineti, 2003

CHAPTER 3: DESIGN AND DEVELOPMENT

The purpose of models is not to fit the data but to sharpen the questions.

- Samuel Karlin, 1983


```
<scanpatterns>
        <pattern name="Climb &amp;amp; Descent">
                <state name="VSI">(&gt;&#61; VSI,500)</state>
                <state name="ATTITUDE">(and,(&gt;, VSI,490),(&lt;, VSI,510))</state>
                <state name="HEADING">(and,(&gt;, heading,328),(&lt;, heading,332))</state>
                <state name="ATTITUDE">(and,(&gt;, bank, -2), (&lt;, bank,2))</state>
                <state name="AIRSPEED">(and,(&gt;,(-,airspeed,desired,5)),(&lt;,(+airspeed,desired,5)))</state>
                <state name="REPEAT">(&lt; altitude,6000)</state>
        </pattern >
</scanpatterns>
```


3.1. System Design - How does it work?

During flight, a pilot continually monitors the flight system status based on the instruments from the aircraft instruments. If a certain demand occurs, the pilot will focus on that specific instrument for information. The Pilot Eye Flight Deck Application simulates the pilots' allocation of attention in the cockpit. Visual attention is when the pilot has to concentrate on several displays at once, while processing information from the outside environment. The capability to simulate attention behavior of pilots is critical for any supervisory control issues, such as interacting with displays and automation from information-rich domains. The Pilot Eye Flight Deck Application is a virtual eye in 2D/3D space that scans the cockpit instruments and simulates eye scanning patterns across various channels of information in a cockpit environment. The capability to simulate eye movement is an important tool that will improve cognitive performance, situation awareness and resilience in NextGen technology.

3.1.1. Program Language

The Pilot Eye Flight Deck Application is written in C#. C# is a computer language developed by Microsoft. It is an object-oriented programming language which semantically and syntactically resembles the C++ language. The advantages of C# over C++ language are: pointer declarations are generally not allowed without specifying an un-safe namespace; it provides automatic garbage collection for all defined-types including integers and floating-point; all objects are distributable across systems through the Component Object Model (COM); and its collection of frameworks, libraries and APIs are extensive.

3.1.2. Compiler

Code for the Pilot Eye Flight Deck Application was compiled and debugged using Microsoft Visual Studio 2008, an *Integrated Development Environment* (IDE) that supports C# and XML. Visual Studio includes a code editor and debugger.

3.2. System Architecture

Figure 4 shows how display data, instrument states, and warning messages interface with the Pilot Eye Flight Deck Application. The external data is streamed into the state machine script engine. The script engine will take the input values and process them. Then, it will determine the next state transition. These transitions will determine the eye movement in the perception block. The output gives instrument names and locations.

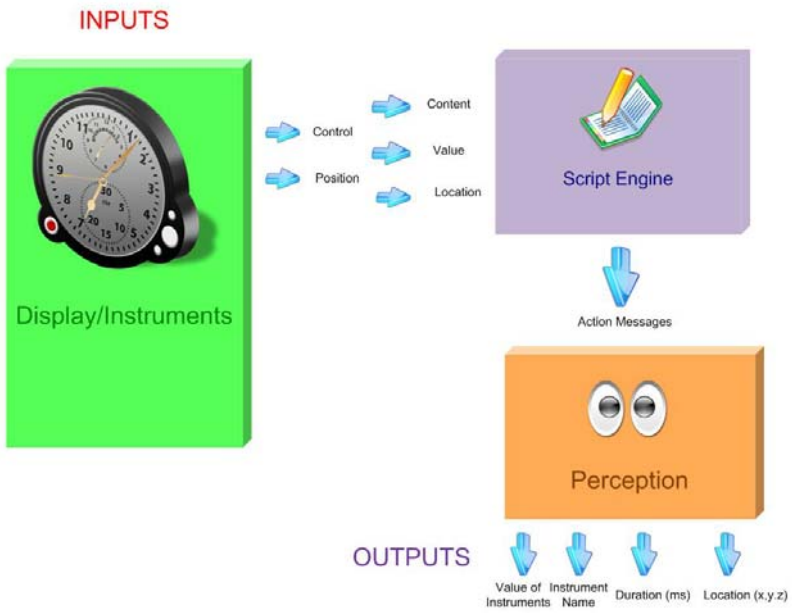


Figure 4: Overview of the System Architecture

As shown in Figure 4, there are three major blocks: Display/Instruments, Script Engine and Perception. The display/instrument block can simulate, generate and read data from external sources. The data from the display/instrument block is sent to the script engine. The arrows in Figure 4 pointing towards the script engine block are examples of data which the script engine can process and interpret. For example, the script engine may process: control, position, content, value and location.

After all data from the display/instrument block is processed by the script engine, the script engine changes the states of the state machine. Different states send different action messages to the perception block. These action messages determine what the eye will look at. An operator can monitor the perception block by observing instrument values, instrument names, durations and instrument locations.

3.3. Program Development

To maintain a straight and level flight, the pilot must maintain the aircraft on a constant heading and at a constant altitude. One of the instruments to determine nose and wing position is the attitude indicator which provides information in relation to the horizon and helps the pilot maintain a constant altitude. However, the attitude indicator may suffer from occasional errors, so it is very important that the pilot scan other instruments to verify the attitude values. The pilot may utilize the heading indicator to verify the aircraft state of constant heading. Any deviation from a constant heading on the heading indicator will notify the pilot that the wings are not at level altitude. The pilot will use the *Vertical Speed Indicator* (VSI) in conjunction with the attitude indicator as a crosscheck. Any change in constant altitude flight will be indicated by a movement on the altimeter and vertical speed indicator. The pilot will perform additional scans of the instruments such as VSI tape and turn rate indicator (FAA and CNATRA, 2009).

3.3.1. Panel Scan Script Pattern Engine

The steps and procedures performed (Section 3.3) by pilots to maintain a straight and level flight are implemented into the state machine. Each node in the state machine reflects an instrument. Each edge that connects two nodes is the transition from one state to another. The condition determines when a state can transition to its next state. Then the pilot will crosscheck with the heading indicator for constant heading. The initial state is the attitude indicator position of the aircraft. The condition checks if the aircraft is above 5,000 feet. Once the altitude reaches 5,000 feet, the pilot will scan the heading indicator (CNATRA, 2009). Therefore, the current state changes from the attitude indicator position state to the heading indicator state. By representing the instruments as nodes and edges, we can simulate a pilot performing a straight and level flight.

To perform condition checks, the Pilot Eye Flight Deck Application provides a scripting language that uses logical semantics to describe state transitions. Each state has an entry condition, so the script engine works by executing this entry script. If the result of the script is non-zero, the state machine changes state. However, if the script returns zero, the script engine will continue to monitor the entry script. Figure 5 depicts an aircraft above 5,000 feet.

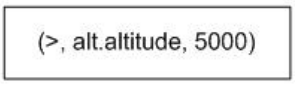


Figure 5: Representation of altitude greater than 5,000 feet.

Each script element begins with a parenthesis followed by the operator. Next, “alt.altitude” returns the current altitude information from the altitude indicator, and compares it with 5,000 feet. If altitude is greater than 5,000, the script element returns a

non-zero result of 1. However, if the alt.altitude parameter is less than 5,000, the script returns zero.

The Pilot Eye Flight Deck Application can also support more complex descriptions such as (and, (>, alt.altitude, 5000), (==, alt.status, enum.functional)). The application also supports nested arguments. The example provides an enumeration called *functional*. The script compares the alt.status with the enumeration. If this condition is true, the element returns true. The script element is nested inside an "and" element. For the "and" element to return true, the conditions (>, alt.altitude, 5000) and (==, alt.status, enum.functional) must be true.

The scripting mechanism formulates the script into a structure tree. Each leaf of the tree calls the evaluate function recursively, and the result is stored onto the stack as shown in Figure 6. This form of processing is also implemented in Scheme which is a functional language. However, the script engine can process expressions similar to imperative languages such as C#. The functional language provided by the script engine is not based on Lambda calculus; however, some aspects of the syntax resemble it. If we write the equivalent statement in C#, it will look like the statement as shown in Figure 7.

Using functional style syntax language has advantages compared to imperative syntax. First, this functional style syntax is much easier to parse and use with the XML description. When parsing an expression, imperative syntax requires a more sophisticated algorithm for pre-processing. First, the algorithm must remove extra parenthesis, invoke operators onto the right expression to evaluate in order from left to right. Second, multiplication and division are evaluated before addition and subtraction. Lastly, the algorithm must check for mismatching logical, conditional and mathematical operations.

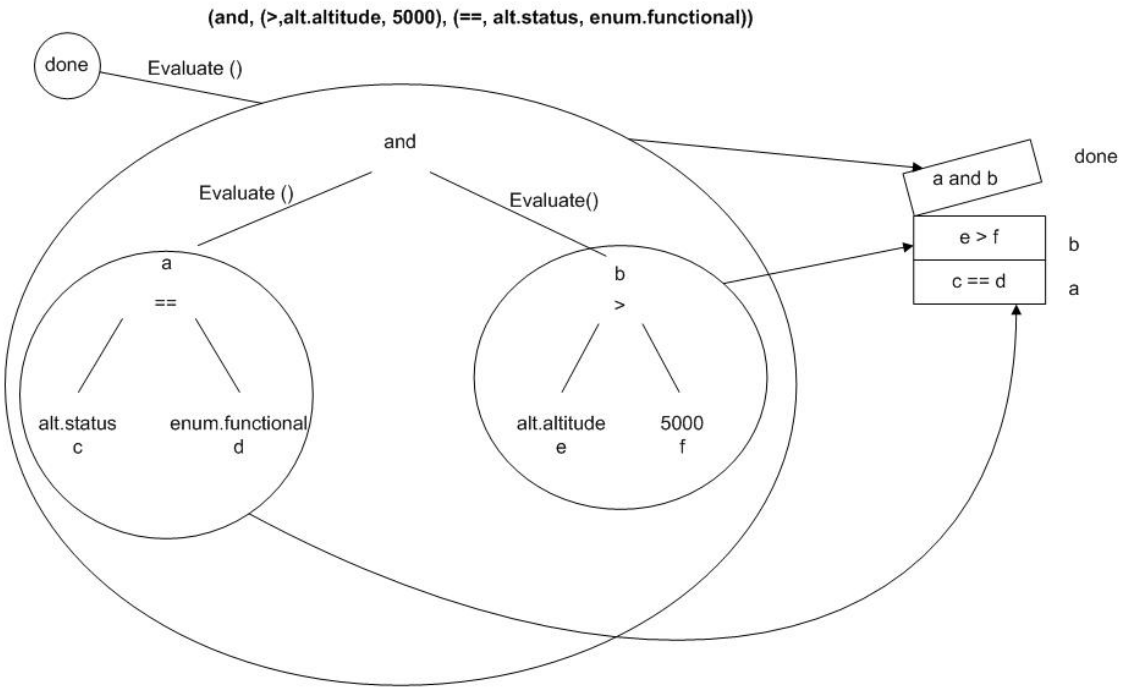


Figure 6: Structure Tree of the nested expression.

```
If (alt.altitude > 5000 && alt.status == enum.functional)
                {
                    //action
                }
```

Figure 7: Sample of C# code.

For the functional script used in the application, these steps are ignored because expressions are executed in a recursive manner and results are stored onto a stack. When expressing using the functional syntax, ordering is described explicitly.

3.3.2. XML

XML (Extensible Markup Language) is a generic syntax used to mark up data with simple, readable tags. A unique program can be written to manipulate the data in documents. XML allows access to a wide range of free libraries in a variety of program languages that can read and write XML to give the software flexibility and customization. Figure 9 describes the state machine for a straight and level task. Each instrument has entry conditions.

The state machine has the starting XML tag called “scanpatterns”. The “patterns” tag can contain multiple pattern scans or state machines. Each pattern represents one state machine. Each “state” tag represents a single state in the state machine. The “name” attribute is the name of the state, and also the name of the instrument the Pilot Eye Flight Deck Application sends to the eye.

Figure 8 shows that the Straight & Level state machine displayed as several states. Since the name of the state is the instrument name, multiple states can have the same name, however; different entry conditions. This state machine demonstrates three different instruments, and each state has an entry condition. If the entry condition evaluates to a non-zero value, the state machine changes to that state. For the “Attitude Indicator” state, the state variables are wings, level.wings, pitch, and level.pitch. The “==” is the operators that computes this expression. Table 4 depicts the XML description of an instrument panel scan pattern for a straight and level task as shown in Figure 8.

The entry condition is used to determine the next state. The scripting language supports all the commands listed in Table 3. The /, > and < symbols are reserved in XML as start tags, end tags, entity references, processing instructions. The “greater than” is

represented as > and “less than” as &lt. If the expression is already reserved in XML there is another representation of the symbol.

Table 3: Script Engine Operations

Expression	Operator Expression	Definition of Expression
(>, ,)	>	Greater than
(<, ,)	<	Less than
(>=, ,)	>=	Greater than or equal
(<=, ,)	<&#1;	Less than or equal
(==, ,)	=&61;	Equality (a==b) true or false
(=, ,)	=	Assignment (a=2)
(/, , ,)	&#frasl;	Division
(*, ,)		Multiplication
(+, ,)		Addition
(-, ,)		Subtraction
(and, ,)		Logical AND
(or, ,)		Logical OR
(not, ,)		Logical NOT

Figure 8 shows how the script operators are used inside the XML file. The script engine supports 13 expressions: Greater than (>), Less than (<), Greater than or equal to (>=), Less than or equal to (<=), Equality (==), Assignment (=), Division (/), Multiplication (*), Addition (+), Subtraction (-), Logical AND (and), Logical OR (or), and Logical NOT (not).

```
<scanpatterns>
<pattern name="Straight & Level">
< state name="Attitude Indicator">(and, (= =, wings,level.wings), (= =, pitch, level.pitch) )</state>
< state name="Altimeter">(&lt;,(-,desired_altitude,20), altitude )</ state >
< state name="Heading Indicator">( = =, heading indicator, heading)</ state >
. . .
. . .
</pattern>
</scanpatterns>
```

Figure 8: Example of script commands listed in Table 4.

Seven operators are reserved by XML, the corresponding operator expressions are shown in Table 3. Table 4 is a scenario for a pilot to maintain straight and level flight at 5,000 feet on a 270° heading. The scripting language, described above, will interpret the following commands listed in Table 4 as shown Figure 8. When the wings are level and pitch is level, the state changes to Altimeter. This description can be shown in the XML as (and, (==, wings, level.wings), (==, pitch, level.pitch)). Normally, equality is not the best operator, but for simplicity it's used in this example. For provide a condition when the desired altitude is below 20 feet, one possible expression is (<, (-, desired_altitude, 20), altitude). Twenty is subtracted from the desired altitude and compared with the value of altitude. For simplicity, the" less than" operator is not replaced with "<" operator. These are examples of how to describe the readings in the chart.

Every state corresponds to an instrument on the displays. The Interrupt state prevents states from locking, by monitoring its duration. The Attitude Indicator is used to check if the wings and pitch are leveled. Once the wings and pitch are leveled, the application will have transition to the Altimeter. The application waits until the altitude is 20 feet below desired altitude. The interrupt will allow the application to observe other

instruments if the uncertainty threshold is reached. The uncertainty threshold represents the pilot's uncertainty about that instrument reading. The uncertainty increases over time. Once the altitude is 20 feet below the desired number, the application transitions to the next state. The application continues to transition to the next state until the task is completed by reaching desired altitude as shown in Figure 9.

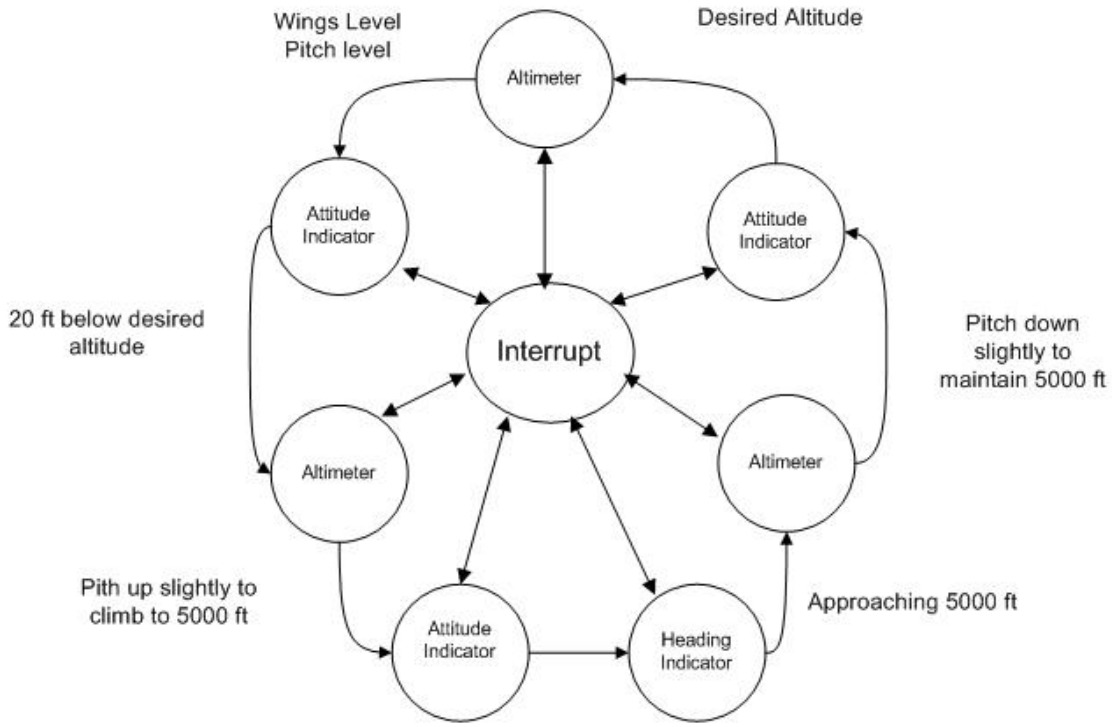


Figure 9: Straight and level state machine diagram

Table 4: A Typical Scan technique for Straight and Level flight

Instrument	**Reading**
Attitude Indicator	Wings level, Pitch level
Altimeter	20 feet below desired altitude
Attitude Indicator	Pitch up slightly to climb to 5,000 feet
Heading Indicator	Right on heading
Attitude Indicator	Maintain wings level
Altimeter	Approaching 5,000 feet
Attitude Indicator	Pitch down slightly to maintain 5,000 feet
Airspeed Indicator	As desired

Source: Stein, 1996

3.3.3. Flight Panel Layout

XML was created to structure, store, and transport information. The <*flightpanel*>, for example, describes the root element of the document. The <*component*> element in the document is contained within <*flightpanel*>. The other tag names are the child elements of the root (name, position, bottomleft, etc). A sample XML sample flight panel is presented in Figure 10.

Each component must have an IDNO which is identification number of the instrument. The identification number is used for analysis and scanning. The "bottomleft" contains the x, y and z position of the instrument. The Pilot Eye Flight Deck Application uses the coordinates to update the eye. The "threshold" is the uncertainty value. This value is used by the Pilot Eye Flight Deck Application to determine how the pilot eye movement is interrupted. The "duration" contains the measure of the average duration of a pilot's focus on that instrument.

```
<flightpanel>
<component IDNO = "0001">
<name>PFD</name>
<position>
<bottomleft> <x>0</x> <y>0</y> <z>0</z></bottomleft>
</position>
<elementtype>display</elementtype>
<threshold>150000</threshold>
<duration>5000</duration></component>
.
```

Figure 10: XML Sample flight panel file.

3.4. Class Diagrams

During a monitoring task, the pilot periodically views each display for an external environment trigger while performing procedural tasks. This is implemented in the brain class which has an interrupt mechanism. The interrupt distracts the virtual eye scanning pattern from the assigned task. The eye moves immediately to the instruments that need the immediate attention.

3.4.1. The Brain Class

The brain class is the center of the program and is shown in Figure 11 (see Appendix B for program code details). This class contains the XML reader that reads the interrupt and the flight panel files. It also contains the clocking or timing system that measures in milliseconds (ms) how much time the eye is focused on each instrument in the cockpit.

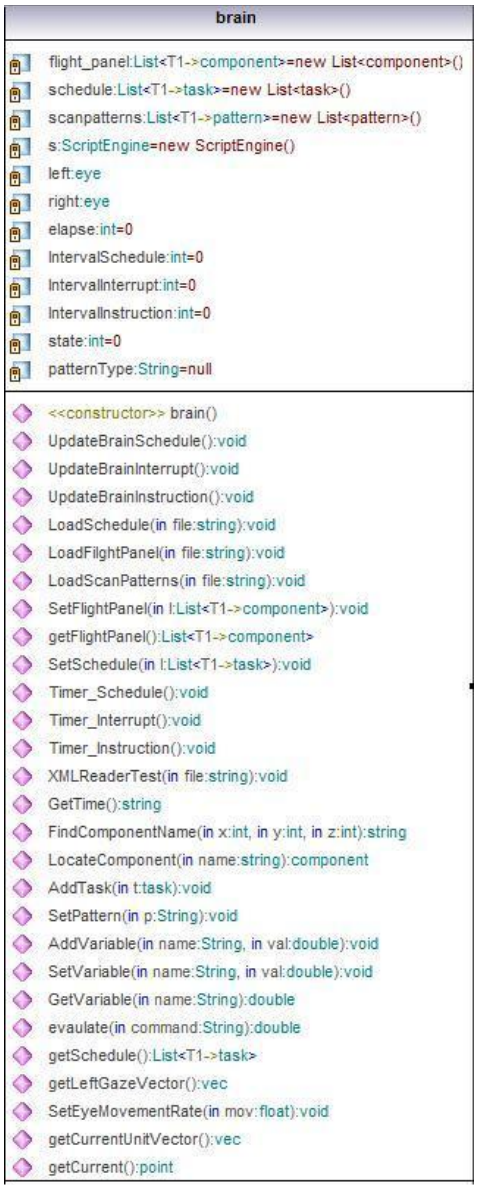


Figure 11: The brain class.

3.4.2. The Eye Class

For the pilot to monitor the flight deck systems and to make adjustments to control settings for the desired performance of the aircraft, the eyes must move from one instrument to another to process information from the environment .To simulate this scenario, an eye class is implemented in the program. The eye class provides the location that the eye is focused on at that time. It also provides the direction of where the eye will focus next in the cockpit. The eye class is shown in Figure 12 (see Appendix B for program code details).

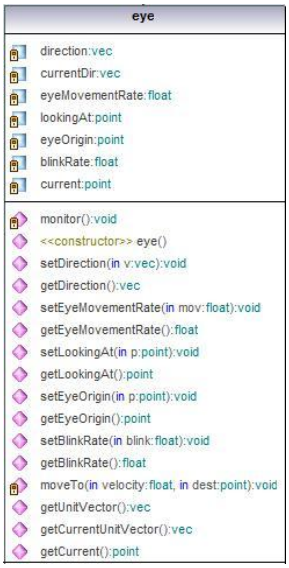


Figure 12: The eye class stores and returns the eye origin, direction and location.

3.4.3. Eye Movement

The Pilot Eye Flight Deck Application is responsible for reading inputs from the brain, moving the eye toward a position, and providing directional data back to the brain.

The eye rotates in three-dimensional space at the origin. The eye movement is restricted to the z direction and +/- π around the x and y axis as in Figure 14.

A thread is used to monitor and coordinate inputs from the brain. It rotates the eyes to the specified coordinates. Using basic trigonometry the angular rotation of the eye is calculated as shown in Figure 15. To rotate the eye to view a point in free space, a transformation must be computed. Streams of eye directional data are sent back to the brain as Smart Eye Tracking UDP packets.

3.5. Flight Panel Visualization

The Pilot Eye Flight Deck Application also provides its own flight panel visual interface for the Boeing737-700. The values in display are static; therefore, do not update according to the scenarios that is implemented. The Pilot Eye Flight Deck Application sends its eye data to the static visualization interface. The panel provides crosshairs to show what instrument the eye is looking at. This flight panel is a visual aid for the users to test their XML descriptions. The application static interface provides two different views of the Boeing 737-700 cockpit in 2D and 3D as shown in Figure 16 and Figure 17.

3.6. Writing Application

The Pilot Eye Flight Deck Application library is a collection of interfaces used to create an application. The library has four major external interfaces: UDP, 2D flight panel, 3D flight panel and Person. The UDP interface sends Smart Eye Tracking packets (see Appendix A) that can be read by Smart Eye Tracking capable application. The 2D flight panel is a simple static interface that reads coordinate data to set the position of the crosshair shown in Figure 16. The 3D flight panel interface is a simple static interface that uses the coordinated data to move a crosshair in a three-dimensional space. The Person interface is divided into the eye and brain. The Brain sends information to the eye such as “look here” and “what are you looking at”. The eye runs on an independent threaded process, therefore the eye is always in motion regardless of what the brain is

processing. The brain is divided into the XML reader, interrupt and state machine. The XML reader reads XML files and loads the interrupt and state machine system. Both the interrupt and the state machine system run on separate threads. The interrupt system can stop the machine to override messages to the eye. The state machine contents consist of the scripting engine and the state machine interpreter.

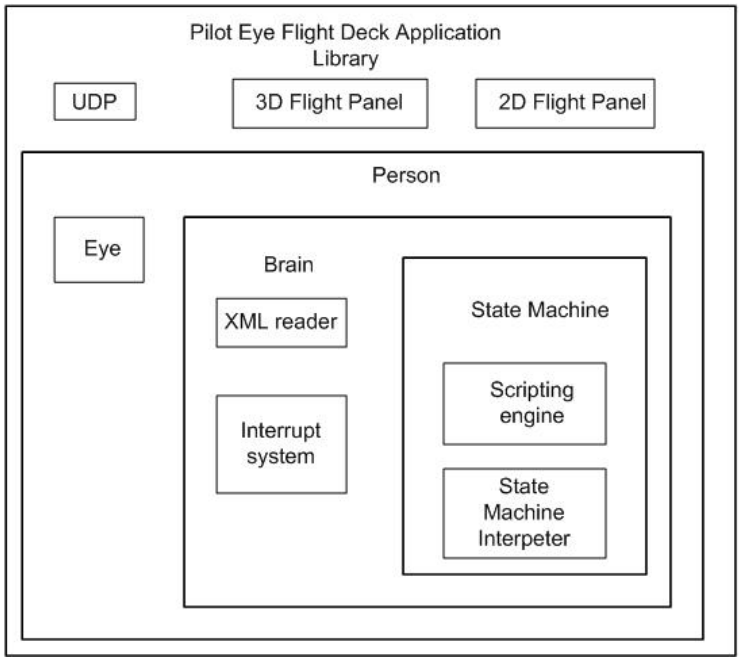


Figure 13: Pilot Eye Flight Deck Application Library Architecture.

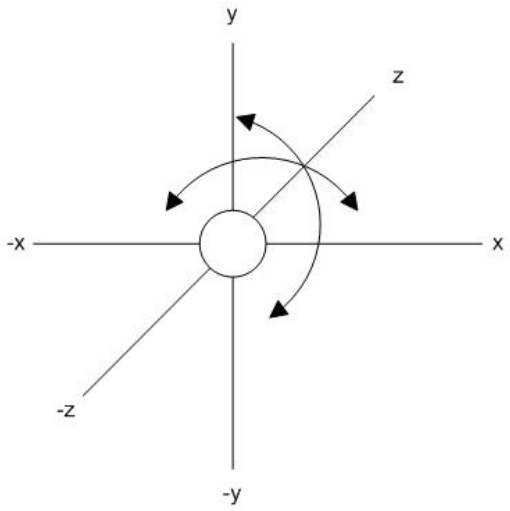
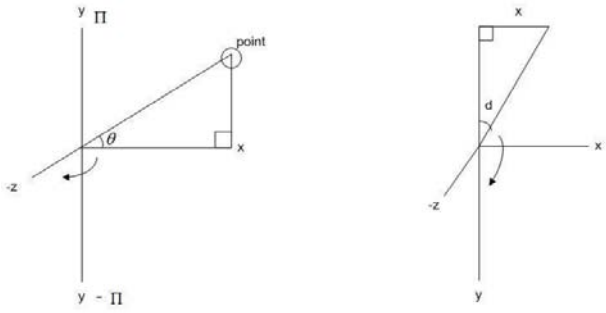


Figure 14: The eye range of rotation at origin.



$$theta = arctan2\frac{z}{x} \qquad delta = arctan2\frac{z}{y}$$

$$\left[arctan2\frac{z}{x}, arctan2\frac{z}{y}, 0\right]$$

Figure 15: Equations used to calculate the angular components of the eye in free space. Basic trigonometry was used to create equations to calculate the angular rotation of the eye.

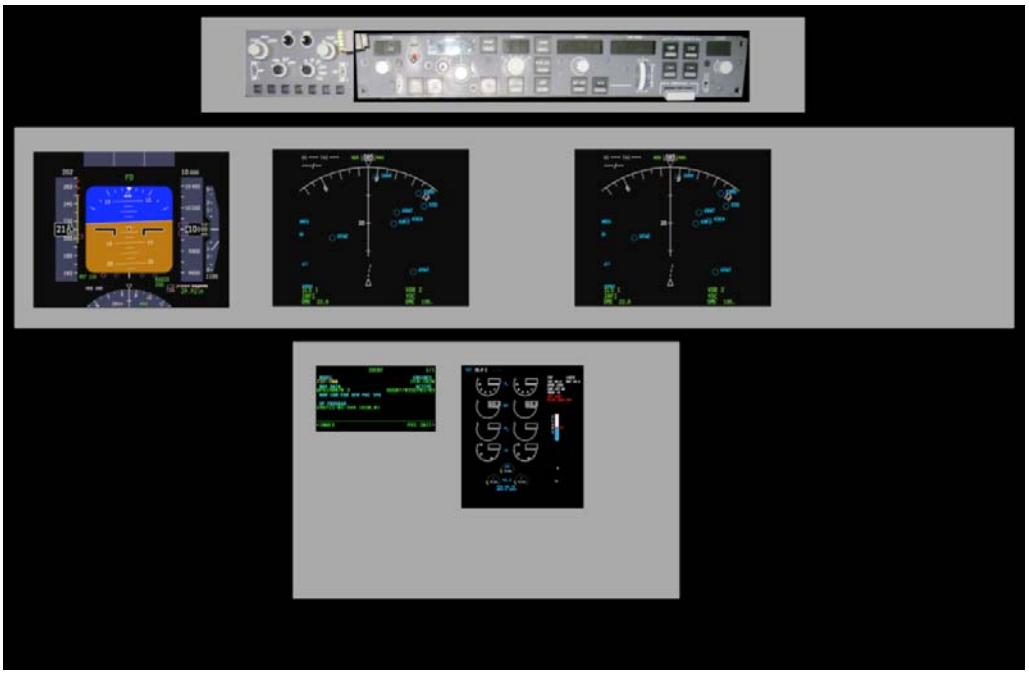


Figure 16: Two-dimensional flight panel visualization static interface.

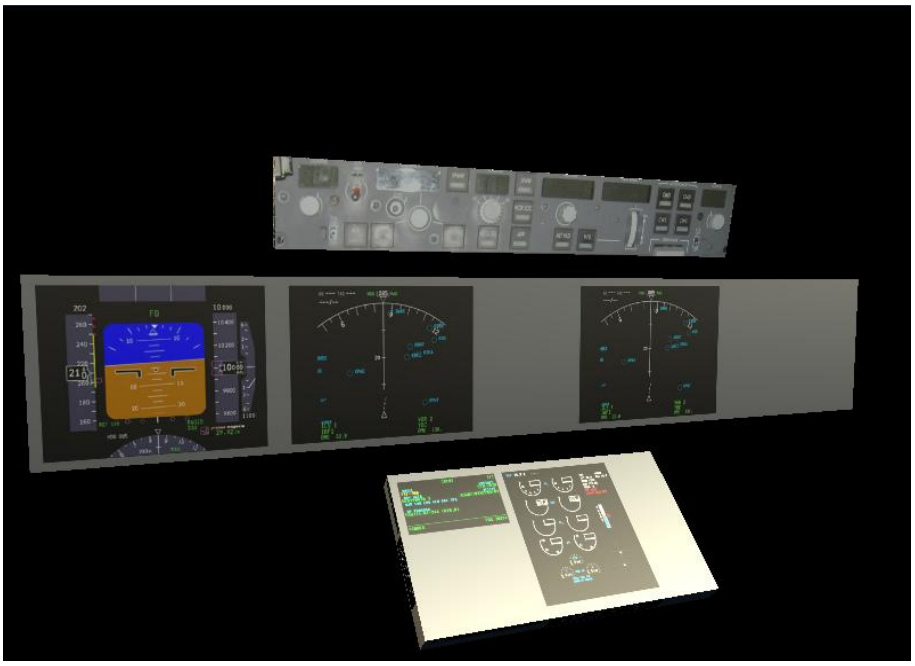


Figure 17: Three-dimensional flight visualization static interface.

CHAPTER 4: ANALYSIS AND RESULTS

But after observation and analysis, when you find that anything agrees with reason and is conducive to the good and benefit of one and all, then accept it and live up to it.

– Buddha, 483 B.C

4.1. Basic Flight Maneuvers

There are four fundamental basic flight maneuvers: straight and level flight, turns, climbs, and descents. All controlled flight consists of either one, or a combination of more than one, of these basic maneuvers. While flying, pilots must constantly be aware of the attitude, altitude, power setting, and performance of their airplanes. Instrument scanning is important because you do not want to fixate on any one instrument, omit an instrument, or over-emphasize any instrument. The human sense of balance is not adequate to keep an airplane flying right side up, the pilot must scan the instruments to make this determination. The information presented in the following paragraphs is referenced from the Instrument Flying Handbook 2008 by the *Federal Aviation Administration* (FAA).

4.1.1. Straight and Level Flight

Straight and level flight is flight in which a constant heading and altitude are maintained. The instruments that directly or indirectly indicate pitch on the *Primary Flight Display* (PFD) are the attitude indicator, altimeter, *vertical speed indicator* (VSI), and *airspeed indicator* (ASI). For straight and level flight, the altimeter provides the primary indication of pitch performance and the heading indicator provides the primary bank performance information. If the altimeter and heading indicators remain on the same indication, the airplane is flying straight and level. The airspeed indicator provides the primary indication of resultant performance from the power setting except when the airspeed indicator is used as the primary pitch instrument. The scan should occasionally

encompass the other instruments, which provide a backup and help identify problems with one or more of the primary instruments.

4.1.2. Turns

At some point, the aircraft will need to be turned to maneuver along airways, GPS courses, and instrument approaches. As in straight and level flight, the altimeter is the primary pitch instrument in a level turn. In a standard rate turn, the pilot will use the Turn indicator to maintain a 3° per second turn which will yield a complete 360° turn in 2 minutes. A turning rate of 3° per second will allow for a timely heading change, as well as allow the pilot sufficient time to cross-check the flight instruments and drastic changes to the aerodynamics forces being exerted on the aircraft. At no time should the aircraft be maneuvered faster than the pilot is comfortable while cross-checking the flight instruments.

4.1.3. Climb

In a straight climb, the heading-indicator is the primary bank instrument. The primary pitch instrument is the vertical speed indicator for a constant rate climb. To climb at a constant airspeed, the airspeed indicator is the primary pitch instrument – rather than the vertical speed indicator. As power is applied, the elevator pressure raises the yellow chevron which represents the nose of the aircraft, to the desired pitch attitude that equates to the desired vertical speed rate. Once the aircraft is stabilized at a constant airspeed and pitch attitude, the primary flight instrument for pitch will be the ASI and the primary bank instrument is the heading indicator. If the pitch attitude is correct, the airspeed should slowly decrease to the desired speed and the pilot is able to devote more time to maintaining an effective scan of all instrumentation.

4.1.4. Descent

Descents can be accomplished with a constant rate and constant airspeed. A reduction in power allows the aircraft to decelerate to the desired airspeed while maintaining straight and level flight. An increase in the scan rate during the transition is important since changes are being made to the aircraft flight path and speed. As the aircraft approaches the desired airspeed, the pilot reduces the power to a predetermined value. The airspeed continues to decrease below the desired airspeed unless a simultaneous reduction in pitch is performed. The primary instrument for pitch is the ASI tape. If any deviation from the desired speed is noted, small pitch corrections are made by referencing the attitude indicator and validating the changes made with the airspeed tape. When performing a constant rate descent, while maintaining a specific airspeed, the coordinated use of pitch and power will be required. Any change in pitch directly affects the airspeed. Conversely, any change in airspeed will have a direct impact on vertical speed as long as the pitch is being held constant.

4.2. Data Analysis

The Pilot Eye Flight Deck Application executes a scenario of climbing from 5,000 to 6,000 feet. The starting state of the aircraft is at 5,000 feet, moving at 190 knots and in a straight and level position. The aircraft will climb at a rate of 500 Feet Per Minute (FPM). The pitch changes until a 500 FPM climb rate is achieved. The aircraft will maintain a constant heading of 330 degrees and maintain a bank angle of 0. The pilot will adjust the throttle, so that the aircraft is traveling at the desired airspeed. This process is repeated again until the aircraft is at 6,000 feet.

4.2.1. Pilot Scan Pattern

The instrument readings are initially input into the Pilot Eye Flight Deck Application before running the scenario. The readings can be simulated data or recorded data. To simulate the pilot attention, the Pilot Eye Flight Deck Application emulates the

steps for a climb procedure. The state machine is designed with the state rules as shown in Figure 18.

```
<scanpatterns>
        <pattern name="Climb &amp; Descent">
                <state name="VSI">(&gt;&#61;,VSI,500)</state>
                <state name="ATTITUDE">(and,(&gt;, VSI,490),(&lt;, VSI,510))</state>
                <state name="HEADING">(and,(&gt;,heading,328),(&lt;,heading,332))</state>
                <state name="ATTITUDE">(and,(&gt;, bank, -2), (&lt;,bank,2))</state>
                <state name="AIRSPEED">(and,(&gt;,(-,airspeed,desired,5)),(&lt;,(+airspeed,desired,5)))</state>
                <state name="REPEAT">(&lt;,altitude,6000)</state>
        </pattern >
</scanpatterns>
```

Figure 18: The state machine for a climb scenario.

Each state in the state machine corresponds to an instrument in the visual display shown in Figure 19.The Pilot Eye Flight Deck Application is initiated in the VSI states, once the climb rate is greater than 500 FPM, the application checks the HEADING. If the heading is between 328 and 332; it checks the ATTITUDE for a bank between -2 degrees and 2 degrees. If the airspeed is +/- 5 feet per second of the desired speed, the state machine enters into the repeat state which checks if the altitude is less than 6000 feet. If the altitude is greater than 6000 feet, the scenario is complete.

The Pilot Eye Flight Deck Application is connected to an external data source that reports the instruments readings. The ideal readings for a climb from 5000 to 6000 feet are shown in Figure 20. The instrument measurement graphs vary for different pilots,

airplanes and tasks. It is unlikely that two pilots flying the same aircraft and performing the same task will have the same data. Therefore, it is very important to perform the same state machine analysis on data flown by different pilots flying the same aircraft and performing the same task.

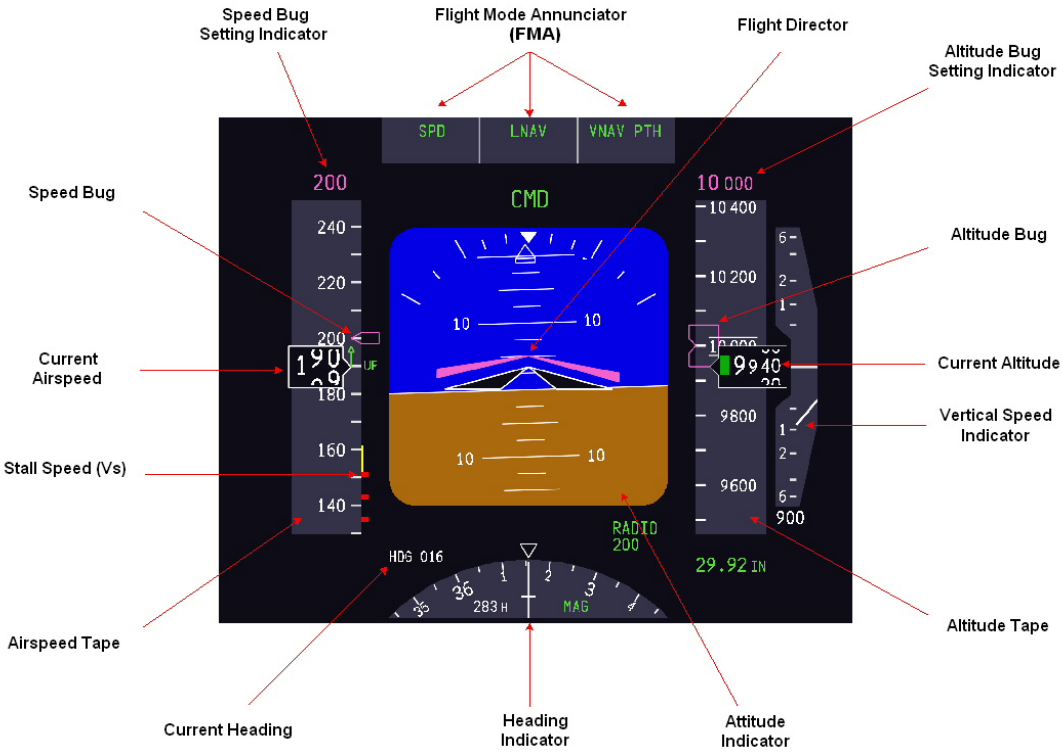


Figure 19: PFD Electronic Flight Instrument System (EFIS).

Source: Ellis, 2009

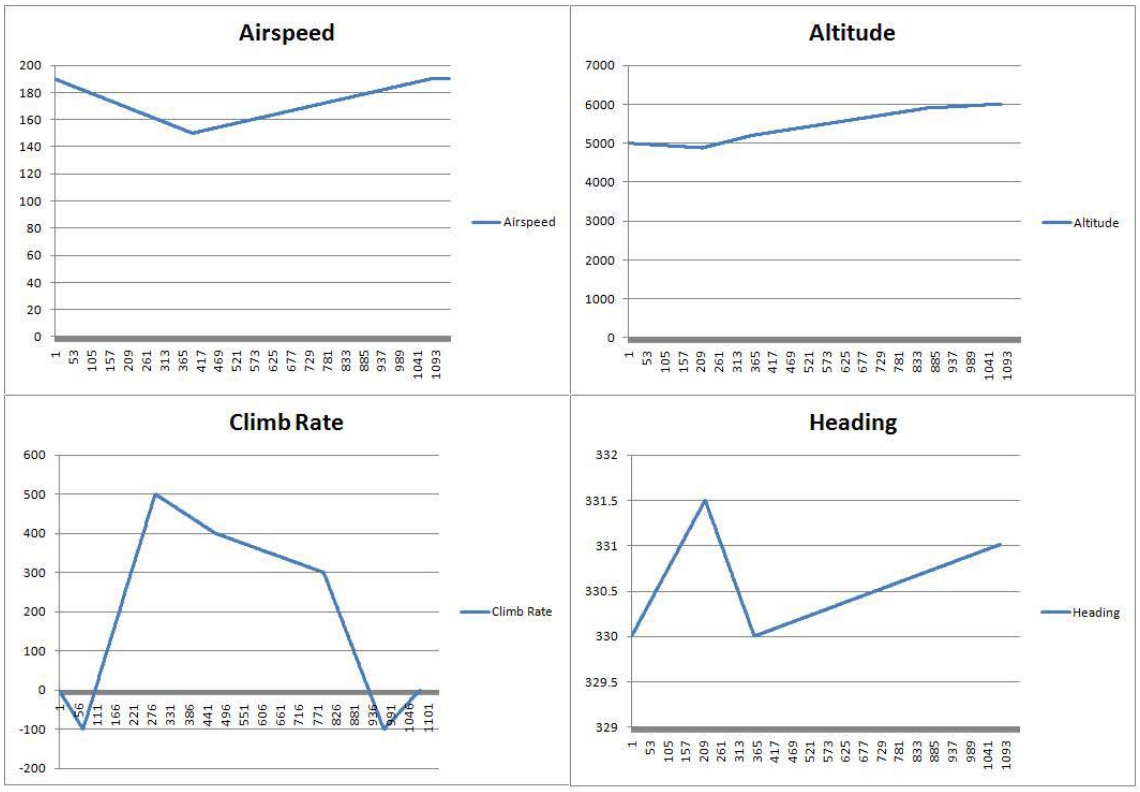


Figure 20: Raw data from Pilot Eye Flight Deck Application. Not to scale. (Note: Static data was used based on state machine scenario.)

The altitude of the aircraft depicted in the graph is a Boeing 737-700 which gradually rises from 5000 to 6000 feet in approximately 3 minutes. The climb rate is a derivative of the altitude. As the altitude increases at a content rate, the climb rate increases to 500 FPM. After the altitude of the aircraft levels off at 6,000 feet, the climb rate decreases. As the aircraft is climbing, the heading changes to 332 degrees, to simulate the pilot attention being drawn away from the heading indicator. Once the pilot's attention is directed towards the heading indicator, the pilot corrects the heading. The aircraft airspeed deceases during the start of the climb, and increases as the pilot

applies more throttle. Figure 21 shows the flight panel for a Boeing 737-700. Crosshairs on the instruments shows where the eye focus is currently situated.

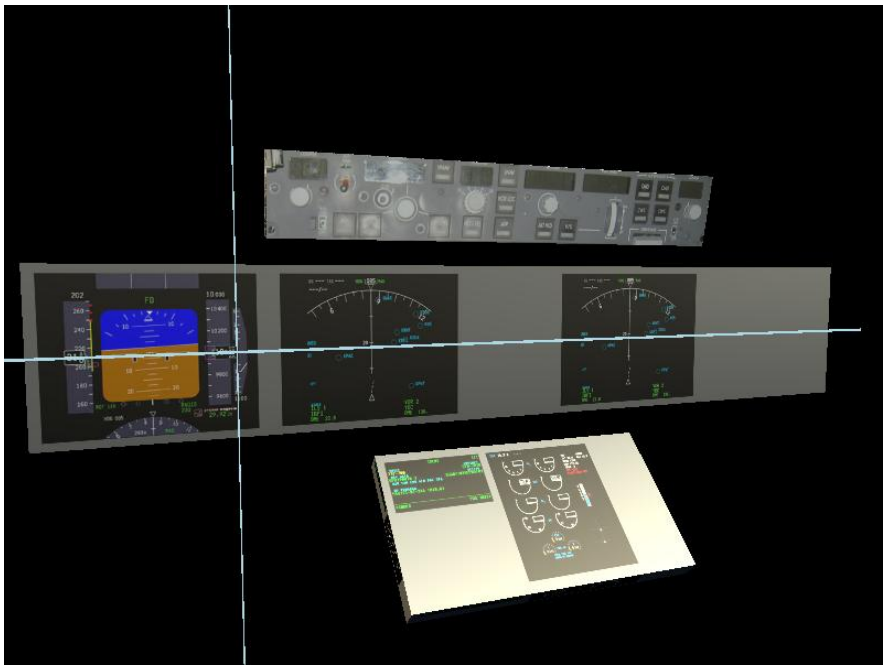


Figure 21: The static visualization of the flight panel for a Boeing 737-700.

Figure 22 shows the phases of a climb flight maneuver. In phase A, the Pilot Eye Flight Deck Application simulates the pilot monitoring the VSI until 500 FPM is reached. At approximately 23 seconds the aircraft will achieved 500 FPM. The application monitors the VSI indicator to verify that the aircraft is climbing at 500 FPM in phase A. After approximately 6 seconds the VSI indicator reads 500 FPM, the application checks the heading to confirm a constant heading of 330 degrees in phase B. At phase C, approximately 51 seconds, the heading is constant at 330 degrees.

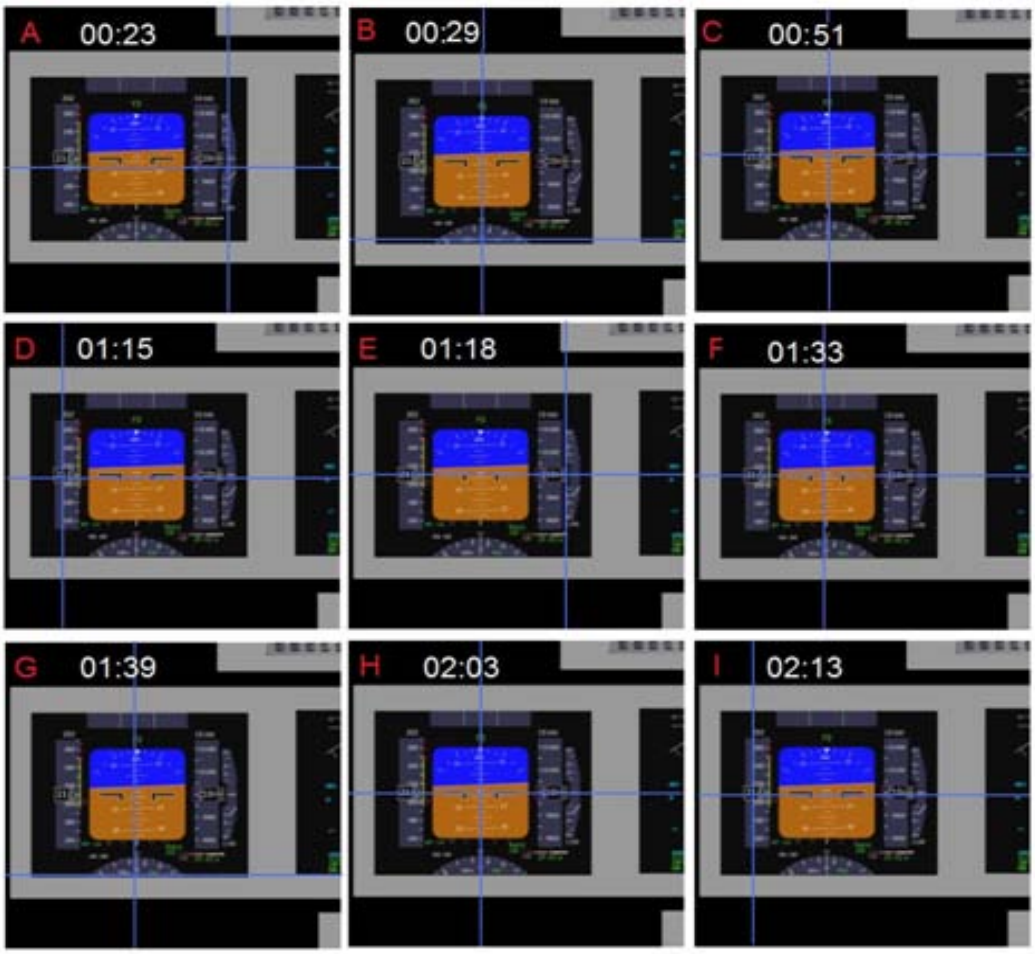


Figure 22: The phases of a climb flight maneuver. (Note: Display data is static. No dynamic data was available.)

The application simulates the pilot's need adjust the airspeed, in phase D. The application detects that the pilot applies more throttle and checks for the climb rate of 500 FPM at approximately 1 minute and 18 seconds, as shown in phase E. The application simulates that pilot's need to maintain pitch and bank, and monitors the attitude indictor until the aircraft is level, as shown in phase F. The application monitor's the heading until the readings is 330 degrees. This process repeats until the 6000 feet is achieved. The Pilot

Eye Flight Deck Application completed the task in approximately 2 minutes and 43 seconds.

Figure 23 shows the scan pattern path of Pilot Eye Flight Deck Application for a climb scenario. The Pilot Eye Flight Deck Application performed the scan pattern in the climb. The order of the scan for this scenario is the VSI, attitude, heading and airspeed indicator(FAA, 2009). As mentioned above, the pattern scan repeats until desired altitude is reached.

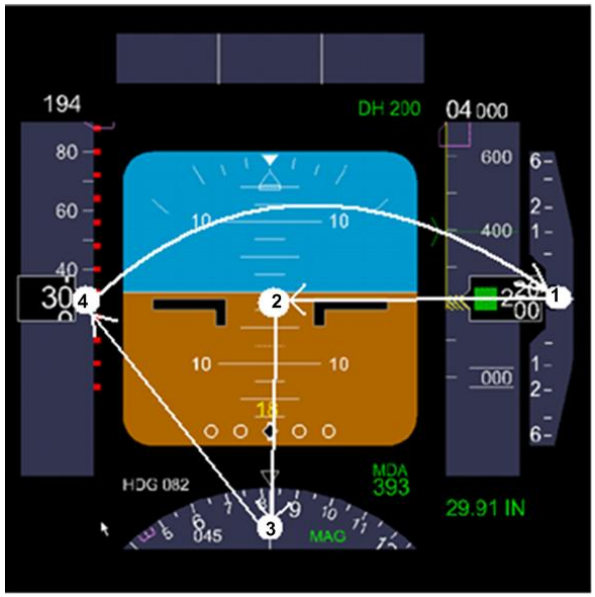


Figure 23: The sequence the eye traveled starting at the VSI.

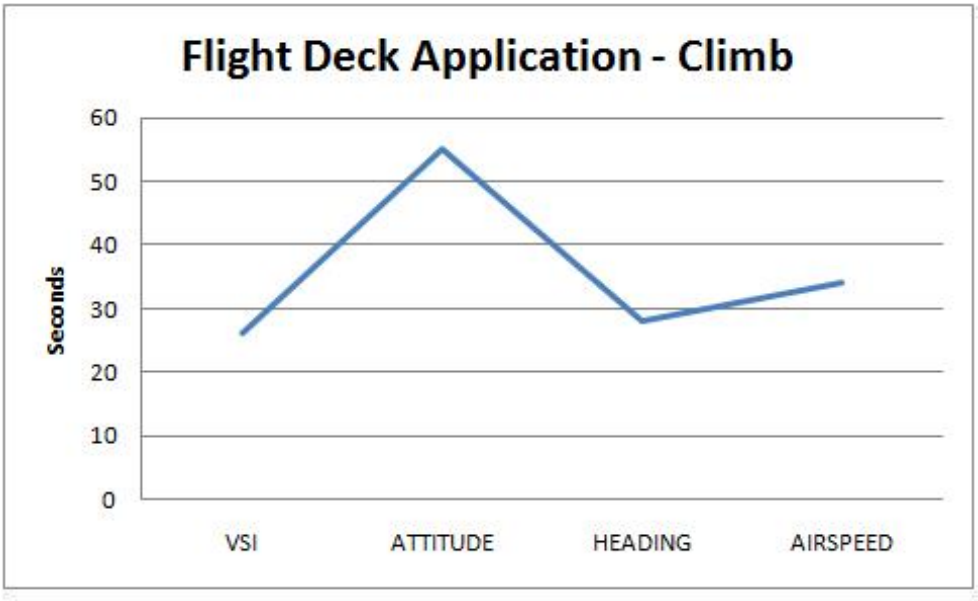


Figure 24: The time the Pilot Eye Flight Deck Application spent monitoring each instrument. (Note: Times are cumulative.)

Figure 24 shows the time monitored on each instrument for the climb. The state movement application monitored VSI for approximately 26 seconds, attitude for approximately 55 seconds, heading for approximately 28 seconds and airspeed for approximately 34 seconds. The data collected with Pilot Eye Flight Deck Application when simulating the pilot's eye scanning behavior can be use to determine if the new technology of NextGen is beneficial or hazardous during aircraft operation. In addition, the data collected will assist in designing better interfaces for pilots to locate valuable information to achieve the goals of a specific task.

CHAPTER 5: CONCLUSION AND DISCUSSION

Follow theological 'reasons' far enough and it always leads to conclusions that are contrary to reason.

– Anonymous

5.1. Conclusion

Eye movements can provide a great source of information for psychologists and engineers. Applications like eye tracking can provide researchers with data concerning the user interaction with a device. Information can be recorded in real time with the use of today's state of the art technology. Eye tracking is helpful in understand a person's gaze and their focus of attention when completing a task. Studying basic gaze patterns may prove helpful in understanding the errors that can occur during execution of a task. These tasks include simulating the effects of introducing new tools and enhancing training strategies. Human performance is vital in the development of the NextGen technologies.

5.2. Pilot Eye Flight Deck Application Limitations

The conclusions drawn were limited by several factors that affect the results obtained. The most significant limitations were due to the state machine that can only detect items that are directly at the focus of attention. Most of the responses for the state machine are pre-programmed responses. Static data was used when dynamic data was not available. The state machine configuration file is limited because it could only operate with one cockpit layout, which is the B737-700 model.

5.3. Recommendations for Future Research

The following models are recommended for future research:

- Use of Soar or ACT-R as one of the cognitive architecture will provide a convenient computational framework in which models automatically abide by

built-in, well tested parameters and constraints on cognition and perceptual-motor processes. The general approach to modeling pilot behavior in cognitive architectures will be extremely productive for future research in a wide range of pilot behavior phenomena.

- Use of model validation will be an important step in creating a useful model. Comparing model predictions to the empirical data will allow it to be connected to the predicted action and the experimental action to validate collected data. The empirical data will be used to analyze the model to determine where the model can be improved. Physical models can be validated by using precise experimental measurements. The goal of model validation is to replicate both the information processing steps and the behavioral outcome of a human performing a task in a theoretically sound manner.

REFERENCES

AAJ. "All About The Journey". http://www.allaboutthejourney.org/human-eye.htm (accessed by July 1, 2009)

A. L. Yarbus (1967) Eye Movements and Vision. New York: Plenum Press (Translated from the 1965 Russian edition by Basil Haigh)

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*, 1036-1060.

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J.R., ACT-R. (2009). http://act-r.psy.cmu.edu/ (accessed by July 1, 2009)

"Answers.com". http://www.answers.com/ (accessed July 1, 2009)

Best, B., Lebiere, C., Schunk, D., Johnson, I, & Archer, R. (2004). *Validating a Cognitive Model of Approach Based on the ACT-R Architecture.* (Technical Report). Boulder, CO: Micro Analysis and Design, Inc.

"Britannica Online Encyclopedia", http://www.britannica.com/ (accessed by July 1, 2009).

Byne, M. D. (2001). ACT-R/PM and Menu Selection: Applying a Cognitive Architecture to HCI, *International Journal of Human-Computer Studies*, 55, 41-84.

Byrne, M. D., & Gray, W. D. (2003). Returning Human Factors to an engineering discipline: Expanding the science base through a new generation of quantitative methods. Preface to the Special Section. *Human Factors, 45*(1), 1-4.

Byrne, M. D. and Kirlik, A. (2004). Integrated Modeling of Cognition and the Information Environment: A Multilevel Investigation (Process and Product Modeling) of Attention Allocation to Visual Displays. (Technical Report AHFD-04-14/NASA-04-4) Savoy, IL: University of Illinois, Institute of Aviation.

Byrne, M. D., & Kirlik, A. (in press). Using computational cognitive modeling to diagnose possible sources of aviation error. To appear in *International Journal of Aviation Psychology*.

Byrne, M.D., Kirlik, A., Fleetwood, M.D., Huss, D.G., Kosorukoff, A., Lin, R., Fick, C.S., (2004). A closed-loop, ACT-R approach to modeling approach and landing with and without synthetic vision system (SVS) technology. Proceedings of the Human Factors and Ergonomics Society 48th Annual Meeting.

Corker, K. M., Human Automation Integration Laboratory: Air MIDAS, 2003. http://www.engr.sjsu.edu/hfe/hail/airmidas.htm (accessed by July 1, 2009)

Corker, K., Gore, B.F., Guneratne, E., Jadhav, A. & Verma, S. (2002). *Integration of Air MIDAS Human Visual Model Requirement and Validation of Human Performance Model for Assessment of Safety Risk Reduction through the implementation of SVS technologies.* NASA Contract Task Order #: NCC2-1307.

Crowe, E.C. (2000). Comparing interfaces based on what users watch and do. *Eye Tracking Research and Applications Symposium, 2002.*

Deutsch, S., D-OMAR, 2002. http://omar.bbn.com/ (accessed by July 1, 2009)

Deutsch, S., & Pew, R. (2004). Examining new flight deck technology using human performance modeling. *Proceedings of the Human Factors and Ergonomics Society 48th Annual Meeting.* (pp.108-112). Santa Monica: Human Factors and Ergonomics Society.

Duchowski, A.T. Eye Tracking Methodology. Theory and Practice. Springer-Verlag London Limited. 2003.

Editure, Worksheet 2: The Human Eye – structure and function http://www.schools.net.au/edu/lesson_ideas/optics/optics_wksht2_p1.html (accessed by July 1, 2009).

Ellis, K., M.S. Thesis. Eye Tracking Metrics for Workload Estimation in Flight Deck. University of Iowa, Iowa City, IA. 2009.

"Federal Aviation Administration". http://www.faa.gov/ (accessed July 1, 2009).

Farmer, E. & Brownson, A., QinetiQ (2003). Review of Workload Measurement, Analysis and Interpretation Methods. European Organization for the safety of air navigation.

Fischer, B. (1998). Attention in saccades. In: Visual Attention. Wright R.D. (ed). Oxford University Press. New York.

Fitts, P.M., Jones, R.E., and Milton, J.L. (1950). Eye movements of aircraft pilots during instrument landing approach. *Aeronautical Engineering Review*, 9, 1-5.

Foyle, D.C. and Hooey, B.L. (2008). *Human performance modeling in aviation*. Boca Raton, FL: CRC Press/Taylor & Francis.

Gobet, F., & Ritter, F. E. (2000). Individual Data Analysis and Unified Theories of Cognition: A methodological proposal. In N. Taatgen & J. Aasman (Eds.), *Proceedings of the 3rd International Conference on Cognitive Modelling*. 150-157. Veenendaal (NL): Universal Press.

Goodman, A., Hooey, B. L., Foyle, D. C., & Wilson, J. R. (2003). Characterizing visual performance during approach and landing with and without a synthetic vision display: A part task study. In D. C. Foyle, A. Goodman & B. L. Hooey (Eds.), *NASA Aviation Safety Program Conference on Human Performance Modeling of Approach and Landing with Augmented Displays.* NASA CP-2003- 212267. Moffett Field, CA: NASA.

Gray, W. D. (Ed.) (2007). *Integrated Models of Cognitive Systems*. Oxford, UK: Oxford University Press.

Haber, R.N., and Hershenson, M. (2nd Ed). Psychology of Visual Perception. Rinehart and Winston. 1973.

Haines, C. D. (editor) (Oct. 1, 2005). Eye Health: The Amazing Human Eye: Your Guide to How the Eye Sees. Retrieved April 25, 2009 from http://www.webmd.com/eye-health/amazing-human-eye.

Hancock, P. A., & Desmond, P. A. (2000). Stress, workload, and fatigue.Mahwah, NJ: Lawrence Erlbaum.

Henkind , P., Hansen, R.I. and Szalay, J. (1979) Ocular circulation. In "Physiology of the human eye and visual system" (Ed. Records, R.E.) pp 98-155. Harper & Row, new York.

Hewett, M., The 737 Virtual Flight Deck. http://www.boeing.com/commercial/aeromagazine/aero_04/textonly/ps02txt.html (accessed by July 1, 2009)

Hill, R.W., "Modeling Perceptual Attention in Virtual Humans". *Proceedings of 8th Conference on Computer Generated Forces and Behavioral Representation.* Orlando, FL., May 1999.

Itti, L. & Koch, C. (2000). A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision Research, 40*(10-12), 1489-1506.

Jacob, R. J. K., & Karn, K. S. (2003). Eye tracking in Human-Computer Interaction and usability research: Ready to deliver the promises, In J. Hyönä, R. Radach, & H. Deubel (Eds.), *The mind's eye: Cognitive and applied aspects of eye movement research* (pp. 573-605). Amsterdam: Elsevier.

Jarrett, D. N. (2005). Cockpit engineering. Aldershot, Hampshire, England; Burlington, VT: Ashgate Publishing Company.

Jones, D. H., (1985). *An error-dependent model of instrument scanning behavior in commercial airplane pilots.* Springfield, Va.: NASA Langley Research Center.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, *33*, 1-64.

Laird, J. E., 2008., Soar. http://sitemaker.umich.edu/soar/home (accessed by July 1, 2009)

Leiden, K. & Best, B.(2005). *A Cross-Model Comparison of Human Performance Modeling Tools Applied to Aviation Safety.* Micro Analysis & Design, Inc. Boulder, CO 80301, 2005

Leiden, K., Keller, J. W., & French, J.W. (2001). *Context of Human Error in Commercial Aviation*. (Technical Report). Boulder, CO: Micro Analysis and Design, Inc.

Leiden, K., Laughery, K. R., Keller, J. W., French, J. W., Warwick, W. & Wood, S.D. (2001). *A Review of Human Performance Models for the Prediction of Human Error*. (Technical Report). Boulder, CO: Micro Analysis and Design, Inc.

Leveson, N. (2001a). The role of software in recent aerospace accidents. Proceedings of the 19th International System Safety Conference, System Safety Society: Unionville, VA.

Leveson, N. (2001b). Systemic factors in software-related spacecraft accidents. AIAA Space 2001 Conference and Exposition. Paper AIAA 2001- 4763. AIAA: Reston, VA.

Montgomery, T. M., Anatomy, Physiology & Pathology of the Human Eye. http: //www.tedmontgomery.com/the_eye/. (accessed July 1, 2009).

Nabatilan,L. Ph.D. Dissertation, The Louisiana State University, Baton Rouge, LA, 2007.

NASA Human Performance Modeling Element (2002). *HPM-SVS part-task simulation documentation: Characterizing pilot performance during approach and landing with and without visual aiding.* Moffett Field, CA: NASA Ames Research Center.

Neece, R., (2007). "Program Background - EHD Element of IIFD," NASA – Ohio University Kickoff Meeting Presentation.

Newell, A., (1990) Unified Theories of Cognition. Harvard University Press.

Operator Performance Laboratory. CCAD. University of Iowa.

"Pearson Education". http://www.pearsoned.com/ (accessed by July 1, 2009).

Ritter, F. E., Van Rooy, D., & St. Amant, R. (2002). A user modeling design tool based on a cognitive architecture for comparing interfaces. In *Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002,* 111-118. Dordrecht, NL: Kluwer Academics Publisher.

Salvucci, D. D. (2001). Predicting the effects of in-car interface use on driver performance: An integrated model approach. *International Journal of Human-Computer Studies*, *55*, 85-107.

Salvucci, D. D. (2002). Modeling driver distraction from cognitive tasks. In *Proceedings of the 24th Annual Conference of the Cognitive Science Society* (pp. 792-797). Mahwah, NJ: Lawrence Erlbaum Associates.

Salvucci, D. D. (2005). A multitasking general executive for compound continuous tasks. *Cognitive Science*, *29*, 457-492.

Salvucci, D. D. (2006). Modeling driver behavior in a cognitive architecture. *Human Factors*, *48*, 362-380.

Salvucci, D. D., & Macuga, K. L. (2002). Predicting the effects of cellular-phone dialing on driver performance. *Cognitive Systems Research*, *3*, 95-102.

Salvucci, D. D., Taatgen, N. A., & Kushleyeva, Y. (2006). Learning when to switch tasks in a dynamic multitasking environment. In *Proceedings of the Seventh International Conference on Cognitive Modeling* (pp. 268-273). Trieste, Italy: Edizioni Goliardiche.

Savetz, K., FlightSimBooks.com. http://www.flightsimbooks.com/flightsimhandbook/ (accessed June 20, 2009).

Taatgen, N.A. (1993). The study of learning mechanisms in unified theories of cognition. University of Groningen, Groningen.

Veltman, J.A., Gailliard, A.W.K. (1993). Measurement of Pilot Workload with Subjective and Physiological Techniques. In *Proceedings of the Workload Assessment and Aviation Conference,* pp. 3.1-3.13.

Verma, S., Corker, K. & Jadhav, A. (2004). Modeling Visual Behavior of Pilots in the Hmuan Performance Model – Air MIDAS. In *Proceedings of the Sixth International Conference on Cognitive Modeling*, 396-397. Mahwah, NJ: Lawrence Earlbaum.

Wetzel, P. A. & Poprik, C (1996). An eye tracking system for analsis of pilots scan paths. Mesa, Ariz.: Human Resources Directorate, Aircrew Training Research Division, U.S. Air Force, Armstrong Laboratory 1997.

Wickens, C. D., & McCarley, J. M. (2008). *Applied Attention Theory.* Boco-Ratan, FL: CRC Press, Taylor & Francis.

Wickens, C. D., McCarley, J. S., Alexander, A. L., Thomas, L. C., Ambinder, M. & Zheng, S. (2005). *Attention-Situation Awareness (A-SA) Model of Pilot Error.* (Technical Report AHFD-04-15, NASA-04-5). Savoy, Illinois: University of Illinois at Urbana-Champaign, Institute of Aviation Human Factors Division.

Wright, R.D., and Ward, L.M. (1998). The control of visual attention. In: Visual Attention. Wright R.D. (ed). Oxford University Press. New York.

APPENDIX A

THE UDP PACKET

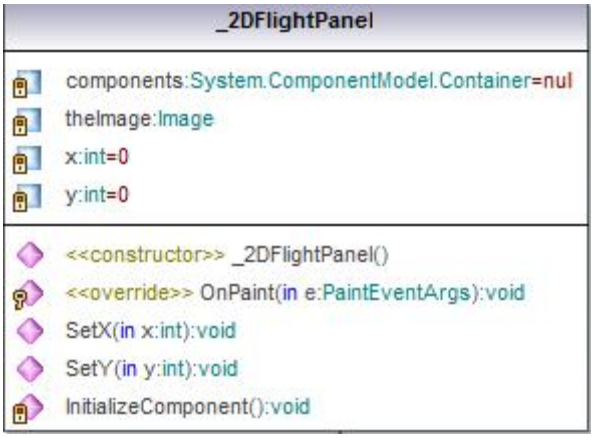
The UDP packet structure for the Smart Eye Pro 3.0

Packet header		**Sync Id** **SEu32, 4 bytes**
		Packet type **SEu16, 2 bytes**
		Packet length **SEu16, 2 bytes**
Subpacket 1	**Subpacket header**	**Id** **SEu16, 2 bytes**
		Length **SEu16, 2 bytes**
	Subpacket data	**Data** . . .
.		.
.		.
Subpacket N	**Subpacket header**	**Id** **SEu16, 2 bytes**
		Length **SEu16, 2 bytes**
	Subpacket data	**Data** . .

APPENDIX B

PILOT EYE FLIGHT DECK APPLICATION SOURCE CODE

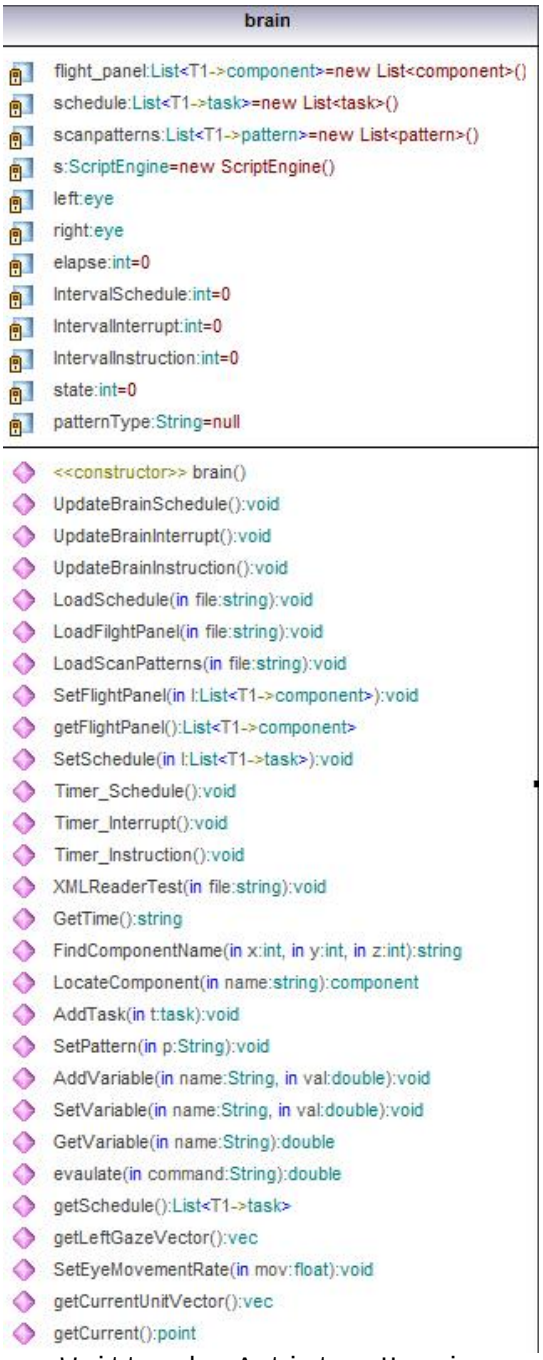
2DFLIGHTPANEL.CS




```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//
//Visualization 2D Flight Panel for viewing eye movement.
//

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Timers;
using System.Threading;
namespace Eyetrack{
    class _2DFlightPanel : System.Windows.Forms.Form{
        private System.ComponentModel.Container components = null;
        private Image theImage;
        private int x = 0;
        private int y = 0;
        public _2DFlightPanel() { InitializeComponent();
            SetStyle(ControlStyles.Opaque, true);
            theImage = new Bitmap("_2Dflightpanel.png");}
        protected override void OnPaint(PaintEventArgs e){
            Pen P = new Pen(Color.Yellow);
            Graphics g = e.Graphics;
            g.DrawImage(theImage, ClientRectangle);
            e.Graphics.DrawLine(P, x, 0, x, 778);
            e.Graphics.DrawLine(P, 0, y, 1197, y); Invalidate();}
```

```
        public void SetX(int x){ Thread.Sleep(100); this.x = x;}
        public void SetY( int y){ Thread.Sleep(100); this.y = y;}
        private void InitializeComponent() {
            this.components = new System.ComponentModel.Container();
            this.Size = new System.Drawing.Size(1197,778);
            this.Text = "2D Flight Panel";}
    }

}
```

BRAIN.CS




```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//
// Reads XML, Interrupt, Script Engine and State Machine
//

using System;
```

```
using System.Xml;
using System.Drawing;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Threading;
namespace Eyetrack{
public class brain{
      List<component>  flight_panel = new List<component>();
      List<task>  schedule = new List<task>();
        List<pattern> scanpatterns = new List<pattern>();
        ScriptEngine s = new ScriptEngine();
        Eyetrack.eye left; Eyetrack.eye right;
            int elapse = 0;
        int IntervalSchedule = 0;
        int IntervalInterrupt = 0;
        int IntervalInstruction = 0;
        int state = 0;
        String patternType = null;
      public brain(){ left = new eye(); right = new eye();}
      public void UpdateBrainSchedule(){
      while (true){ Thread.Sleep(IntervalSchedule); Timer_Schedule();}
      }
      public void UpdateBrainInterrupt(){
      while (true){Thread.Sleep(IntervalInterrupt);Timer_Interrupt();}
      }
      public void UpdateBrainInstruction(){while (true){
                Thread.Sleep(IntervalInstruction);
                Timer_Instruction();}
      }
      public void LoadSchedule( string file ){
      XmlTextReader reader = new XmlTextReader( file );
      task item = new task();
                while ( reader.Read() ) {
                        if( reader.NodeType == XmlNodeType.EndElement
){
                              if( string.Equals( reader.Name, "task" )

){
                                    schedule.Add( item );
                              }
                        }
                if (reader.NodeType == XmlNodeType.EndElement)
continue;
                        switch( reader.Name ){
                        case "task":
                              item = new task();
                              while ( reader.MoveToNextAttribute() ){
                                    if( string.Equals( reader.Name,
"duration") {
                                          item.duration = reader.Value;
                                    }
                              }
                              break;
                        case "x":
                              reader.Read();
                        item.location.x = System.Convert.ToSingle(
reader.Value );
                              break;
```

```
                        case "y":
                              reader.Read();
                        item.location.y = System.Convert.ToSingle(
reader.Value );
                              break;
                        case "z":
                              reader.Read();
                        item.location.z = System.Convert.ToSingle(
reader.Value );
                              break;
                        case "instrument":
                              reader.Read();
                              item.instrument = reader.Value;
                              break;
                        case "idno":
                              reader.Read();
                              item.idno = reader.Value;
                              break;
                        }
                  }
            }
            public void LoadFilghtPanel( string file ){
                  XmlTextReader reader = new XmlTextReader( file );/
                  component item = new component();
                  bool cont = true;
                  bool skip = false;
            while (skip || reader.Read() ) {
                        skip = false;
                        if( reader.NodeType == XmlNodeType.EndElement
){
                              if( string.Equals( reader.Name,
"component" ) )
                              {
                                    flight_panel.Add( item );
                              }
                        }
                        if( reader.NodeType == XmlNodeType.EndElement )
continue;
                              switch( reader.Name ){
                              case "component":
                                    item = new component();
                                    while (
reader.MoveToNextAttribute() ){
                                          if( string.Equals(
reader.Name, "IDNO") ){
                                                item.IDNO =
reader.Value;
                                          }
                                    }
                                    break;
                              case "name":
                                    reader.Read();
                                    item.name = reader.Value;
                                    break;
                              case "topleft":
                                    cont = true;
                                    while( cont && reader.Read() ){
```

```
                                        if( reader.NodeType ==
XmlNodeType.EndElement ) continue;
                                        reader.Read();
                                        switch( reader.Name ){
                                        case "x":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                              item.topleft.x =
System.Convert.ToSingle( reader.Value.Trim() );
                                              break;
                                        case "y":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                              item.topleft.y =
System.Convert.ToSingle( reader.Value.Trim() );
                                              break;
                                        case "z":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                              item.topleft.z =
System.Convert.ToSingle( reader.Value.Trim() );
                                              break;
                                        default:
                                              skip = true;
                                              cont = false;
                                              break;
                                        }
                                  }
                                  break;
                            case "topright":
                                  cont = true;
                                  while( cont && reader.Read() ){
                                        if( reader.NodeType ==
XmlNodeType.EndElement ) continue;
                                        reader.Read();
                                        switch( reader.Name ){
                                        case "x":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                              item.topright.x =
System.Convert.ToSingle( reader.Value.Trim() );
                                              break;
                                        case "y":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                              item.topright.y =
System.Convert.ToSingle( reader.Value.Trim() );
                                              break;
                                        case "z":
                                              reader.Read();
                                              if( reader.NodeType !=
XmlNodeType.Text ) continue;
```

```
                                                item.topright.z =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                          default:
                                                skip = true;
                                                cont = false;
                                                break;
                                          }
                                    }
                                    break;
                              case "bottomleft":
                                    cont = true;
                                    while( cont && reader.Read() ){
                                          if( reader.NodeType ==
XmlNodeType.EndElement ) continue;
                                          reader.Read();
                                          switch( reader.Name ){
                                          case "x":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                                item.bottomleft.x =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                          case "y":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                                item.bottomleft.y =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                          case "z":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                                item.bottomleft.z =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                          default:
                                                skip = true;
                                                cont = false;
                                                break;
                                          }
                                    }
                                    break;
                              case "bottomright":
                                    cont = true;
                                    while( cont && reader.Read() ){
                                          if( reader.NodeType ==
XmlNodeType.EndElement ) continue;
                                          reader.Read();
                                          switch( reader.Name ){
                                          case "x":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
```

```
                                                item.bottomright.x =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                            case "y":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                                item.bottomright.y =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                            case "z":
                                                reader.Read();
                                                if( reader.NodeType !=
XmlNodeType.Text ) continue;
                                                item.bottomright.z =
System.Convert.ToSingle( reader.Value.Trim() );
                                                break;
                                            default:
                                                skip = true;
                                                cont = false;
                                                break;
                                            }
                                        }
                                        break;
                                    case "elementtype":
                                        reader.Read();
                                        item.elementtype = reader.Value;
                                        break;
                                    case "threshold":
                                        reader.Read();
                                        item.theshold =
System.Convert.ToSingle( reader.Value.Trim() );
                                        break;
                                    case "duration":
                                        reader.Read();
                                        item.duration =
System.Convert.ToSingle( reader.Value.Trim() );
                                        break;
                                    }
                            }
                            }
                    public void LoadScanPatterns(string file){
                        XmlTextReader reader = new XmlTextReader(file);//
                        pattern item = new pattern();
                        instrument item2 = new instrument();

                        while (reader.Read()){
                            if (reader.NodeType == XmlNodeType.EndElement){
                                if (string.Equals(reader.Name, "pattern")){
                                    scanpatterns.Add(item);
                                }
                                else if (string.Equals(reader.Name, "state"))
                                {
                                    item.instr.Add(item2);
                                }
                            }
```

```
                if (reader.NodeType == XmlNodeType.EndElement)
continue;
                switch (reader.Name){
                    case "pattern":
                        item = new pattern();
                        while (reader.MoveToNextAttribute()){
                            if (string.Equals(reader.Name, "name"))
                            {
                                item.name = reader.Value;
                            }
                        }
                        break;
                    case "state":
                        item2 = new instrument();
                        while (reader.MoveToNextAttribute()){
                            if (string.Equals(reader.Name, "name")){
                                item2.name = reader.Value;
                            }
                        }
                        reader.Read();
                        item2.command = reader.Value;
                        break;
                }
            }
        }
            public void SetFlightPanel( List<component> l ){
                  this.flight_panel = l;
            }
            public List<component> getFlightPanel(){
                  return this.flight_panel;
            }
            public void SetSchedule( List<task> l ){
                  this.schedule = l;
            }

            public void Timer_Schedule()
                        if( schedule.Count == 0 ) return;
                    IntervalSchedule =
(int)(System.Convert.ToSingle(schedule[0].duration));
                        Console.WriteLine( GetTime() );
                        Console.WriteLine("Duration = " +
schedule[0].duration );
                        Console.WriteLine("Instrument = " +
schedule[0].instrument );
                        Console.WriteLine("IDNO = " + schedule[0].idno
);

                left.setLookingAt( new point(schedule[0].location.x,
schedule[0].location.y, schedule[0].location.z) );
                        right.setLookingAt( new
point(schedule[0].location.x, schedule[0].location.y,
schedule[0].location.z) );

                Console.WriteLine("X: " +
this.getLeftGazeVector().getUnitValue().getX() + " y: " +
this.getLeftGazeVector().getUnitValue().getY() + " z: " +
this.getLeftGazeVector().getUnitValue().getY());
```

```
                        schedule.Add( schedule[0] );
                        schedule.RemoveAt(0);
                        Console.WriteLine(" ");
                }
            public void Timer_Interrupt()
                if (flight_panel.Count == 0) return;
                DateTime startTime = DateTime.Now;
                int timeOffset = startTime.Millisecond - this.elapse;
                if (timeOffset < 0) timeOffset = this.elapse;
                for (int i = 0; i < flight_panel.Count; i++){
                    float val = flight_panel[i].getUncertainty() +
timeOffset + 1;
                    flight_panel[i].setUncertainty(val);
                    if (val > flight_panel[i].getThreshold()){
                        task item = new task();
                        item.instrument = flight_panel[i].name;
                        item.idno = flight_panel[i].IDNO;
                        item.location = flight_panel[i].getTopleft();
                        item.duration =
flight_panel[i].duration.ToString();
                        flight_panel[i].setUncertainty(0);
                        IntervalInterrupt =
(int)(System.Convert.ToSingle(item.duration));
                        Console.WriteLine("Interruption");
                        Console.WriteLine(GetTime());
                        Console.WriteLine("Duration = " + item.duration);
                        Console.WriteLine("Instrument = " +
item.instrument);
                        Console.WriteLine("IDNO = " + item.idno);
                        left.setLookingAt(new point(item.location.x,
item.location.y, item.location.z));
                        right.setLookingAt(new point(item.location.x,
item.location.y, item.location.z));
                        Console.WriteLine("X: " +
this.getLeftGazeVector().getUnitValue().getX() + " y: " +
this.getLeftGazeVector().getUnitValue().getY() + " z: " +
this.getLeftGazeVector().getUnitValue().getY());
                        Console.WriteLine("");
                    }
                }
                this.elapse = startTime.Millisecond;
            }
            public void Timer_Instruction(){for (int i = 0; i <
scanpatterns.Count; i++){
                    if (scanpatterns[i].name == patternType){
                        s.removeSpecialChar( ref
scanpatterns[i].instr[state].command );
                        if
(s.evaulate(scanpatterns[i].instr[state].command) > 0.0){
                            if ( scanpatterns[i].instr[state].name ==
"REPEAT" ){
                                state = 0;
                                return;
                            }
                            component c = LocateComponent(
scanpatterns[i].instr[state].name );
                            point p = c.getBottomleft();
```

```
                        Random random = new Random();
                        if (c.duration == 0.0f)
                            c.duration = 500;
                        do{
                            IntervalInstruction = (int)c.duration -
random.Next(-500, 500);
                        }
                        while (IntervalInstruction <= 0 ||
IntervalInstruction < 500) ;
                        Console.WriteLine(GetTime());
                        Console.WriteLine("Instrument = " +
scanpatterns[i].instr[state].name);
                        left.setLookingAt(new point(p.x, p.y, p.z));
                        Console.WriteLine("X: " +
this.getLeftGazeVector().getUnitValue().getX() + " y: " +
this.getLeftGazeVector().getUnitValue().getY() + " z: " +
this.getLeftGazeVector().getUnitValue().getY());
                        Console.WriteLine(" ");
                        state++;
                        if (state >= scanpatterns[i].instr.Count){
                            state = 0;
                            return;
                        }
                    }
                }
            }
        }
        public void XMLReaderTest(string file){
            XmlTextReader reader = new XmlTextReader("test.xml");
            while (reader.Read()){
                switch (reader.NodeType){
                    case XmlNodeType.Element:
                        Console.Write("<" + reader.Name);
                        while (reader.MoveToNextAttribute())
                            Console.Write(" " + reader.Name + "='" +
reader.Value + "'");
                        Console.WriteLine(">");
                        break;
                    case XmlNodeType.Text:
                        Console.WriteLine(reader.Value);
                        break;
                    case XmlNodeType.EndElement:
                        Console.Write("</" + reader.Name);
                        Console.WriteLine(">");
                        break;
                }
            }
            Console.ReadLine();
        }
            public string GetTime(){
                  string TimeInString="";
                  int hour=DateTime.Now.Hour;
                  int min=DateTime.Now.Minute;
                  int sec=DateTime.Now.Second;
                  TimeInString=(hour < 10)?"0" + hour.ToString()
:hour.ToString();
```

```
                  TimeInString+=":" + ((min<10)?"0" + min.ToString()
:min.ToString());
                  TimeInString+=":" + ((sec<10)?"0" + sec.ToString()
:sec.ToString());
                  return TimeInString;
            }
            public string FindComponentName(int x, int y, int z){
                  for( int i = 0; i < flight_panel.Count; i++ ){
                  }
                  return "panel";
            }
            public component LocateComponent( string name ){
                  for( int i = 0; i < flight_panel.Count; i++ ){
                        if( flight_panel[i].getName().Equals( name ) )
                              return flight_panel[i];
                  }
                  return null;
            }
            public void AddTask( task t ){
            IntervalInterrupt =
(int)(System.Convert.ToInt16(t.duration));
                  Console.WriteLine( GetTime() );
                  Console.WriteLine( "New task added: " + t.instrument
);
                  left.setLookingAt( new point(t.location.x,
t.location.y, t.location.z) );
                  right.setLookingAt( new point(t.location.x,
t.location.y, t.location.z) );
                  schedule.Add( t );
                  Console.WriteLine( " " );
            }
        public void SetPattern(String p){
            patternType = p;
        }
        public void AddVariable(String name, double val){
            s.AddVariable(name, val);
        }
        public void SetVariable(String name, double val){
            s.SetVariable(name, val);
        }
        public double GetVariable(String name){
            return s.GetVariable(name);
        }
        public double evaulate(String command){
            return s.evaulate(command);
        }
        public List<task> getSchedule(){
            return this.schedule;
        }
            public vec getLeftGazeVector(){
                  return left.getUnitVector();
            }
        public void SetEyeMovementRate( float mov ){
            left.setEyeMovementRate(mov);
            right.setEyeMovementRate(mov);
        }
        public vec getCurrentUnitVector(){
```

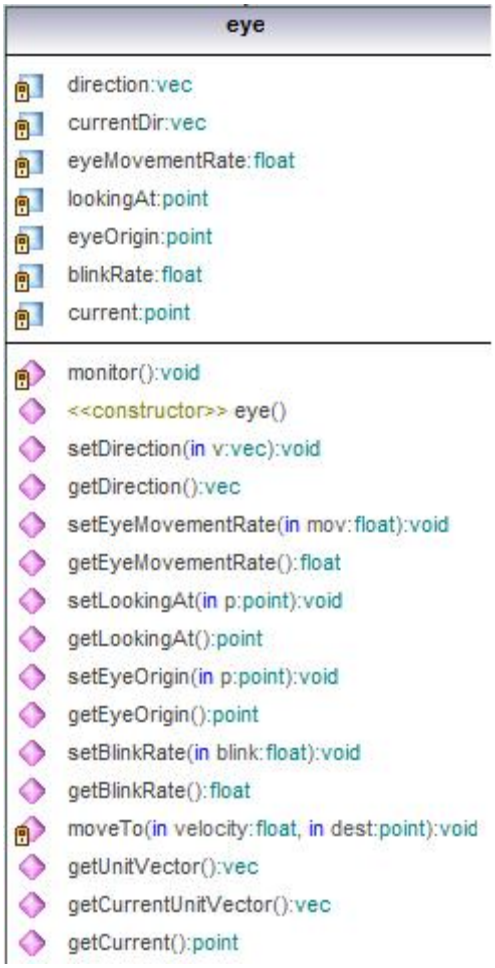
```
            return left.getCurrentUnitVector();
        }
        public point getCurrent()
        {
            return left.getCurrent();
        }
    }

}
```

COMPONENT.CS


```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//
// Description of instrument locations
//
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Eyetrack{
      public class component{
          public string IDNO;
            public string name;
            public point topleft = new point();
            public point topright = new point();
            public point bottomleft = new point();
            public point bottomright = new point();
            public string elementtype;
            public float uncertainty = 0;
            public float theshold = 0;
            public float duration = 0;
            public void setIDNO(string v){ this.IDNO = v;}
            public string getIDNO(){return IDNO;}
            public void setName( string s ){this.name = s;}
            public string getName(){return this.name;}
            public void setTopleft( point p ){this.topleft = p;}
            public point getTopleft(){return this.topleft;}
            public void setTopright( point p ){this.topright = p;}
            public point getTopright(){return this.topright;}
            public void setBottomleft( point p ){this.bottomleft = p;}
            public point getBottomleft(){return this.bottomleft;}
      public void setBottomright( point p ){this.bottomright = p;}
            public point getBottomright(){return this.bottomright;}
      public void setElementtype( string s ){this.elementtype = s;}
            public string getElementtype(){return this.elementtype;}
      public void setUncertainty( float val ){this.uncertainty = val;}
      public float getUncertainty( ){return this.uncertainty;}
      public void setThreshold( float val ){this.theshold = val;}
            public float getThreshold( ){return this.theshold;}
      }

}
```

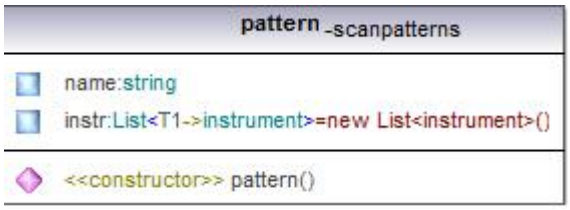
EYE.CS




```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//
// Controls and computes look movement
//

using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Eyetrack
```

```
{
    public class eye
    {
        private Eyetrack.vec direction;
        private Eyetrack.vec currentDir;
        private float eyeMovementRate;
        private point lookingAt;
        private point eyeOrigin;
        private float blinkRate;
        private point current;
        void monitor(){
            while (true){
                if (current.x != lookingAt.x
                    || current.y != lookingAt.y
                    || current.z != lookingAt.z){
                    double angle = Math.Atan2( lookingAt.y - current.y
, lookingAt.x - current.x );

                    current.x = current.x + eyeMovementRate *
(float)Math.Cos(angle);
                    current.y = current.y + eyeMovementRate *
(float)Math.Sin(angle);

                    this.currentDir.setDest(current);
                }
            }
        }
            public eye(){
            this.current = new point(0, 0, 0);
                  this.lookingAt = new point(0,0,0);
                  this.eyeOrigin = new point(0,0,0);
                  this.direction = new
vec(this.eyeOrigin,this.lookingAt);
            this.currentDir = new vec(this.eyeOrigin, this.lookingAt);
            new Thread(this.monitor).Start();
            }
        public void setDirection(Eyetrack.vec v){this.direction = v;}
        public Eyetrack.vec getDirection(){return direction;}
public void setEyeMovementRate(float mov){this.eyeMovementRate = mov;}
        public float getEyeMovementRate(){return eyeMovementRate;}
        public void setLookingAt(point p){this.direction.setDest(p);
this.lookingAt = p;}
        public point getLookingAt(){return lookingAt;}
        public void setEyeOrigin(point p){this.eyeOrigin = p;}
        public point getEyeOrigin(){return eyeOrigin;}
        public void setBlinkRate(float blink){this.blinkRate = blink;}
        public float getBlinkRate(){
        return blinkRate;}
        void moveTo(float velocity, point dest){
        this.setLookingAt(dest);this.eyeMovementRate = velocity;
        this.direction.moveStepDest(velocity);  }
      public vec getUnitVector(){return direction.getUnitVector();}
        public vec getCurrentUnitVector(){
return currentDir.getUnitVector();}
        public point getCurrent(){return current;}
```

PATTERN.CS




```
//Eye Tracking Program Written by Artistee Harris
Reference: Microsoft Framework .NET
//
// Description node for state machine elements
//

using System;using System.Collections.Generic;
using System.Text;
namespace Eyetrack{
    class instrument{
        public String name;
        public String command;
    }
      class pattern{
            public string name;
        public List<instrument> instr = new List<instrument>();

        public pattern(){
            }
      }

      }
```

PERSON.CS

```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
// Passes data to the brain, eye script engine, xml reader and state
//machine
//
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using System.Threading;
namespace Eyetrack{
    class Person{
        private point headPosition;
        private point headPositionQuality;
        private Eyetrack.vec noseVector;
        private Eyetrack.vec headVector;
        private Eyetrack.vec earVector;
        private float headAzimuth;
        private float headPitch;
        private float headRoll;
        private float spacingEyes;
        private point lookAt;
        private point whatItsLooking;
        private point leftGazeOrigin;
        private point rightGazeOrigin;
        private int leftGazeOriginQuality;
        private int rightGazeOriginQuality;
        private Eyetrack.vec leftGazeVectorFiltered;
        private Eyetrack.vec rightGazeVectorFiltered;
        private int leftGazeQuality;
        private int rightGazeQuality;
            private brain mybrain;
        _2DFlightPanel fp;

            public Person(){
                  mybrain = new brain();
            }
        public void loadFlightPanelInterruption(string s){
            mybrain.LoadFilghtPanel(s);
        }
        public void learnSchedule(string s){
            mybrain.LoadSchedule(s);
        }
        public void loadScanPatterns(string s){
            mybrain.LoadScanPatterns(s);
        }
        public void StartFlighting(){
            new Thread(mybrain.UpdateBrainInstruction).Start();
        }
        public void addTask(task t){
            mybrain.AddTask(t);
        }
        public void AddVariable(string name, double val){
            mybrain.AddVariable(name, val);
        }
        public void SetVariable(string name, double val){
            mybrain.SetVariable(name, val);
        }
```

```
public double GetVariable(string name){
    return mybrain.GetVariable(name);
}
public double evaulate(string command){
    return mybrain.evaulate(command);
}
public void SetPattern(string p){
    mybrain.SetPattern( p );
}
public void SetEyeMovementRate(float mov){
    this.mybrain.SetEyeMovementRate(mov);
}
public vec getEyeUnitVector(){
    return this.mybrain.getCurrentUnitVector();
}
public void setHeadPosition( point p){
   this.headPosition = p;
}
public point getHeadPosition(){
    return headPosition;
}
public void setHeadPositionQuality(point p){
    this.headPositionQuality = p;
}
public point getHeadPositonQuality(){
    return headPositionQuality;
}
public void setNoseVector(Eyetrack.vec v){
    this.noseVector = v;
}
public Eyetrack.vec getNoseVector(){
    return noseVector;
}
public void setHeadVector(Eyetrack.vec v){
    this.headVector = v;
}
public Eyetrack.vec getHeadVector(){
    return headVector;
}
public void setEarVector( Eyetrack.vec v){
    this.earVector = v;
}
public Eyetrack.vec getEarVector(){
    return earVector;
}
public void setHeadAzimuth(float num){
    this.headAzimuth = num;
}
public float getHeadAzimuth(){
    return headAzimuth;
}
public void setHeadPitch(float num){
    this.headAzimuth = num;
}
public float getHeadPitch(){
    return headPitch;
}
```

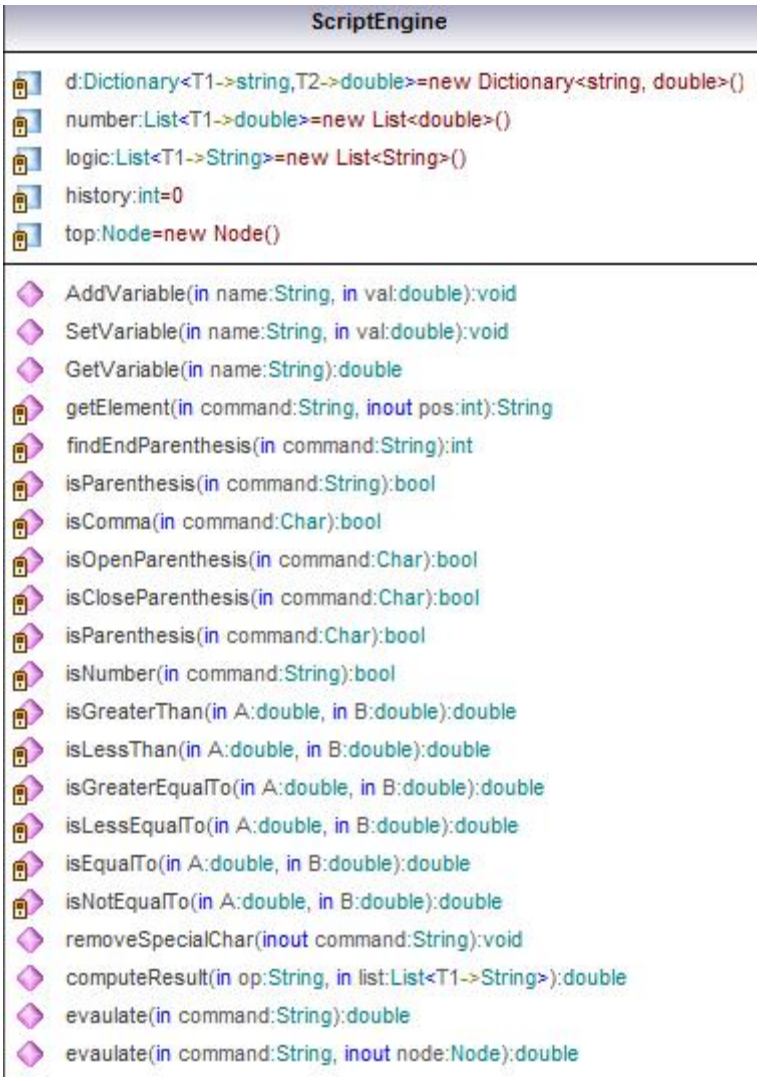
```
public void setHeadRoll(float num){
    this.headPitch = num;
}
public float getHeadRoll(){
    return headRoll;
}
public void setSpaceEyes(float spacing){
    this.spacingEyes = spacing;
}
public float getSpaceEyes(){
    return spacingEyes;
}
public void setLookAt( point p){
    this.lookAt = p;
}
public point getLookAt(){
    return lookAt;
}
public void setWhatItsLooking(point p){
    this.lookAt = p;
}
public point getWhatItsLooking(){
        return whatItsLooking;
}
public void setLeftGazeOrigin(point p){
    this.leftGazeOrigin= p;
}
public point getLeftGazeOrigin()
{
    return leftGazeOrigin;
}
public void setRightGazeOrigin(point p){
    this.rightGazeOrigin= p;
}
public point getRightGazeOrigin(){
    return rightGazeOrigin;
}
public void setLeftGazeOriginQuality(int numb){
    this.leftGazeOriginQuality = numb;
}
public int getLeftGazeOriginQuality(){
    return rightGazeOriginQuality;
}
public void setRightGazeOriginQuality(int numb){
    this.rightGazeOriginQuality = numb;
}
public int getRightGazeOriginQuality(){
    return rightGazeOriginQuality;
}
public void setLeftGazeVectorFiltered(Eyetrack.vec v){
    this.leftGazeVectorFiltered = v;
}
public Eyetrack.vec getLeftGazeVectorFiltered(){
    return leftGazeVectorFiltered;
}
public void setLeftGazeQuality(int numb){
    this.leftGazeQuality = numb;
```

```
        }
        public int getLeftGazeQuality(){
            return leftGazeQuality;
        }
        public void setRightGazeQuality(int numb){
            this.rightGazeQuality = numb;
        }
        public int getRightGazeQuality(){
            return rightGazeOriginQuality;
        }
        public void Run2DFlightPanel(){
            fp = new _2DFlightPanel();
            Application.Run(fp);
        }
        public void SetFlightPanelX(int x){
            fp.SetX(x);
        }
        public void SetFlightPanelY(int y){
            fp.SetY(y);
        }
        public point getCurrent(){
            return mybrain.getCurrent();
        }

    }
}
```

POINT.CS

```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET

//Holds information about point

using System;
using System.Collections.Generic;
using System.Text;
namespace Eyetrack{
    public class point{
        public float x, y, z;
            public point(){
                  x = 0;
                  y = 0;
                  z = 0;
            }
        public void setX(float x){this.x = x;}
        public float getX(){return x;}
        public void setY(float y){this.y = y;}
        public float getY(){return y;}
        public void setZ(float z){this.z = z;}
        public float getZ(){return z;}
        public point(float x, float y, float z){this.x = x;
            this.y = y;this.z = z;}
        public static point operator +(point p){return p;}
        public static point operator -(point p){return p;}
        public static point operator /(point p1, point p2){
            return p1;}
        public static point operator *(point p1, point p2){
            return p1;}
        public static point operator +(point p1, point p2){
                 return p1;}
        public static point operator -(point p1, point p2){
            return p1;}
    }
}
```

SCRIPTENGINE.CS




```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//Processes script engine to be sent to the state machine

using System;
using System.Collections.Generic;
using System.Text;
namespace Eyetrack{
    class Node{
```

```
        public String command;
        public String opcode;
        public List<Node> child;

    }
    class ScriptEngine{
        Dictionary<string, double> d = new Dictionary<string,
double>();

        List<double> number = new List<double>();
        List<String> logic = new List<String>();
        int history = 0;
        Node top = new Node();
        public void AddVariable( String name, double val ){
            d.Add(name, val);
        }
        public void SetVariable(String name, double val){
            if (d.ContainsKey(name)) // True{
                d[name] = val;
            }
        }
        public double GetVariable(String name){
            if (d.ContainsKey(name)) // True{
                return d[name];
            }
            return double.NaN;
        }
        String getElement(String command, ref int pos){
            int i = 0;
            for ( i = 0; i < command.Length; i++){
                if (command[i].Equals(',') || command[i].Equals(')')){
                    pos = i;
                    return command.Substring(0, i);
                }
            }
            pos = i;
            return command;
        }
        int findEndParenthesis(String command)
        {
            int counter = 0;
            if (!command[0].Equals('(')){
                return -1;
            }
            for (int i = 0; i < command.Length; i++){
                if (command[i].Equals('(')){
                    counter++;
                }
                else if (command[i].Equals(')')){
                    counter--;
                }
                if (counter == 0){
                    return i;
                }
            }
            return -1;
        }
```

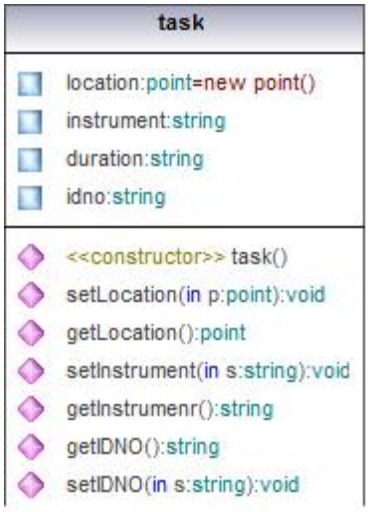
```
        bool isParenthesis(String command){
            return command.Equals("(") || command.Equals(")");
        }
        bool isComma(Char command){
            return command.Equals(',');
        }
        bool isOpenParenthesis(Char command)//new{
            return command.Equals('(');
        }
        bool isCloseParenthesis(Char command)//new{
            return command.Equals(')');
        }
        bool isParenthesis(Char command){
            return command.Equals('(') || command.Equals(')');
        }
        bool isNumber(String command){
            if (command[0].Equals('-')){
                command = command.Remove(0, 1);
            }
            for (int i = 0; i < command.Length; i++){
                if ( !char.IsDigit(command, i) ){
                    return false;
                }
            }
            return true;
        }
        double isGreaterThan(double A, double B){
            return A > B ? 1.0 : 0.0;
        }
        double isLessThan(double A, double B){
            return A < B ? 1.0 : 0.0;
        }
        double isGreaterEqualTo(double A, double B){
            return A >= B ? 1.0 : 0.0;
        }
        double isLessEqualTo(double A, double B){
            return A <= B ? 1.0 : 0.0;
        }
        double isEqualTo(double A, double B){
            return A == B ? 1.0 : 0.0;
        }
        double isNotEqualTo(double A, double B){
            return A != B ? 1.0 : 0.0;
        }
        public void removeSpecialChar(ref String command){
            for(int i =0; i < command.Length; i++){
                if (command[i].Equals(' ') || command[i].Equals('\t')
|| command[i].Equals('\n') || command[i].Equals('\r')){
                    command = command.Remove(i, 1);
                    i--;
                }
            }
        }
        public double computeResult(String op, List<String> list){
            double [] num = new double[2];
            double result = 0;
            for (int i = 0; i < list.ToArray().Length; i++){
```

```
                if (isNumber(list[i])){
                    num[i] = Convert.ToDouble(list[i]);
                }
                else{
                    num[i] = GetVariable(list[i]);
                }
            }

            switch (op){
                case ">":
                    result = num[0] > num[1] ? 1.0 : 0.0;
                    break;
                case "<":
                    result = num[0] < num[1] ? 1.0 : 0.0;
                    break;
                case ">=":
                    result = num[0] >= num[1] ? 1.0 : 0.0;
                    break;
                case "<=":
                    result = num[0] <= num[1] ? 1.0 : 0.0;
                    break;
                case "==":
                    result = num[0] == num[1] ? 1.0 : 0.0;
                    break;
                case "=":
                    result = num[0];
                    break;
                case "/":
                    result = num[0] / num[1];
                    break;
                case "*":
                    result = num[0] * num[1];
                    break;
                case "+":
                    result = num[0] + num[1];
                    break;
                case "-":
                    result = num[0] - num[1];
                    break;
                case "and":
                    result = (num[0] > 0 && num[1] > 0) == true ? 1 :
0;
                    break;
                case "or":
                    result = (num[0] > 0 || num[1] > 0) == true ? 1 :
0;
                    break;
                case "not":
                    result = (num[0]) >  0 ? 0 : 1;
                    break;
                default:
                    result = double.NaN;
                    break;
            }
            return result;
        }
        public double evaulate(String command){
```

```
            command = command.Substring(1, findEndParenthesis(command)
- 1);
            top.child = new List<Node>();
            int pos = 0;
            String temp;
            temp = command;
            top.command = command;
            top.opcode = getElement(top.command, ref pos);
            temp = top.command.Substring(pos);
            temp = temp.Remove(0, 1);
            List<String> number = new List<String>();
            while (temp.Length > 0){
                if (findEndParenthesis(temp) == -1){
                    Node child = new Node();
                    child.command = getElement(temp, ref pos);
                    top.child.Add(child);
                    temp = temp.Substring(pos);
                    //add number to list
                    number.Add(child.command);
                    if (temp.Length == 0){
                        return computeResult(top.opcode, number);
                    }
                    temp = temp.Remove(0, 1);
                }
                else{
                    Node child = new Node();
                    top.child.Add(child);
                    double result = evaulate(temp, ref child);
                    //add number to list
                    number.Add(result.ToString());
                    temp = temp.Substring(findEndParenthesis(temp) +
1);
                    if (temp.Length == 0){
                        return computeResult(top.opcode, number);
                    }
                    temp = temp.Remove(0, 1);
                }
            }
            return computeResult(top.opcode, number);
        }
        public double evaulate(String command, ref Node node){
            command = command.Substring(1, findEndParenthesis(command)
- 1);
            node.child = new List<Node>();
            int pos = 0;
            String temp;
            temp = command;
            node.command = command;
            node.opcode = getElement(node.command, ref pos);
            temp = node.command.Substring(pos);
            temp = temp.Remove(0, 1);
            List<String> number = new List<String>();
            while ( temp.Length > 0 ){
                if (findEndParenthesis(temp) == -1){
                    Node child = new Node();
                    child.command = getElement(temp, ref pos);
                    node.child.Add(child);
```

```
                    temp = temp.Substring(pos);
                    //add number to list
                    number.Add(child.command);
                    if (temp.Length == 0){
                        return computeResult(node.opcode, number);
                    }
                    temp = temp.Remove(0, 1);
                }else{
                    Node child = new Node();
                    node.child.Add(child);
                    double result = evaulate(temp, ref child);
                    //add number to list
                    number.Add(result.ToString());
                    temp = temp.Substring(findEndParenthesis(temp) +
1);
                    if (temp.Length == 0){
                        return computeResult(node.opcode, number);
                    }

                    temp = temp.Remove(0, 1);
                }
            }
            return computeResult(top.opcode, number);

        }
    }

      }
```

TASK.CS




```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//Hold information about interrupts
```

```
using System;
namespace Eyetrack{
      public class task{
            public point location = new point();
            public string instrument;
            public string duration;
            public string idno;
            public task(){
            }
            public void setLocation( point p ){
                  this.location = p;
            }
            public point getLocation(){
                  return this.location;
            }
            public void setInstrument( string s ){
                  this.instrument = s;
            }
            public string getInstrumenr(){
                  return this.instrument;
            }
            public string getIDNO()
{
                  return this.idno;
            }
            public void setIDNO( string s ){
                  this.idno = s;
            }
      }

}
```

UDP.CS


```
      //Eye Tracking Program Written by Artistee Harris
      //Reference: Microsoft Framework .NET
      //Sends Smart Eye UDP packets

      using System;
      using System.Net;
      using System.Text;
      using System.Net.Sockets;
      using System.Collections.Generic;
      namespace Eyetrack{
         class ByteSwap{
             public static UInt16 swap(UInt16 input){
                 return ((UInt16)(((0xFF00 & input) >> 8) |
                 ((0x00FF & input) << 8)));}
```

```
            public static UInt32 swap(UInt32 input){
                return ((UInt32)(
                ((0xFF000000 & input) >> 24) |
                ((0x00FF0000 & input) >> 8) |
                ((0x0000FF00 & input) << 8) |
                ((0x000000FF & input) << 24)));}
            public static float swap(float input){
                byte[] tmpIn = BitConverter.GetBytes(input);
                byte[] tmpOut = new byte[4];
                tmpOut[0] = tmpIn[3];
                tmpOut[1] = tmpIn[2];
                tmpOut[2] = tmpIn[1];
                tmpOut[3] = tmpIn[0];
                return BitConverter.ToSingle(tmpOut, 0);}
            public static double swap(double input){
                byte[] tmpIn = BitConverter.GetBytes(input);
                byte[] tmpOut = new byte[8];
                tmpOut[0] = tmpIn[7];
                tmpOut[1] = tmpIn[6];
                tmpOut[2] = tmpIn[5];
                tmpOut[3] = tmpIn[4];
                tmpOut[4] = tmpIn[3];
                tmpOut[5] = tmpIn[2];
                tmpOut[6] = tmpIn[1];
                tmpOut[7] = tmpIn[0];
                return BitConverter.ToSingle(tmpOut, 0);}
            public static byte[] swapb(double input){
                byte[] tmpIn = BitConverter.GetBytes(input);
                byte[] tmpOut = new byte[8];
                tmpOut[0] = tmpIn[7];
                tmpOut[1] = tmpIn[6];
                tmpOut[2] = tmpIn[5];
                tmpOut[3] = tmpIn[4];
                tmpOut[4] = tmpIn[3];
                tmpOut[5] = tmpIn[2];
                tmpOut[6] = tmpIn[1];
                tmpOut[7] = tmpIn[0];
                return tmpOut;}}
        public class SEVector{
            public SEf64 x;
            public SEf64 y;
            public SEf64 z;
            public int GetSize(){
                return x.GetSize() + y.GetSize() + z.GetSize();}
            public byte[] GetByteArray(){
                byte[] nb = new byte[GetSize()];
                x.GetByteArray().CopyTo(nb, 0);
                y.GetByteArray().CopyTo(nb, x.GetSize());
                z.GetByteArray().CopyTo(nb, x.GetSize() +
    y.GetSize());
                return nb;}}
        public class SEu8{
            byte b = new byte();
            private SEu8(byte ba) { b = ba; }
            public SEu8() { }
            public static implicit operator SEu8(byte val){
                return(new SEu8(val));}
```

```
        public int GetSize(){
            return 1;
        }
        public byte[] GetByteArray(){
            byte[] ret = new byte[1];
            ret[0] = b;
            return ret;}}
   public class SEu16{byte[] b = new byte[2];
private SEu16(byte[] ba) { b = ba; }
        public SEu16() { }
        public static implicit operator SEu16(UInt16 val){
            byte[] ba = new byte[8];
            ba = BitConverter.GetBytes(ByteSwap.swap(val));
            return (new SEu16(ba));
        }
        public int GetSize(){
            return b.Length;
        }
        public byte[] GetByteArray(){
            return b;
        }
   }
   public class SEu32{
        byte[] b = new byte[4];
        private SEu32(byte[] ba) { b = ba; }
        public SEu32() { }
        public static implicit operator SEu32(UInt32 val){
            byte[] ba = new byte[8];
            ba = BitConverter.GetBytes(ByteSwap.swap(val));
            return (new SEu32(ba));
        }
        public int GetSize(){
            return b.Length;
        }
        public byte[] GetByteArray(){
            return b;
        }
   }
   public class SEs32{
        byte[] b = new byte[4];
        private SEs32(byte[] ba) { b = ba; }
        public SEs32() { }
        public static implicit operator SEs32(Int32 val){
            byte[] ba = new byte[8];
            ba = BitConverter.GetBytes(ByteSwap.swap(val));
            return (new SEs32(ba));
        }
        public int GetSize(){
            return b.Length;
        }
        public byte[] GetByteArray(){
            return b;
        }
   }
   public class SEu64{
        byte[] b = new byte[8];
        private SEu64(byte[] ba) { b = ba; }
```

```
    public SEu64() { }
    public static implicit operator SEu64(UInt64 val){
        byte[] ba = new byte[8];
        ba = BitConverter.GetBytes(ByteSwap.swap(val));
        return (new SEu64(ba));
    }
    public int GetSize(){
        return b.Length;
    }
    public byte[] GetByteArray(){
        return b;
    }
}
public class SEf64{
    byte[] b = new byte[8];
    private SEf64(byte[] ba) { b = ba; }
    public SEf64() { }
    public static implicit operator SEf64(double val)
    {
        byte[] ba = new byte[8];
        ba = ByteSwap.swapb(val);
        return (new SEf64(ba));
    }
    public int GetSize(){
        return b.Length;
    }
    public byte[] GetByteArray(){
        return b;
    }
}
public class SEPoint2D{
    public SEf64 x;
    public SEf64 y;
    public int GetSize(){
        return x.GetSize() + y.GetSize();
    }
    public byte[] GetByteArray(){
        byte[] nb = new byte[GetSize()];
        x.GetByteArray().CopyTo(nb,0);
        y.GetByteArray().CopyTo(nb,x.GetSize());
        return nb;
    }
}
public class SEVect2D{
    public SEf64 x;
    public SEf64 y;
    public int GetSize(){
        return x.GetSize() + y.GetSize();
    }
    public byte[] GetByteArray(){
        byte[] nb = new byte[GetSize()];
        x.GetByteArray().CopyTo(nb, 0);
        y.GetByteArray().CopyTo(nb, x.GetSize());
        return nb;
    }
}
public class SEPoint3D : SEVector{
```

```
        public SEf64 x;
        public SEf64 y;
        public SEf64 z;
        public int GetSize(){
            return x.GetSize() + y.GetSize() + z.GetSize();
        }
        public byte[] GetByteArray(){
            byte[] nb = new byte[GetSize()];
            x.GetByteArray().CopyTo(nb, 0);
            y.GetByteArray().CopyTo(nb, x.GetSize());
            z.GetByteArray().CopyTo(nb, x.GetSize() +
y.GetSize());
            return nb;
        }
    }
    public class SEVect3D : SEVector{
        public SEf64 x;
        public SEf64 y;
        public SEf64 z;
        public int GetSize()
        {
            return x.GetSize() + y.GetSize() + z.GetSize();
        }
        public byte[] GetByteArray(){
            byte[] nb = new byte[GetSize()];
            x.GetByteArray().CopyTo(nb, 0);
            y.GetByteArray().CopyTo(nb, x.GetSize());
            z.GetByteArray().CopyTo(nb, x.GetSize() +
y.GetSize());
            return nb;
        }
    }
    public class SEString{
        byte[] b = new byte[0];
        public SEString() { }
        public SEString(String val){
            b = new byte[val.Length+1];
            int i;
            for(i=0;i<val.Length;i++) b[i] =
Convert.ToByte(val[i]);
            b[i] = 0;
        }
        public int GetSize(){
            return b.Length;
        }
        public byte[] GetByteArray(){
            return b;
        }
    }
    public class UDP{
        private int _port = 80;
        private string _address = 80;
        public void SetPort(int val){
            _port = val;
        }
        public void SetAddress(string val){
            _address = val;
```

```
        }
        public void Send(){
            UdpClient sock = new UdpClient();
            IPEndPoint iep = new
IPEndPoint(IPAddress.Parse(_address), _port);
            List<byte[]> message = new List<byte[]>();
            List<byte> id = new List<byte>();
            if (SEFrameNumberSpecified){
                message.Add(SEFrameNumber.GetByteArray());
                id.Add((byte)Id.SEFrameNumber);
            }
            if (SEEstimatedDelaySpecified){
                message.Add(SEEstimatedDelay.GetByteArray());
                id.Add((byte)Id.SEEstimatedDelay);
            }
            if (SETimeStampSpecified){
                message.Add(SETimeStamp.GetByteArray());
                id.Add((byte)Id.SETimeStamp);
            }
            if (SEUserTimeStampSpecified){
                message.Add(SEUserTimeStamp.GetByteArray());
                id.Add((byte)Id.SEUserTimeStamp);
            }
            if (SEFrameRateSpecified){
                message.Add(SEFrameRate.GetByteArray());
                id.Add((byte)Id.SEFrameRate);
            }
            if (SEHeadPositionSpecified){
                message.Add(SEHeadPosition.GetByteArray());
                id.Add((byte)Id.SEHeadPosition);
            }
            if (SECameraPositionsSpecified){
                message.Add(SECameraPositions.GetByteArray());
                id.Add((byte)Id.SECameraPositions);
            }
            if (SECameraRotationsSpecified){
                message.Add(SECameraRotations.GetByteArray());
                id.Add((byte)Id.SECameraRotations);
            }
            if (SEHeadPositionQSpecified){
                message.Add(SEHeadPositionQ.GetByteArray());
                id.Add((byte)Id.SEHeadPositionQ);
            }
            if (SEHeadRotationRodriguesSpecified){

 message.Add(SEHeadRotationRodrigues.GetByteArray());
                id.Add((byte)Id.SEHeadRotationRodrigues);
            }
            if (SEHeadNoseDirectionSpecified){
                message.Add(SEHeadNoseDirection.GetByteArray());
                id.Add((byte)Id.SEHeadNoseDirection);
            }
            if (SEHeadUpDirectionSpecified){
                message.Add(SEHeadUpDirection.GetByteArray());
                id.Add((byte)Id.SEHeadUpDirection);
            }
            if (SEHeadLeftEarDirectionSpecified){
```

```
message.Add(SEHeadLeftEarDirection.GetByteArray());
                id.Add((byte)Id.SEHeadLeftEarDirection);
            }
            if (SEHeadHeadingSpecified){
                message.Add(SEHeadHeading.GetByteArray());
                id.Add((byte)Id.SEHeadHeading);
            }
            if (SEHeadPitchSpecified){
                message.Add(SEHeadPitch.GetByteArray());
                id.Add((byte)Id.SEHeadPitch);
            }
            if (SEHeadRollSpecified){
                message.Add(SEHeadRoll.GetByteArray());
                id.Add((byte)Id.SEHeadRoll);
            }
            if (SEHeadRotationQSpecified){
                message.Add(SEHeadRotationQ.GetByteArray());
                id.Add((byte)Id.SEHeadRotationQ);
            }
            if (SEEyePositionSpecified){
                message.Add(SEEyePosition.GetByteArray());
                id.Add((byte)Id.SEEyePosition);
            }
            if (SEGazeOriginSpecified){
                message.Add(SEGazeOrigin.GetByteArray());
                id.Add((byte)Id.SEGazeOrigin);
            }
            if (SELeftGazeOriginSpecified){
                message.Add(SELeftGazeOrigin.GetByteArray());
                id.Add((byte)Id.SELeftGazeOrigin);
            }
            if (SERightGazeOriginSpecified){
                message.Add(SERightGazeOrigin.GetByteArray());
                id.Add((byte)Id.SERightGazeOrigin);
            }
            if (SEGazeDirectionSpecified){
                message.Add(SEGazeDirection.GetByteArray());
                id.Add((byte)Id.SEGazeDirection);
            }
            if (SEGazeDirectionQSpecified){
                message.Add(SEGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEGazeDirectionQ);
            }
            if (SELeftEyePositionSpecified){
                message.Add(SELeftEyePosition.GetByteArray());
                id.Add((byte)Id.SELeftEyePosition);
            }
            if (SELeftGazeDirectionSpecified){
                message.Add(SELeftGazeDirection.GetByteArray());
                id.Add((byte)Id.SELeftGazeDirection);
            }
            if (SELeftGazeDirectionQSpecified){
                message.Add(SELeftGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SELeftGazeDirectionQ);
            }
            if (SERightEyePositionSpecified){
```

```
                message.Add(SERightEyePosition.GetByteArray());
                id.Add((byte)Id.SERightEyePosition);
            }
            if (SERightGazeDirectionSpecified){
                message.Add(SERightGazeDirection.GetByteArray());
                id.Add((byte)Id.SERightGazeDirection);
            }
            if (SERightGazeDirectionQSpecified){
                message.Add(SERightGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SERightGazeDirectionQ);
            }
            if (SEGazeHeadingSpecified){
                message.Add(SEGazeHeading.GetByteArray());
                id.Add((byte)Id.SEGazeHeading);
            }
            if (SEGazePitchSpecified){
                message.Add(SEGazePitch.GetByteArray());
                id.Add((byte)Id.SEGazePitch);
            }
            if (SEFilteredGazeDirectionSpecified){

message.Add(SEFilteredGazeDirection.GetByteArray());
                id.Add((byte)Id.SEFilteredGazeDirection);
            }
            if (SEFilteredGazeDirectionQSpecified){

message.Add(SEFilteredGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEFilteredGazeDirectionQ);
            }
            if (SEFilteredLeftGazeDirectionSpecified){

message.Add(SEFilteredLeftGazeDirection.GetByteArray());
                id.Add((byte)Id.SEFilteredLeftGazeDirection);
            }
            if (SEFilteredLeftGazeDirectionQSpecified){

message.Add(SEFilteredLeftGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEFilteredLeftGazeDirectionQ);
            }
            if (SEFilteredRightGazeDirectionSpecified){

message.Add(SEFilteredRightGazeDirection.GetByteArray());
                id.Add((byte)Id.SEFilteredRightGazeDirection);
            }
            if (SEFilteredRightGazeDirectionQSpecified){

message.Add(SEFilteredRightGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEFilteredRightGazeDirectionQ);
            }
            if (SEFilteredGazeHeadingSpecified){
                message.Add(SEFilteredGazeHeading.GetByteArray());
                id.Add((byte)Id.SEFilteredGazeHeading);
            }
            if (SEFilteredGazePitchSpecified){
                message.Add(SEFilteredGazePitch.GetByteArray());
                id.Add((byte)Id.SEFilteredGazePitch);
            }
```

```
if (SESaccadeSpecified){
    message.Add(SESaccade.GetByteArray());
    id.Add((byte)Id.SESaccade);
}
if (SEBlinkSpecified){
    message.Add(SEBlink.GetByteArray());
    id.Add((byte)Id.SEBlink);
}
if (SEEyelidOpeningSpecified){
    message.Add(SEEyelidOpening.GetByteArray());
    id.Add((byte)Id.SEEyelidOpening);
}
if (SEEyelidOpeningQSpecified){
    message.Add(SEEyelidOpeningQ.GetByteArray());
    id.Add((byte)Id.SEEyelidOpeningQ);
}
if (SELeftEyelidOpeningSpecified){
    message.Add(SELeftEyelidOpening.GetByteArray());
    id.Add((byte)Id.SELeftEyelidOpening);
}
if (SELeftEyelidOpeningQSpecified){
    message.Add(SELeftEyelidOpeningQ.GetByteArray());
    id.Add((byte)Id.SELeftEyelidOpeningQ);
}
if (SERightEyelidOpeningSpecified){
    message.Add(SERightEyelidOpening.GetByteArray());
    id.Add((byte)Id.SERightEyelidOpening);
}
if (SERightEyelidOpeningQSpecified){
    message.Add(SERightEyelidOpeningQ.GetByteArray());
    id.Add((byte)Id.SERightEyelidOpeningQ);
}
if (SEEstimatedGazeOriginSpecified){
    message.Add(SEEstimatedGazeOrigin.GetByteArray());
    id.Add((byte)Id.SEEstimatedGazeOrigin);
}
if (SEPupilDiameterSpecified){
    message.Add(SEPupilDiameter.GetByteArray());
    id.Add((byte)Id.SEPupilDiameter);
}
if (SEPupilDiameterQSpecified){
    message.Add(SEPupilDiameterQ.GetByteArray());
    id.Add((byte)Id.SEPupilDiameterQ);
}
if (SELeftPupilDiameterSpecified){
    message.Add(SELeftPupilDiameter.GetByteArray());
    id.Add((byte)Id.SELeftPupilDiameter);
}
if (SELeftPupilDiameterQSpecified){
    message.Add(SELeftPupilDiameterQ.GetByteArray());
    id.Add((byte)Id.SELeftPupilDiameterQ);
}
if (SERightPupilDiameterSpecified){
    message.Add(SERightPupilDiameter.GetByteArray());
    id.Add((byte)Id.SERightPupilDiameter);
}
if (SERightPupilDiameterQSpecified){
```

```
                message.Add(SERightPupilDiameterQ.GetByteArray());
                id.Add((byte)Id.SERightPupilDiameterQ);
            }
            if (SEFilteredPupilDiameterSpecified){

message.Add(SEFilteredPupilDiameter.GetByteArray());
                id.Add((byte)Id.SEFilteredPupilDiameter);
            }
            if (SEFilteredPupilDiameterQSpecified){

message.Add(SEFilteredPupilDiameterQ.GetByteArray());
                id.Add((byte)Id.SEFilteredPupilDiameterQ);
            }

            if (SEFilteredLeftPupilDiameterSpecified){

message.Add(SEFilteredLeftPupilDiameter.GetByteArray());
                id.Add((byte)Id.SEFilteredLeftPupilDiameter);
            }

            if (SEFilteredLeftPupilDiameterQSpecified){

message.Add(SEFilteredLeftPupilDiameterQ.GetByteArray());
                id.Add((byte)Id.SEFilteredLeftPupilDiameterQ);
            }
            if (SEFilteredRightPupilDiameterSpecified){

message.Add(SEFilteredRightPupilDiameter.GetByteArray());
                id.Add((byte)Id.SEFilteredRightPupilDiameter);
            }
            if (SEFilteredRightPupilDiameterQSpecified){

message.Add(SEFilteredRightPupilDiameterQ.GetByteArray());
                id.Add((byte)Id.SEFilteredRightPupilDiameterQ);
            }
            if (SEEstimatedLeftGazeOriginSpecified){

message.Add(SEEstimatedLeftGazeOrigin.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftGazeOrigin);
            }
            if (SEEstimatedRightGazeOriginSpecified){

message.Add(SEEstimatedRightGazeOrigin.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightGazeOrigin);
            }
            if (SEEstimatedEyePositionSpecified){

message.Add(SEEstimatedEyePosition.GetByteArray());
                id.Add((byte)Id.SEEstimatedEyePosition);
            }
            if (SEEstimatedGazeDirectionSpecified){

message.Add(SEEstimatedGazeDirection.GetByteArray());
                id.Add((byte)Id.SEEstimatedGazeDirection);
            }
            if (SEEstimatedGazeDirectionQSpecified){
```

```
message.Add(SEEstimatedGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEEstimatedGazeDirectionQ);
            }
            if (SEEstimatedGazeHeadingSpecified){

message.Add(SEEstimatedGazeHeading.GetByteArray());
                id.Add((byte)Id.SEEstimatedGazeHeading);
            }
            if (SEEstimatedGazePitchSpecified){
                message.Add(SEEstimatedGazePitch.GetByteArray());
                id.Add((byte)Id.SEEstimatedGazePitch);
            }
            if (SEEstimatedLeftEyePositionSpecified){

message.Add(SEEstimatedLeftEyePosition.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftEyePosition);
            }
            if (SEEstimatedLeftGazeDirectionSpecified){

message.Add(SEEstimatedLeftGazeDirection.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftGazeDirection);
            }
            if (SEEstimatedLeftGazeDirectionQSpecified){

message.Add(SEEstimatedLeftGazeDirectionQ.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftGazeDirectionQ);
            }
            if (SEEstimatedLeftGazeHeadingSpecified){

message.Add(SEEstimatedLeftGazeHeading.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftGazeHeading);
            }
            if (SEEstimatedLeftGazePitchSpecified){

message.Add(SEEstimatedLeftGazePitch.GetByteArray());
                id.Add((byte)Id.SEEstimatedLeftGazePitch);
            }
            if (SEEstimatedRightEyePositionSpecified){

message.Add(SEEstimatedRightEyePosition.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightEyePosition);
            }
            if (SEEstimatedRightGazeDirectionSpecified){

message.Add(SEEstimatedRightGazeDirection.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightGazeDirection);
            }
            if (SEEstimatedRightGazeDirSpecified){

message.Add(SEEstimatedRightGazeDir.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightGazeDir);
            }
            if (SEEstimatedRightGazeHeadingSpecified){

message.Add(SEEstimatedRightGazeHeading.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightGazeHeading);
            }
```

```
            if (SEEstimatedRightGazePitchSpecified){

 message.Add(SEEstimatedRightGazePitch.GetByteArray());
                id.Add((byte)Id.SEEstimatedRightGazePitch);
            }

            if (SEKeyboardStateSpecified){
                message.Add(SEKeyboardState.GetByteArray());
                id.Add((byte)Id.SEKeyboardState);
            }

            if (SEASCIIKeyboardStateSpecified){
                message.Add(SEASCIIKeyboardState.GetByteArray());
                id.Add((byte)Id.SEASCIIKeyboardState);
            }
            if (SELeftDiagnosisSpecified){
                message.Add(SELeftDiagnosis.GetByteArray());
                id.Add((byte)Id.SELeftDiagnosis);
            }
            if (SERightDiagnosisSpecified){
                message.Add(SERightDiagnosis.GetByteArray());
                id.Add((byte)Id.SERightDiagnosis);
            }
            if (SELeftGlintsFoundSpecified){
                message.Add(SELeftGlintsFound.GetByteArray());
                id.Add((byte)Id.SELeftGlintsFound);
            }
            if (SERightGlintsFoundSpecified){
                message.Add(SERightGlintsFound.GetByteArray());
                id.Add((byte)Id.SERightGlintsFound);
            }
            int packetlength = 0;
            List<byte[]> subpackets = new List<byte[]>();
            for (int i = 0; i < message.Count; i++){
                byte[] idbytes = new byte[2];
                idbytes[0] = 0x00;
                idbytes[1] = id.ToArray()[i]
                byte[] lengthbytes = new byte[2];
                lengthbytes =
BitConverter.GetBytes((UInt16)ByteSwap.swap((UInt16)message[i].Le
ngth));
                byte[] subpacket = new byte[2 + 2 +
message[i].Length];
                idbytes.CopyTo(subpacket, 0);
                lengthbytes.CopyTo(subpacket, 2);
                message[i].CopyTo(subpacket, 2 + 2);
                subpackets.Add(subpacket);
                packetlength += 2 + 2 + message[i].Length;
            }
            byte[] packet = new byte[4 + 2 + 2 + packetlength];
            //header
            packet[0] = 0x0000;
            packet[1] = 0x0000;
            packet[2] = 0x0000;
            packet[3] = 0x0000;

 BitConverter.GetBytes(ByteSwap.swap((UInt16)4)).CopyTo(packet,
```

```
4);

 BitConverter.GetBytes(ByteSwap.swap((UInt16)packetlength)).CopyT
o(packet, 6);
            int placement = 4 + 2 + 2;
            for (int i = 0; i < subpackets.Count; i++){
                subpackets[i].CopyTo(packet, placement);
                placement += subpackets[i].Length;
            }
            sock.Send(packet, packet.Length, iep);
            sock.Close();
        }
        public void clear(){
            SEFrameNumberSpecified = false;
            SEEstimatedDelaySpecified = false;
            SETimeStampSpecified = false;
            SEUserTimeStampSpecified = false;
            SEFrameRateSpecified = false;
            SEHeadPositionSpecified = false;
            SECameraPositionsSpecified = false;
            SECameraRotationsSpecified = false;
            SEHeadPositionQSpecified = false;
            SEHeadRotationRodriguesSpecified = false;
            SEHeadNoseDirectionSpecified = false;
            SEHeadUpDirectionSpecified = false;
            SEHeadLeftEarDirectionSpecified = false;
            SEHeadHeadingSpecified = false;
            SEHeadPitchSpecified = false;
            SEHeadRollSpecified = false;
            SEHeadRotationQSpecified = false;
            SEEyePositionSpecified = false;
            SEGazeOriginSpecified = false;
            SELeftGazeOriginSpecified = false;
            SERightGazeOriginSpecified = false;
            SEGazeDirectionSpecified = false;
            SEGazeDirectionQSpecified = false;
            SELeftEyePositionSpecified = false;
            SELeftGazeDirectionSpecified = false;
            SELeftGazeDirectionQSpecified = false;
            SERightEyePositionSpecified = false;
            SERightGazeDirectionSpecified = false;
            SERightGazeDirectionQSpecified = false;
            SEGazeHeadingSpecified = false;
            SEGazePitchSpecified = false;
            SEFilteredGazeDirectionSpecified = false;
            SEFilteredGazeDirectionQSpecified = false;
            SEFilteredLeftGazeDirectionSpecified = false;
            SEFilteredLeftGazeDirectionQSpecified = false;
            SEFilteredRightGazeDirectionSpecified = false;
            SEFilteredRightGazeDirectionQSpecified = false;
            SEFilteredGazeHeadingSpecified = false;
            SEFilteredGazePitchSpecified = false;
            SESaccadeSpecified = false;
            SEBlinkSpecified = false;
            SEEyelidOpeningSpecified = false;
            SEEyelidOpeningQSpecified = false;
            SELeftEyelidOpeningSpecified = false;
```

```
        SELeftEyelidOpeningQSpecified = false;
        SERightEyelidOpeningSpecified = false;
        SERightEyelidOpeningQSpecified = false;
        SEEstimatedGazeOriginSpecified = false;
        SEPupilDiameterSpecified = false;
        SEPupilDiameterQSpecified = false;
        SELeftPupilDiameterSpecified = false;
        SELeftPupilDiameterQSpecified = false;
        SERightPupilDiameterSpecified = false;
        SERightPupilDiameterQSpecified = false;
        SEFilteredPupilDiameterSpecified = false;
        SEFilteredPupilDiameterQSpecified = false;
        SEFilteredLeftPupilDiameterSpecified = false;
        SEFilteredLeftPupilDiameterQSpecified = false;
        SEFilteredRightPupilDiameterSpecified = false;
        SEFilteredRightPupilDiameterQSpecified = false;
        SEEstimatedLeftGazeOriginSpecified = false;
        SEEstimatedRightGazeOriginSpecified = false;
        SEEstimatedEyePositionSpecified = false;
        SEEstimatedGazeDirectionSpecified = false;
        SEEstimatedGazeDirectionQSpecified = false;
        SEEstimatedGazeHeadingSpecified = false;
        SEEstimatedGazePitchSpecified = false;
        SEEstimatedLeftEyePositionSpecified = false;
        SEEstimatedLeftGazeDirectionSpecified = false;
        SEEstimatedLeftGazeDirectionQSpecified = false;
        SEEstimatedLeftGazeHeadingSpecified = false;
        SEEstimatedLeftGazePitchSpecified = false;
        SEEstimatedRightEyePositionSpecified = false;
        SEEstimatedRightGazeDirectionSpecified = false;
        SEEstimatedRightGazeDirSpecified = false;
        SEEstimatedRightGazeHeadingSpecified = false;
        SEEstimatedRightGazePitchSpecified = false;
        SEKeyboardStateSpecified = false;
        SEASCIIKeyboardStateSpecified = false;
        SELeftDiagnosisSpecified = false;
        SERightDiagnosisSpecified = false;
        SELeftGlintsFoundSpecified = false;
        SERightGlintsFoundSpecified = false;
    }
    public enum Id{
        SEFrameNumber = 0x0001,
        SEEstimatedDelay = 0x0002,
        SETimeStamp = 0x0003,
        SEUserTimeStamp = 0x0004,
        SEFrameRate = 0x0005,
        SEHeadPosition = 0x0010,
        SECameraPositions = 0x0006,
        SECameraRotations = 0x0007,
        SEHeadPositionQ = 0x0011,
        SEHeadRotationRodrigues = 0x0012,
        SEHeadNoseDirection = 0x0013,
        SEHeadUpDirection = 0x0014,
        SEHeadLeftEarDirection = 0x0015,
        SEHeadHeading = 0x0016,
        SEHeadPitch = 0x0017,
        SEHeadRoll = 0x0018,
```

```
SEHeadRotationQ = 0x0019,
SEEyePosition = 0x0020,
SEGazeOrigin = 0x001a,
SELeftGazeOrigin = 0x001b,
SERightGazeOrigin = 0x001c,
SEGazeDirection = 0x0021,
SEGazeDirectionQ = 0x0022,
SELeftEyePosition = 0x0023,
SELeftGazeDirection = 0x0024,
SELeftGazeDirectionQ = 0x0025,
SERightEyePosition = 0x0026,
SERightGazeDirection = 0x0027,
SERightGazeDirectionQ = 0x0028,
SEGazeHeading = 0x0029,
SEGazePitch = 0x002a,
SEFilteredGazeDirection = 0x0030,
SEFilteredGazeDirectionQ = 0x0031,
SEFilteredLeftGazeDirection = 0x0032,
SEFilteredLeftGazeDirectionQ = 0x0033,
SEFilteredRightGazeDirection = 0x0034,
SEFilteredRightGazeDirectionQ = 0x0035,
SEFilteredGazeHeading = 0x0036,
SEFilteredGazePitch = 0x0037,
SESaccade = 0x003d,
SEBlink = 0x003f,
SEEyelidOpening = 0x0050,
SEEyelidOpeningQ = 0x0051,
SELeftEyelidOpening = 0x0052,
SELeftEyelidOpeningQ = 0x0053,
SERightEyelidOpening = 0x0054,
SERightEyelidOpeningQ = 0x0055,
SEEstimatedGazeOrigin = 0x007a,
SEPupilDiameter = 0x0060,
SEPupilDiameterQ = 0x0061,
SELeftPupilDiameter = 0x0062,
SELeftPupilDiameterQ = 0x0063,
SERightPupilDiameter = 0x0064,
SERightPupilDiameterQ = 0x0065,
SEFilteredPupilDiameter = 0x0066,
SEFilteredPupilDiameterQ = 0x0067,
SEFilteredLeftPupilDiameter = 0x0068,
SEFilteredLeftPupilDiameterQ = 0x0069,
SEFilteredRightPupilDiameter = 0x006a,
SEFilteredRightPupilDiameterQ = 0x006b,
SEEstimatedLeftGazeOrigin = 0x007b,
SEEstimatedRightGazeOrigin = 0x007c,
SEEstimatedEyePosition = 0x0080,
SEEstimatedGazeDirection = 0x0081,
SEEstimatedGazeDirectionQ = 0x0082,
SEEstimatedGazeHeading = 0x0083,
SEEstimatedGazePitch = 0x0084,
SEEstimatedLeftEyePosition = 0x0085,
SEEstimatedLeftGazeDirection = 0x0086,
SEEstimatedLeftGazeDirectionQ = 0x0087,
SEEstimatedLeftGazeHeading = 0x0088,
SEEstimatedLeftGazePitch = 0x0089,
SEEstimatedRightEyePosition = 0x008a,
```

```
        SEEstimatedRightGazeDirection = 0x008b,
        SEEstimatedRightGazeDir = 0x0082,
        SEEstimatedRightGazeHeading = 0x008d,
        SEEstimatedRightGazePitch = 0x008e,
        SEKeyboardState = 0x0056,
        SEASCIIKeyboardState = 0x00a4,
        SELeftDiagnosis = 0x00a0,
        SERightDiagnosis = 0x00a1,
        SELeftGlintsFound = 0x00a2,
        SERightGlintsFound = 0x00a3,
    };
    SEu32 _SEFrameNumber;
    public bool SEFrameNumberSpecified = false;
    public SEu32 SEFrameNumber
    {
        set{
            SEFrameNumberSpecified = true;
            _SEFrameNumber = value;
        }
        get{
            SEFrameNumberSpecified = true;
            return _SEFrameNumber;
        }
    }
    SEu32 _SEEstimatedDelay = new SEu32();
    public bool SEEstimatedDelaySpecified = false;
    public SEu32 SEEstimatedDelay{
        set{
            SEEstimatedDelaySpecified = true;
            _SEEstimatedDelay = value;
        }
        get{
            SEEstimatedDelaySpecified = true;
            return _SEEstimatedDelay;
        }
    }
    SEu64 _SETimeStamp = new SEu64();
    public bool SETimeStampSpecified = false;
    public SEu64 SETimeStamp{
        set{
            SETimeStampSpecified = true;
            _SETimeStamp = value;
        }
        get{
            SETimeStampSpecified = true;
            return _SETimeStamp;
        }
    }
    SEu64 _SEUserTimeStamp = new SEu64();
    public bool SEUserTimeStampSpecified = false;
    public SEu64 SEUserTimeStamp{
        set{
            SEUserTimeStampSpecified = true;
            _SEUserTimeStamp = value;
        }
        get{
            SEUserTimeStampSpecified = true;
```

```
            return _SEUserTimeStamp;
        }
    }
    SEf64 _SEFrameRate = new SEf64();
    public bool SEFrameRateSpecified = false;
    public SEf64 SEFrameRate{
        set{
            SEFrameRateSpecified = true;
            _SEFrameRate = value;
        }
        get{
            SEFrameRateSpecified = true;
            return _SEFrameRate;
        }
    }
    SEPoint3D _SEHeadPosition = new SEPoint3D();
    public bool SEHeadPositionSpecified = false;
    public SEPoint3D SEHeadPosition{
        set{
            SEHeadPositionSpecified = true;
            _SEHeadPosition = value;
        }
        get{
            SEHeadPositionSpecified = true;
            return _SEHeadPosition;
        }
    }
    SEVector _SECameraPositions = new SEVector();
    public bool SECameraPositionsSpecified = false;
    public SEVector SECameraPositions{
        set{
            SECameraPositionsSpecified = true;
            _SECameraPositions = value;
        }
        get{
            SECameraPositionsSpecified = true;
            return _SECameraPositions;
        }
    }
    SEVector _SECameraRotations = new SEVector();
    public bool SECameraRotationsSpecified = false;
    public SEVector SECameraRotations{
        set{
            SECameraRotationsSpecified = true;
            _SECameraRotations = value;
        }
        get{
            SECameraRotationsSpecified = true;
            return _SECameraRotations;
        }
    }
    SEf64 _SEHeadPositionQ = new SEf64();
    public bool SEHeadPositionQSpecified = false;
    public SEf64 SEHeadPositionQ{
        set
        {
            SEHeadPositionQSpecified = true;
```

```
            _SEHeadPositionQ = value;
        }
        get{
            SEHeadPositionQSpecified = true;
            return _SEHeadPositionQ;
        }
    }
    SEVect3D _SEHeadRotationRodrigues = new SEVect3D();
    public bool SEHeadRotationRodriguesSpecified = false;
    public SEVect3D SEHeadRotationRodrigues{
        set{
            SEHeadRotationRodriguesSpecified = true;
            _SEHeadRotationRodrigues = value;
        }
        get{
            SEHeadRotationRodriguesSpecified = true;
            return _SEHeadRotationRodrigues;
        }
    }
    SEVect3D _SEHeadNoseDirection = new SEVect3D();
    public bool SEHeadNoseDirectionSpecified = false;
    public SEVect3D SEHeadNoseDirection{
        set{
            SEHeadNoseDirectionSpecified = true;
            _SEHeadNoseDirection = value;
        }
        get{
            SEHeadNoseDirectionSpecified = true;
            return _SEHeadNoseDirection;
        }
    }
    SEVect3D _SEHeadUpDirection = new SEVect3D();
    public bool SEHeadUpDirectionSpecified = false;
    public SEVect3D SEHeadUpDirection{
        set{
            SEHeadUpDirectionSpecified = true;
            _SEHeadUpDirection = value;
        }
        get{
            SEHeadUpDirectionSpecified = true;
            return _SEHeadUpDirection;
        }
    }
    SEVect3D _SEHeadLeftEarDirection = new SEVect3D();
    public bool SEHeadLeftEarDirectionSpecified = false;
    public SEVect3D SEHeadLeftEarDirection{
        set{
            SEHeadLeftEarDirectionSpecified = true;
            _SEHeadLeftEarDirection = value;
        }
        get{
            SEHeadLeftEarDirectionSpecified = true;
            return _SEHeadLeftEarDirection;
        }
    }
    SEf64 _SEHeadHeading = new SEf64();
    public bool SEHeadHeadingSpecified = false;
```

```
public SEf64 SEHeadHeading{
    set{
        SEHeadHeadingSpecified = true;
        _SEHeadHeading = value;
    }
    get{
        SEHeadHeadingSpecified = true;
        return _SEHeadHeading;
    }
}
SEf64 _SEHeadPitch = new SEf64();
public bool SEHeadPitchSpecified = false;
public SEf64 SEHeadPitch{
    set{
        SEHeadPitchSpecified = true;
        _SEHeadPitch = value;
    }
    get{
        SEHeadPitchSpecified = true;
        return _SEHeadPitch;
    }
}
SEf64 _SEHeadRoll = new SEf64();
public bool SEHeadRollSpecified = false;
public SEf64 SEHeadRoll{
    set{
        SEHeadRollSpecified = true;
        _SEHeadRoll = value;
    }
    get{
        SEHeadRollSpecified = true;
        return _SEHeadRoll;
    }
}
SEf64 _SEHeadRotationQ = new SEf64();
public bool SEHeadRotationQSpecified = false;
public SEf64 SEHeadRotationQ{
    set{
        SEHeadRotationQSpecified = true;
        _SEHeadRotationQ = value;
    }
    get{
        SEHeadRotationQSpecified = true;
        return _SEHeadRotationQ;
    }
}
SEPoint3D _SEEyePosition = new SEPoint3D();
public bool SEEyePositionSpecified = false;
public SEPoint3D SEEyePosition{
    set{
        SEEyePositionSpecified = true;
        _SEEyePosition = value;
    }
    get{
        SEEyePositionSpecified = true;
        return _SEEyePosition;
    }
```

```
}
SEPoint3D _SEGazeOrigin = new SEPoint3D();
public bool SEGazeOriginSpecified = false;
public SEPoint3D SEGazeOrigin{
    set{
        SEGazeOriginSpecified = true;
        _SEGazeOrigin = value;
    }
    get{
        SEGazeOriginSpecified = true;
        return _SEGazeOrigin;
    }
}
SEPoint3D _SELeftGazeOrigin = new SEPoint3D();
public bool SELeftGazeOriginSpecified = false;
public SEPoint3D SELeftGazeOrigin{
    set{
        SELeftGazeOriginSpecified = true;
        _SELeftGazeOrigin = value;
    }
    get{
        SELeftGazeOriginSpecified = true;
        return _SELeftGazeOrigin;
    }
}
SEPoint3D _SERightGazeOrigin = new SEPoint3D();
public bool SERightGazeOriginSpecified = false;
public SEPoint3D SERightGazeOrigin{
    set{
        SERightGazeOriginSpecified = true;
        _SERightGazeOrigin = value;
    }
    get{
        SERightGazeOriginSpecified = true;
        return _SERightGazeOrigin;
    }
}
SEVect3D _SEGazeDirection = new SEVect3D();
public bool SEGazeDirectionSpecified = false;
public SEVect3D SEGazeDirection{
    set{
        SEGazeDirectionSpecified = true;
        _SEGazeDirection = value;
    }
    get{
        SEGazeDirectionSpecified = true;
        return _SEGazeDirection;
    }
}
SEf64 _SEGazeDirectionQ = new SEf64();
public bool SEGazeDirectionQSpecified = false;
public SEf64 SEGazeDirectionQ{
    set{
        SEGazeDirectionQSpecified = true;
        _SEGazeDirectionQ = value;
    }
    get{
```

```
            SEGazeDirectionQSpecified = true;
            return _SEGazeDirectionQ;
        }
    }
    SEPoint3D _SELeftEyePosition = new SEPoint3D();
    public bool SELeftEyePositionSpecified = false;
    public SEPoint3D SELeftEyePosition{
        set{
            SELeftEyePositionSpecified = true;
            _SELeftEyePosition = value;
        }
        get{
            SELeftEyePositionSpecified = true;
            return _SELeftEyePosition;
        }
    }
    SEVect3D _SELeftGazeDirection = new SEVect3D();
    public bool SELeftGazeDirectionSpecified = false;
    public SEVect3D SELeftGazeDirection{
        set{
            SELeftGazeDirectionSpecified = true;
            _SELeftGazeDirection = value;
        }
        get{
            SELeftGazeDirectionSpecified = true;
            return _SELeftGazeDirection;
        }
    }
    SEf64 _SELeftGazeDirectionQ = new SEf64();
    public bool SELeftGazeDirectionQSpecified = false;
    public SEf64 SELeftGazeDirectionQ{
        set{
            SELeftGazeDirectionQSpecified = true;
            _SELeftGazeDirectionQ = value;
        }
        get{
            SELeftGazeDirectionQSpecified = true;
            return _SELeftGazeDirectionQ;
        }
    }
    SEPoint3D _SERightEyePosition = new SEPoint3D();
    public bool SERightEyePositionSpecified = false;
    public SEPoint3D SERightEyePosition{
        set{
            SERightEyePositionSpecified = true;
            _SERightEyePosition = value;
        }
        get{
            SERightEyePositionSpecified = true;
            return _SERightEyePosition;
        }
    }
    SEVect3D _SERightGazeDirection = new SEVect3D();
    public bool SERightGazeDirectionSpecified = false;
    public SEVect3D SERightGazeDirection{
        set{
            SERightGazeDirectionSpecified = true;
```

```
            _SERightGazeDirection = value;
        }
        get{
            SERightGazeDirectionSpecified = true;
            return _SERightGazeDirection;
        }
    }
    SEf64 _SERightGazeDirectionQ = new SEf64();
    public bool SERightGazeDirectionQSpecified = false;
    public SEf64 SERightGazeDirectionQ{
        set{
            SERightGazeDirectionQSpecified = true;
            _SERightGazeDirectionQ = value;
        }
        get{
            SERightGazeDirectionQSpecified = true;
            return _SERightGazeDirectionQ;
        }
    }
    SEf64 _SEGazeHeading = new SEf64();
    public bool SEGazeHeadingSpecified = false;
    public SEf64 SEGazeHeading{
        set{
            SEGazeHeadingSpecified = true;
            _SEGazeHeading = value;
        }
        get{
            SEGazeHeadingSpecified = true;
            return _SEGazeHeading;
        }
    }       SEf64 _SEGazePitch = new SEf64();
    public bool SEGazePitchSpecified = false;
    public SEf64 SEGazePitch{
        set{
            SEGazePitchSpecified = true;
            _SEGazePitch = value;
        }
        get{
            SEGazePitchSpecified = true;
            return _SEGazePitch;
        }
    }
    SEVect3D _SEFilteredGazeDirection = new SEVect3D();
    public bool SEFilteredGazeDirectionSpecified = false;
    public SEVect3D SEFilteredGazeDirection{
        set{
            SEFilteredGazeDirectionSpecified = true;
            _SEFilteredGazeDirection = value;
        }
        get{
            SEFilteredGazeDirectionSpecified = true;
            return _SEFilteredGazeDirection;
        }
    }
    SEf64 _SEFilteredGazeDirectionQ = new SEf64();
    public bool SEFilteredGazeDirectionQSpecified = false;
    public SEf64 SEFilteredGazeDirectionQ{
```

```
                set{
                    SEFilteredGazeDirectionQSpecified = true;
                    _SEFilteredGazeDirectionQ = value;
                }
                get{
                    SEFilteredGazeDirectionQSpecified = true;
                    return _SEFilteredGazeDirectionQ;
                }
            }
            SEVect3D _SEFilteredLeftGazeDirection = new SEVect3D();
            public bool SEFilteredLeftGazeDirectionSpecified = false;
            public SEVect3D SEFilteredLeftGazeDirection{
                set{
                    SEFilteredLeftGazeDirectionSpecified = true;
                    _SEFilteredLeftGazeDirection = value;
                }
                get{
                    SEFilteredLeftGazeDirectionSpecified = true;
                    return _SEFilteredLeftGazeDirection;
                }
            }
            SEf64 _SEFilteredLeftGazeDirectionQ = new SEf64();
            public bool SEFilteredLeftGazeDirectionQSpecified = false;
            public SEf64 SEFilteredLeftGazeDirectionQ{
                set{
                    SEFilteredLeftGazeDirectionQSpecified = true;
                    _SEFilteredLeftGazeDirectionQ = value;
                }
                get{
                    SEFilteredLeftGazeDirectionQSpecified = true;
                    return _SEFilteredLeftGazeDirectionQ;
                }
            }
            SEVect3D _SEFilteredRightGazeDirection = new SEVect3D();
            public bool SEFilteredRightGazeDirectionSpecified = false;
            public SEVect3D SEFilteredRightGazeDirection{
                set{
                    SEFilteredRightGazeDirectionSpecified = true;
                    _SEFilteredRightGazeDirection = value;
                }
                get{
                    SEFilteredRightGazeDirectionSpecified = true;
                    return _SEFilteredRightGazeDirection;
                }
            }
            SEf64 _SEFilteredRightGazeDirectionQ = new SEf64();
            public bool SEFilteredRightGazeDirectionQSpecified =
false;
            public SEf64 SEFilteredRightGazeDirectionQ{
                set{
                    SEFilteredRightGazeDirectionQSpecified = true;
                    _SEFilteredRightGazeDirectionQ = value;
                }
                get{
                    SEFilteredRightGazeDirectionQSpecified = true;
                    return _SEFilteredRightGazeDirectionQ;
                }
```

```
}
SEf64 _SEFilteredGazeHeading = new SEf64();
public bool SEFilteredGazeHeadingSpecified = false;
public SEf64 SEFilteredGazeHeading{
    set{
        SEFilteredGazeHeadingSpecified = true;
        _SEFilteredGazeHeading = value;
    }
    get{
        SEFilteredGazeHeadingSpecified = true;
        return _SEFilteredGazeHeading;
    }
}
SEf64 _SEFilteredGazePitch = new SEf64();
public bool SEFilteredGazePitchSpecified = false;
public SEf64 SEFilteredGazePitch{
    set{
        SEFilteredGazePitchSpecified = true;
        _SEFilteredGazePitch = value;
    }
    get{
        SEFilteredGazePitchSpecified = true;
        return _SEFilteredGazePitch;
    }
}
SEVector _SESaccade = new SEVector();
public bool SESaccadeSpecified = false;
public SEVector SESaccade{
    set{
        SESaccadeSpecified = true;
        _SESaccade = value;
    }
    get{
        SESaccadeSpecified = true;
        return _SESaccade;
    }
}
SEVector _SEBlink = new SEVector();
public bool SEBlinkSpecified = false;
public SEVector SEBlink{
    set{
        SEBlinkSpecified = true;
        _SEBlink = value;
    }
    get{
        SEBlinkSpecified = true;
        return _SEBlink;
    }
}
SEf64 _SEEyelidOpening = new SEf64();
public bool SEEyelidOpeningSpecified = false;
public SEf64 SEEyelidOpening{
    set{
        SEEyelidOpeningSpecified = true;
        _SEEyelidOpening = value;
    }
    get{
```

```
            SEEyelidOpeningSpecified = true;
            return _SEEyelidOpening;
        }
    }
    SEf64 _SEEyelidOpeningQ = new SEf64();
    public bool SEEyelidOpeningQSpecified = false;
    public SEf64 SEEyelidOpeningQ{
        set{
            SEEyelidOpeningQSpecified = true;
            _SEEyelidOpeningQ = value;
        }
        get{
            SEEyelidOpeningQSpecified = true;
            return _SEEyelidOpeningQ;
        }
    }
    SEf64 _SELeftEyelidOpening = new SEf64();
    public bool SELeftEyelidOpeningSpecified = false;
    public SEf64 SELeftEyelidOpening{
        set{
            SELeftEyelidOpeningSpecified = true;
            _SELeftEyelidOpening = value;
        }
        get{
            SELeftEyelidOpeningSpecified = true;
            return _SELeftEyelidOpening;
        }
    }
    SEf64 _SELeftEyelidOpeningQ = new SEf64();
    public bool SELeftEyelidOpeningQSpecified = false;
    public SEf64 SELeftEyelidOpeningQ{
        set{
            SELeftEyelidOpeningQSpecified = true;
            _SELeftEyelidOpeningQ = value;
        }
        get{
            SELeftEyelidOpeningQSpecified = true;
            return _SELeftEyelidOpeningQ;
        }
    }
    SEf64 _SERightEyelidOpening = new SEf64();
    public bool SERightEyelidOpeningSpecified = false;
    public SEf64 SERightEyelidOpening{
        set{
            SERightEyelidOpeningSpecified = true;
            _SERightEyelidOpening = value;
        }
        get{
            SERightEyelidOpeningSpecified = true;
            return _SERightEyelidOpening;
        }
    }
    SEf64 _SERightEyelidOpeningQ = new SEf64();
    public bool SERightEyelidOpeningQSpecified = false;
    public SEf64 SERightEyelidOpeningQ{
        set{
            SERightEyelidOpeningQSpecified = true;
```

```
            _SERightEyelidOpeningQ = value;
        }
        get{
            SERightEyelidOpeningQSpecified = true;
            return _SERightEyelidOpeningQ;
        }
    }
    SEPoint3D _SEEstimatedGazeOrigin = new SEPoint3D();
    public bool SEEstimatedGazeOriginSpecified = false;
    public SEPoint3D SEEstimatedGazeOrigin{
        set{
            SEEstimatedGazeOriginSpecified = true;
            _SEEstimatedGazeOrigin = value;
        }
        get{
            SEEstimatedGazeOriginSpecified = true;
            return _SEEstimatedGazeOrigin;
        }
    }
    SEf64 _SEPupilDiameter = new SEf64();
    public bool SEPupilDiameterSpecified = false;
    public SEf64 SEPupilDiameter{
        set{
            SEPupilDiameterSpecified = true;
            _SEPupilDiameter = value;
        }
        get{
            SEPupilDiameterSpecified = true;
            return _SEPupilDiameter;
        }
    }
    SEf64 _SEPupilDiameterQ = new SEf64();
    public bool SEPupilDiameterQSpecified = false;
    public SEf64 SEPupilDiameterQ{
        set{
            SEPupilDiameterQSpecified = true;
            _SEPupilDiameterQ = value;
        }
        get{
            SEPupilDiameterQSpecified = true;
            return _SEPupilDiameterQ;
        }
    }
    SEf64 _SELeftPupilDiameter = new SEf64();
    public bool SELeftPupilDiameterSpecified = false;
    public SEf64 SELeftPupilDiameter{
        set{
            SELeftPupilDiameterSpecified = true;
            _SELeftPupilDiameter = value;
        }
        get{
            SELeftPupilDiameterSpecified = true;
            return _SELeftPupilDiameter;
        }
    }
    SEf64 _SELeftPupilDiameterQ = new SEf64();
    public bool SELeftPupilDiameterQSpecified = false;
```

```
public SEf64 SELeftPupilDiameterQ{
    set{
        SELeftPupilDiameterQSpecified = true;
        _SELeftPupilDiameterQ = value;
    }
    get{
        SELeftPupilDiameterQSpecified = true;
        return _SELeftPupilDiameterQ;
    }
}
SEf64 _SERightPupilDiameter = new SEf64();
public bool SERightPupilDiameterSpecified = false;
public SEf64 SERightPupilDiameter{
    set{
        SERightPupilDiameterSpecified = true;
        _SERightPupilDiameter = value;
    }
    get{
        SERightPupilDiameterSpecified = true;
        return _SERightPupilDiameter;
    }
}
SEf64 _SERightPupilDiameterQ = new SEf64();
public bool SERightPupilDiameterQSpecified = false;
public SEf64 SERightPupilDiameterQ{
    set{
        SERightPupilDiameterQSpecified = true;
        _SERightPupilDiameterQ = value;
    }
    get{
        SERightPupilDiameterQSpecified = true;
        return _SERightPupilDiameterQ;
    }
}
SEf64 _SEFilteredPupilDiameter = new SEf64();
public bool SEFilteredPupilDiameterSpecified = false;
public SEf64 SEFilteredPupilDiameter{
    set{
        SEFilteredPupilDiameterSpecified = true;
        _SEFilteredPupilDiameter = value;
    }
    get{
        SEFilteredPupilDiameterSpecified = true;
        return _SEFilteredPupilDiameter;
    }
}
SEf64 _SEFilteredPupilDiameterQ = new SEf64();
public bool SEFilteredPupilDiameterQSpecified = false;
public SEf64 SEFilteredPupilDiameterQ{
    set{
        SEFilteredPupilDiameterQSpecified = true;
        _SEFilteredPupilDiameterQ = value;
    }
    get{
        SEFilteredPupilDiameterQSpecified = true;
        return _SEFilteredPupilDiameterQ;
    }
```

```
        }
        SEf64 _SEFilteredLeftPupilDiameter = new SEf64();
        public bool SEFilteredLeftPupilDiameterSpecified = false;
        public SEf64 SEFilteredLeftPupilDiameter{
            set{
                SEFilteredLeftPupilDiameterSpecified = true;
                _SEFilteredLeftPupilDiameter = value;
            }
            get{
                SEFilteredLeftPupilDiameterSpecified = true;
                return _SEFilteredLeftPupilDiameter;
            }
        }
        SEf64 _SEFilteredLeftPupilDiameterQ = new SEf64();
        public bool SEFilteredLeftPupilDiameterQSpecified = false;
        public SEf64 SEFilteredLeftPupilDiameterQ{
            set{
                SEFilteredLeftPupilDiameterQSpecified = true;
                _SEFilteredLeftPupilDiameterQ = value;
            }
            get{
                SEFilteredLeftPupilDiameterQSpecified = true;
                return _SEFilteredLeftPupilDiameterQ;
            }
        }
        SEf64 _SEFilteredRightPupilDiameter = new SEf64();
        public bool SEFilteredRightPupilDiameterSpecified = false;
        public SEf64 SEFilteredRightPupilDiameter
        {
            set{
                SEFilteredRightPupilDiameterSpecified = true;
                _SEFilteredRightPupilDiameter = value;
            }
            get{
                SEFilteredRightPupilDiameterSpecified = true;
                return _SEFilteredRightPupilDiameter;
            }
        }
        SEf64 _SEFilteredRightPupilDiameterQ = new SEf64();
        public bool SEFilteredRightPupilDiameterQSpecified =
false;
        public SEf64 SEFilteredRightPupilDiameterQ{
            set{
                SEFilteredRightPupilDiameterQSpecified = true;
                _SEFilteredRightPupilDiameterQ = value;
            }
            get{
                SEFilteredRightPupilDiameterQSpecified = true;
                return _SEFilteredRightPupilDiameterQ;
            }
        }

        SEPoint3D _SEEstimatedLeftGazeOrigin = new SEPoint3D();
        public bool SEEstimatedLeftGazeOriginSpecified = false;
        public SEPoint3D SEEstimatedLeftGazeOrigin{
            set{
                SEEstimatedLeftGazeOriginSpecified = true;
```

```
            _SEEstimatedLeftGazeOrigin = value;
        }
        get{
            SEEstimatedLeftGazeOriginSpecified = true;
            return _SEEstimatedLeftGazeOrigin;
        }
    }
    SEPoint3D _SEEstimatedRightGazeOrigin = new SEPoint3D();
    public bool SEEstimatedRightGazeOriginSpecified = false;
    public SEPoint3D SEEstimatedRightGazeOrigin{
        set{
            SEEstimatedRightGazeOriginSpecified = true;
            _SEEstimatedRightGazeOrigin = value;
        }
        get{
            SEEstimatedRightGazeOriginSpecified = true;
            return _SEEstimatedRightGazeOrigin;
        }
    }
    SEPoint3D _SEEstimatedEyePosition = new SEPoint3D();
    public bool SEEstimatedEyePositionSpecified = false;
    public SEPoint3D SEEstimatedEyePosition{
        set{
            SEEstimatedEyePositionSpecified = true;
            _SEEstimatedEyePosition = value;
        }
        get{
            SEEstimatedEyePositionSpecified = true;
            return _SEEstimatedEyePosition;
        }
    }
    SEVect3D _SEEstimatedGazeDirection = new SEVect3D();
    public bool SEEstimatedGazeDirectionSpecified = false;
    public SEVect3D SEEstimatedGazeDirection{
        set{
            SEEstimatedGazeDirectionSpecified = true;
            _SEEstimatedGazeDirection = value;
        }
        get{
            SEEstimatedGazeDirectionSpecified = true;
            return _SEEstimatedGazeDirection;
        }
    }
    SEf64 _SEEstimatedGazeDirectionQ = new SEf64();
    public bool SEEstimatedGazeDirectionQSpecified = false;
    public SEf64 SEEstimatedGazeDirectionQ{
        set{
            SEEstimatedGazeDirectionQSpecified = true;
            _SEEstimatedGazeDirectionQ = value;
        }
        get{
            SEEstimatedGazeDirectionQSpecified = true;
            return _SEEstimatedGazeDirectionQ;
        }
    }
    SEf64 _SEEstimatedGazeHeading = new SEf64();
    public bool SEEstimatedGazeHeadingSpecified = false;
```

```
        public SEf64 SEEstimatedGazeHeading{
            set{
                SEEstimatedGazeHeadingSpecified = true;
                _SEEstimatedGazeHeading = value;
            }
            get{
                SEEstimatedGazeHeadingSpecified = true;
                return _SEEstimatedGazeHeading;
            }
        }
        SEf64 _SEEstimatedGazePitch = new SEf64();
        public bool SEEstimatedGazePitchSpecified = false;
        public SEf64 SEEstimatedGazePitch{
            set{
                SEEstimatedGazePitchSpecified = true;
                _SEEstimatedGazePitch = value;
            }
            get{
                SEEstimatedGazePitchSpecified = true;
                return _SEEstimatedGazePitch;
            }
        }
        SEPoint3D _SEEstimatedLeftEyePosition = new SEPoint3D();
        public bool SEEstimatedLeftEyePositionSpecified = false;
        public SEPoint3D SEEstimatedLeftEyePosition{
            set{
                SEEstimatedLeftEyePositionSpecified = true;
                _SEEstimatedLeftEyePosition = value;
            }
            get{
                SEEstimatedLeftEyePositionSpecified = true;
                return _SEEstimatedLeftEyePosition;
            }
        }
        SEVect3D _SEEstimatedLeftGazeDirection = new SEVect3D();
        public bool SEEstimatedLeftGazeDirectionSpecified = false;
        public SEVect3D SEEstimatedLeftGazeDirection{
            set{
                SEEstimatedLeftGazeDirectionSpecified = true;
                _SEEstimatedLeftGazeDirection = value;
            }
            get{
                SEEstimatedLeftGazeDirectionSpecified = true;
                return _SEEstimatedLeftGazeDirection;
            }
        }
        SEf64 _SEEstimatedLeftGazeDirectionQ = new SEf64();
        public bool SEEstimatedLeftGazeDirectionQSpecified =
false;
        public SEf64 SEEstimatedLeftGazeDirectionQ{
            set{
                SEEstimatedLeftGazeDirectionQSpecified = true;
                _SEEstimatedLeftGazeDirectionQ = value;
            }
            get{
                SEEstimatedLeftGazeDirectionQSpecified = true;
                return _SEEstimatedLeftGazeDirectionQ;
```

```
            }
        }
        SEf64 _SEEstimatedLeftGazeHeading = new SEf64();
        public bool SEEstimatedLeftGazeHeadingSpecified = false;
        public SEf64 SEEstimatedLeftGazeHeading{
            set{
                SEEstimatedLeftGazeHeadingSpecified = true;
                _SEEstimatedLeftGazeHeading = value;
            }
            get{
                SEEstimatedLeftGazeHeadingSpecified = true;
                return _SEEstimatedLeftGazeHeading;
            }
        }
        SEf64 _SEEstimatedLeftGazePitch = new SEf64();
        public bool SEEstimatedLeftGazePitchSpecified = false;
        public SEf64 SEEstimatedLeftGazePitch{
            set{
                SEEstimatedLeftGazePitchSpecified = true;
                _SEEstimatedLeftGazePitch = value;
            }
            get{
                SEEstimatedLeftGazePitchSpecified = true;
                return _SEEstimatedLeftGazePitch;
            }
        }
        SEPoint3D _SEEstimatedRightEyePosition = new SEPoint3D();
        public bool SEEstimatedRightEyePositionSpecified = false;
        public SEPoint3D SEEstimatedRightEyePosition{
            set{
                SEEstimatedRightEyePositionSpecified = true;
                _SEEstimatedRightEyePosition = value;
            }
            get{
                SEEstimatedRightEyePositionSpecified = true;
                return _SEEstimatedRightEyePosition;
            }
        }
        SEVect3D _SEEstimatedRightGazeDirection = new SEVect3D();
        public bool SEEstimatedRightGazeDirectionSpecified =
false;
        public SEVect3D SEEstimatedRightGazeDirection{
            set{
                SEEstimatedRightGazeDirectionSpecified = true;
                _SEEstimatedRightGazeDirection = value;
            }
            get{
                SEEstimatedRightGazeDirectionSpecified = true;
                return _SEEstimatedRightGazeDirection;
            }
        }
        SEf64 _SEEstimatedRightGazeDir = new SEf64();
        public bool SEEstimatedRightGazeDirSpecified = false;
        public SEf64 SEEstimatedRightGazeDir{
            set{
                SEEstimatedRightGazeDirSpecified = true;
                _SEEstimatedRightGazeDir = value;
```

```
    }
    get{
        SEEstimatedRightGazeDirSpecified = true;
        return _SEEstimatedRightGazeDir;
    }
}
SEf64 _SEEstimatedRightGazeHeading = new SEf64();
public bool SEEstimatedRightGazeHeadingSpecified = false;
public SEf64 SEEstimatedRightGazeHeading{
    set{
        SEEstimatedRightGazeHeadingSpecified = true;
        _SEEstimatedRightGazeHeading = value;
    }
    get{
        SEEstimatedRightGazeHeadingSpecified = true;
        return _SEEstimatedRightGazeHeading;
    }
}       SEf64 _SEEstimatedRightGazePitch = new SEf64();
public bool SEEstimatedRightGazePitchSpecified = false;
public SEf64 SEEstimatedRightGazePitch{
    set{
        SEEstimatedRightGazePitchSpecified = true;
        _SEEstimatedRightGazePitch = value;
    }
    get{
        SEEstimatedRightGazePitchSpecified = true;
        return _SEEstimatedRightGazePitch;
    }
}
SEString _SEKeyboardState = new SEString();
public bool SEKeyboardStateSpecified = false;
public SEString SEKeyboardState{
    set{
        SEKeyboardStateSpecified = true;
        _SEKeyboardState = value;
    }
    get{
        SEKeyboardStateSpecified = true;
        return _SEKeyboardState;
    }
}
SEu16 _SEASCIIKeyboardState = new SEu16();
public bool SEASCIIKeyboardStateSpecified = false;
public SEu16 SEASCIIKeyboardState{
    set{
        SEASCIIKeyboardStateSpecified = true;
        _SEASCIIKeyboardState = value;
    }
    get{
        SEASCIIKeyboardStateSpecified = true;
        return _SEASCIIKeyboardState;
    }
}
SEf64 _SELeftDiagnosis = new SEf64();
public bool SELeftDiagnosisSpecified = false;
public SEf64 SELeftDiagnosis{
    set{
```

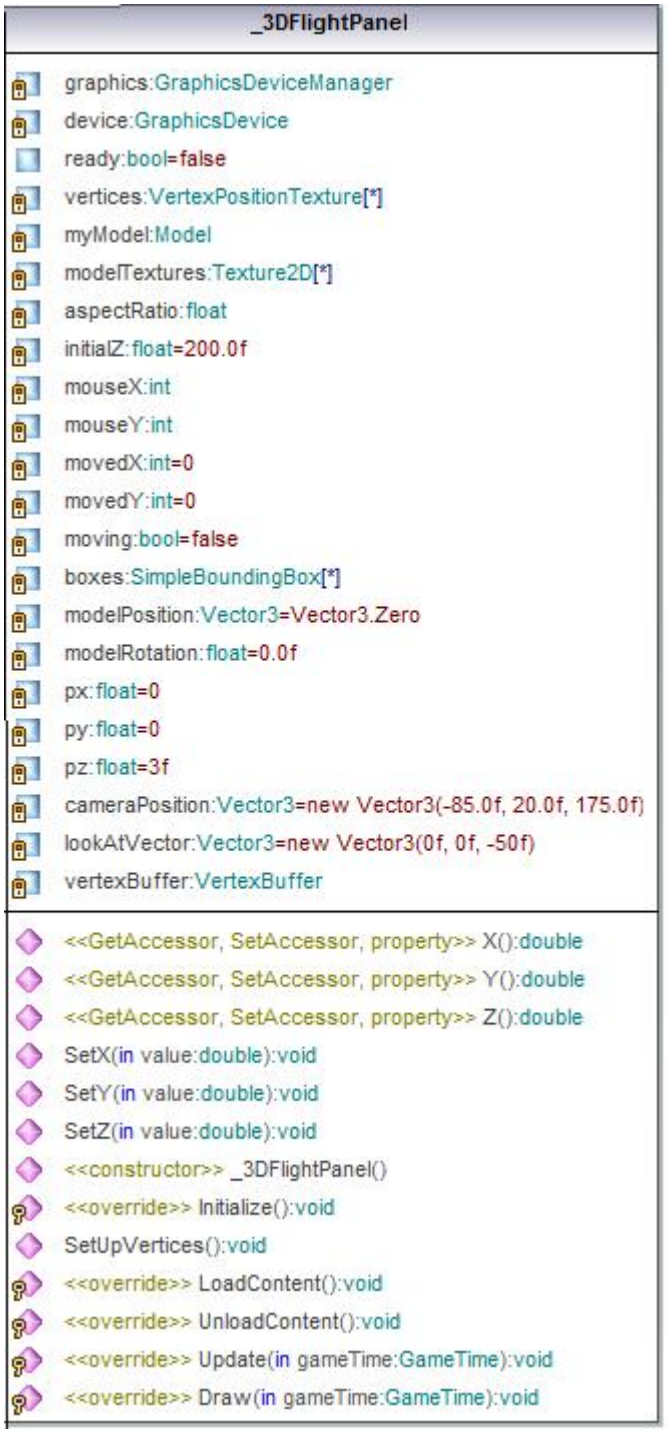
```
                SELeftDiagnosisSpecified = true;
                _SELeftDiagnosis = value;
            }
            get{
                SELeftDiagnosisSpecified = true;
                return _SELeftDiagnosis;
            }
        }
        SEf64 _SERightDiagnosis = new SEf64();
        public bool SERightDiagnosisSpecified = false;
        public SEf64 SERightDiagnosis{
            set{
                SERightDiagnosisSpecified = true;
                _SERightDiagnosis = value;
            }
            get{
                SERightDiagnosisSpecified = true;
                return _SERightDiagnosis;
            }
        }
        SEu32 _SELeftGlintsFound = new SEu32();
        public bool SELeftGlintsFoundSpecified = false;
        public SEu32 SELeftGlintsFound{
            set{
                SELeftGlintsFoundSpecified = true;
                _SELeftGlintsFound = value;
            }
            get{
                SELeftGlintsFoundSpecified = true;
                return _SELeftGlintsFound;
            }
        }
        SEu32 _SERightGlintsFound = new SEu32();
        public bool SERightGlintsFoundSpecified = false;
        public SEu32 SERightGlintsFound{
            set{
                SERightGlintsFoundSpecified = true;
                _SERightGlintsFound = value;
            }
            get{
                SERightGlintsFoundSpecified = true;
                return _SERightGlintsFound;
            }
        }
    }
}
```

VECTOR.CS


```
//Eye Tracking Program Written by Artistee Harris
//Reference: Microsoft Framework .NET
//Description of a vector

using System;
using System.Collections.Generic;
using System.Text;
namespace Eyetrack{
    public class vec{
        private point source;
        private point dest;
        public vec(point s, point d){
            source = s;
            dest = d;
        }
        public void setSource(point p){
            this.source = p;
```

```
        }
        public point getSource(){
            return source;
        }
        public void setDest(point d){
            this.dest = d;
        }
        public point getDest(){
            return dest;
        }
            public point getUnitValue(){
                  return dest;
            }
        public static vec operator +(vec addVec1){
            return addVec1;
        }
        public static vec operator -(vec minusVec1){
            return minusVec1;
        }
        public static vec operator /(vec divideVec1, vec divideVec2){
            return divideVec1;
        }
        public static vec operator *(vec multVec1, vec multVec2){
            return multVec1;
        }
        public static vec operator +(vec addVec1, vec addVec2){

            return addVec1;
        }
        public static vec operator -(vec minusVec1, vec minusVec2){
            return minusVec1;
        }
        public void moveStepDest(float velocity){
        }
            public vec getUnitVector(){
                  float divisor = (float)Math.Sqrt( Math.Pow(
(double)dest.x, (double)2.0 ) + Math.Pow( (double)dest.y, (double)2.0 )
+ Math.Pow( (double)dest.z, (double)2.0 ) );
                  if( divisor == 0 ) divisor = 1;
                  vec v = new vec( new point( 0, 0, 0 ), new point(
dest.x/divisor, dest.y/divisor, dest.x/divisor) );
                  return v;
            }
    }
}
```

3DFLIGHTPANEL.CS




```
//Eye Tracking Program Written by Artistee Harris
```

```
//Reference: Microsoft Framework .NET

//3D Flight panel for visualizing the eye movement
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace Eyetrack
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
public class _3DFlightPanel : Microsoft.Xna.Framework.Game{
GraphicsDeviceManager graphics;
GraphicsDevice device;
public bool ready = false;
public double X{get { return (double)this.px; } set { this.px =
(float)value; }}
public double Y{get { return (double)this.py; }set { this.py =
(float)value; }}
public double Z{get { return (double)this.pz; }set { this.pz =
(float)value; }}
public void SetX(double value){this.px = (float)value;}
public void SetY(double value){this.py = (float)value;}
public void SetZ(double value){this.pz = (float)value;}
public _3DFlightPanel(){graphics = new GraphicsDeviceManager(this);
Content.RootDirectory = "Content";}
protected override void Initialize(){graphics.PreferredBackBufferWidth
= 800;
graphics.PreferredBackBufferHeight = 600;graphics.IsFullScreen = false;
graphics.ApplyChanges();Window.Title = "3d Cockpit";base.Initialize();
ready = true;}
VertexPositionTexture[] vertices;
public void SetUpVertices(){vertices = new VertexPositionTexture[4];
vertices[0].Position = new Vector3(-10.0F, 10.0F, 0.0F);
vertices[0].TextureCoordinate.X = 0;vertices[0].TextureCoordinate.Y =
0;
vertices[1].Position = new Vector3(10.0F, 10.0F,
0.0F);vertices[1].TextureCoordinate.X = 1;
vertices[1].TextureCoordinate.Y = 0;vertices[2].Position = new
Vector3(10.0F, -10.0F, 0.0F);
vertices[2].TextureCoordinate.X = 1;vertices[2].TextureCoordinate.Y =
1;
vertices[3].Position = new Vector3(-10.0F, -10.0F,
0.0F);vertices[3].TextureCoordinate.X = 0;
vertices[3].TextureCoordinate.Y = 1;}
Model myModel;Texture2D[] modelTextures;
protected override void LoadContent(){device = graphics.GraphicsDevice;
```

```
myModel = Content.Load<Model>(@"Models\cockpit");
aspectRatio = graphics.GraphicsDevice.Viewport.AspectRatio;
modelTextures = new Texture2D[]
{Content.Load<Texture2D>(@"Textures\panel1"),
Content.Load<Texture2D>(@"Textures\panel3"),Content.Load<Texture2D>(@"T
extures\panel2")
            };
        }
float aspectRatio;float initialZ = 200.0f;protected override void
UnloadContent(){}
int mouseX;int mouseY;int movedX = 0;int movedY = 0;bool moving =
false;
struct VertexPositionNormal{public Vector3 Position;public Vector3
Normal;}
private SimpleBoundingBox[] boxes;
public class SimpleBoundingBox {
private Vector3 topLeft;private Vector3 bottomRight;
public SimpleBoundingBox(Vector3 topLeft, Vector3 bottomRight) {
this.topLeft = topLeft;this.bottomRight = bottomRight;
}
public Vector3 TopLeft {get { return topLeft; }}
public Vector3 BottomRight {get { return bottomRight; }}
public float? GetZPosition(Vector2 vector) {
if (vector.X >= topLeft.X && vector.X <= bottomRight.X) {
if (vector.Y >= bottomRight.Y && vector.Y <= topLeft.Y) {
float diffTop = topLeft.Y - bottomRight.Y;float diffZ = topLeft.Z -
bottomRight.Z;
float relY = topLeft.Y - vector.Y;float perc = relY / diffTop;
float relZ = diffZ * perc + topLeft.Z;return relZ;}}return null;}}
protected override void Update(GameTime gameTime){
if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
ButtonState.Pressed)this.Exit();
if (Keyboard.GetState().IsKeyDown(Keys.Escape))this.Exit();
if (Keyboard.GetState().IsKeyDown(Keys.NumPad4))px = px - 0.1f;
if (Keyboard.GetState().IsKeyDown(Keys.NumPad6))px = px + 0.1f;
if (Keyboard.GetState().IsKeyDown(Keys.NumPad8))py = py + 0.1f;
if (Keyboard.GetState().IsKeyDown(Keys.NumPad2))py = py - 0.1f;
if (Keyboard.GetState().IsKeyDown(Keys.NumPad9))pz = pz + 0.1f;
if (Keyboard.GetState().IsKeyDown(Keys.NumPad3))pz = pz - 0.1f;
int translateX = 0;
int translateZ = 0;
if (Keyboard.GetState().IsKeyDown(Keys.Left) ||
Keyboard.GetState().IsKeyDown(Keys.A))translateX -= 5;
if (Keyboard.GetState().IsKeyDown(Keys.Right) ||
Keyboard.GetState().IsKeyDown(Keys.D))translateX += 5;
if (Keyboard.GetState().IsKeyDown(Keys.Up) ||
Keyboard.GetState().IsKeyDown(Keys.W))translateZ -= 5;
if (Keyboard.GetState().IsKeyDown(Keys.Down) ||
Keyboard.GetState().IsKeyDown(Keys.S))translateZ += 5;
cameraPosition = Vector3.Add(cameraPosition, new Vector3(translateX, 0,
translateZ));
if (cameraPosition.X <= -170){cameraPosition = new Vector3(-170,
cameraPosition.Y, cameraPosition.Z);}
if (cameraPosition.X > 170){cameraPosition = new Vector3(170,
cameraPosition.Y, cameraPosition.Z);}
if (cameraPosition.Z > initialZ){cameraPosition = new
Vector3(cameraPosition.X, cameraPosition.Y, initialZ);}
```

```
if (cameraPosition.Z < 50){cameraPosition = new
Vector3(cameraPosition.X, cameraPosition.Y, 50);}
base.Update(gameTime);}
Vector3 modelPosition = Vector3.Zero;float modelRotation = 0.0f;float
px = 0;float py = 0;float pz = 3f;
Vector3 cameraPosition = new Vector3(-85.0f, 20.0f, 175.0f);Vector3
lookAtVector = new Vector3(0f, 0f, -50f);
VertexBuffer vertexBuffer;
protected override void Draw(GameTime gameTime){
graphics.GraphicsDevice.Clear(Color.Black);Matrix[] transforms = new
Matrix[myModel.Bones.Count];
myModel.CopyAbsoluteBoneTransformsTo(transforms);Matrix world = new
Matrix();
foreach (ModelMesh mesh in myModel.Meshes){
foreach (BasicEffect effect in mesh.Effects){int index =
myModel.Meshes.IndexOf(mesh);
if (modelTextures.Length > index){effect.Texture =
modelTextures[index];
effect.TextureEnabled = true;}
effect.EnableDefaultLighting() ;
effect.World = transforms[mesh.ParentBone.Index] *
Matrix.CreateRotationY(modelRotation)*
Matrix.CreateTranslation(modelPosition);
effect.View = Matrix.CreateLookAt(cameraPosition,lookAtVector,
Vector3.Up);
effect.Projection =
Matrix.CreatePerspectiveFieldOfView(MathHelper.ToRadians(50.0f),
aspectRatio,
1.0f, 10000.0f);world = effect.World;}mesh.Draw();}
int points = 8;
VertexPositionColor[] pointList = new VertexPositionColor[8];
pointList[0] = new VertexPositionColor(new Vector3(px, py, pz),
Color.Blue);
pointList[1] = new VertexPositionColor(new Vector3(px, py + 200, pz),
Color.White);
pointList[2] = new VertexPositionColor(new Vector3(px, py, pz),
Color.White);
pointList[3] = new VertexPositionColor(new Vector3(px, py - 200, pz),
Color.White);
pointList[4] = new VertexPositionColor(new Vector3(px, py, pz),
Color.White);
pointList[5] = new VertexPositionColor(new Vector3(px + 200, py, pz),
Color.White);
pointList[6] = new VertexPositionColor(new Vector3(px, py, pz),
Color.White);
pointList[7] = new VertexPositionColor(new Vector3(px - 200, py, pz),
Color.White);
BasicEffect effect1 = new BasicEffect(graphics.GraphicsDevice,null);
effect1.View = Matrix.CreateLookAt(cameraPosition,lookAtVector,
Vector3.Up);
effect1.LightingEnabled = true;effect1.Projection =
Matrix.CreatePerspectiveFieldOfView(
MathHelper.ToRadians(50.0f), aspectRatio,1.0f,
10000.0f);effect1.DiffuseColor = Color.LightBlue.ToVector3();
for (int x = 0; x < 3; x++){for (int i = 0; i < pointList.Length; i++){
pointList[i].Position.X += 0.05f;}for (int y = 0; y < 3; y++){
```

```
for (int i = 0; i < pointList.Length; i++){pointList[i].Position.Y +=
0.05f;}
effect1.AmbientLightColor =
Color.White.ToVector3();effect1.VertexColorEnabled = false;
effect1.Begin();foreach (EffectPass p in
effect1.CurrentTechnique.Passes){p.Begin();
GraphicsDevice.DrawUserPrimitives<VertexPositionColor>(PrimitiveType.Li
neList,
pointList,0,  4 );p.End();}effect1.End();}}base.Draw(gameTime);}}}
```