Fall 2018

Cognitive-model-driven pilot attention for commercial airline scenarios

Mathew Brian Cover
University of Iowa

COGNITIVE-MODEL-DRIVEN PILOT ATTENTION FOR COMMERCIAL AIRLINE SCENARIOS

by

Mathew Brian Cover

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

December 2018

Thesis Supervisor: Professor Thomas Schnell

To Grandpa Purdy, the man who gave me the inspiration to become an engineer ever since I was a child. He fed me a steady diet of computer books, toys, electronic kits and more, and I knew I wanted to be just like him when I grew up.

Ad astra per aspera
"To the stars through difficulties"

Latin Proverb

ACKNOWLEDGMENTS

I would like to thank Professor Tom Schnell for his guidance during my research. His experience and support was invaluable to completing my dissertation. I'd also like to extend my gratitude to Professor David Andersen. As my advisor for my master's thesis and academic advisor for my PhD, he has provided me guidance and helped pushed me toward the finish line over the last few years.

Next, I need to give acknowledgements to the graduate students and full-time employees at the Operator Performance Lab: Mike Yocius for blazing the graduate student trail ahead of me and giving me hope it can be done. Zeke Gunnink for allowing me to pick his brain whenever I ran into problems with writing code or debugging a nasty side-effect I may have discovered and/or introduced. Chris Reuter for being there as moral support from a fellow graduate student that understood the same stresses I was going through to get this finished.

Finally, I'd like to thank all of my friends and family who constantly gave me support during the periods when I was really struggling with writing or finding ways to stay motivated when I felt lost. You all listened to my thoughts and concerns, and that meant the world to me. Without your words of encouragement, support, and suggestions over the last few years, I would have surely failed during this time.

ABSTRACT

Bringing airline pilots to remote locations for evaluation of new software/hardware tools and procedures is an expensive process in terms of both money and time. Estimating the design and outcome of a study to evaluate these new tools can be tricky as there are many new variables for which there is little to no data. However, sometimes even after careful vetting of scenarios in the simulator prior to bringing subject-matter experts into the simulation facility, few to no metrics of statistical significance can be found. While it may be valid that there are no metrics of statistical significance, it is perhaps a missed opportunity to take advantage of the precious time and resources of having a subject-matter expert at the research facility.

The research presented in this paper has developed a software tool for simulating a pilot's visual perception of working in various configurations of cockpits. This may provide researchers insight into what types of scenarios and tactics would be of interest to use with real subject-matter experts. In other words, this should help identify the best use of resources to take advantage of having pilots at the facility and avoid scenarios/procedures that don't generate data of interest.

Another useful possibility with this tool is identifying cockpits that may be inefficiently designed. Instruments that should be grouped together can be easily identified by analyzing the eye-scan pattern of the model with different cockpit-configuration files. The results that this new software-evaluation tool provides have implications for several different evaluations beyond estimating pilot reactions.

PUBLIC ABSTRACT

Bringing airline pilots to remote locations for evaluation of new software/hardware tools and procedures is an expensive process in terms of both money and time. Designing scenarios to evaluate these new tools can be tricky as there are many new variables for which there is little to no data. However, sometimes even after careful vetting of scenarios in the simulator prior to bringing airline pilots into the simulation facility, few to no points of data that illustrate something interesting can be found. If nothing else, it is perhaps a missed opportunity to take advantage of the precious time and resources of having a pilot at the research facility.

The research presented in this paper has developed a software tool for simulating a pilot's visual perception in various types of aircraft. This should help researchers identify the best use of their time to take advantage of having pilots at the facility and avoid scenarios/procedures that could be a waste of time. Another useful possibility with this tool is identifying cockpits that may be inefficiently designed. The results that this new software-evaluation tool provides have implications for several different evaluations.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Statement of the Problem

Pilots can only make decisions and act on the information that is available to them from inside and outside the cockpit. Understanding how pilots acquire and prioritize information while conducting operations in aircraft is critical for addressing data gaps that might be formed in a pilot's mind.

This dissertation seeks to increase the understanding of how pilots acquire information using an approach that hasn't been used before. Fusing the Soar cognitive-modeling software with a complex aircraft environment in X-Plane, a tool for investigating such queries has been created to help answer these questions outside of a human-in-the-loop environment.

Cognitive modeling is a field that has evolved over the last several decades, starting with serious attempts by Card, Moran, Newell in the 1950s [1] to develop a model that would satisfy behavior prediction of humans.

Research in aviation safety often involves conducting studies wherein airline pilots are brought into a simulator environment to evaluate new procedures and displays/notifications and to give feedback on what is useful and what is not useful from the pilot's perspective. It is an expensive endeavor to bring airline pilots into a simulator facility as well as to staff the simulator for the duration of the study. It is important to maximize the product of the study for the time and resources put into it. In order to minimize waste while conducting such a study, elements of the proposed study could be screened for effectiveness, and some great cost savings could be realized. One method for

screening the design of the study would be to evaluate the scenarios in the simulation with an artificial airline pilot controlled by a cognitive model.

A cognitive model for an airline pilot would support decision making and interaction with a dynamic environment, in this case a commercial airline cockpit. This would allow researchers to identify scenarios that are of interest before actual airline pilots visit a simulation facility to participate in a study.

Background

There are a few different levels/layers of modeling a human in software: the physical layer, the cognitive layer, and the knowledge layer. The cognitive architecture layer exists at the interface between the physical layer and the cognitive layer. One of the goals of cognitive modeling is creating a model of human behavior that closely matches behavioral data such as error rates and reaction times.

Several models of human behavior have been developed and have evolved over the last half century. Psychology-based models are designed to emulate the architecture of human cognitive patterns to help decode stages of perception from knowledge retrieval through decision evaluation to motor action execution.

Each style of cognitive model that has been developed takes different feature and implementation approaches based on the researcher's goals for the use of the model. However, most of these models have many common elements that allow other researchers to focus on the tasks they are interested in rather than thinking about the lower-level programming details needed to implement the cognitive capabilities from scratch. The breadth of models developed and their applications will be explored in Chapter 2.

Objectives

The objective of this dissertation research is the development and verification of a cognitive model of a commercial airline pilot. Using X-Plane as the simulator environment, the pilot model will be exposed to different types of scenarios and the data collected from the model will be compared against data collected from human pilot participants in previous studies. After the comparison of the data sets between the cognitive model and actual human participants, an evaluation of future use of the cognitive model will be performed and elaborated upon.

Influences on Architecture Design

There are three primary components that influence the design of the cognitive architecture: environment, tasks, and agent structure. For this research topic and scope, the environment will be confined to the cockpit of a Boeing 737-800 simulator with the notion that this could be expanded to many different types of simulated cockpits in the future. The tasks for this project will be kept to standard in-flight procedures such as small changes in altitude and heading, but as with the environment, the tasks could be expanded upon in the future. The agent structure will be covered extensively in Chapters 3 and 4 of this research, but the key components are that it has interfaces to its environment and the knowledge to complete aviation-related tasks.

For this research, the "end state" or goal of the agent flying the airplane is not explicit since the goal is just to maintain certain aspects of a state. While explicit representation can be useful in choosing which operators to select, it is not a requirement [2] for this model.

This iteration of the model is not expected to produce scientifically significant results for any one specific study at this time, but it does demonstrate that the framework for a cognitive model for airline pilots is accurate for a number of use cases. This model should be able to be expanded upon to achieve a higher fidelity for future uses to predict pilot performance and, eventually, workload of pilots in a wide variety of situations. Common effects of eye-scan pattern behavior that has been of research interest for decades include fixation durations, number of fixations, saccadic movements, and scan pattern changes. Definitions of eye-scan terms and relevant metrics can be found in the Table 1 below.

Table 1: Definitions of eye-related metrics

Term	Definition
Area of Interest	An area of interest is a region over a field of view that is significant due to a particular source of information that may be available there for a human to look at. In an airplane cockpit, there could be several areas of interest, such as the primary flight displays, mode control panel, EICAS, and flight management system.
Fixation Frequency	Fixations that are "a relatively stable eye-in-head position within some threshold of dispersion (~2 degrees) over some minimum duration (200ms), and with a velocity threshold of 15-100 degrees per second" [3]. This shows a positive correlation to subject workload similar to fixation total.

Table 1 - continued

Gaze (or Dwell Time)	Gaze is similar to fixations in that it is the grouping of fixations within a single region of interest. The area around which a gaze is calculated is dependent on the size of the area of interest.
Saccade	A saccade is the movement of the eye from one fixation to the next. The speed of a saccade is obtained by calculating the distance from one fixation to the next and calculating the time difference to result in an angular velocity in degrees per second.
Link Analysis	A link analysis measures the relative strength among transitions of fixations between any two areas of interest. This was a useful tool that Fitts et al. [4] used in their study of eye movements of pilots during instrument flight.

The research presented in this paper will also perform a link analysis and an analysis of region of interest of the gaze of the software pilot model for different tasks. The results of this analysis are covered in Chapter 5. Finally, Chapter 6 summarizes the effort of developing this cognitive model as well as future work that can be pursued to develop this capability further for projects outside of commercial aviation.

CHAPTER 2: HUMAN MODEL FRAMEWORKS

Cognitive architectures instantiate "united theories of cognition" consisting of implementations of theories of the mechanisms utilized by humans in processing information. Since Newell articulated the goal of United Theories of Cognition in his prominent book [5], a number of cognitive architectures have been developed, with varying degrees of depth and breadth of model applications [6]. This section will provide a quick overview of a few of the more common models used in human modeling.

There are different types of goals of cognitive architecture research which can generally be grouped into three categories: biological modeling, psychological modeling, and AI functionality. Table 2 below illustrates the different groupings of cognitive architectures and which time scales each type is best suited for.

Table 2: Newell's Time Scale of Human Action

Scale (sec)	Time Units	System	Cognitive Category
10^{7}	Months		Social
10^{6}	Weeks		Social
10^{5}	Days		Social
10^{4}	Hours	Task	Rational
10^{3}	10 Minutes	Task	Rational
10^{2}	Minutes	Task	Rational
10^{1}	10 Seconds	Unit Task	Cognitive
10^{0}	1 Second	Operations	Cognitive
10^{-1}	100 ms	Deliberate Act	Cognitive
10^{-2}	10 ms	Neural Circuit	Biological
10^{-3}	1 ms	Neuron	Biological
10^{-4}	100 us	Organelle	Biological

There are several different modeling methods and scopes that cover several bands of this scale. No one modeling method encompasses all of the bands however. To

sufficiently model the behavior of an airline pilot, one does not need to get down to the level of modeling neurons firing as there is not much intelligent action going on at that level. These biological bands on the lower end of the time scale model what we know about the brain: neurons, neural circuits, etc. They predict neural activity and cognitive behavior. Two examples of models that detail this part of the spectrum are LEABRA [7] [8], and SPAUN [9].

Next there is psychological modeling tools and methods that target human performance in a wide range of cognitive tasks. They predict human reaction time and error rates for psychological tasks such as ACT-R [10] [11], EPIC [12], CLARION [13], LIDA [14], CHREST [15], and 4CAPS [16].

Finally, there are modeling techniques that emulate AI functionality and moves closer toward human-level intelligence inspired by psychology and biology. The emphasis is more on complex cognitive processing and longer time-scales (upwards of hours). Examples of these models include Soar [17], Companions [18], Sigma [19] [20], ICArUS [21], and CogPrime [22].

Several candidate modeling techniques will be reviewed in the remainder of this chapter. Their role in the development of human performance modeling is illustrated as well as their benefits and drawbacks to being used in modeling pilot performance in commercial cockpits.

GOMS

In 1983, Card, Moran, and Newell published a book titled *The Psychology of Human-Computer Interaction* that helped bridge the gap for cognitive modeling between

the science and useful applications. They recognized that the human-computer interface at the time was rapidly becoming the most important domain in human factors practice [1]. One of the more prominent models put forward in this book is called GOMS, or Goals, Operators, Methods, and Selections rules. It is a specialized human-information-processor model for human-computer interactions.

Most of the earlier models were created with desktop computer applications in which the computer only does something in response to an action by the user. Aircraft cockpits don't have such simple conditions; forces and other agents external to the pilot's control, including weather, mechanical systems, air traffic control, and traffic often dictate the pace of the tasks to the pilot.

GOMS reduces a user's interactions with a computer to their elementary actions (physical, cognitive, or perceptual). Goals are what the user intends to accomplish. Operators are actions that are performed to reach the goal. Methods are sequences of operators that accomplish a goal. There can be more than one method available to accomplish a single goal. If this is the case, then selection rules are used to describe when a user would select a certain method over the others.

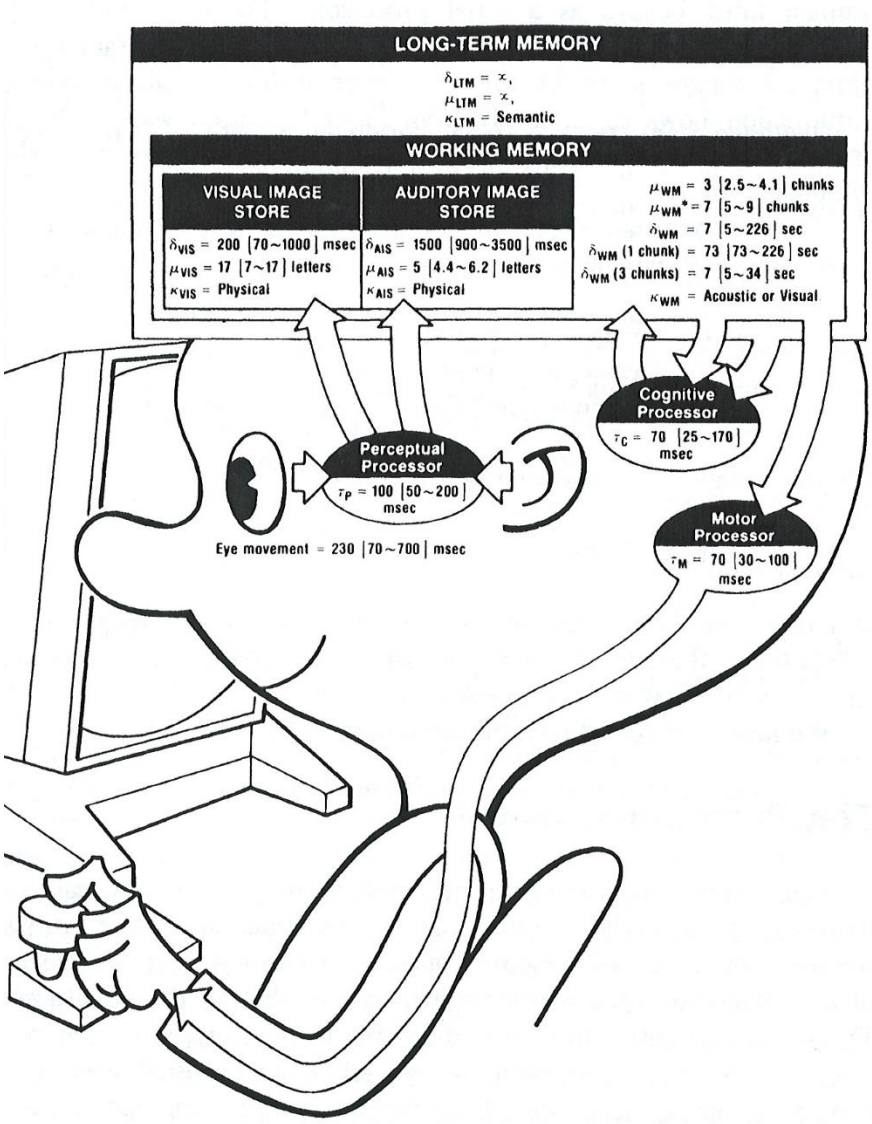


Figure 1: Model Human Processor - memories and processors [1]

A GOMS estimate of a particular interaction can be calculated with little effort, at little cost, and in a short amount of time if the average methods-time measurement data for each task has been previously measured experimentally to a high degree of accuracy. With a careful investigation into all of the detailed steps necessary for a user to successfully interact with an interface, the measurement of how long it will take a user to successfully interact with an interface is a simple calculation. Summing the times

necessary to complete the detailed steps provides an estimate for how long it will take a user to successfully complete the desired task. A sample of time estimates can be found in Figure 1.

None of the techniques address user unpredictability – such as user behavior being affected by fatigue, social surroundings, or organizational factors. The techniques are very explicit about basic movement operators, but are generally less rigid with basic cognitive actions. In the real world, splits cannot be prevented, but none of the GOMS models allow for any type of error. Further, all of the techniques work under the assumption that a user will know what to do at any given point – so they apply only to expert users, not novices.

User personalities, habits, or physical restrictions (for example, disabilities) are not accounted for in any of the GOMS models. All users are assumed to be exactly the same. Extensions of the baseline GOMS model have been developed to allow the formulation of GOMS models describing the interaction behavior of disabled users.

Except for Keystroke Level Modeling (KLM), the evaluators are required to have a fairly deep understanding of the theoretical foundations of GOMS, Cognitive Complexity Theory (CCT), or Model Human Processor (MHP). This limits the effective use of GOMS to large entities with the financial power to hire a dedicated human-computer interaction (HCI) specialist or contract with a consultant with such expertise.

Variants of GOMS

The plain, or "vanilla flavored" GOMS first introduced by Card, Moran, and Newell is now referenced as CMN-GOMS. Keystroke Level Modeling is the next GOMS

technique and was also introduced by Card, Moran, and Newell in their 1983 book [1]. This technique makes several simplifying assumptions that make it really just a restricted version of GOMS. The third major variant of the GOMS technique is the Natural GOMS Language (NGOMSL). This technique gives a very strict, but natural, language for building GOMS models. The final variation of GOMS is Cognitive Perceptual Motor-GOMS (CPM-GOMS). This technique is based on the MHP. The main advantage of CPM-GOMS is that it allows for the modeling of parallel information processing by the user; however, it also is the most difficult GOMS technique to implement.

Importance of GOMS Analysis

Before applying the average times for detailed functions, it is very important that an experimenter make sure he or she has accounted for as many variables as possible by using assumptions. Experimenters should design the GOMS analysis for the users who will most likely be using the system that is being analyzed. Consider, for example, that an experimenter wishes to determine how long it will take an F22 Raptor pilot to interact with an interface he or she has used for years. It can probably be assumed that the pilot has outstanding vision and is in good physical health. In addition, it can be assumed that the pilot can interact with the interface quickly because of the vast hours of simulation and previous use he or she has endured. All things considered, it is fair to use "fastman" times in this situation. Contrarily, consider a 65-year-old individual with no flight experience, let alone fighter pilot experience, attempting to interact with the same F22 Raptor interface. It is fair to say that the two people would have much different skill sets, and those skill sets should be accounted for subjectively.

Cognitive Perceptual Motor-GOMS

The CPM-GOMS was developed by Bonnie E. John while at the school of Computer Science at Carnegie Mellon University in 1988. Bonnie John was a student of Allen Newell. Unlike other GOMS variations, CPM-GOMS does not assume that the user's interaction is a serial process, and hence it can model multi-tasking behavior that can be exhibited by experienced users. The technique is also based directly on the MHP – a simplified model of human response.

The tasks are first joined serially and then examined to see which actions can be overlapped so that they happen in parallel. This technique facilitates representation of overlapping and the very efficient "chunks" of activity characteristic of expert users. The times estimated by CPM-GOMS are generally faster since they do not allocate as much time to the "prepare for action" type of operations.

This is the most difficult GOMS technique to implement. Therefore, it has the problem of discrepancies between evaluators. Research is currently being conducted to improve the CPM-GOMS technique so that it can be used without the evaluator having a high-level understanding of the GOMS theoretical foundations.

CogTool

A team at Carnegie Mellon University, headed by Bonnie John, has created an open-source tool to support KLM analysis. CogTool was developed to enable low-cost, rapid construction of interactive prototypes that focus on systems involving deliberate commands that the user invokes by some motor action [23]. It automatically evaluates user interface designs with a predictive human performance model (a "cognitive crash dummy"). Types of systems include cell phones, handheld terminals, in-vehicle driver

information systems, and computers that run desktop applications. Cog Tool has increased the accuracy of the KLM because it applies the theory more consistently through its "modeling by demonstration" approach, and has been reported to be within about 10% of empirical data [24].

ACT-R

The Adaptive Control of Thought-Rational (ACT-R) is a cognitive architecture developed at Carnegie Mellon University. ACT-R is a proposed unified theory of cognition realized as a production system designed to predict human behavior by processing information and generating intelligent behavior itself. In short, the ACT-R model for cognition tries to provide a comprehensive explanation for high-level cognitive control behavior. ACT-R is a computation cognitive architecture that takes as inputs knowledge about how to do the task, both procedural and declarative, and a simulated world or environment in which to run.

ACT-R has two types of long-term memory: declarative and procedural. Declarative memory defines things that are factual in nature, such as "George Washington was the first president of the United States" and "2+3=5." The basic unit of declarative knowledge is known as chunks. Procedural knowledge consists of production rules that encode skills and take the form of condition-action pairs (if/then statements). These production rules correspond to goals or sub-goals and mainly consist of retrieval and storage of declarative knowledge.

In ACT-R, a chunk's activation decreases as a function of time since the chunk was created and increases with the number of times the chunk has been retrieved from memory. When retrieving items from memory, ACT-R looks at the most active chunk in

memory; if it is above the threshold, it is retrieved, otherwise an "error of omission" has occurred, i.e., the item has been forgotten.

An important aspect of the ACT-R system is that it operates in real time: Each covert step of cognition (production firing, retrieval from declarative memory) or overt action (mouse-click, moving attention) has latencies associated with it that are based on psychological theories and data. For example, firing a production rule typically takes 50 milliseconds, and the time needed to scan a part of a computer screen is calculated using Fitt's law. In this way, the system allows application of psychological knowledge in real time.

Fitt's Law:

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right)$$

where:

- *T* is the average time taken to complete the movement.
- *a* represents the start/stop time of the device.
- *b* represents the inherent speed of the device.
- *D* is the distance from the starting point to the center of the target.
- *W* is the width of the target measured along the axis of motion. *W* can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within +/- *W*/2 of the target's center.

From Fitt's Law, the speed-accuracy tradeoff can be seen where targets that are smaller and/or further away require more time to acquire.

ACT-R/PM

ACT-R is used primarily to model experiment psychology data. However, there is a version of ACT-R that has also been used to model behavior in synthetic environments. It was developed at Carnegie Mellon University and is called the ACT-R Perceptual Motor (ACT-R/PM) model. The enhancements provide any model created with ACT-R/PM the ability to interact with a simulated device such as a computer, driving/flight simulator, video game, etc.

The research from [25] developed a computational model of a closed-loop, pilot-displays-aircraft system designed to evaluate the impact of the addition of a synthetic vision system (SVS) to a commercial airliner cockpit on pilot's attention-allocation behaviors. ACT-R was used for the pilot model and was coupled to the flight simulator package, X-Plane, via a low-level UDP network connection.

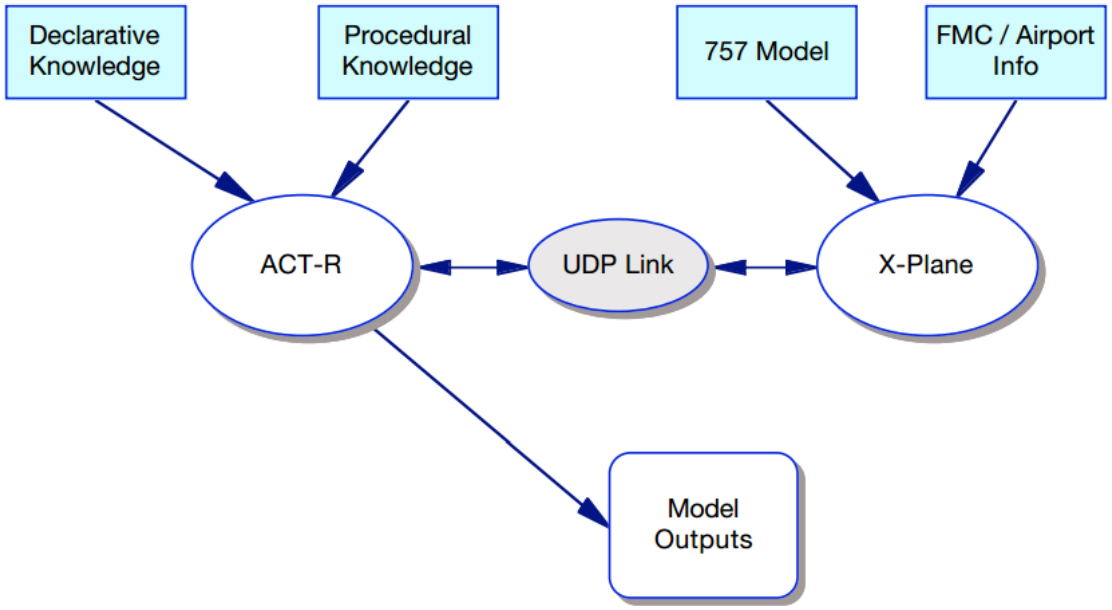


Figure 2: Byrne's ACT-R pilot model overview [25]

Soar

Soar originally stood for State, Operator And Result, and was created by John Laird, Allen Newell, and Paul Rosenbloom at Carnegie Mellon University, and is now maintained by John Laird's research group at the University of Michigan. The Soar development community, over time, no longer regarded Soar as an acronym, which is why it is no longer written in upper case. At Larid's research group, graduate students work on both cognitive model architecture improvements to Soar as well as finding new applications for Soar.

The views of the cognition underlying Soar are tied to the psychological theory expressed in Allen Newell's book, *Unified Theories of Cognition* [5]. Soar is based on a production system, much like ACT-R, using production rules similar to the form of "if…then…" conditions. Solving of problems in Soar analyzes the problem space (achievable states that can be reached by the system) for a goal state (the solution of the problem). A search is conducted to find a solution that brings the system closer to its goal state.

While Soar can be used to help understand cognition and solve problems with production rules, it has a documented programmer interface that provides a means to allow Soar agents to interact with external environments. It is this feature that made Soar extremely desirable to develop the SoarEye pilot perception module.

There are several key abilities for controlling aircraft simulations that do not coincide with the strengths of production systems, such as performing a large amount of numerical calculations and optimization problems. These tasks need to be off-loaded to an external interface that can be written in more traditional languages such as C++, Java,

and Python. The Soar Markup Language (SML) allows simplified interaction of a Soar agent with external environments and other software systems that complement Soar's strengths (e.g., neural networks, state estimation techniques, and object recognition methods).

The SML protocol has been used for giving robots a form of cognition by allowing them to make decisions based on sensor input from their surroundings. The Cognitive Robots System (CRS) is an excellent example [26] in the last few years of how embedded systems can take advantage of a production system architecture to create a symbolic representation of the environment to a Soar agent.

For the CRS, three sensors and two infrared sensors are used to detect obstacles in front of and to the sides of the SuperDroid. Wheel encoders installed on the two front wheels are used to estimate position of the SuperDroid. Two web cameras are also installed on the SuperDroid and implemented as a stereo pair to measure distances to edge pixels. A laptop onboard receives all the sensor information and concentrates it for the Soar model to process on the input branch of its working memory [9].

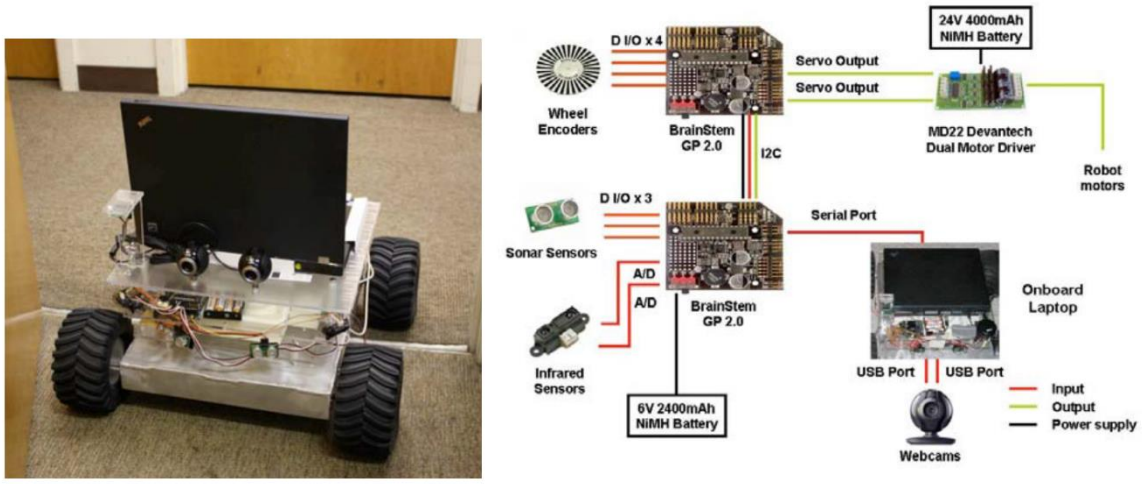


Figure 3: The SuperDroid and a schematic of the hardware used on the SuperDroid for map generation [26]

Jones et al. [27] have used the Soar architecture to autonomously fly U.S. military fixed-wing aircraft during missions in a simulated environment for the TacAir-Soar project using Soar agents with 5200 production rules, 450 total operators, and 130 abstract operators. These agents were used as AI entities in simulated war games to help create large-scale battles without requiring a large number of people. This helped demonstrate the scalability of Soar to thousands of rules, due to the use of the Rete algorithm [28]. The TacAir-Soar project also demonstrated that Soar is capable of performing high-level activities, such as reasoning, using large agents in a simulated environment with real-time constraints. Since cognitive architectures are able to model general decision making, a single Soar agent can be used for multiple missions and can be capable of using multiple approaches to the same problem.

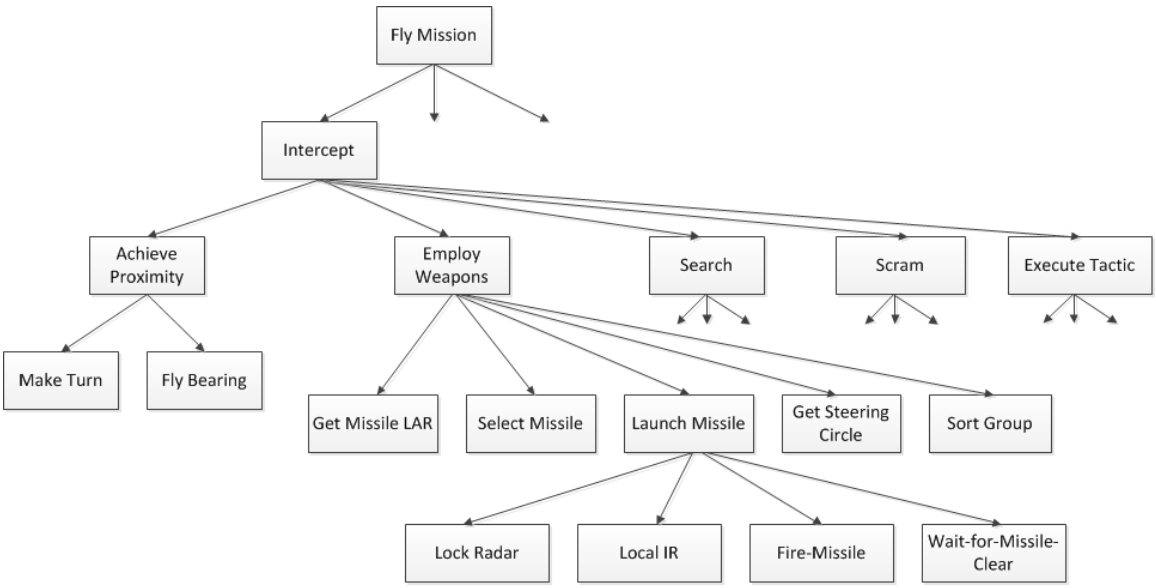


Figure 4: Example part of the goal hierarchy of TacAir-Soar

Other Considerations

In software engineering, different steps use one or more representations [29]. For instance, data flow diagrams (DFDs), entity life histories (ELHs), entity relationship diagrams (ERDs), and process outlines (POs) are used in Structured Systems Analysis and Design Methodology (SSADM), in the same way as use cases, activity diagrams, and interaction diagrams are representations used in the Unified Modeling Language (UML). It would therefore seem desirable for task analysis output representations to be in the form of one or more of such software engineering representations or to be easily translated into them. However, many task analysis methods [29] have been developed by researchers with a psychological background and have focused mainly on the first steps of user interface design, so their outputs often do not integrate well with those from software engineering.

The classical software engineering distinction (adapted from IEEE-STD-610) is between verification (verity that a clearly specified problem is solved properly) and

validation (ensuring that what is done meets the actual requirements). Verification is generally made by semi-automatic tools, while validation requires human interpretation.

Neural Networks

Another type of cognitive model is a neural network. This has become more popular in the last couple of decades due to the processing power that has become available in computers. Neural networks are "trained" to become highly skilled at one or two tasks. These networks are fed massive amounts of historical training data from previous experts conducting the same task that the network then "learns" from to adjust the weights and biases of the network of artificial neurons. Training neural networks is a time-intensive task that can take hours, days or weeks even with just a few variables to observe.

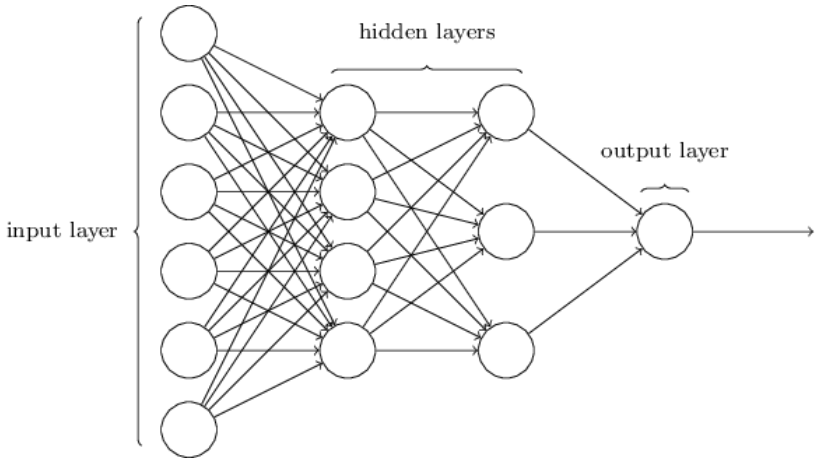


Figure 5: Simple neural network illustration

In Figure 5, each circle represents a 'neuron' of a neural network. The leftmost layer is called the input layer of the network and the rightmost, or output layer, which in this illustration is just a single output neuron. The middle layers are called the "hidden

layer" since they are not exposed to the world like the input or output layer is. In more complex neural networks, there are several hidden layers in the middle of the network.

Since the expertise of a neural network is only as good as the training data that is fed into it, extreme caution must be used in how neural networks are applied to general problem solving. The neural network (named AlphaGo) that bested the world best Go players in 2016, is highly skilled at playing the game Go, but would fail at chess, or any other board game, for that matter. In much the same way, IBM's Watson can become highly trained to defeat the best human players in *Jeopardy!*, but would in no way become a medical professional using the same training dataset.

CHAPTER 3: SOAR BASICS

Due to previous successes with Soar design being used in task execution duration on the order of minutes to hours and the relative ease of interfacing Soar agents with external environments, Soar was selected as the cognitive architecture to use in this research endeavor. In this chapter, a quick summary of the core components of Soar: production memory, reinforcement learning, substates and impasses, chunking, as well as semantic and episodic memory are presented. This will prepare for coverage of how the SoarEye tool was constructed in the following chapter.

Production System Components

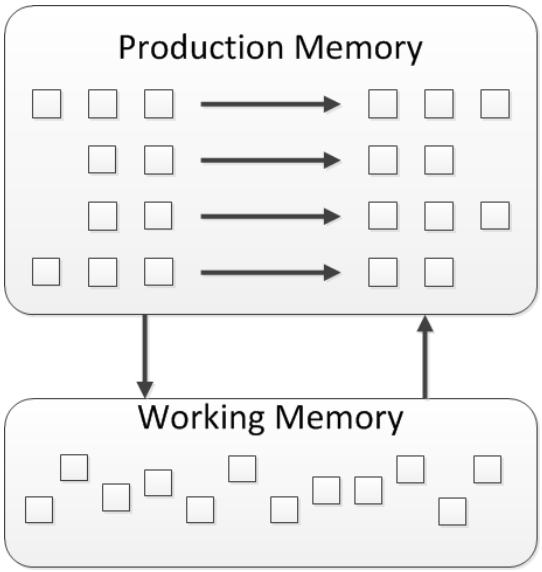


Figure 6: Production System Structure [2]

Long-term procedural knowledge is implemented in Soar as a series of if-then rules, or production rules [30]. Figure 6 contains a block diagram of the structure of a generic production system. The “if” part of a rule consists of conditions (on the left side of the arrow in the long-term memory block) that must be satisfied by working memory

for the rule to match. The "then" parts of the production rules are actions that add, modify, or delete structures in working memory.

Preferences for production rules can be classified in Soar as seen in Table 3.

Table 3: Symbolic preference for selecting operators

Preference	Syntax	Meaning
Acceptable	+	All operators with acceptable preference are collected into the candidate set
Reject	-	Any operator in the candidate set with a reject preference is removed from the set
Better/worse	>>/<<	All operators in the candidate set that are worse than another operators in the candidate set are removed from the set
Best	>	If there are any operators in the candidate set with best preference, then all operators without best preference are removed from the candidate set
Worst	<	If there are any operators in the candidate set without worst preference, then all operators with worst preference are removed from the candidate set
Indifferent	=	Indifferent preference specifies that all operators are equally good and a random selection can be made between them

Working Memory

Working memory contains all of a Soar agent's dynamic information about its world and its internal reasoning. It contains sensor data, intermediate calculations, current operators, and goals. In Soar, every element in working memory, or a working memory element (WME), consists of three parts, an identifier, attribute, and value (which can be a constant or another identifier) (See Figure 7). All of working memory is organized as a graph structure in states that are connected and directed [31]. Every WME therefore is connected directly or indirectly to a state symbol (see example in Figure 8).

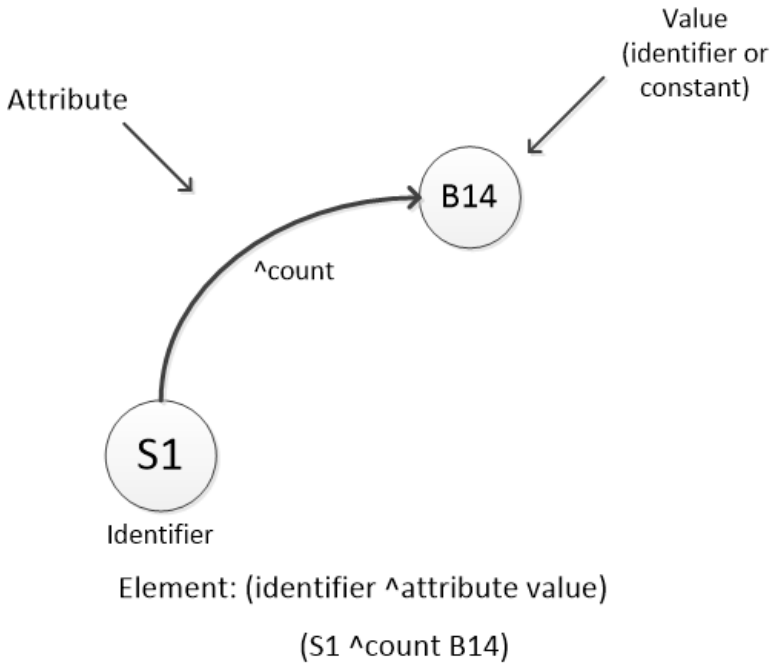


Figure 7: Example of a Working Memory Element Structure

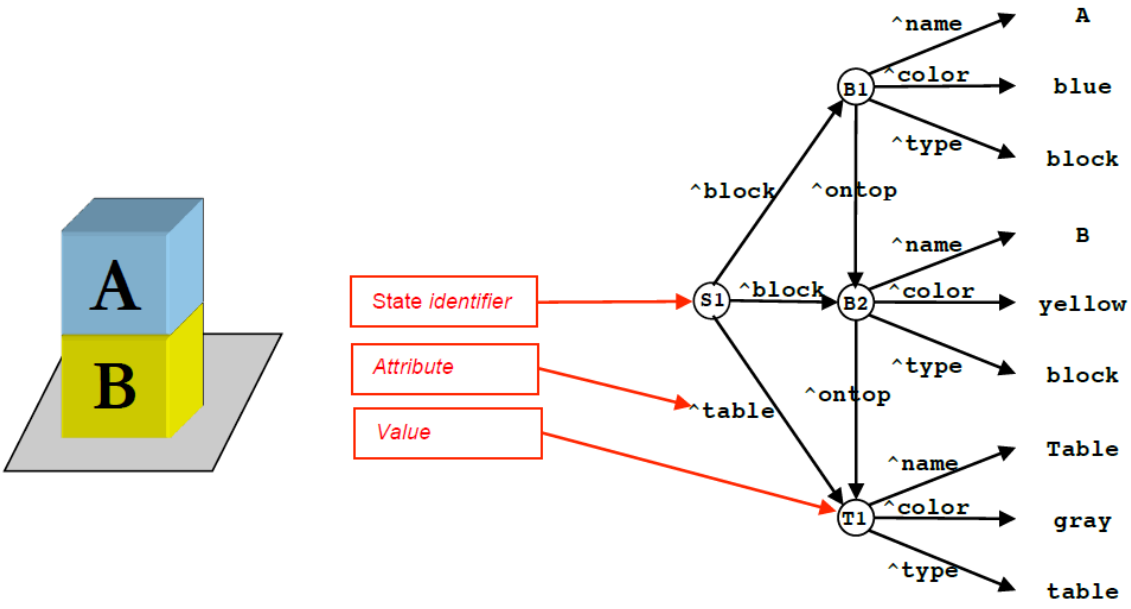


Figure 8: Example of structure of working memory [31]

The Soar Processing Cycle

The Soar processing cycle, presented in Figure 9, shows the high-level sequence of events when a Soar agent evaluates a situation. All the processes represented by rectangles are performed by production rules. The round-cornered rectangles in Figure 9 are fixed task-independent processes. The input and output phases provide Soar's means to interact with an environment, while the decision phase chooses the current operator.

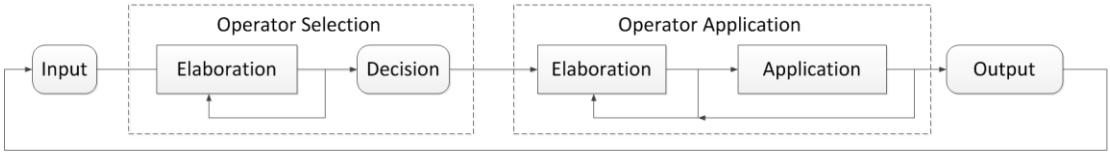


Figure 9: The Soar processing cycle

Input Phase

During the input phase, new working-memory elements are added to reflect changes in perception. To extract relevant sensory information from the external environment (simulated vision, hearing, touch, or another type of interaction with another program) and transfer it to working memory, a perception module must be written in a language that interfaces with Soar, such as C++, Java, C#, or Python [2]. Similarly, to initiate commands in an external environment, an output module must be created.

The interface with perception and output systems is via working memory through the input-link and output-link structures, which are substructures of the input-output (io) state structure [2]. By having an area reserved for external input, Soar can distinguish between structures created by perception and those generated internally by its own reasoning.

Decision Procedure

The results of the decision procedure are either the selection of a new operator or an impasse, which are discussed later in this chapter.

Parallel Operators

In Soar, only one operator can be selected at a time, forcing a sequential bottleneck. This design decision dates back to the original tasks implemented in Soar in which an operator's taking a step in a problem space would generate a new state. With that strategy, parallel operators would have generated multiple new states, pushing selection knowledge to the selection of the next state.

Once Soar gained the capability to interact with external environments, the restriction against parallel operators was maintained on the grounds that parallel operators could have conflicting actions that would be difficult for an agent to detect and recover from. Even with that restriction, there are multiple ways to generate parallel action in Soar:

- Operator switching: As long as Soar is fast enough relative to the environment (which is the case in most uses), Soar can switch back and forth between operators that initiate independent external actions, giving the appearance of parallelism in the environment, even though only one operator is selected at a time [2].
- Overlapping output actions: for temporally extended actions that do not require constant cognitive attention, multiple actions can be initiated either through a sequential operator application or through mega-operators.

Goal Detection

There is no separate phase for goal detection in Soar. The knowledge to detect the achievement of a goal can be encoded either as state elaborations or as a separate operator. For more complex agents that can pursue multiple goals, goals can be created by operators or operators can act as goals by being selected, but where there are no application rules, the achievement of the operator is pursued in a substate (discussed later in this chapter) so that the operator becomes a goal.

The basic processing cycle starts by matching conditions of production rules against the contents of working memory. This matching process determines which production rules have all of their conditions satisfied by the elements in working memory. If a rule tests whether working memory includes a representation of a blue block and a yellow block, and those elements exist in working memory, the rule matches.

The results of the matching process is a set of rule instantiations. There is one rule instantiation for each successful match of a rule to working memory. The match process can also compute which rules that previously matched no longer match. Although not used in many rule-based systems, this ability to detect when a rule retracts is built into Soar.

The major computational cost of a production system is matching the rules against working memory. The naïve approach is to compare all of the conditions of all the rules to all elements of working memory on each cycle. This is an expensive approach to matching rules, as the cost would be W^{C*R}, where W is the number of elements in working memory, C is the average number of conditions in each rule, and R is the number of rules.

The Rete algorithm was designed to avoid this problem. Instead of matching all conditions to all of the working memory in each cycle, Rete processes only the changes in working memory. This method explicitly trades space for speed; Rete maintains a memory of all partial matches for all rules and processes the changes in working memory to update the partial matches and determine which rules completely match and which no longer match. Given that there usually are only a small number of changes in working memory during each cycle and that those changes only affect a small number of rules, it is possible to create rule matchers that can efficiently process a very large number of rules.

Instantiation and Operator Support

The persistence of WMEs is determined by the type of rule that selected them. A rule instantiation that does not test the current operator and makes changes in the state will remove all WMEs it created whenever any of the WMEs tested in the conditions are removed from working memory. This is called instantiation-support, or i-support.

The other class of support, which leads to persistent WMEs, is called operator-support, or o-support. All WMEs created by an operator-application rule have o-support and persist until they are removed by the actions of the rule, or when they become disconnected from the state through removal of other WMEs.

Reinforcement Learning

Reinforcement learning (RL) is one of the core tasks in machine learning (ML). It is one of the two main primary methods of learning procedural knowledge, as seen in Figure 10. The other method is called "chunking" and will be reviewed later in this chapter. Reinforcement learning allows an agent to modify or tune existing rules by adjusting numeric preferences in operator-evaluation rules.

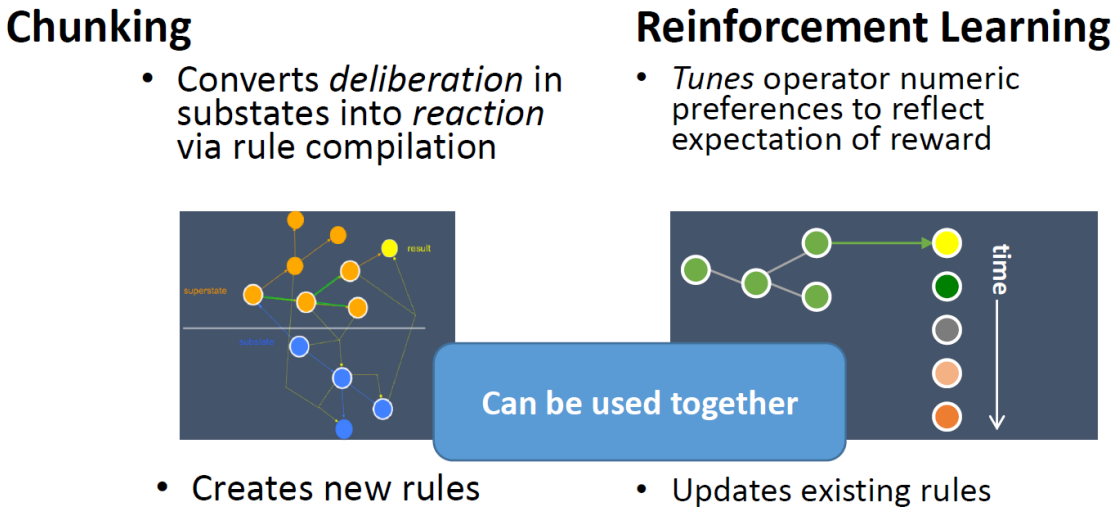


Figure 10: Methods for Learning Procedural Knowledge [31]

RL algorithms are dependent on making adjustments to rules over long periods of time and not from a single experience. This allows an agent to become robust to noise with the agent's interaction with the environment in which it is operating in over time. A simple illustration of the RL process cycle can be found in Figure 11. Inside the agent, there is a value function, called the Q function, which maintains the current expected reward for every action. Referring to this value function, the agent tries to select an action that will maximize the future reward.

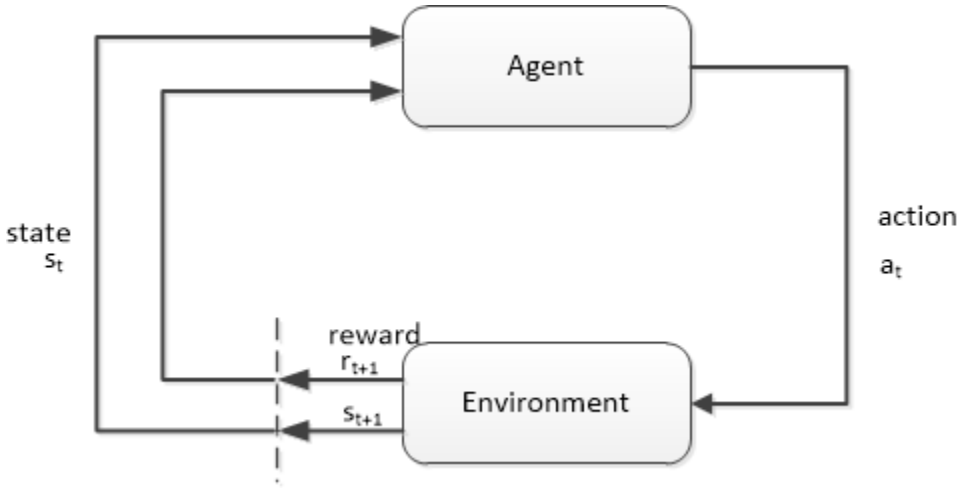


Figure 11: Reinforcement Learning Cycle

All the numeric preferences created for an operator are added together to evaluate the Q value for that operator. In Soar, the symbolic preferences are given priority as that provides boundaries of the desirability of the operators. Then after the symbolic preferences have filtered out the desired operators, the Q values are used to select from the remaining operators. Figure 12 shows an example of three operators and their calculated Q values. In this example, O1 would be the most likely selected operator, although O2 and O3 are still possible as well. The exact probability of operator selection depends on which selection scheme is chosen by the agent designer.

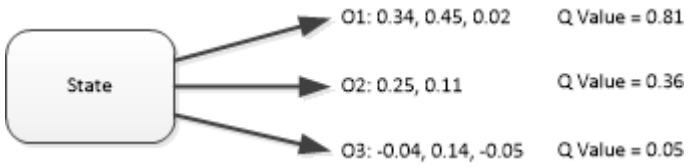


Figure 12: Example calculation of Q Value for three operators (O1, O2, and O3)

Impasses and Substates in Soar

If an agent always has sufficient knowledge, the model will just do the task at hand. There is no planning or simulation of external actions, no reasoning/simulation

about other agents/entities, no subgoals for task decomposition, and no reasoning about reasoning (metacognition). Metacognition arises from scenarios where there is insufficient or conflicting knowledge and the model has to take a step back and create a separate state from which to reason (without disrupting the original reasoning). This process of learning compiles metacognition into direct knowledge for future situations.

An impasse arises if there is insufficient/conflicting procedural knowledge to select an operator. In Soar, there are four ways in which an impasse can manifest itself:

- [state no-change] No operator is proposed
- [operator tie] Multiple operators are proposed by insufficient preferences to select between them
- [operator no-change] An operator is selected, but it can't be applied by a single rule
- [operator conflict] Multiple operators are proposed with conflicting better/worse preferences

To resolve any of these three types of impasses, a substate is created. The substate created is a framework for deliberate reasoning and accessing additional knowledge sources (long-term memories, external environment, or internal reasoning (planning)) to resolve the impasse. An impasse is considered resolved when results, sometimes through recursive impasses, lead to a decision.

Figure 13 illustrates the WMEs that are created for a tie impasse among three operators (O31, O32, and O33) in state S20. Soar creates the substate identifier S23 and the WMEs as well as augmentations with the type of impasse (no-change, conflict, or tie)

and an “attribute” augmentation that indicates if the cause is “state” or “operator.” Substates in Soar are managed by the architecture and cannot be created/modified/deleted by rules.

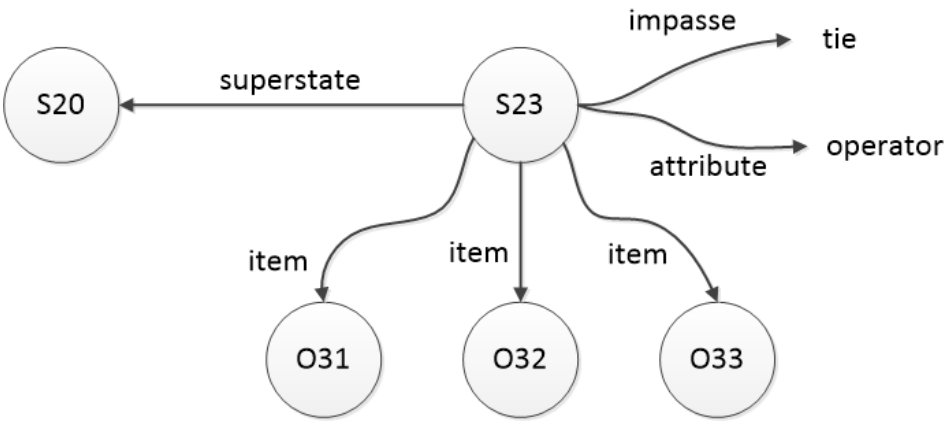


Figure 13: Substate structures

Chunking

The addition of using substates to resolve impasses is useful, but the problem is that, by itself, the knowledge discovered with the substates is lost after each problem solving episode. Chunking is Soar’s first learning mechanism and is the capability to build new rules that summarize processing. These new “chunks” of rules are built as soon as a result is produced. There is one chunk for each result, where a result consists of connected WMEs that become results at the same time. Soar will only learn what it “thinks” about and is impasse driven; in other words, learning arises from a lack of knowledge.

Semantic and Episodic Memory

Semantic memory in Soar is designed to support deliberate storage and retrieval of long-term “objects,” features, and relations. Semantic memory is similar to working

memory in that it consists of symbolic triples (Figure 14); however, attributes cannot be identifiers, and the resulting graph is not necessarily connected. Semantic memory is disabled by default in Soar and needs to be explicitly turned on by a user of Soar. Agents can acquire and store semantic knowledge either manually via a user using the command line (especially useful for loading larger external data sources), or via deliberate (via rules) addition/modification by the agent.

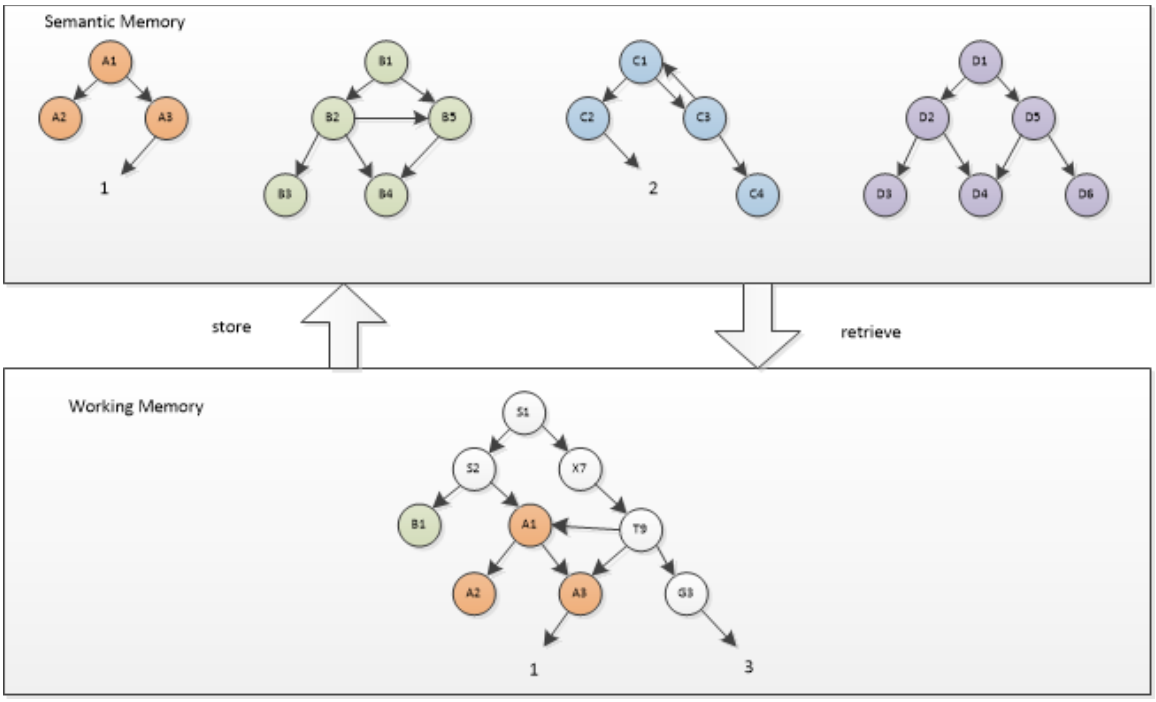


Figure 14: Example of interaction between working memory and semantic memory

Since semantic memory can become very large, it was implemented with a SQLite backend so it has the ability to save semantic stores to disk and use disk-based databases. However, this often causes semantic memory to be the slowest portion of most Soar models.

Episodic memory is a form of a weak learning mechanism in Soar. When enabled, it automatically captures, stores, and temporally indexes agent state to create an

autobiographical prior experience memory structure. Episodic memory is what you "remember," and semantic memory is what you "know." As with semantic memory, episodic memory representation is similar to working memory in that it consists of symbolic triples and attributes cannot be identifiers (Figure 15). Structures within an episode are connected; separate episodes are disconnected.

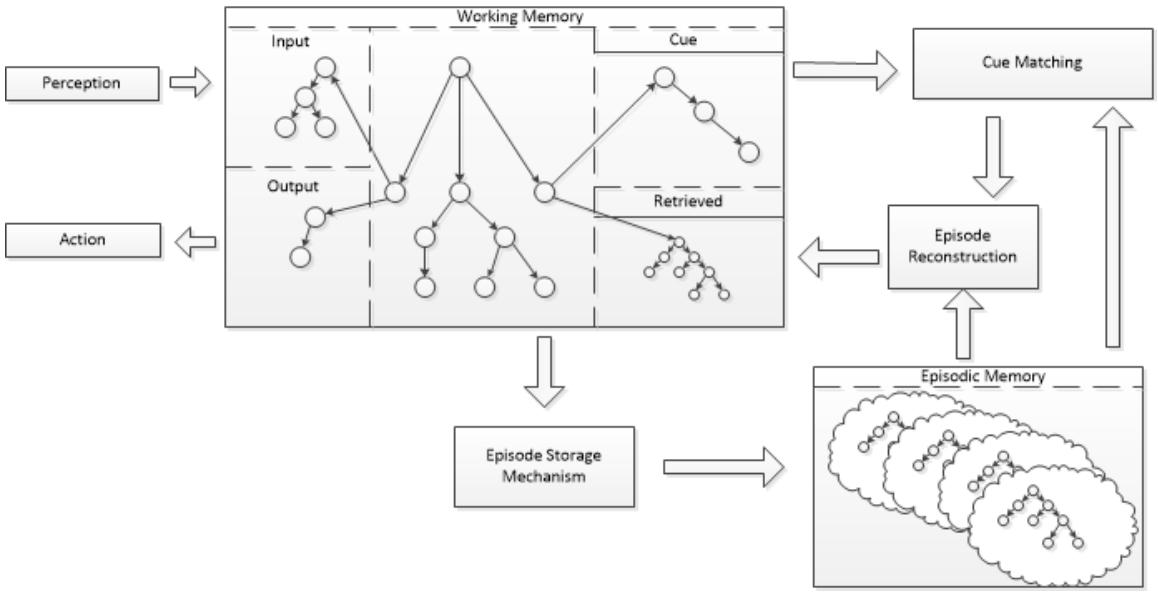


Figure 15: Processing and memory modules supporting episodic memory

Episodic memory could have a role in a pilot model by storing and retrieving times when it last saw a specific configuration of the autopilot to anticipate the outcome/behavior and environmental dynamics in similar situations. Just as is the case with semantic memory, the goal of episodic memory is to support a form of long-term memory that interacts with working memory for real-time agents that have long lifespans (hours to days).

CHAPTER 4: SOAREYE ARCHITECTURE AND BEHAVIOR

This chapter explores the development of the eye-movement behavior for the pilot model. There are some constraints to the cognitive model software that make it easier to unload some tasks to external pieces of software. The mechanics of eye movement are math oriented, and Soar is better suited for non-math behavior. The software tool SoarEye, which brings everything together, was designed to deal with the eye-movement mechanics. The commands for eye movement still come from Soar, but the implementation of how the eyes move from point A to point B is done inside of SoarEye and not the cognitive model.

Data Publishing

First, the eye-movement model needs to capture the metrics needed for analysis. Ideally, they should match the metrics that would be collected from actual airline pilots. The system that the Operator Performance Laboratory (OPL) has used in the 737 simulator for numerous studies is called SmartEye. The SmartEye system reports on the 3D coordinates of several facial features and eye metrics and publishes that as User Datagram Protocol (UDP) packets at 60 hertz. The Cognitive Avionics Tool Set (CATS) data collection system picks up those packets and stores them in a Structured Query Language (SQL) database with a timestamp for subsequent analysis.

The pilot-model interface has been coded so that it also sends out the SmartEye system packets in the exact same format, so the CATS data system can collect eye movement data as if a real human were being eye tracked. This makes data collection trivial, as CATS was already designed to collect and store these UDP-formatted packets. The SQL database can be queried for eye-tracking data for subsequent analysis as would

be done with regular human subject-matter expert participants in studies conducted in the simulator environment as well.

Coordinate System Definition

The coordinate system for the pilot model implemented the same coordinate system that the SmartEye system uses so the data-recording system saw the data exactly as if it were coming from a real human in the cockpit. This is defined with a world model in 3D space using x, y, and z coordinates.

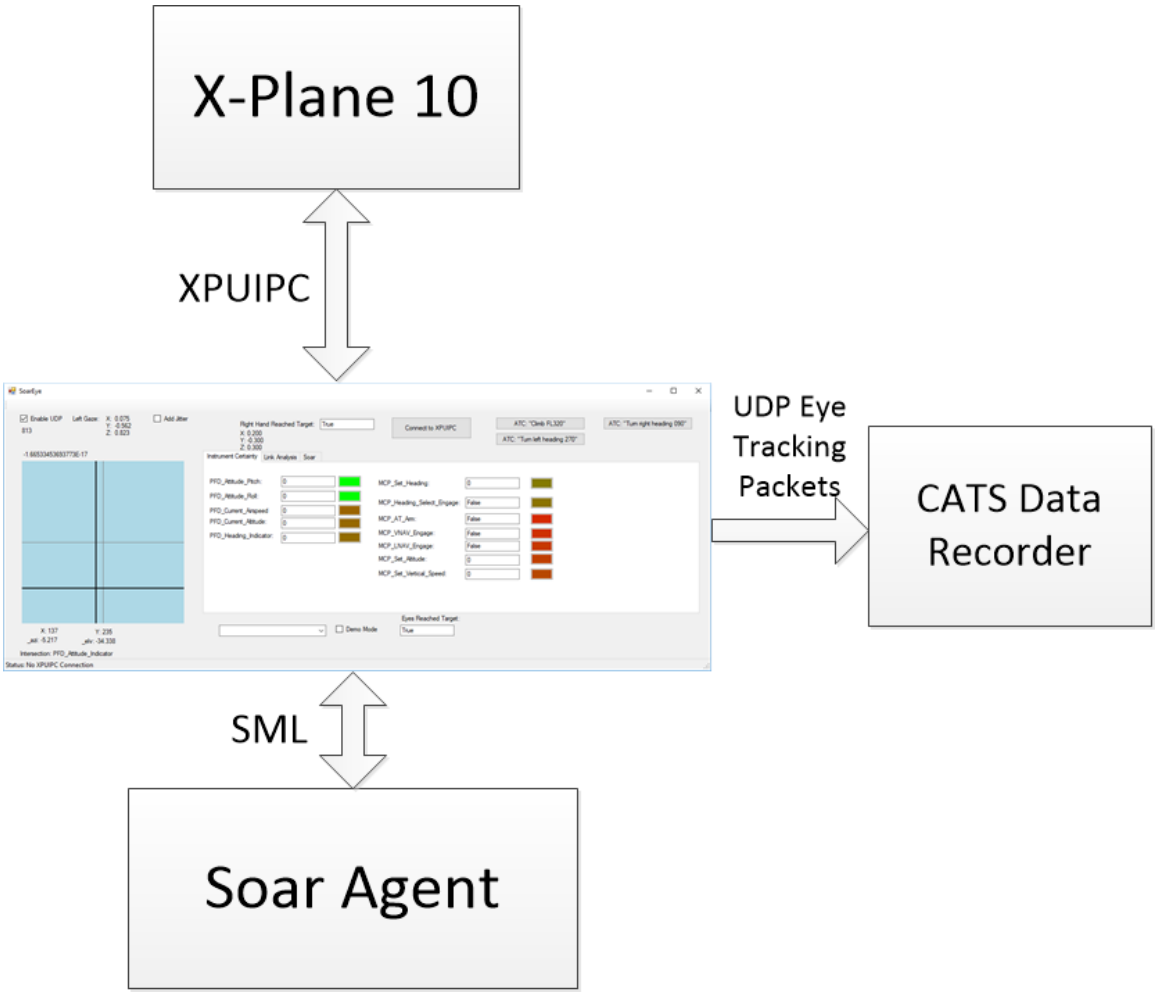


Figure 16: High-level diagram showing SoarEye data flow

Table 4: Orientation of the coordinate system

Axis	Orientation
X	Positive X to the left of the pilot facing forward
Y	Positive Y upwards towards the roof of the plane
Z	Orthogonal to the XY plane with positive Z toward the nose of the aircraft

World Model

The world model of the 737 simulator was constructed from measuring key instruments/controls with respect to the world origin (0, 0, 0), which is located near where the pilot's head would be in the cockpit. Using the coordinate system defined in the previous section, we can see that looking straight ahead would "see" things to the left with a positive X value and things to the right with a negative X value. Things that are above the pilot's head would be seen with a positive Y value, and any items below the pilot's head would have a negative Y value. A visual representation of a pilot sitting in the left seat of the aircraft looking forward illustrates the X & Y axes in Figure 17.

Two primary interfaces of concern in the world model for the SoarEye software is the primary flight display (PFD) and the mode control panel (MCP). Figure 18 and Figure 19 illustrate the coordinates for various elements of the PFD and MCP, respectively, as they were measured in the OPL simulator.

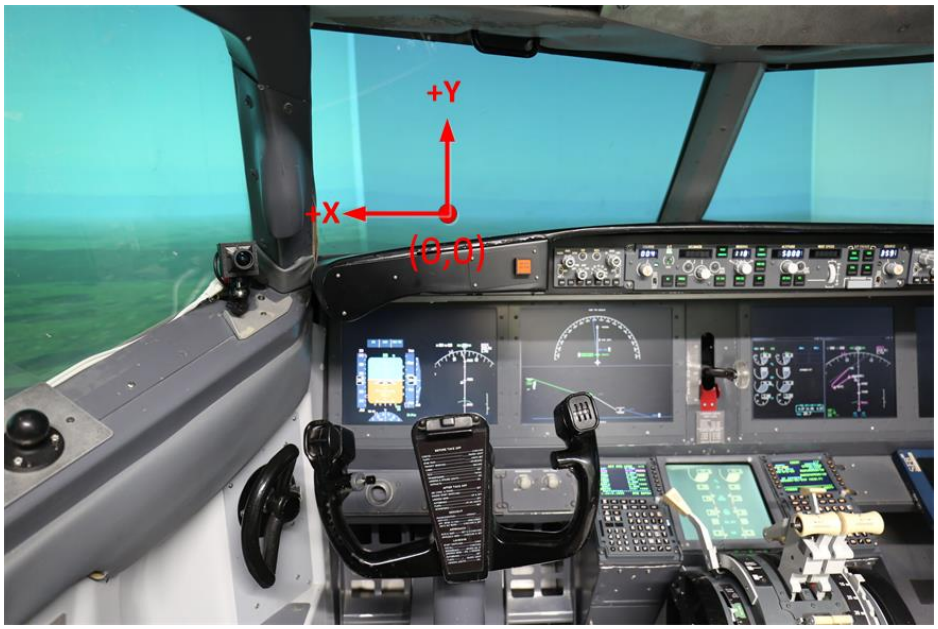


Figure 17: X & Y Axis of pilot looking forward in the cockpit

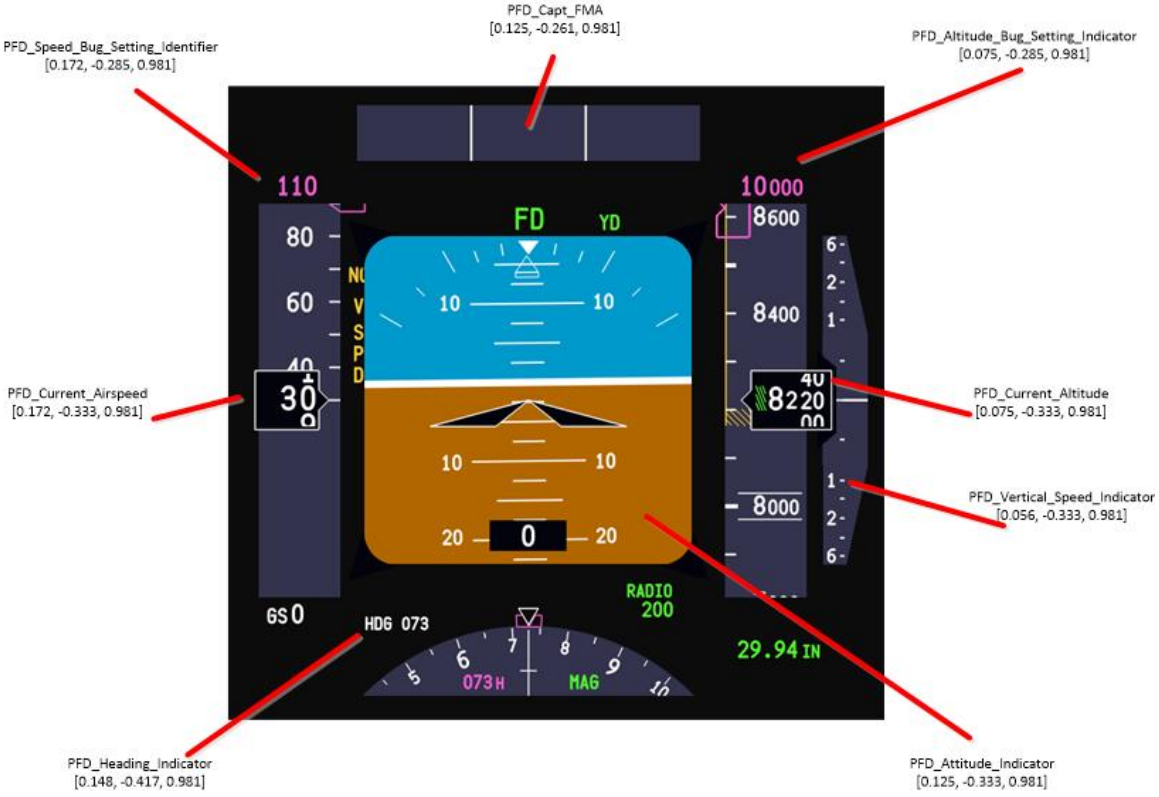


Figure 18: PFD with (x,y,z) coordinates specified for regions of interest

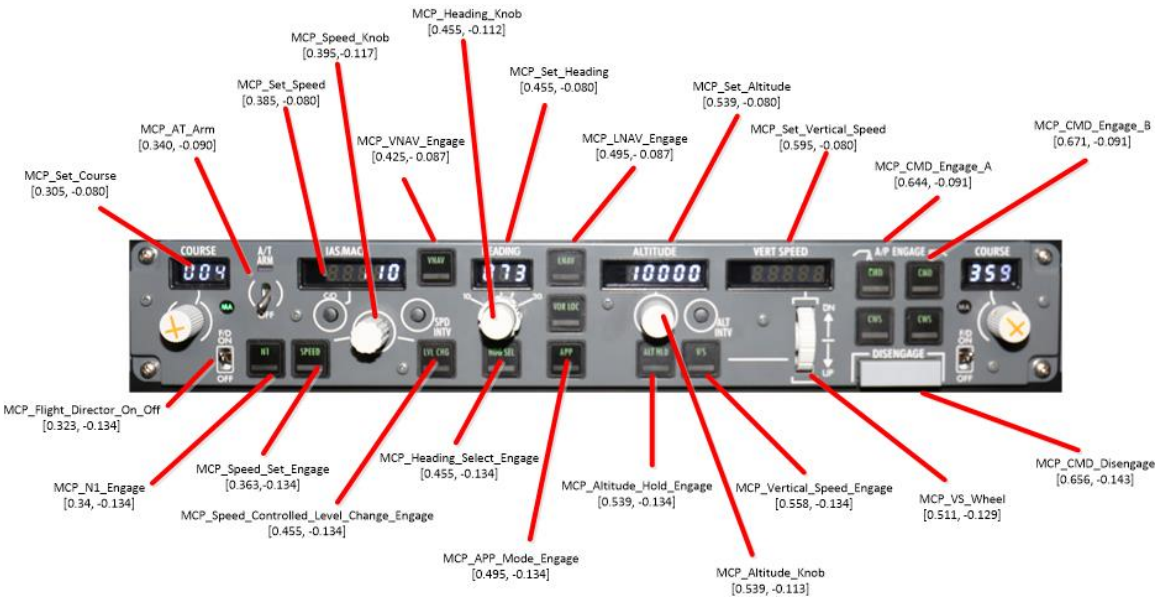


Figure 19: MCP with (x, y) coordinates specified for regions of interest (z=0.734 for all MCP elements)

Eye Movement

When an event triggered either internally via the SoarEye tool or via a command from the Soar agent itself causes a command to move the eyes from one fixation to the next, a short series of steps takes place. A simple example will illustrate these steps as seen in Figure 20. The calculations for eye gaze vectors, angles, and saccade movements between fixations is done in the SoarEye tool and not the Soar agent.

Assume that the pilot model is currently fixated on the LNAV engage button on the Mode Control Panel (MCP) and Soar issues a command to look at the attitude indicator on the primary flight display (PFD). The 3D coordinates for both the left and right eyes are known, as are the coordinates of the current fixation (MCP_LNAV_Engage) and the goal of the next fixation (PFD_Attitude_Indicator). Gaze vectors for both the current fixation and the goal fixation are easily calculated. Next, the angle between those two vectors is calculated (in this case, it is about 43.54°). Finally, the

time for the eyes to traverse the angle between those two locations is assumed to travel on the shortest path between the current and goal fixation points. Again, for this particular case, it takes the eyes approximately 0.22 seconds (assuming a saccade speed of about 200° per second).

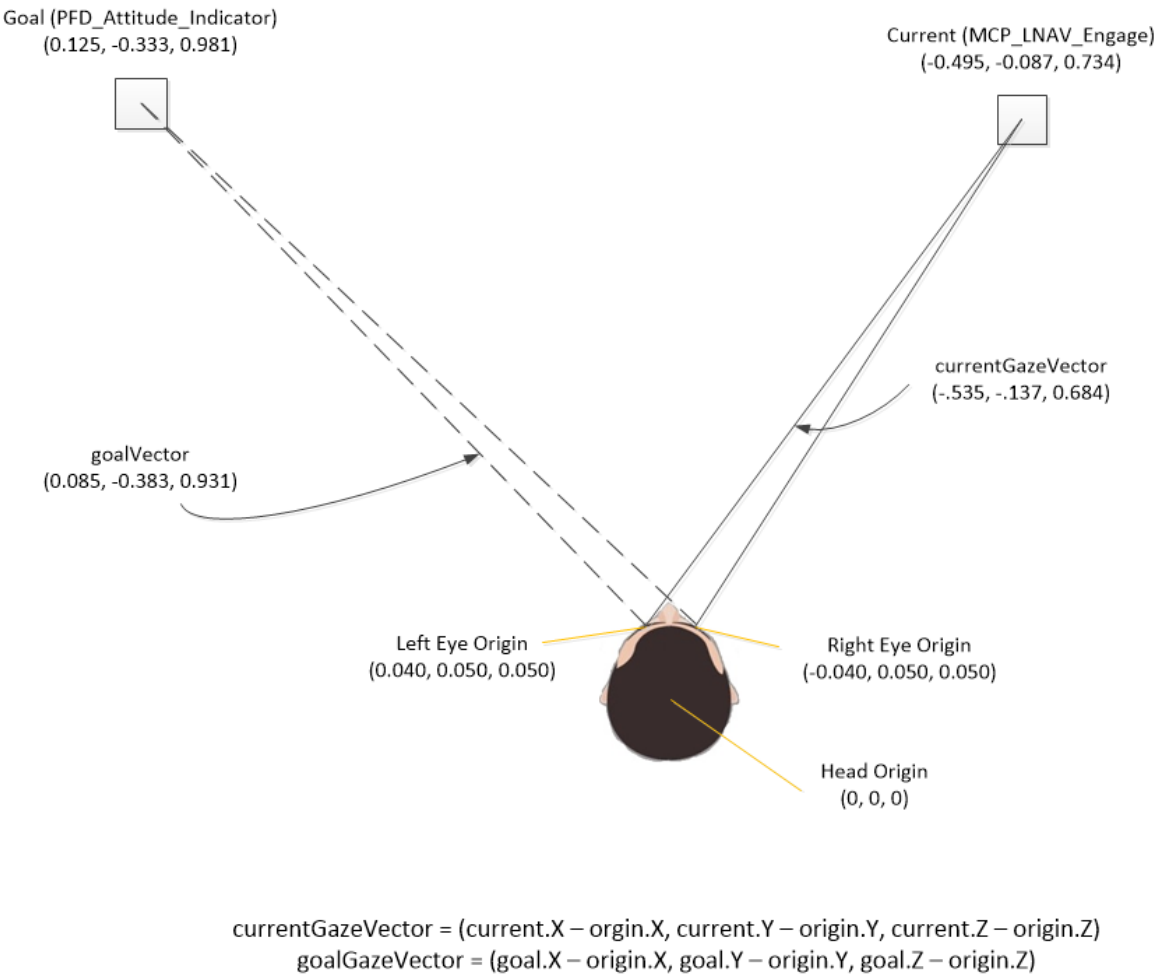


Figure 20: Example Eye Gaze Vector Transition (Not to Scale)

Instrument Uncertainty

Every instrument in the cockpit displays real-time data of the aircraft. However, the human is only capable of perceiving a small selection of all the instruments due to the simple fact that the human eye can only focus in about a 1.5 degree field of view [32].

This is because the central region of the human eye is where the foveal vision has a densely packed region of cones for near 100% visual acuity (Figure 21). The Soar pilot interface emulates this by only providing updates to the Soar agent of whatever instrument it is currently looking at with the model's eyes. Once the pilot looks away from a particular instrument, it no longer has the 100% certainty of its value/state.

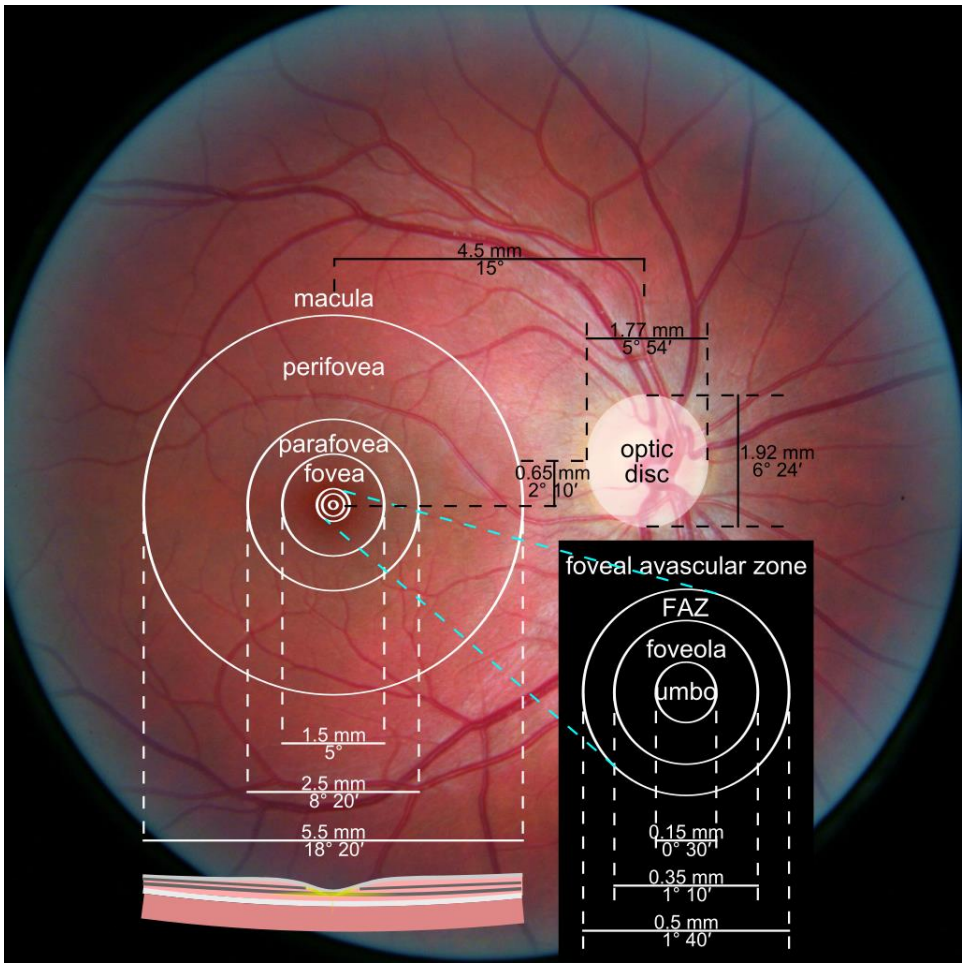


Figure 21: Photograph of the retina of the human eye, with overlay diagrams showing the positions and sizes of the macula, fovea, and optic disc

Some instruments are capable of changing faster than others, so each channel can be set up with its own unique decay rate. The type of decay can also vary per instrument, as each instrument changes at different rates, and within an instrument, depending on phase of flight. An example is that the altimeter of an aircraft may not change much or at

all during a cruise phase of a flight, but it can change dramatically during a descent/approach into an airport. While all three decay types depicted in Figure 22, only the Type II (linear) decay was used for the scenarios outlined later in this paper.

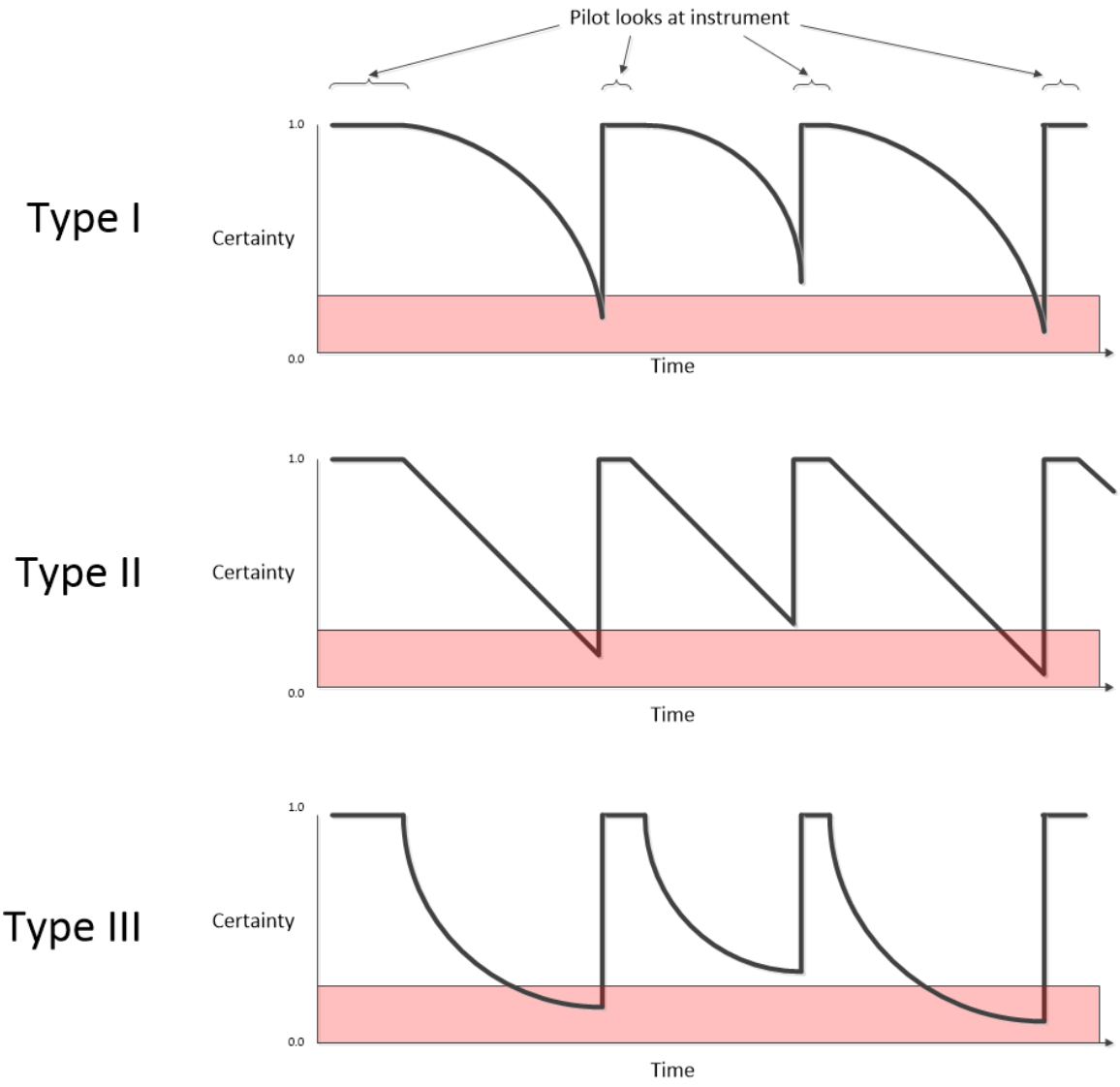


Figure 22: Example of the three different uncertainty decay types implemented in the SoarEye model

There are four different visually guided saccade movement categories: visually guided saccade, antisaccade, memory-guided saccade, and predictive saccade. Of these four categories, only memory-guided saccades for prescribed procedures and visually

guided saccades (scanning) have been implemented in the SoarEye model. A complete list of the eye metrics of the data packets (and their definitions) from the SoarEye model can be found in Appendix A.

Cockpit Configuration File

The SoarEye model needs to accommodate a wide variety of aircraft types, but every aircraft type will have a different physical layout of instruments and controls. In order to adapt to this reality, an aircraft-specific configuration file can be made for as many aircraft as needed. To make a fair comparison to the simulator environment at the OPL, an aircraft file for a Boeing 737-800 cockpit was developed using the physical dimensions of the 737 simulator located at the facility. This required dozens of measurements with respect to the world origin to determine the location of dozens of controls and instruments related to basic aircraft tasks.

These measurements were taken down to the millimeter and placed into the Extensible Markup Language (XML) file (see Appendix B). Every region of interest (ROI) was given a name as well as its x, y, z world coordinates in meters. A certainty decay value and decay type is also specified per instrument in this file. This allows a user of the SoarEye tool to customize the parameters for uncertainty as they see fit for the particular aircraft they are defining in that file. There is no one correct value for any instrument and it may take some experimentation to find a set of values that works for the user of the tool.

Passing Data to the Agent

Data needs to be passed along from the SoarEye interface to the Soar agent. This is done via the input-link of the Soar agent as specified in the previous chapter. Figure 24

shows an example of what the input-link branch of the WMEs of the pilot agent would look like. There are two elements added to the input-link for every instrument that the SoarEye controller is aware of. Those two elements would have the value/state of the instrument specified and another for the certainty of the instrument.

Using the SML .dll resource in the C# project permits simple function calls to add new WMEs as seen in the example in Figure 23. Every time the update Soar timer is triggered in the application, all of the WMEs that need to be updated are removed and then re-added with their new values. This is because a WME in Soar cannot be modified; it can only be removed or added.

```
/// <summary>
/// function to add WME with an integer value
/// </summary>
/// <param name="id">identifier of wme</param>
/// <param name="attribute">attribute of wme</param>
/// <param name="value">value of wme</param>
/// <returns>Working memory element that has an integer value</returns>
protected IntElement _createWME(Identifier id, string attribute, int value)
{
    IntElement wme = id.CreateIntWME(attribute, value);
    wmes.AddLast(wme);
    return wme;
}
```

Figure 23: Code sample of function create to add WME with an integer value

In Soar there is no pre-defined structure on the input-link, which is up to the designer/programmer of the agent. For this application, a simple structure of one element per variable is added to the input-link branch of working memory. A sample subset of what that structure looks like can be found in Figure 24. With these values present in working memory, the production rules can make decisions based on what it has as its most recent value for each instrument/control relevant to the task being completed.

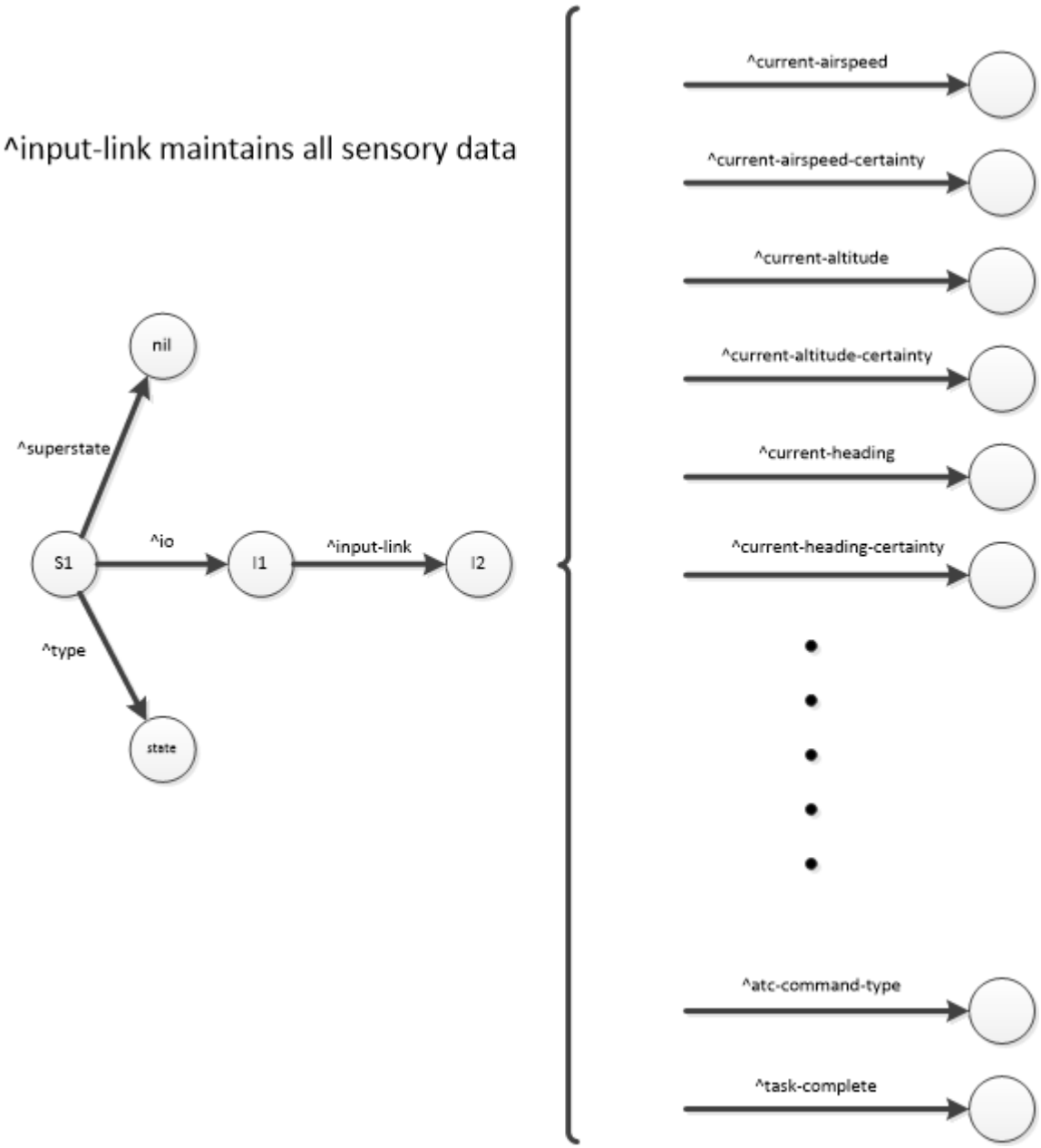


Figure 24: Subset of the SoarEye model input-link

Motor Movement Model

For the purposes of this project, a high-fidelity motor movement model of hand movements was not required. A more simple approximation of hand and arm movement is required only to create the appropriate delay in manipulating controls in the cockpit. A reasonable approximation of time for a hand to move from “point A” to “point B”

(translational motion) is to use Fitt's Law. Fitt's Law was introduced in Chapter 2 while discussing ACT-R visual model performance and is presented here again:

Fitt's Law

The equation for Fitt's law [33] is defined as follows:

$$T = a + b \log_2 \left(\frac{D}{W} + 1 \right)$$

where

- T is the average time taken to complete the movement.
- a represents the start/stop time of the device.
- b represents the inherent speed of the device.
- D is the distance from the starting point to the center of the target.
- W is the width of the target measured along the axis of motion. W can also be thought of as the allowed error tolerance in the final position, since the final point of the motion must fall within $\pm W/2$ of the target's center.

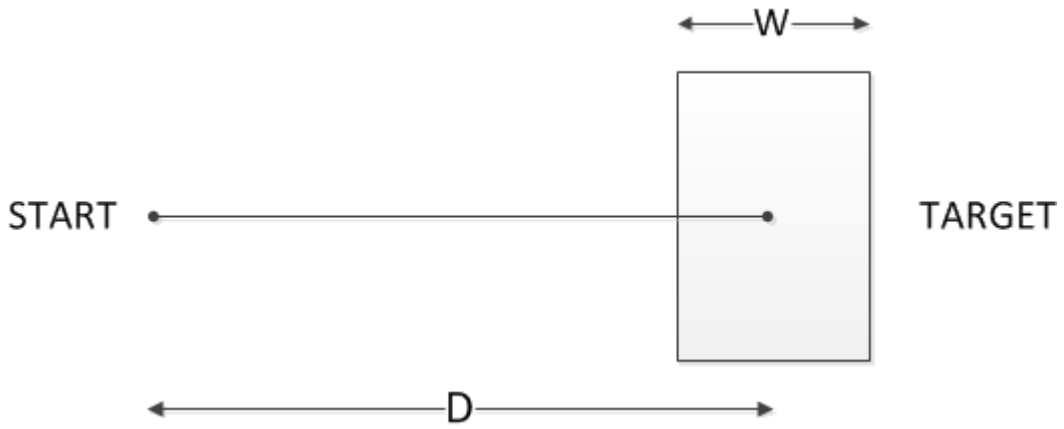


Figure 25: Analysis of the movement of a user's hand to a target

Figure 25 shows how D and W are determined with a simple start point and the target is drawn. From Fitt's law, the speed-accuracy tradeoff can be seen where targets that are smaller and/or farther away require more time to acquire.

For the SoarEye tool, it was assumed that each target for the hand was approximately 1 cm across in the direction of motion. The tasks that the SoarEye model is asked to perform for the scope of this research only involve pressing buttons and rotating dials on the MCP, all of which are approximately 1 cm across.

In 1994, Kondraske [34] wrote a paper building upon Fitt's translational motion model to also account for angular motion. This more comprehensive model could be incorporated in the future to predict performance in tasks that involved one or more jointed body segments with even more precision.

CHAPTER 5: RESULTS AND ANALYSIS

Methodology

The SoarEye model was tested with the Operator Performance Laboratory's 737-800 flight simulator (Figure 26) located at the Iowa City airport. The 737-800 simulator is comprised of full glass cockpit displays, 180 degree outside visual projection system, a mode control panel, driven throttle quadrant, hardware control display units (CDUs) with a functional flight management system (FMS) and enhanced display control panels (EDCPs). On the glass cockpit displays, there is a left and right seat PFD, left and right seat multi-function display (MFDs), and upper and lower engine indicating and crew alerting system (EICAS). For purposes of working with and initial evaluation of the SoarEye tool, only the left/captain subset of these systems is considered. The right/first officer (FO) side of the cockpit would normally be used by the first officer, which isn't considered here.

Figure 26: OPL Boeing 737-800 flight deck simulator

The SoarEye model connects to the simulator via XPUIPC software which allows SoarEye to access memory offsets containing the variables of the flight simulator that are of interest. Since the XPUIPC application transmits simulator state variables via UDP packets, the SoarEye and Soar agent can run either on the same machine as X-Plane or on any computer connected to the same network as the computers running the simulator. For the experimental setup, SoarEye was run on a laptop using Windows 10 with an i7 processor and 8.0GB of RAM so as not to interfere with simulator performance and vice versa. An illustration depicting a high-level architecture of the data collection setup can be found in Figure 27.

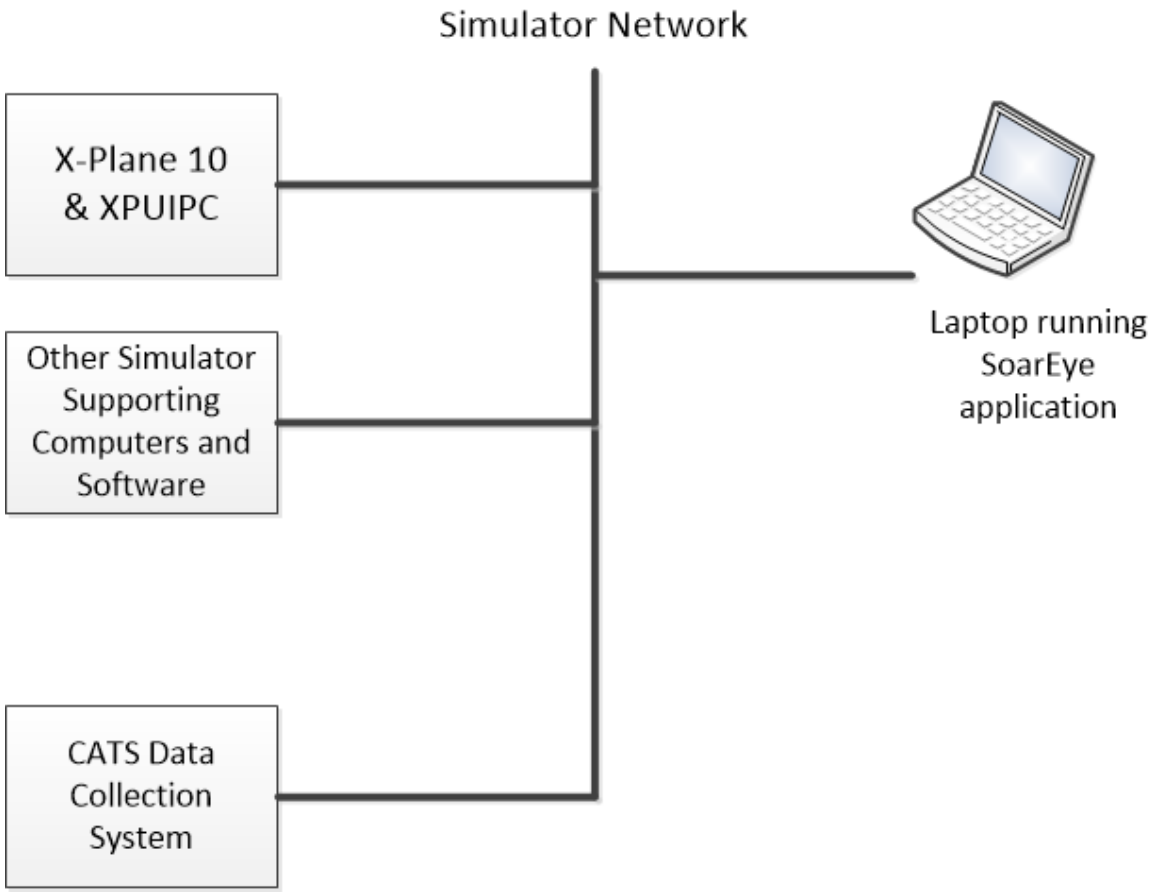


Figure 27: High level network connection diagram of setup for data collection

Scenarios

To test the SoarEye model with this configuration, four different scenarios/tasks were designed to exercise the model, collect and analyze data from. They are listed in Table 5 below.

Table 5: Scenarios/tasks to collect data from SoarEye

Scenario 1	Straight and Level flight at 28,000 feet, 270 heading
Scenario 2	Left-hand turn
Scenario 3	Right-hand turn
Scenario 4	Climb from 28,000 feet (FL280) to 32,000 feet (FL320)

Scenario 1 is a simple baseline task of the aircraft flying in cruise configuration. This was selected for the simple reason that the pilot's primary job is to observe the automation flying the aircraft for the pilot and no motor action is required by the pilot. This allowed data collection of a pure eye-gaze only task to compare to the next three tasks. A cruise altitude of 28,000 feet (FL280) and heading of 270 degrees (west) were selected.

Scenario 2 is also a simple task of performing a left-hand turn. This tasks consists of maintaining altitude (staying in vertical navigation (VNAV)), and having the SoarEye model select heading mode (HDG) on the MCP and dial a new heading (90 degrees left of current heading). The pilot monitors the aircraft heading as the turn is executed through the aircraft leveling wings.

Scenario 3 is nearly identical to Scenario 2 in that it's a turn but in the opposite direction (right). Just as in the previous scenario, the pilot maintains altitude (staying in VNAV), and selected the aircraft into heading mode (HDG) on the MCP. This task should show very similar eye gaze metrics since nearly all the same tasks are completed.

Scenario 4 is a climb from 28,000 feet (FL280) to 32,000 feet (FL320). This scenario could occur in the real world for a number of reasons (avoiding turbulence, traffic, convective weather avoidance, etc.). However, the reason is not important for the SoarEye data collection as the actions taken to complete the task would be the same regardless. The pilot turns the altitude knob on the MCP to the target altitude (FL320) and changes the cruise altitude in the flight management system by entering the new altitude into the control display unit (CDU). The aircraft then enters a climb and the pilot monitors the aircraft until it levels off at the new altitude.

Each scenario was run in the simulator 10 times. The data collected in CATS was compiled and inserted into a spreadsheet separated by scenario. Tables were then organized for both area of interest (AOI) analysis and a link analysis. The results of those analyses are presented in the following sections.

Regions of Interest

Region of Interest is one mechanism of analyzing the eye gaze data and compare it to other sets of data to gauge how close to actual pilot performance the SoarEye model is. For Scenario 1 of straight and level flight, Figure 28 highlights what regions of interest the SoarEye model was fixating on. This makes sense that the MCP items accounted for less than 7% of the fixations in areas of interest as there is nothing there that requires the pilot's input/attention during straight and level flight.

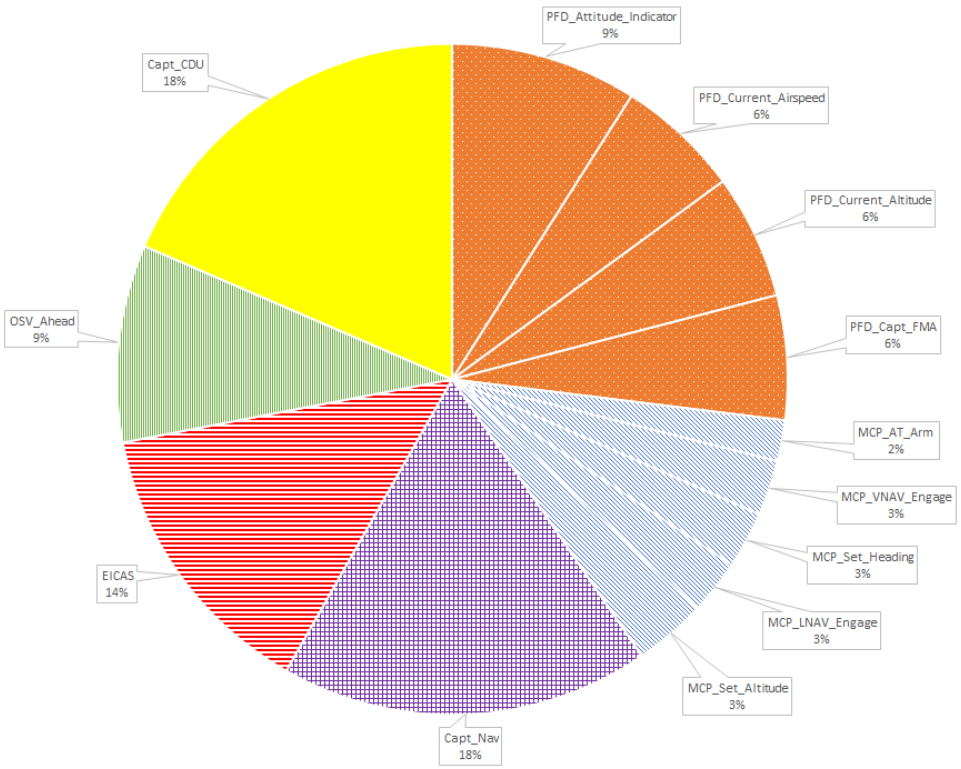


Figure 28: Region of Interest percentage of SoarEye fixations during Scenario 1, straight and level flight

Figure 29 shows what the SoarEye model fixations are during Scenario 2, a left turn of 90 degrees. Note the increase in amount of time spent looking at the captain navigation display. This is due to the Soar model checking more frequently the heading of the aircraft to verify correct operation of the flight management system. There are also some noticeable increases in fixations on the mode control panel. This is from the pilot model needing to switch the aircraft over to heading mode and dial in the 90 degree turn left. Very similar changes between straight and level and Scenario 3, the 90 degree turn right, can be found in Figure 30.

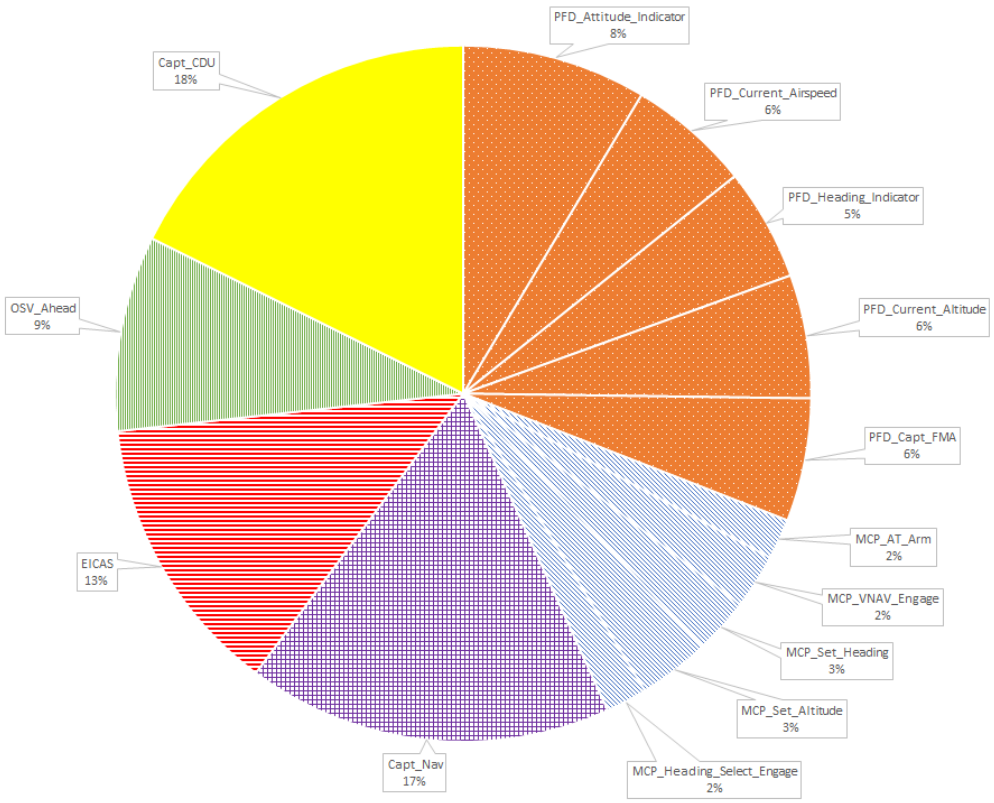


Figure 29: Region of Interest percentage of SoarEye fixations during Scenario 2, the 90 degree left turn

Figure 31 shows the SoarEye model fixations during scenario 4, the climb from FL280 to FL320. The largest difference between scenario 4 and scenario 1 is the decrease in fixation time on the captain navigation (Capt_Nav) display. The two new fixations regions of the PFD introduced are the altitude bug setting and the vertical speed indicator. The altitude bug becomes part of the scan pattern for the model once a new altitude is selected in the MCP to ensure the autopilot is targeting the correct altitude. The pilot model looks at the vertical speed indicator as part of the scan pattern to ensure the aircraft is in fact climbing and to ensure the rate slows down once aircraft reaches the target altitude (FL320).

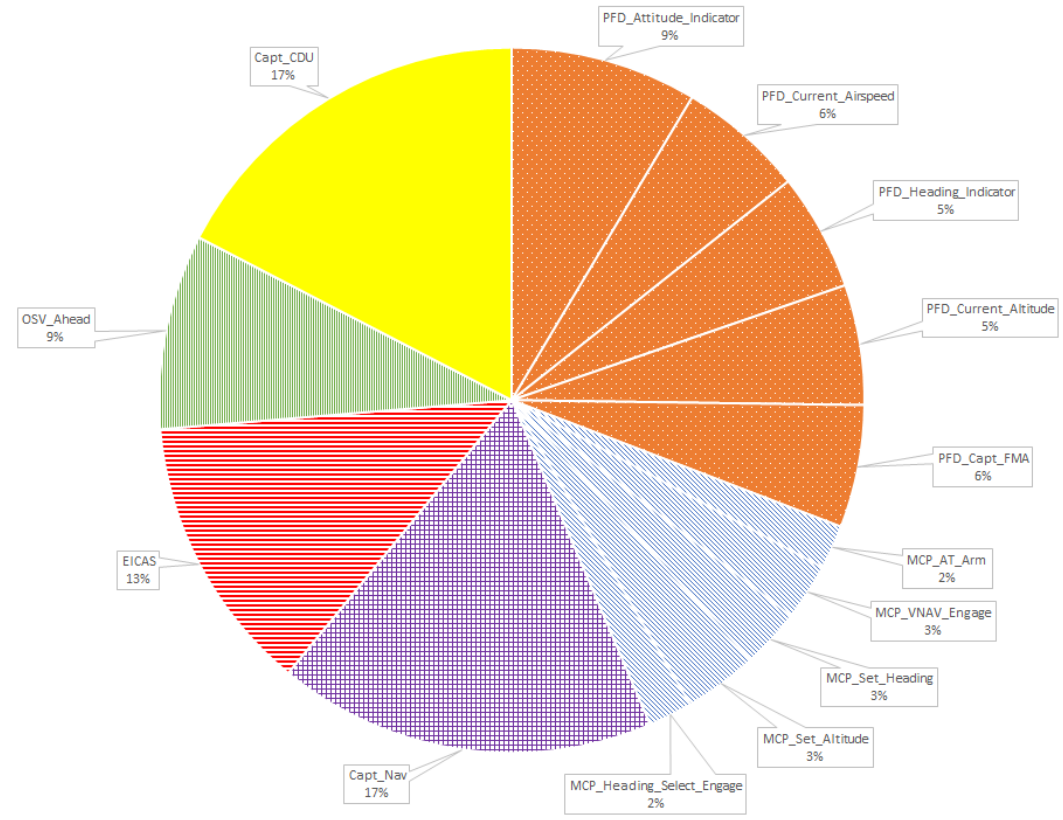


Figure 30: Region of Interest percentage of SoarEye fixations during Scenario 3, the 90 degree right turn

Table 6 summarizes the region of interest percentages among the four scenarios. The regions of interest that comprise the overall PFD and MCP were consolidated for ease of comparison as not every component within each was used in each scenario. The most significant item of note from this table was the difference between the baseline (scenario 1) and the scenario that the pilot model performs the climb (scenario 4). There was an increase of nearly 7% of fixations on the PFD during the climb compared to the baseline. Digging into the PFD components of the display, the increase of 7% to the PFD can be attributed to the fixations on the altitude bug setting and vertical speed indicator which were not part of the scan pattern during scenario 1.

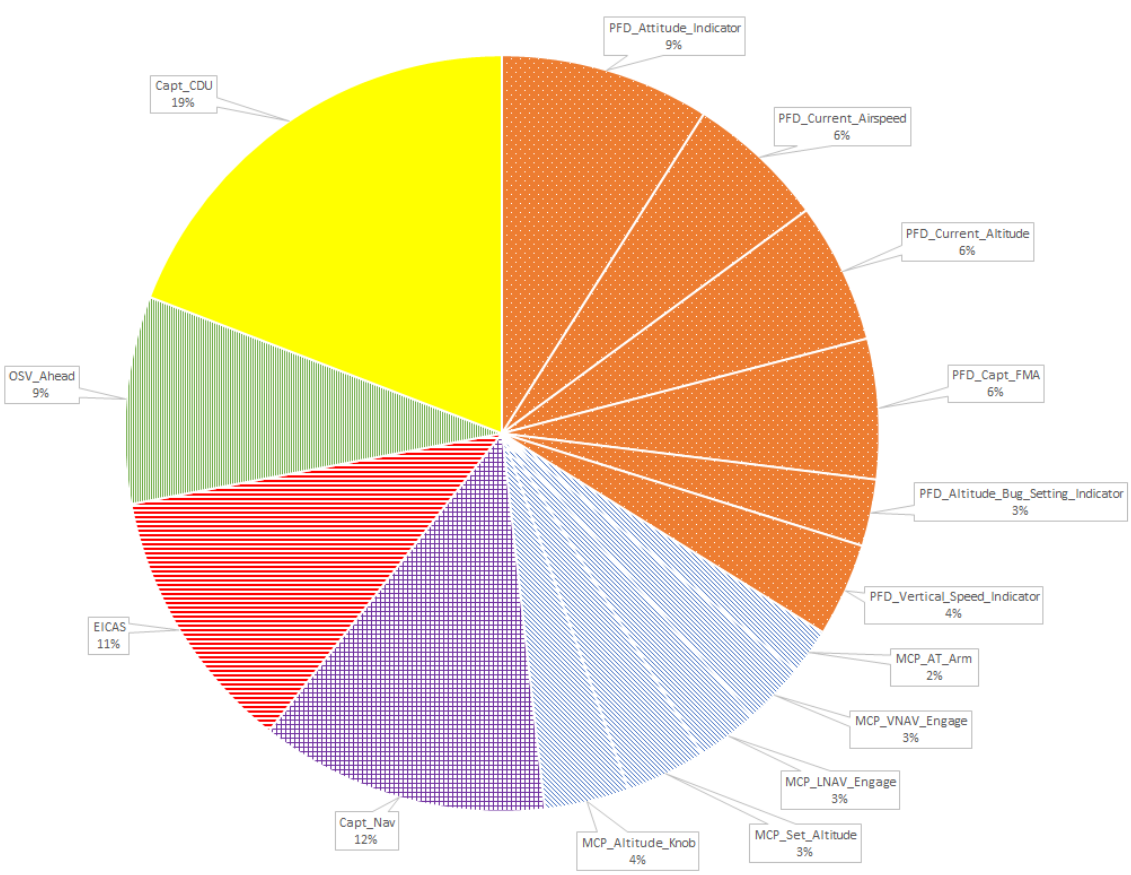


Figure 31: Region of Interest percentage of SoarEye fixations during Scenario 4, the climb to FL320

As a consequence of the increase in attention to the PFD, the navigation display saw a drop-off of nearly 5% of fixations during the climb. The EICAS also saw a marginal drop in fixations during the climb of approximately 3% (14% to 11%). There were only minor changes of approximately 1% among all the scenarios for the other regions of interest (Captain CDU, outside visual, MCP).

Table 6: Region of Interest percentage comparison among scenarios

	Scenario 1	Scenario 2	Scenario 3	Scenario 4
PFD	27%	31%	31%	34%
MCP	14%	12%	13%	15%
Capt_Nav	18%	17%	17%	12%
EICAS	14%	13%	13%	11%
Capt_CDU	18%	18%	17%	19%
OSV_Ahead	9%	9%	9%	9%

Link Analysis

A link analysis was also done on the data collected from the SoarEye model for all four scenarios. This counts the raw number of links/transitions between regions of interest. A link is defined as a fixation from one region on interest, followed by a saccade, to another fixation on another region of interest. This link analysis was conducted as a one-way transition (e.g. a transition from the MCP to the EICAS is unique from a transition from the EICAS to the MCP). These types of analyses have been done for years with some of the earlier ones conducted by [4]. An example illustration of what the link analysis looked like from that study can be found in Figure 32.

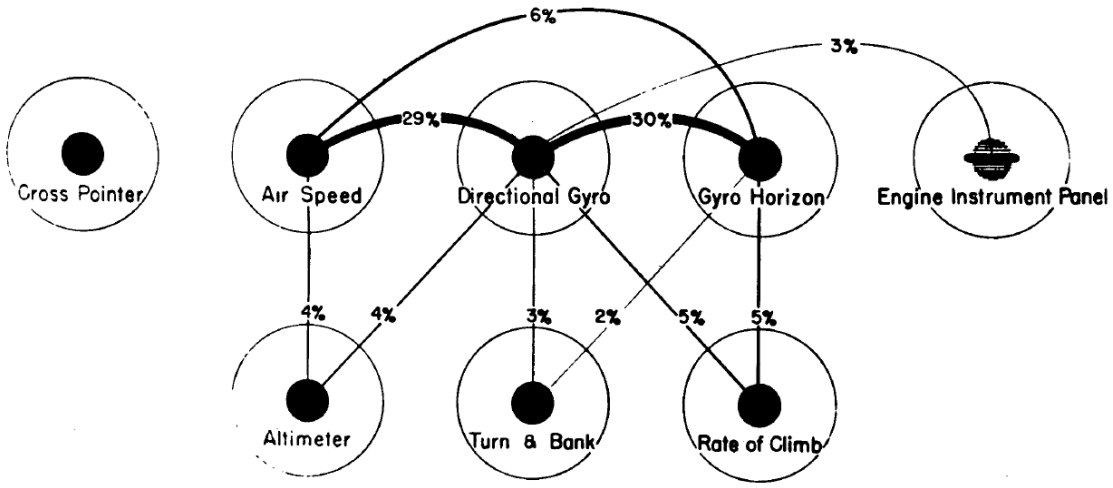
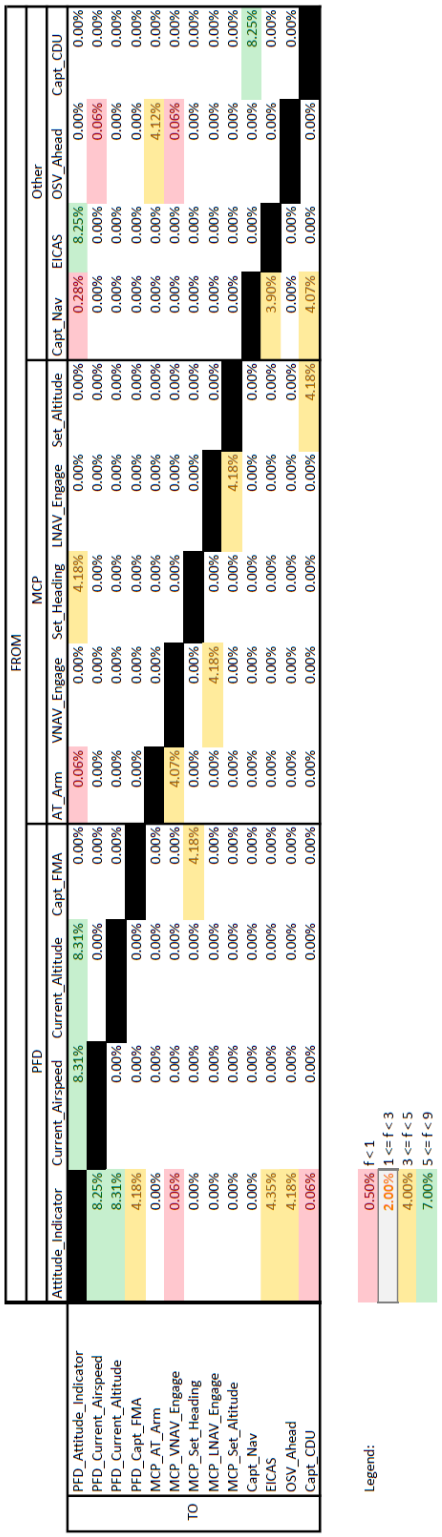


Figure 32: Adjacency layout diagram of eye-movement link values between aircraft instruments during an approach [4]

AOI transition data was collected within SoarEye and exported to a spreadsheet for analysis. Table 7 contains the data for link analysis during Scenario 1, straight-and-level flight. The numbers within these tables add up to 100% and represent the percentage values that fixations move from one AOI to the others.

Table 7: Link analysis for straight-and-level flight



An adjacency layout diagram was constructed for straight-and-level flight was generated from the data produced by SoarEye. Figure 33 illustrates the links between AOI for any that had a frequency percentage greater than 1%. Links are color-coded to depict the ranges of frequency, as shown in the legend within the figure.

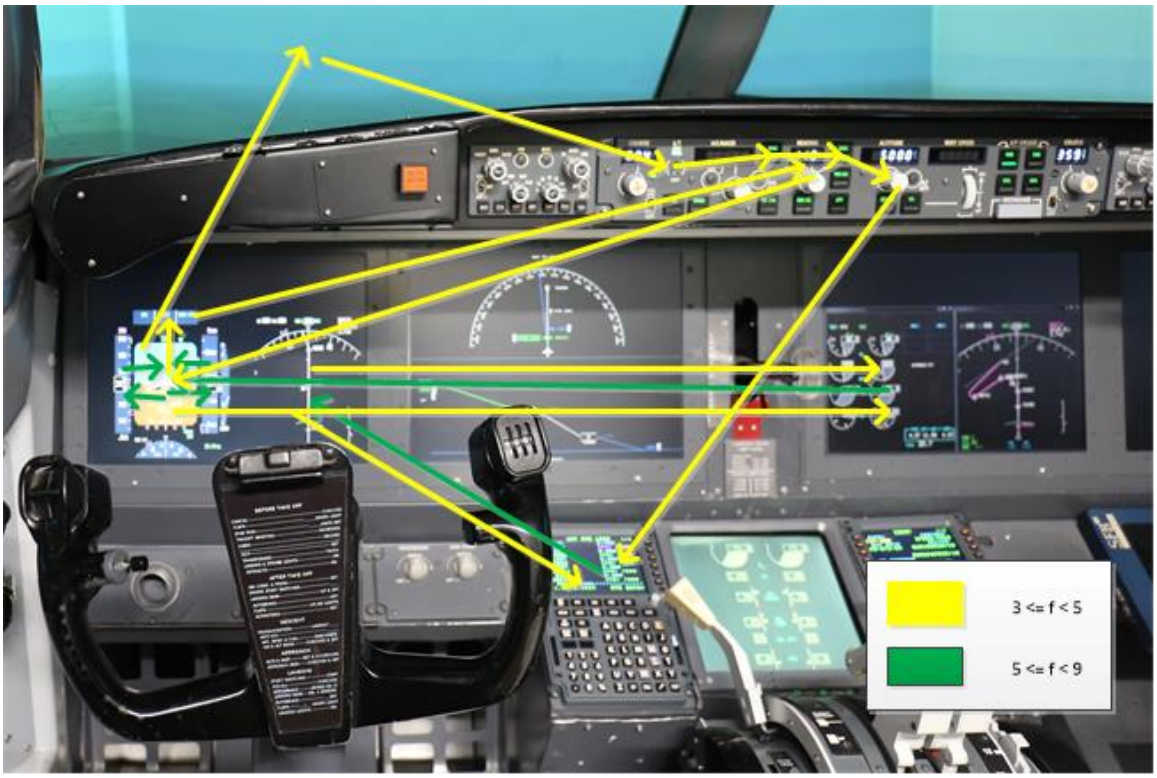
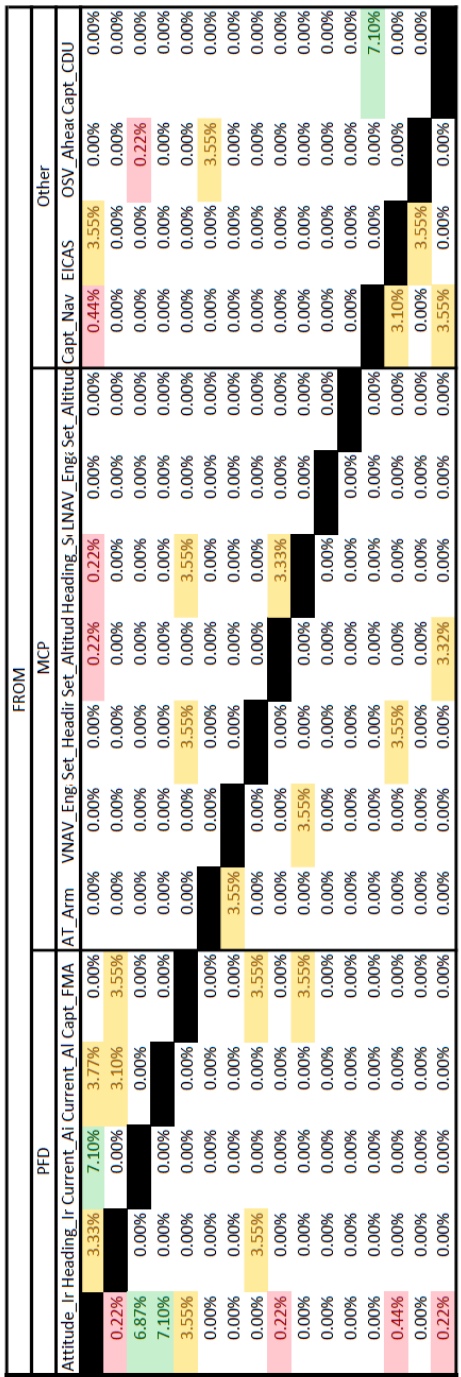


Figure 33: Adjacency layout diagram of SoarEye model eye movement among instruments during straight-and-level flight

An adjacency layout diagram was constructed for scenario 2 (left-hand turn) by data that was generated from SoarEye. The data for the left-hand turn can be found in Table 8. Figure 34 illustrates the links between AOI for any that had a frequency percentage greater than 1%. Links are color-coded to depict the ranges of frequency, as shown in the legend within the figure.

Table 8: Link analysis for left-hand turn



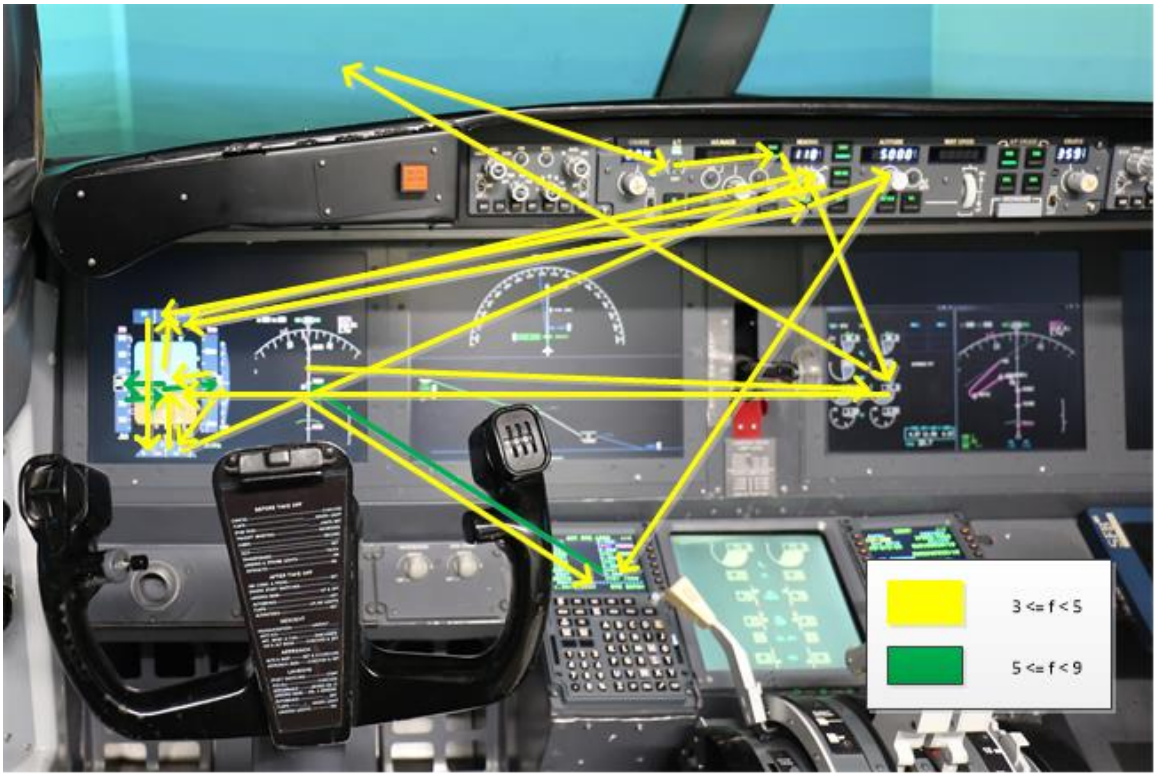
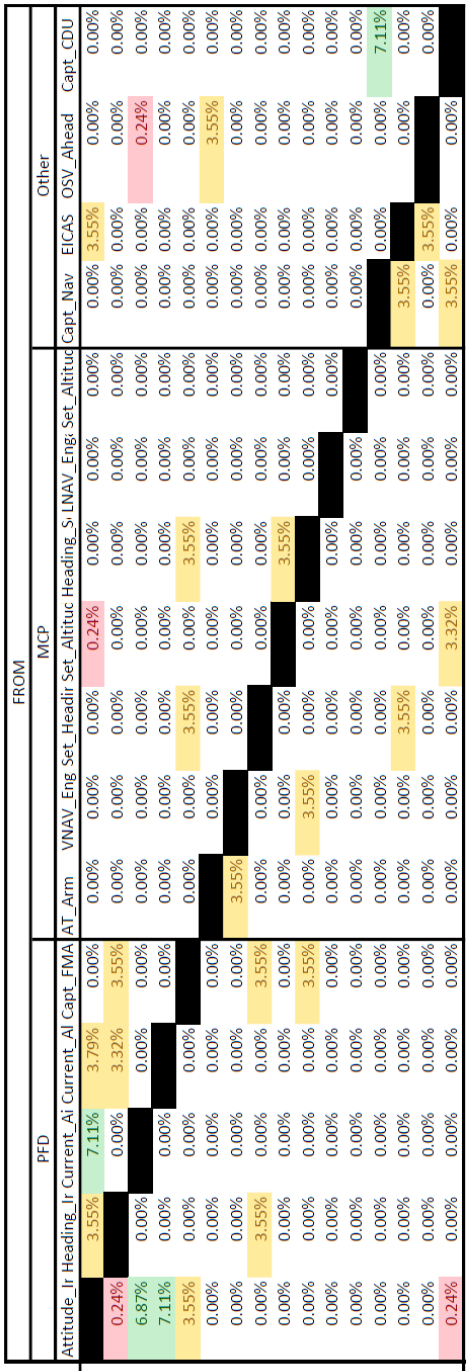


Figure 34: Adjacency layout diagram of SoarEye model eye movement among instruments during a left-hand turn (scenario 2)

A link-analysis was also performed on the eye tracking data collected from the SoarEye model during a right-hand turn as well. The data can be found in Table 9, and resulting adjacency layout diagram in Figure 35. The data from the link-analysis was near identical to what was found in the left-hand turn (scenario 2). This corroborates what was expected since the exact same production rules to drive eye movement in the Soar agent were used regardless if the turn was left or right handed.

Table 9: Link analysis for right-hand turn



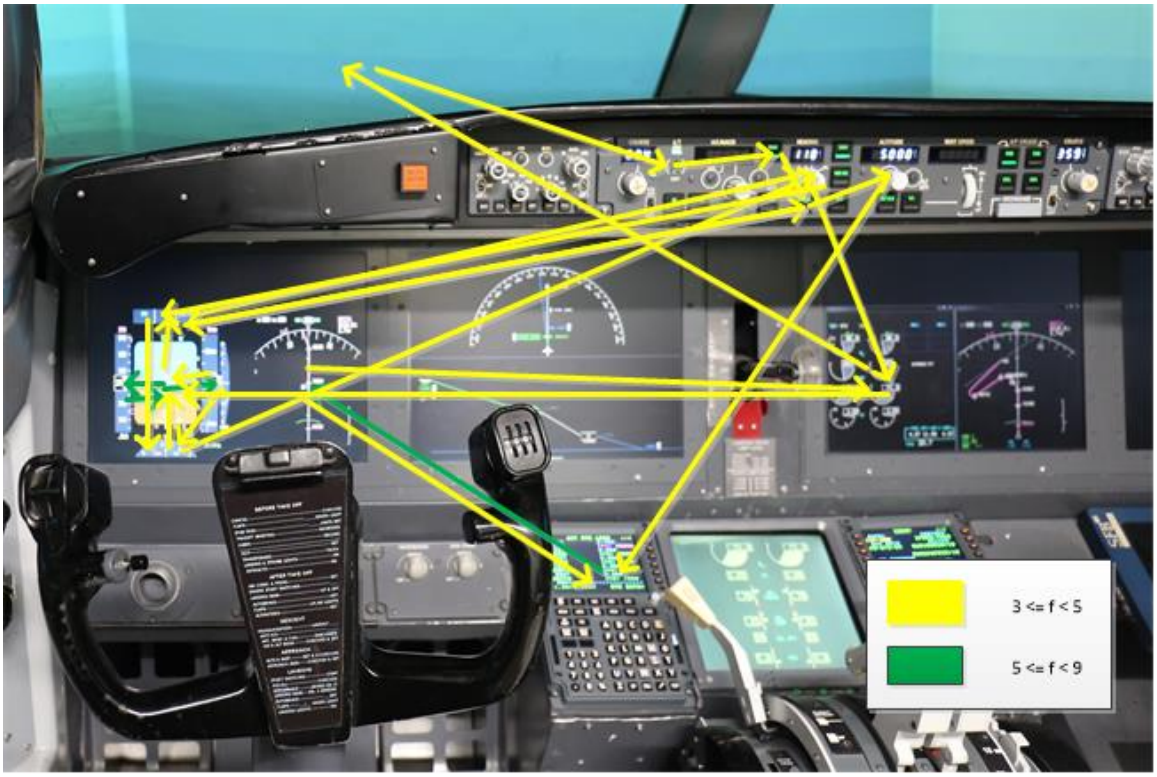
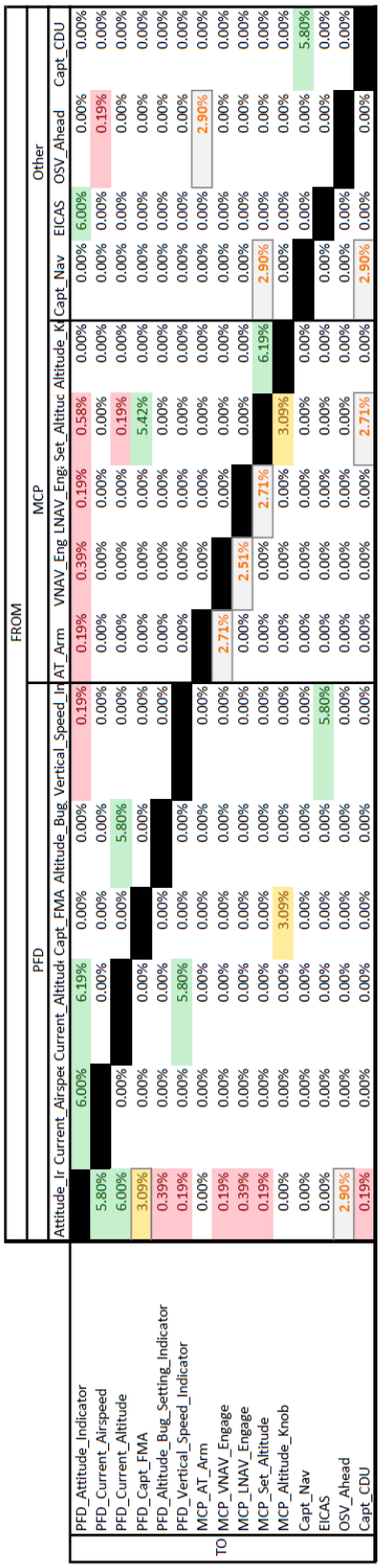


Figure 35: Adjacency layout diagram of SoarEye model eye movement among instruments during a right-hand turn (scenario 3)

Finally, a link analysis was performed on the data collected from running scenario 4, a climb from FL280 to FL320. Data can be found in Table 10 and the adjacency layout diagram can be found in Figure 36 for this scenario.

Table 10: Link analysis of a climb from FL280 to FL320



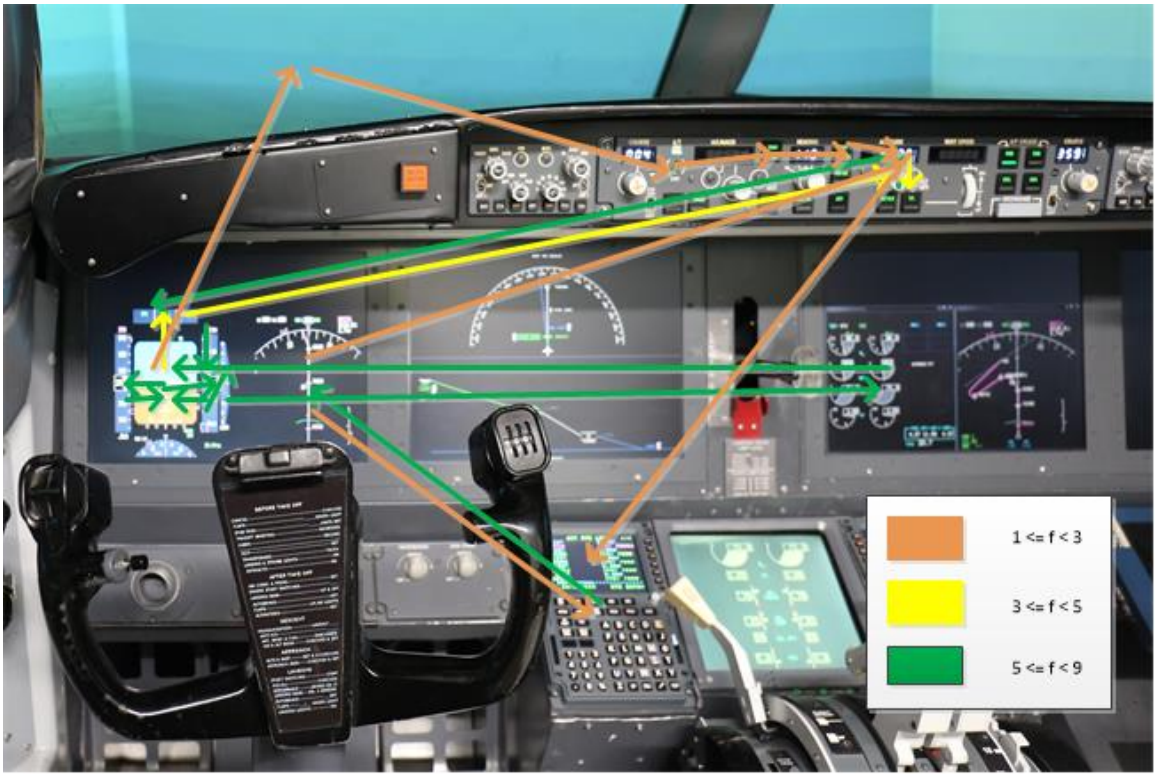


Figure 36: Adjacency layout diagram of SoarEye model eye movement among instruments during a climb between FL280 and FL320 (scenario 4)

Tool Verification

To establish a reference to look at the accuracy of the SoarEye model, data collected from a simulator study conducted in August of 2018 at the Operator Performance Laboratory was analyzed. This data collection effort took place in the same Boeing 737 simulator that the SoarEye model was connected to for its data collection effort. This study collected eye tracking data from seven (N = 7) commercial airline pilots performing an assortment of maneuvers while on autopilot. The eye tracking data collected from the pilots was done with a Dikablis head-worn eye tracker and the CATS data collection system. All seven of the pilots were Boeing qualified (first officers and captains) at the time of the data collection.

This simulator study with the airline pilots was for the evaluation a new type of system display in the cockpit to assist pilots in identifying autopilot mode change behaviors quickly. In order to make a comparison to the standard setup, half of the scenarios for each pilot were done with the new display available. The other half of the scenarios were done with the simulator configured in a standard setup to be used as a baseline. For purposes of the research in this chapter, only the eye tracking data from the simulator runs with the baseline configuration were used. A comparison of the data collected from the SoarEye model and the airline pilots can be found in Table 11.

Table 11: Comparison of SoarEye vs Actual Region of Interest Data

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	SoarEye	Pilots	SoarEye	Pilots	SoarEye	Pilots	SoarEye	Pilots
PFD	27%	46.4%	31%	22.0%	31%	39.0%	34%	16.9%
MCP	14%	1.0%	12%	1.9%	13%	0.6%	15%	3.7%
Capt_Nav	18%	20.3%	17%	27.8%	17%	30.0%	12%	16.0%
EICAS	14%	0.5%	13%	0.5%	13%	0.0%	11%	0.1%
Capt_CDU	18%	7.0%	18%	4.9%	17%	7.3%	19%	2.2%
OSV_Ahead	9%	-	9%	-	9%	-	9%	-
Other	-	24.0%		41.4%		22.7%		60.7%

The data collected from the airline pilots' shows that of the regions of interest tracked in the simulator, the PFD and the NAV displays were the most focused on regions of interest which generally agreed with what the SoarEye model predicted. However, the margins by which the airline pilots looked at the PFD and NAV were far greater than what the SoarEye model predicted. Another data point that stands out is that the airline pilots rarely looked at the EICAS display. In none of the scenarios did the pilots look at the EICAS for even more than 1% of the scenario duration. This makes some sense as the EICAS would primarily be consulted during engine run-up on takeoff and if there were cautions/warning/alerts that need to be addressed in non-normal

situations. Neither of these events were part of the four scenarios. The production rules in the Soar model spent far more time gazing at components inside of the EICAS which was a major contributor to the discrepancy.

The SoarEye model production rules also estimated a greater period of time required to complete tasks on the CDU and MCP than what the airline pilots actually did. The estimates for task completion in the SoarEye model for CDU and MCP procedures need to accommodate faster execution times to emulate expert behavior in the future.

In summary, there are three big differences between the SoarEye model and the airline pilot behavior witnessed in the simulator across these four scenarios. First, while pilots did have the EICAS display as part of their scan pattern, the dwell time was very short (less than a second) as they were only glancing at it to verify there wasn't a caution/warning/alert that needed to be addressed. Second, the speed at which pilots completed steps on the CDU and MCP was much quicker than the SoarEye model predicted. This could be addressed with adjustments to execution times of the Soar production rules that approximates the behavior of a novice pilot. Third, in the simulator, there are several conversations that the airline pilots have with the copilot in the right seat of the simulator. When a pilot talks with their copilot, their eyes concentrate on their cockpit companion and not the instrumentation. The SoarEye model does not accommodate for this event (talking with a copilot) at this time which in turn, inflates the percentages of all regions of interest across the board. Future iterations of the SoarEye model will need to take this into account for more realistic eye scan behavior as conversations with other crew members is essential in today's cockpit environments and should not be ignored.

CHAPTER 6: SUMMARY AND FUTURE WORK

Summary

This project has shown that it is possible to obtain datasets of simulated eye scan behavior to help researchers identify opportunities to design scenarios for human subject matter experts in simulator studies. The SoarEye model demonstrates that a cognitive model driven software tool can be integrated with a simulated environment and data collection system to complete and analyze a wide-range of tasks that humans could perform. The region of interest, link analysis and model verification demonstrated that the model output, with adjustments, could reasonably approximate what would be expected of a human pilot sitting in the cockpit of the same aircraft/simulator.

Future Research Areas

Throughout the course of this project, a number of issues and uses were identified which could be explored in the future. Firstly, only a small subset of tasks/scenarios were evaluated for this dissertation. This was due to the complexity of breaking down and converting task analysis for human operators into Soar rules in the cognitive model. Diagnosing behavior within Soar working with a real-time environment/simulation is an intensive endeavor. Future work should be expanded to explore a wider variety of tasks by adding more Soar rules to the SoarEye model.

Another key issue that could be investigated is how efficiently cockpits are designed. By doing a link-analysis of the scan pattern, it could be determined if there are instruments/controls in the cockpit that are frequently used together, but not closely spaced physically. This could be useful in both existing cockpit designs and evaluation of new cockpit design layouts for future aircraft.

As was mentioned in the methodology section, the analysis of the SoarEye tool was constrained to the captain's perspective (left seat). Further research could be done with two instances of SoarEye running simultaneously with a concurrent simulation to emulate the perception of a flight crew conducting the same flight. Pursing research with simulation perception and cognitive modeling of a flight crew could result in new insights into optimizations of procedures by eliminating redundant checks and/or fixing oversights in attention deficits of other items on checklists. In the discussion of the tool verification in Chapter 5, it was noted that the interaction of flight crew plays a key role for flight crews. Even verbal communication among crew has a large influence on visual attention, can disrupt eye scan behavior and needs to be taken into account.

The saliency of pilot perception in the cockpit is an additional capability that would be desirable for the SoarEye model. Being able to integrate mechanisms into the visual system of the pilot model to recognize visual events (new objects, changes in luminance, etc.) that might capture attention not in the line of sight, has been researched by many over several decades [35]. Another good place to start would be incorporating some form of Wickens Salience Effort Expectancy Value (SEEV) or N-SEEV models [36] [37].

The physical capabilities of the SoarEye model could be enhanced as well. The SoarEye head/eye movement at the time of this writing is attached to a stationary torso that does not move. Developing a more accurate physical model of the pilot could enhance the realism for reaction times for both motor movements and perception of the environment by the cognitive model. While the differences would result in changes to

individual actions on the order of milliseconds, over simulation runs that last hours, this could be a significant development in accuracy of the model.

Finally, it would be of interest to be able to implement tools into SoarEye that can quiz the metrics in the Soar agent itself to estimate the workload of the pilot. The SoarEye model helps determine the mental representation airline pilots have for flight-deck information. This is the initial step for developing cognitive measures of crew performance. There are a number of metrics in Soar that could potentially be correlated to how much effort the model is required to use to complete tasks the cockpit. How often memory is accessed, monitoring the reinforcement learning values being assigned to production rules specific to certain tasks, and how long individual inputs are ignored/neglected due to the model focusing on other tasks. Using data collected from simulator studies with human pilots, numerical weights could be assigned to the cognitive model metrics to achieve comparable workload ratings such as one would find with NASA-TLX and/or Bedford workload scores.

APPENDIX A: SMARTEYE PACKET DATA AND FORMAT

Packet Header		Sync ID u32, 4 bytes
		Packet Type u16, 2 bytes
		Packet Length u16, 2 bytes
Subpacket 1	Subpacket header	ID u16, 2 bytes
		Length u16, 2 bytes
	Subpacket data	Data . . .
.		.
.		.
Subpacket N	Subpacket header	ID u16, 2 bytes
		Length u16, 2 bytes
	Subpacket data	Data *

Figure A.1: Packet structure/format of the eye metric packets

Table A.1: Data Types

Data Type	Consists of	Type Alias
u8	Unsigned integer 1 byte	Type_u8
u16	Unsigned integer 2 bytes	Type_u16
u32	Unsigned integer 4 bytes	Type_u32
s32	Signed integer 4 bytes	Type_s32
u64	Unsigned integer 8 bytes	Type_u64
f64	Floating point 8 bytes	Type_f64
Point2D	f64 x f64 y	Type_Point2D
Vect2D	f64 x f64 y	Type_Vect2D
Point3D	f64 x f64 y f64 z	Type_Point3D
Vect3D	f64 x f64 y f64 z	Type_Vect3D
String	u16 size u8 ptr[1024]	Type_String
WorldIntersectionStruct	Point3D worldPoint Point3D objectPoint String objectName	
WorldIntersection	u16 size = 0..1 WorldIntersectionStruct	Type_WorldIntersection
WorldIntersections	u16 size = 0..1 WorldIntersectionStruct	Type_WorldIntersection
Vector	u16 size = 0..1 N items of any Type Each Type is prefixed with u16 SubPacket ID	Type_Vector

Table A.2: SubPacket IDs

Enum ID	Enum number	Data type
SEFrameNumber	0001	Type_u32
SEHeadPosition	0010	Type_Point3D
SEHeadPositionQ	0011	Type_f64
SEHeadNoseDirection	0013	Type_Vect3D
SEHeadUpDirection	0014	Type_Vect3D
SEHeadLeftEarDirection	0015	Type_Vect3D
SEFilteredGazeHeading	0036	Type_f64
SEFilteredGazePitch	0037	Type_f64
SEHeadRoll	0018	Type_f64
SELeftGazeOrigin	001B	Type_Point3D
SERightGazeOrigin	001C	Type_Point3D
SELeftGazeDirectionQ	0025	Type_f64
SERightGazeDirectionQ	0028	Type_f64
SEFilteredLeftGazeDirection	0032	Type_Vect3D
SEFilteredRightGazeDirection	0034	Type_Vect3D
SEEyelidOpening	0050	Type_f64
SEEyelidOpeningQ	0051	Type_f64
SEClosestWorldIntersection	0040	Type_WorldIntersection

Table A.3: SubPacket item descriptions

Data Item	Unit	Description
Frame number	-	A sequential frame number. The frame number starts counting from 0 at application start and increments by 1 for each successfully built data packet.
Head Position	m	The head position in 3D given in the defined world coordinate system
Head Position Quality		0..1 (0 = no tracking, 1 = full tracking)
Head Nose Direction		Unit vector defining the 'nose' direction. Identical to the z-axis of the head rotation matrix.
Head Up Direction		Unit vector defining the 'up' direction. Identical to the y-axis of the head rotation matrix.
Head Left Ear Direction		Unit vector defining the 'left ear' direction. Identical to the x-axis of the head rotation matrix.
Filtered Gaze Heading	rad	The left/right angle of the gaze vector.
Filtered Gaze Pitch	rad	The up/down angle of the gaze vector.
Head Roll	rad	The tilt-rotation of the head. Also known as 'maybe' rotation.
Left Gaze Origin	m	The center of the iris of the left eye, where the gaze vector originates.
Right Gaze Origin	m	The center of the iris of the right eye, where the gaze vector originates.
Left Gaze Direction Quality		0..1. Depends on the quality of the iris detection of the left eye
Right Gaze Direction Quality		0..1 Depends on the quality of the iris detection of the right eye
Filtered Left Gaze Direction		A unit vector originating in the left gaze origin, describing the direction of the gaze of the left eye (filtered).
Filtered Right Gaze Direction		A unit vector originating in the right gaze origin, describing the direction of the gaze of the right eye (filtered).
Eyelid Opening	m	The average distance between the eyelids of both eyes
Eyelid Opening Quality		Normally in the range 0..1. Calculates as the average quality of the both physical eyes.
Closest World Intersection		The closest intersection with any of the world objects. The intersection information contains name of the object, intersection

Table A.3 - continued

		point in world coordinates and intersection point in object coordinates. Each sub packet of this type starts with an integer indicating the number of world intersections contained in the sub packet. If there are no gaze intersections with world objects for the current frame, this integer will be 0, otherwise 1.

APPENDIX B: 737 COCKPIT DEFINITION FILE

```
<?xml version="1.0" encoding="utf-8"?>
<ArrayOfRoi xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Roi>
     <Name>PFD_Attitude_Indicator</Name>
     <X>50</X>
     <Y>200</Y>
     <Xm>0.125</Xm>
     <Ym>-0.333</Ym>
     <Zm>0.981</Zm>
     <DecayTO>.005</DecayTO>
     <DecayTOType>Linear</DecayTOType>
     <DecayCRZ>.001</DecayCRZ>
     <DecayCRZType>Linear</DecayCRZType>
     <DecayAPP>.005</DecayAPP>
     <DecayAPPType>Linear</DecayAPPType>
  </Roi>
  <Roi>
     <Name>PFD_Heading_Indicator</Name>
     <X>31</X>
     <Y>220</Y>
     <Xm>.148</Xm>
     <Ym>-.417</Ym>
     <Zm>0.981</Zm>
     <DecayTO>.005</DecayTO>
     <DecayTOType>Linear</DecayTOType>
     <DecayCRZ>.001</DecayCRZ>
     <DecayCRZType>Linear</DecayCRZType>
     <DecayAPP>.005</DecayAPP>
     <DecayAPPType>Linear</DecayAPPType>
  </Roi>
  <Roi>
     <Name>PFD_Current_Airspeed</Name>
     <X>20</X>
     <Y>198</Y>
     <Xm>0.172</Xm>
     <Ym>-0.333</Ym>
     <Zm>0.981</Zm>
     <DecayTO>.005</DecayTO>
     <DecayTOType>Linear</DecayTOType>
     <DecayCRZ>.001</DecayCRZ>
     <DecayCRZType>Linear</DecayCRZType>
```

```
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>PFD_Current_Altitude</Name>
    <X>81</X>
    <Y>203</Y>
    <Xm>0.075</Xm>
    <Ym>-0.333</Ym>
    <Zm>0.981</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>PFD_Capt_FMA</Name>
    <X>49</X>
    <Y>181</Y>
    <Xm>0.125</Xm>
    <Ym>-0.261</Ym>
    <Zm>0.981</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>PFD_Speed_Bug_Setting_Indicator</Name>
    <X>23</X>
    <Y>185</Y>
    <Xm>0.172</Xm>
    <Ym>-0.285</Ym>
    <Zm>0.981</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
```

```
</Roi>
<Roi>
   <Name>PFD_Altitude_Bug_Setting_Indicator</Name>
   <X>84</X>
   <Y>182</Y>
   <Xm>0.075</Xm>
   <Ym>-0.285</Ym>
   <Zm>0.981</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>PFD_Vertical_Speed_Indicator</Name>
   <X>92</X>
   <Y>203</Y>
   <Xm>.056</Xm>
   <Ym>-0.33</Ym>
   <Zm>0.981</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Set_Course</Name>
   <X>203</X>
   <Y>124</Y>
   <Xm>-0.305</Xm>
   <Ym>-0.08</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
```

```
    <Name>MCP_AT_Arm</Name>
    <X>212</X>
    <Y>123</Y>
    <Xm>-0.34</Xm>
    <Ym>-0.09</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Set_Speed</Name>
    <X>228</X>
    <Y>117</Y>
    <Xm>-0.385</Xm>
    <Ym>-0.08</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Speed_Knob</Name>
    <X>233</X>
    <Y>130</Y>
    <Xm>-0.395</Xm>
    <Ym>-0.117</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_VNAV_Engage</Name>
    <X>242</X>
```

```
    <Y>117</Y>
    <Xm>-0.425</Xm>
    <Ym>-0.087</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Heading_Knob</Name>
    <X>250</X>
    <Y>130</Y>
    <Xm>-0.455</Xm>
    <Ym>-0.08</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Set_Heading</Name>
    <X>253</X>
    <Y>117</Y>
    <Xm>-0.455</Xm>
    <Ym>-0.08</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_LNAV_Engage</Name>
    <X>264</X>
    <Y>118</Y>
    <Xm>-0.495</Xm>
```

```
    <Ym>-0.087</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Set_Altitude</Name>
    <X>278</X>
    <Y>120</Y>
    <Xm>-0.539</Xm>
    <Ym>-0.08</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_Set_Vertical_Speed</Name>
    <X>295</X>
    <Y>121</Y>
    <Xm>-0.595</Xm>
    <Ym>-0.08</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_CMD_Engage_A</Name>
    <X>307</X>
    <Y>122</Y>
    <Xm>-0.644</Xm>
    <Ym>-0.091</Ym>
    <Zm>0.734</Zm>
```

```
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_CMD_Engage_B</Name>
    <X>314</X>
    <Y>124</Y>
    <Xm>-0.671</Xm>
    <Ym>-0.091</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_CMD_Disengage</Name>
    <X>312</X>
    <Y>140</Y>
    <Xm>-0.656</Xm>
    <Ym>-0.143</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>MCP_VS_Wheel</Name>
    <X>299</X>
    <Y>132</Y>
    <Xm>-0.511</Xm>
    <Ym>-0.129</Ym>
    <Zm>0.734</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
```

```
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Vertical_Speed_Engage</Name>
   <X>284</X>
   <Y>135</Y>
   <Xm>-0.558</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Altitude_Knob</Name>
   <X>285</X>
   <Y>138</Y>
   <Xm>-0.539</Xm>
   <Ym>-0.113</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Altitude_Hold_Engage</Name>
   <X>276</X>
   <Y>136</Y>
   <Xm>-0.539</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
```

```
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_APP_Mode_Engage</Name>
   <X>263</X>
   <Y>135</Y>
   <Xm>-0.495</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Heading_Select_Engage</Name>
   <X>254</X>
   <Y>135</Y>
   <Xm>-0.455</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Speed_Controlled_Level_Change_Engage</Name>
   <X>244</X>
   <Y>135</Y>
   <Xm>-0.425</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
```

```
</Roi>
<Roi>
   <Name>MCP_Speed_Set_Engage</Name>
   <X>220</X>
   <Y>133</Y>
   <Xm>-0.363</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_N1_Engage</Name>
   <X>212</X>
   <Y>135</Y>
   <Xm>-0.34</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
   <Name>MCP_Flight_Director_On_Off</Name>
   <X>205</X>
   <Y>135</Y>
   <Xm>-0.323</Xm>
   <Ym>-0.134</Ym>
   <Zm>0.734</Zm>
   <DecayTO>.005</DecayTO>
   <DecayTOType>Linear</DecayTOType>
   <DecayCRZ>.001</DecayCRZ>
   <DecayCRZType>Linear</DecayCRZType>
   <DecayAPP>.005</DecayAPP>
   <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
```

```
    <Name>Capt_Nav</Name>
    <X>150</X>
    <Y>200</Y>
    <Xm>-0.297</Xm>
    <Ym>-0.333</Ym>
    <Zm>0.923</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>Lower_EICAS</Name>
    <X>240</X>
    <Y>260</Y>
    <Xm>-0.56</Xm>
    <Ym>-0.585</Ym>
    <Zm>0.82</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>EICAS</Name>
    <X>280</X>
    <Y>200</Y>
    <Xm>-0.671</Xm>
    <Ym>-0.262</Ym>
    <Zm>0.981</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>OSV_Ahead</Name>
    <X>50</X>
```

```
    <Y>80</Y>
    <Xm>0.00</Xm>
    <Ym>0.10</Ym>
    <Zm>1.0</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>Landing_Gear</Name>
    <X>220</X>
    <Y>200</Y>
    <Xm>-0.59</Xm>
    <Ym>-0.262</Ym>
    <Zm>0.858</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>Overhead_Panel</Name>
    <X>280</X>
    <Y>20</Y>
    <Xm>-0.505</Xm>
    <Ym>0.389</Ym>
    <Zm>0.249</Zm>
    <DecayTO>.005</DecayTO>
    <DecayTOType>Linear</DecayTOType>
    <DecayCRZ>.001</DecayCRZ>
    <DecayCRZType>Linear</DecayCRZType>
    <DecayAPP>.005</DecayAPP>
    <DecayAPPType>Linear</DecayAPPType>
</Roi>
<Roi>
    <Name>Capt_CDU</Name>
    <X>180</X>
    <Y>270</Y>
    <Xm>-0.367</Xm>
```

```
      <Ym>-0.531</Ym>
      <Zm>0.82</Zm>
      <DecayTO>.005</DecayTO>
      <DecayTOType>Linear</DecayTOType>
      <DecayCRZ>.001</DecayCRZ>
      <DecayCRZType>Linear</DecayCRZType>
      <DecayAPP>.005</DecayAPP>
      <DecayAPPType>Linear</DecayAPPType>
  </Roi>
</ArrayOfRoi>
```

REFERENCES

1. Card, S.K., T.P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*. 1983: Taylor & Francis.
2. Laird, J., *The Soar Cognitive Architecture*. 2012: MIT Press.
3. J. K. Jacob, R. and K. Karn, *Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises*. Vol. 2. 2003. 573-605.
4. Fitts, P.M.J., Richard E. ; Milton, John L., *Eye Fixations of Aircraft Pilots. III. Frequency, Duration, and Sequence Fixations When Flying Air Force Ground-Controlled Approach System (GCA).* AIR MATERIEL COMMAND WRIGHT-PATTERSON AFB OH, 1949: p. 25.
5. Newell, A., *Unified Theories of Cognition*. 1994: Harvard University Press.
6. Taatgen, N. and J.R. Anderson, *The Past, Present, and Future of Cognitive Architectures.* Topics in Cognitive Science, 2010. **2**(4): p. 693-704.
7. Jilk, D.J., et al., *SAL: An explicitly pluralistic cognitive architecture.* Journal of Experimental and Theoretical Artificial Intelligence, 2008. **20**(3): p. 197-218.
8. O'Reilly, R.C., T.E. Hazy, and S.A. Herd, *The Leabra Cognitive Architecture: How to Play 20 Principles with Nature.* The Oxford handbook of cognitive science, 2016: p. 91.
9. Eliasmith, C., et al., *A large-scale model of the functioning brain.* science, 2012. **338**(6111): p. 1202-1205.
10. Anderson, J.R., *ACT: A simple theory of complex cognition.* American Psychologist, 1996. **51**(4): p. 355.
11. Anderson, J.R., *The Architecture of Cognition*. 1996: Lawrence Erlbaum.
12. Kieras, D.E. and D.E. Meyer, *An overview of the EPIC architecture for cognition and performance with application to human-computer interaction.* Human–Computer Interaction, 1997. **12**(4): p. 391-438.
13. Sun, R., *The CLARION cognitive architecture: Extending cognitive modeling to social simulation.* Cognition and multi-agent interaction, 2006: p. 79-99.
14. Franklin, S. and F. Patterson Jr, *The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent.* pat, 2006. **703**: p. 764-1004.
15. Gobet, F., et al., *Chunking mechanisms in human learning.* Trends in cognitive sciences, 2001. **5**(6): p. 236-243.
16. Just, M.A., P.A. Carpenter, and S. Varma, *Computational modeling of high-level cognition and brain function.* Human brain mapping, 1999. **8**(2-3): p. 128-136.
17. Wray, R.E. and R.S. Chong, *Comparing Cognitive Models and Human Behavior Models: Two Computational Tools for Expressing Human Behavior.* Journal of Aerospace Computing, Information, and Communication, 2007. **4**(5): p. 836-852.
18. Lim, M.Y., *Memory Models for Intelligent Social Companions*, in *Human-Computer Interaction: The Agency Perspective*, M. Zacarias and J.V. de Oliveira, Editors. 2012, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 241-262.
19. Rosenbloom, P.S., *The Sigma cognitive architecture and system.* AISB Quarterly, 2013. **136**: p. 4-13.

20. Rosenbloom, P.S., A. Demski, and V. Ustun, *The Sigma cognitive architecture and system: Towards functionally elegant grand unification.* Journal of Artificial General Intelligence, 2016. **7**(1): p. 1-103.
21. Watson, P.D., et al., *A COMPUTATIONAL MODEL OF RELATIONAL MEMORY BINDING IN THE HIPPOCAMPUS.* MECHANISMS OF HIPPOCAMPAL RELATIONAL BINDING. **1001**: p. 157.
22. Goertzel, B., *CogPrime: An Inegrative Architecture for Embodied Artificial General Intelligence.* 2012.
23. John, B.E. and D.D. Salvucci, *Multipurpose prototypes for assessing user interfaces in pervasive computing systems.* Pervasive Computing, IEEE, 2005. **4**(4): p. 27-34.
24. John, B.E. *CogTool Website*. 2014 [cited 2014 June 17, 2014]; Available from: https://cogtool.wordpress.com/.
25. Byrne, M.D.a.K., A, *Integrated Modeling of Cognition and the Information Environment: A Closed-Loop, ACT-R Approach to Modeling Approach and Landing With and Without Synthetic Vision System (SVS) Technology.* Proceedings of the 2003 Conference on Human Performance Modeling of Approach and Landing with Augmented Displays, 2003: p. 27.
26. Long, L.N., S.D. Hanford, and O. Janrathitikarn. *Cognitive robotics using vision and mapping systems with Soar*. 2010.
27. Jones, R.M., Laird, J.E., Nielsen, P.E., Coulter, K.J., Kenny, P., & Koss, F.V., *Automated Intelligent Pilots for Combat Flight Simulation.* AI Magazine, 1999. **20**(1): p. 27-41.
28. Forgy, C.L., *On the Efficient Implementation of Production Systems*, in *Department of Computer Science*. 1979, Carnegie-Mellon University.
29. Balbo, S., Ozkan, N., Paris, C., *Choosing the right task-modeling notation: a taxonomy*, in *The Handbook of Task Analysis for Human-Computer Interaction*, S. Dan Diaper, N.A., Editor. 2004, Lawrence Erlbaum Associates. p. 445-466.
30. Newell, A. and H.A. Simon, *Human problem solving*. 1972: Prentice-Hall.
31. Laird, J.E., *The Soar 9 Tutorial*. 2014, University of Michigan: University of Michigan.
32. Zelinsky, G.J., et al., *Eye Movements Reveal the Spatiotemporal Dynamics of Visual Search.* Psychological Science, 1997. **8**(6): p. 448-453.
33. Fitts, P.M., *The information capacity of the human motor system in controlling the amplitude of movement.* Journal of Experimental Psychology, 1954. **47**(6): p. 381-391.
34. Kondraske, G.V. *An angular motion Fitt's Law for human performance modeling and prediction*. in *Proceedings of 16th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. 1994.
35. Hillstrom, A.P. and S. Yantis, *Visual motion and attentional capture.* Perception & Psychophysics, 1994. **55**(4): p. 399-411.
36. Wickens, C.D., et al., *Attentional models of multitask pilot performance using advanced display technology.* Hum Factors, 2003. **45**(3): p. 360-80.
37. Steelman-Allen, K.S., et al., *N-SEEV: A Computational Model of Attention and Noticing.* Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 2009. **53**(12): p. 774-778.