IN-63
28220
P. 97

NASA Contractor Report 194976

Knowledge-Based Processing for Aircraft Flight Control

John H. Painter, Emily Glass,
Gregory Economides, and Paul Russell
Texas A&M University, College Station, Texas

NASA GRANT NAG1-1066
October 1994

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION.

OBJECTIVES.

Domain-Specific Software Architecture for Intelligent Control.

The overall objective was that of defining and developing software architectures for specific application to a class of dynamic system control problems which are called, variously, "Intelligent"[1] [2], "Expert"[3], "Declarative"[4], "Knowledge-based"[5], and/or "Fuzzy"[6]. Development was for the purpose of exposing the issues key to verification and validation of the resulting architectures and was carried only to the point of demonstration. A necessary part of that definition was the setting down of as strong a theoretical base as possible, in the usual control-theoretic and computer science senses. The research was multi-disciplinary, in that it enfolded control, artificial intelligence, and software engineering disciplines.

The overall objective was to approach the creation of software architectures in a balanced way, giving just as much attention to the software engineering issues as to the abstract theoretical control issues. Moreover, the control issues were to be approached in a way which blends computer science, decision science, and stochastic control. In particular, the issue of controlling systems whose performance is modeled semi-qualitatively was approached by expressing and exploiting *dualities* which exist between stochastic control and expert systems theories. The overall objective was addressed through several sub-objectives. These were naturally divided between decision/control and software engineering.

The specific domain which was chosen for application and demonstration was that of Aircraft Flight Control. The approach was that of symbolic interpretation and management of flight operational modes, to satisfy operational clearances. This was done by implementing expert, rule-based interpretation and meta-control of the aircraft automatic flight control system, AFCS. Rules of flight were taken from those published by

American Airlines for the Boeing-737 aircraft.

Qualitative Decision/Control Inference Under Uncertainty.

The Intelligent Control problem is to control dynamic systems which are modeled in two different ways. These systems possess the usual numerical state-variable model (linear or non-linear), defined on a direct-product space of numerical (measurable or observable) variables. However, at the highest level of modeling abstraction, the systems are described by performance measures, such as *operating modes*, which may be partitions of the numerical state space. Moreover, these measures are not unique, in that they possess uncertainty as to the partition boundaries. As subsets of a direct-product space, the performance measures overlap. This leads to describing qualitative dynamic system performance measures by conditional probabilities, p(performance event | num. var.), or *fuzzy* membership functions. The control paradigm is to process numerical sensor measurements to first infer the system's qualitative performance and then control it, much as a human would do.

System performance decision is embedded in Intelligent Control, either implicitly or explicitly. If a human operator is present, explicit decision is useful for explanation of controller actions. In any event, implicit decision is present, as may be seen from Kosko's formulation of Fuzzy Control[7]. This implicit decision, as an element of control, may be thought of as *conjunctive*, in the Decision Sciences[8] sense. As such, it is not hard decision, but soft, carrying with it a measure of its correctness. Thus, Intelligent Control may be said to be (Soft) Decision-Directed Control[9]. Moreover, a particular Fuzzy implementation (one of Kosko's[7] several) was shown to be dual with decision-directed conditional-mean (Bayesian) estimation and control.

This objective was to model and implement Intelligent Control for Dynamic Systems in such a way as to recognize and exploit dualities existing between Stochastic Decision/Estimation/Control, Fuzzy Control, Decision Sciences, and Expert Systems.

SIGNIFICANCE.

The significance of the research lay in two areas. The first was in the creation of theoretical architectures and algorithms for Intelligent Control. The second was in the implementation issues which were solved in order to make the architectures and algorithms workable and acceptable in practice.

Intelligent Control holds promise of providing a new class of control techniques applicable to systems of higher orders of complexity than is at present practical. One significance of the research was to make this extension in ways that are recognizable abstractions, or duals, of well understood methods of stochastic decision, estimation, and control. Such abstractions suggest ways to characterize and compare performance and to validate Intelligent Control algorithms, something that has been heretofore absent. This leads to contributions in a well-founded supporting theory for Intelligent Control, which has been slow in appearing.

RELATION TO PRESENT STATE-OF-KNOWLEDGE.

The present state of knowledge of Intelligent Control is a result of many individual efforts dating back to roughly 1985. In that year, there was formed an IEEE Committee on Intelligent Control. Since then, annual symposia have been held, with published proceedings (cf.:). Previous work has been principally driven by computer science (software) issues[10], such as *Knowledge Representation*[11], and by the long history of work in Fuzzy Logic [12]. Lately, however, there has appeared work on the fundamental mathematical modeling of Intelligent Control systems. The works of Kohn[4] and Nerode[13] are representative. This latter work attempts to set down a well-founded and complete supporting theory for symbolic control, based on modern algebra and autonoma theory. In its present state of development, the theory is deeply algebraic and topological, combining concepts from group theory with the more geometric concepts of state-space control theory.

The Kohn-Nerode work has already spawned a top-level architecture for an Intelligent Controller, within which they are working to complete their theory. Our architecture, at the first two levels of explication, is compatible with theirs. (Our architecture will be shown in the following section.) Our approach, however, was not based on a formal logical (algebraic) model, but on a more geometric (and visualizable) object-oriented[14] model, which uses *Frames*[11] for knowledge structuring and modified-Bayes (Fuzzy-like) knowledge representation and inference implementation. That is, knowledge of the higher level qualitative performance abstractions is contained in functions, like fuzzy Membership Functions, or Bayes a posteriori conditional probability functions. Inference processing combines fuzzy (as in Kosko's Fuzzy Associative Memories[7]), crisp-rule-based, and procedural processing, as appropriate to the level of inference. Note, however, that our fuzzy inference is computed using stochastic control dualities. That is, classical logical set combining (soft "and/or") is used, rather than the newer fuzzy operations (hard conjunction/disjunction)[15].

The research reported here complements other ongoing work to complete a topological foundation for Intelligent Control, by dealing specifically with knowledge representation and inference implementation issues, and by developing dualities with stochastic decision/estimation/control.

A SYNOPSIS OF THE RESEARCH.

The present work has been funded for six years from a variety of sources. Early funding was from E-Systems of Dallas, TX., matched by funding from a NASA Center for Space Power, established at Texas A&M University. Recent support was from NASA Headquarters and NASA Langley Research Center, under grants, NGT-50401 and NAG1-1066. These grants expired in 1992 and 1994, respectively. This report compiles a collection of conference papers written under those grants.

The work reported here has focused on the specific application of managing the flight of a transport-class jet aircraft. However, the architectures and algorithms developed have been generic to dynamic systems. (Indeed, previous work focused on symbolic interpretation of radio signals and intelligent control of an AC power distribution system.) This is because the symbolic processing draws from data-bases (libraries) which can be loaded with knowledge specific to a particular application. This follows the *new thinking* in Expert Systems which favors model-based reasoning in specific, knowledge-rich problem domains[16], as opposed to knowledge-poor, general problem solvers[17]. Thus, progress has been made in the general area of Intelligent Control of dynamic systems, even though a specific aircraft control application has been the recent focus of the work. What is being said here is that the technology is transferable. And, the implemented software is being done in such a way that it is reusable.

At the present point in time (December, 1992), an architecture has been formulated, as in Figure-1, below, which is shown in *Data-flow* (Block Diagram) form.

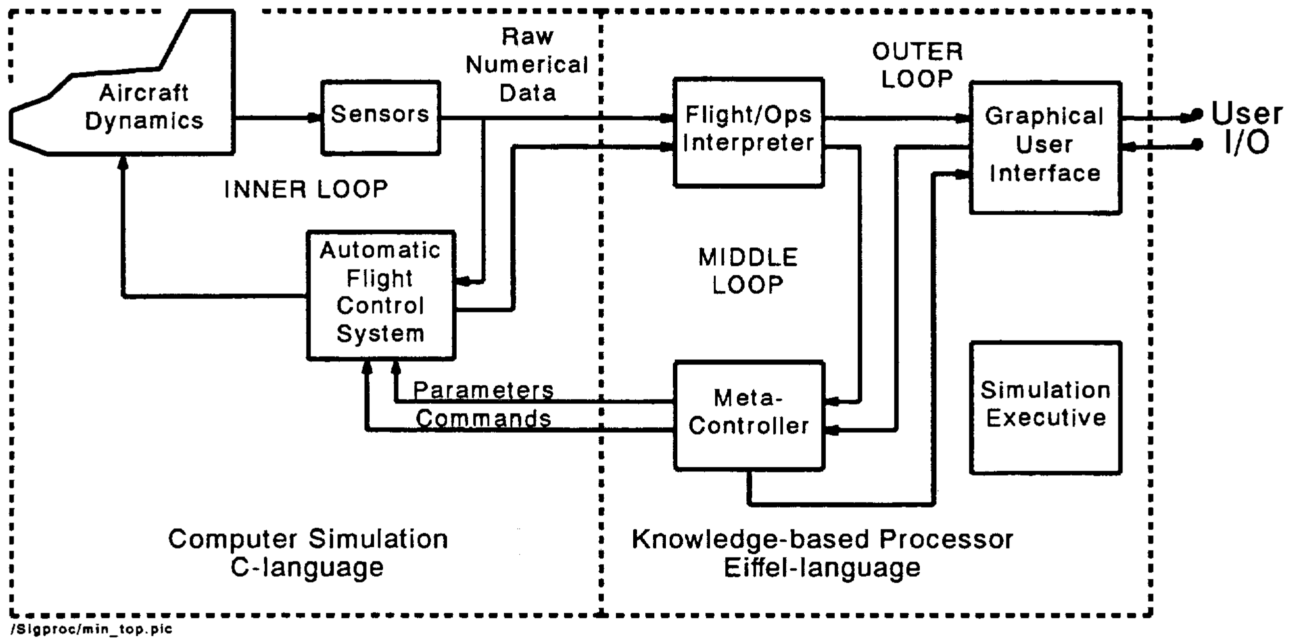
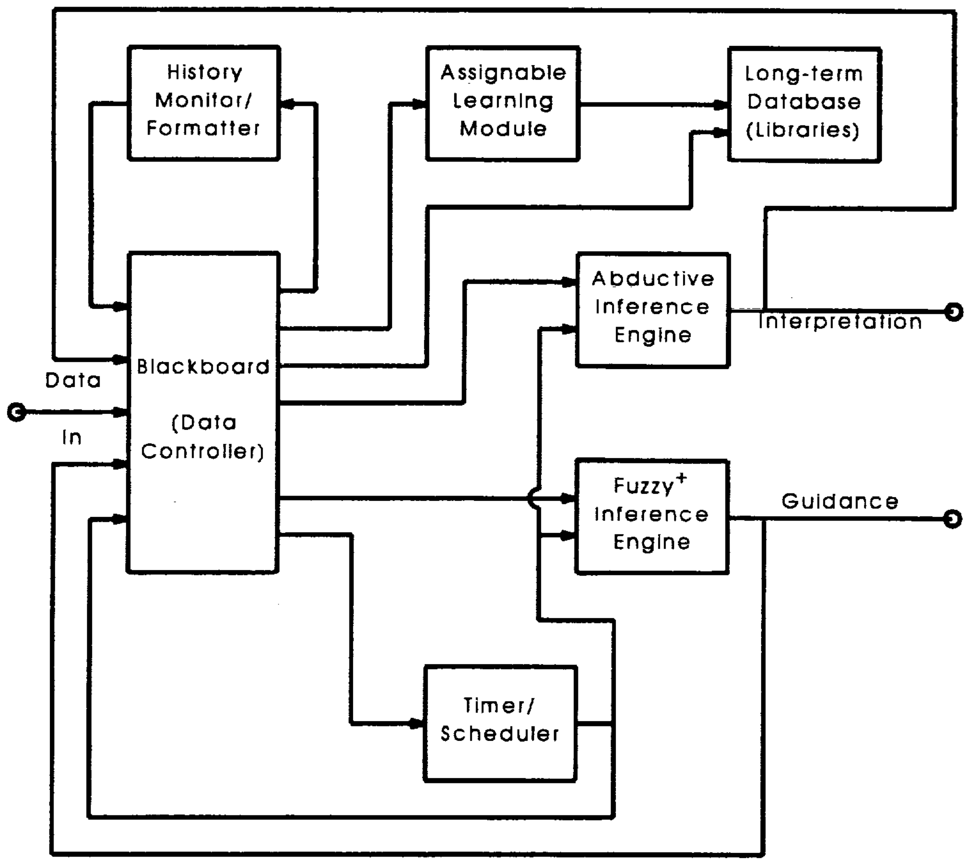


Figure 1. Top-level Data-flow Diagram.

This top-level architecture shows three major elements, being a symbolic interpreter for operating mode, a meta-controller, and a graphical user interface (GUI). The interpreter is a soft-decision element, dealing with a pre-defined, hierarchical set of system operating modes and conditions[5]. It employs modified-Bayes decision, using pre-defined conditional probability functions (fuzzy membership functions), and an *abductive* scheme for computing decision confidence[18] and for testing for conflicting evidence. The Meta-Controller employs a combination of fuzzy-, crisp-, and procedural processing to formulate inputs to the standard automatic flight control system, in response to input directions in English language, using stored procedural rules. For the fuzzy (abductive Bayes) processing, the Interpreter computes the necessary decision elements and passes them to the Controller. The GUI provides necessary input/output to the symbolic processors and to the simulation of the dynamic system (aircraft).


History Monitor/ Formatter
Assignable Learning Module
Long-term Database (Libraries)
Abductive Inference Engine
Interpretation
Data In
Blackboard (Data Controller)
Fuzzy+ Inference Engine
Guidance
Timer/ Scheduler

/Sigproc/mln_2nd.plc rev.: 2/10/92

Figure 2. Symbolic Processor.

Figure-2 shows a second-level view of the architecture, sans GUI. Of the modules shown in the figure, only three are currently under development. These are the Blackboard[19], the Abductive Inference Engine (Interpreter), and the Fuzzy$^+$ Inference Engine (Meta-controller). The Blackboard is used principally to limit the *ripple effect* on other modules of modifications to any single module. It is essentially an internal data controller. A classical Blackboard approach would internalize the Interpreter and Meta-Controller. We have removed them from the Blackboard, proper, as a matter of emphasis, and to facilitate development. But, note that our total architecture is similar to the Blackboard-based controller of Skillman, Kohn, et. al.[20]

WORK IN PROGRESS ELSEWHERE.

There is significant work being performed elsewhere, as indicated by the annual Proceedings of the IEEE International Symposium on Intelligent Control. That work is too voluminous to recount here. Also, we have referenced other work in the previous section entitled, RELATION TO PRESENT STATE OF KNOWLEDGE. In addition, there is work on Intelligent Control being presently performed under the DARPA Domain-Specific Software Architectures Initiative, as recently reported in the Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design. Reported work by Hayes-Roth, et. al.[21] also uses the concept of *meta-controller*, as we do. More work by Platek and Nerode on *Hybrid Control* was verbally reported[22], though not documented, as was the work by Kohn and Nerode[13].

Curiously, not a significant percentage of the ongoing work has been published in technical journals. This may reflect the fact that there seems to be no single journal which is well suited to Intelligent Control. Publications are scattered through a variety of journals.

CHAPTER-1 BIBLIOGRAPHY

1. Fu, K.S.; Learning Control Systems and Intelligent Control Systems: An Intersection of Artificial Intelligence and Automatic Control; *IEEE Trans. Auto. Control*; vol. AC-16, No. 1, Jan. 1971, pp. 70-72.

2. Saridis, G.N.; Knowledge Implementation: Structures of Intelligent Control Systems; *Proc. 1987 IEEE Intl. Symp. Intelligent Control*; Jan. 19-20, 1987, pp. 9-17.

3. Astrom, K.J., Anton, J.G., and Arzen, K.E.; Expert Control; *Automatica*; No. 13, 1986, pp. 277-286.

4. Kohn, W.; A Declarative Theory for Rational Controllers; *Proc. 27th Conf. Decision & Control*; Dec. 1988, pp. 130-136.

5. Painter, J.H.; Knowledge-based Control; *Proc. 1992 IEEE Symp. Computer-aided Control System Design*; Mar. 17-19, 1992, pp. 138-147.

6. Zadeh, L.A.; A Rationale for Fuzzy Control; *Trans. ASME; Series G (USA)*, No. 94, 1974, pp. 3-4.

7. Kosko, B.; *Neural Networks and Fuzzy Systems*, Chap.-8; Prentice-Hall; 1992.

8. Sutherland, J.W.; Assessing the Artificial Intelligence Contribution to Decision Technology; *IEEE Trans. Syst., Man, & Cyb.*; vol. SMC-16, No. 1, Jan/Feb. 1986, pp. 3-20.

9. Painter, J.H. & Jones, S.K.; Results on Discrete-Time Decision-Directed Integrated Detection, Estimation, and Identification; *IEEE Trans. Comm.* vol. COM-25, No. 7, July, 1977.

10. Porter, B.; Issues in the Design of Intelligent Control Systems; *Plenary Lecture, 12th IMACS World Congress on Scientific Computation*; Paris, France, July 18-22, 1988.

11. Minsky, M.; A Framework for Representing Knowledge; *The Psychology of Computer Vision; McGraw-Hill*, 1975, pp. 211-277.

12. Maiers, J. & Sherif, Y.S.; Applications of Fuzzy Set Theory; *IEEE Trans. Syst., Man, & Cyb.*; vol. SMC-15, No. 1, Jan./Feb., 1985, pp. 175-189.

13. Nerode, A. & Kohn, W.; An Autonomous Systems Control Theory: An Overview; *Proc. 1992 IEEE Symp. Computer-aided Control System Design*; Mar. 17-19, 1992, pp. 204-210.

14. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W.; *Object-Oriented Modeling and Design*; Prentice-hall, 1991.

15. Bellman, R.E. and Zadeh, L.A.; Decision-making in a Fuzzy Environment; *Management Science*; No. 4, 1970, pp. 141-164.

16. Chandrasekaran, B.; Generic Tasks in Knowledge-based Reasoning:High-Level Building Blocks for Expert System Design; *IEEE Expert*; Fall, 1986, pp. 23-30.

17. Newell, A. & Simon, H.A.; *Human Problem Solving*; Prentice-Hall, 1972.

18. Punch, W.F., Tanner, M.C., Josephson, J.R., & Smith, J.W.; Peirce, A Tool for Experimenting With Abduction; *IEEE Expert*; Oct. 1990, pp. 34-44.

19. Nii, H.P.; Blackboard systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures; *The AI Magazine;* Summer, 1986, pp. 38-53.

20. Skillman, T.L., Kohn, W., Nguyen, D., Ling, C., & Dodhiawala, R.; Class of Hierarchical Controllers and Their Blackboard Implementations; *J. Guidance, Control, & Dynamics*; vol. 13, No. 1, Jan./Feb. 1990, pp. 176-182.

21. Hayes-Roth, R., Erman, L., & Hayes-Roth, b.; Domain-Specific Software Architectures: Distributed Intelligent Control and Management; Op. Cit., *1992 IEEE CACSD*; pp. 117-128.

22. Platek, R. & Nerode, A.; Hybrid Control; Op. Cit., *1992 IEEE CACSD;* page-137.

CHAPTER 2

KNOWLEDGE-BASED CONTROL.

by
John H. Painter
Department of Electrical Engineering
Texas A&M University
College Station, Texas 77843-3128
409, 845-7441
painter@zadok.tamu.edu

"If we could first know where we are, and whither we are tending, we could better judge what to do, and how to do it."
- A. Lincoln; June 16, 1858.

"The heart of the prudent getteth knowledge; and the ear of the wise seeketh knowledge." [Proverbs 18:15].
- King Solomon; circa 1,000 B.C..

INFERENCE AND CONTROL.

INFERENCE EMBEDDED IN CONTROL.

In 1858, Abraham Lincoln pretty well made the case for Knowledge-based Control with his statement about knowing first and doing second. Down through the centuries, it has been appreciated that knowledge is the precursor to prudent action. So it is, with the emerging area which is the subject of the present paper.

Knowledge-based Control, also known as Intelligent Control, comes from the *Systems and Control* discipline, as a branch which is traceable back to a seminal paper by K.S. Fu [1]. Twenty years ago, he recognized that automatic control should benefit from developments in *Artificial Intelligence* (AI), if not vice versa. That these benefits have been so long in coming, is a comment upon the difficulties inherent in AI development and in transfer between two very different technologies. It is one purpose of the present paper to point out dualities between AI and Control which make the transfer easier.

Since the 1950's, it has been recognized that inference is an embedded element of modern control. Bellman's [2] development of dynamic programming employed explicit inference, in the form of decision, as an integral step of control. Kalman's [3] famous contribution also showed inference, specifically prediction, to be inherent in (state) control. Both examples of inference embedded in control illustrate what is formally codified in the now famous *Separation Theorem* [4] of *stochastic control*. The theorem states that, under fairly general conditions, a *reasonable* closed-loop control system may be formally partitioned into three parts; comprising the system (plant) under control, an inferential estimator, and a memoryless controller.

The difference between traditional *Modern Control* and Knowledge-based Control is that the embedded inference and memoryless control algorithms now contain qualitative, as well as numerical, processing elements. That is, the inferences and resulting controls are focused on qualitative performance measures, as well as numerical. As a dual to the usual required state-space modeling of the plant, now there is the required modeling of *qualitative states*.

THE PARADIGM.

The general pattern in knowledge-based control of a dynamic system is as follows: First, the usual numerical state-variable modeling of the plant takes place, with a preference given to states which are available as sensor readings, or are, at least, easily derivable (observable) therefrom. Next, a state-space is constructed, which is the *Cartesian product* of the states. (Actually, it is a direct-product space, since the door is left open to measurable states which are inherently qualitative, and not numerical.) Then, and most importantly, qualitative states are defined as partitions of the space. These qualitative states are most generally "*operating modes*," as defined by a human operator. Such modes are not directly measurable, but are observed by (human) inference processing of the available sensor measurements. Control is then exerted to maintain or modify these inferred states (modes).

As an example, consider the management of the flight of a transport-type aircraft. By *management* is meant an automation of the control activities traditionally performed by the pilot and/or flight engineer. It is assumed that the usual numerical automatic flight control system (AFCS) is present. Thus, the pilot concerns himself with evaluation of flight performance and formulation of inputs to the autopilot, to accomplish the goals of the flight.

The usual sensor readings are assumed to be available, as in Figure-1, below. Based on these available readings, and others derivable from them, qualitative operating modes are defined, as in Figure-2. Given the sensor readings and the defined qualitative modes, mode is inferred and control exerted, based on knowledge of the aircraft and of the rules of flight.

The list of operating modes is assumed finite, with *mode_unknown* covering the rest of the possibilities. Thus, the required inference is deci-

SPEED	ALT.	FORW_∠	LAT_∠	HEADING	POWER	RESOURC.	AUX_DEV.
IAS Mach.	Alt.	Pitch	Bank	Mag. Comp. (TH)	EPR EGT N(rpm) Eng-#	Fuel_Qty.	Flaps Slats Gear Spd_Brk

/Sigproc/lst_raw.tbl

Figure 1. Raw Numerical Data List.

MODE	STATUS	CONTROL	MANUVER	CONSTRAINTS
Takeoff Clean_up Climb_out Climb_on_course Cruise Descend_on_course Vector Initial_approach Final_approach Go_around Land Unknown	Normal Emergency Alarms: Systems, Fuel.	Pilot in Command. Air Traffic Control Other	Pilot Computer/ Autopilot/ Flight Control System.	Manuver Power Fuel Systems Weather

/Sigproc/lst_sops.tbl

Figure 2. Flight Operations List.

sion, rather than estimation. That is, given the sensor measurements, it is to be decided into which partition of the direct-product space the observed *"point"* fits. Given that decision, an input to the AFCS is then formulated, based on known goals and characteristics of the aircraft.

ANTECEDENTS.

Although the above paradigm sounds simple (and it is accomplished routinely by human pilots) the automation thereof is not trivial for several reasons. First, in the definition of operating modes, the partitioning of the state space, is not clean. That is, the operating modes are not unique, in terms of the sensor measurements. Second, the decision inference method is not obvious. This is because reasonable limiting of the number of operating modes may cause some sensor data to apparently contradict a mode that is "*essentially true*". The decision method must accomodate these anomalies. Finally, the method for synthesizing qualitative commands, accompanied by numerical prescriptions, based on qualitative performance interpretations, is also not obvious. Thus, is examined the *information and control* discipline, as well as the *artificial intelligence* discipline to determine what prior results might apply to the present problem.

DISCRETE-EVENT DYNAMIC SYSTEMS.

A particular branch of control theory which has been active for twenty years or more is that of discrete-event dynamic systems (DEDS). This discipline has attempted to create, for dynamic systems exhibiting a finite (or at least countable) number of states, a theoretical basis parallel to that existing for continuous-state dynamic systems. Upon these discrete state spaces, "*events*" are defined as the causes of the abrupt transitions of the state-vector between its discrete values. A recent paper by Ramage and Wonham [5] provides a clear summary of the application and methods of DEDS. Effort has focused on *admissable event trajectories* through the state space (sequences of events). In the inference context, effort is to determine if, given a certain sequence property, an observed sequence exhibits that property. In the control context, a trajectory is controlled to have the desired property. Methods of analysis have been *logical*, if time is not a consideration, or *timed*. Among the timed approaches are non-stochastic (Petri nets) or stochastic (Markov processes). Generally, the analytical approaches have been arithmetic. That is, they have not been geometric.

In the paradigm previously described (aircraft example), a finite set of events is defined on a continuous state-space. In the decision context, the events are not the causes of transitions from one region of the space to another, but are the partitions of the space, itself. Interest, here, is not in a sequence of operational modes, but in identifying the individual modes. In the control context, interest is not in some average control of a sequence of modes, but in just moving very deterministically to a succeeding mode. Finally, modeling and processing methods are desired, which give maximum insight (visibility) into the nature of the problem, itself. Thus, the search through antecedents is continued.

EXPERT SYSTEMS: FIRST GENERATION.

Next is that branch of AI, known as Expert Systems. Therein are programs particularly constructed to perform qualitative inference. It is natural to inquire whether such software might be naturally employed to perform the embedded inference required in Knowledge-based Control.

The success of early Expert System work devolved from the development of the "General Problem Solver (GPS)" by Newall and Simon[6], in the 1960's, which provided the foundation for much of the subsequent development in Expert Systems. According to Giarratano and Riley[7], GPS produced the now commonly accepted architecture for an expert system, comprising i)-long-term memory (rules), ii)-short-term memory (working memory), and iii)-cognitive processor (inference engine). GPS was founded upon the assumption that much machine *reasoning* could be done, based on IF-THEN types of rules. This assumption was shown by subsequent events to be true.

Winston[8] related that GPS focused on a state description of the problem to be solved. In particular, it was goal driven, to move the current state of the problem to some goal state. A distance measure was employed upon current and goal states to measure progress toward a solution. This procedure is dual, of course, to feedback control theory, wherein an error between input and output drives the system. Since the state of the GPS problem was not numerical, the problem control was by search through a tree, to move current state to goal state. In modern expert system parlance, this was *forward-chaining, depth-first search*.

The original Newell and Simon approach dominated expert system work for twenty or more years. And, this approach was marked by a preference for algebraic, logical processing over what might be termed the geometric approach, the latter based on appreciation of natural organizations of knowledge-rich domain-specific problems. Thus, expert systems tools and shells were created, to which the knowledge representation and inference procedures for diverse domain-specific problems must needs be conformed. This paradigm came to be known as the "First Generation" of expert systems. It reached a limit to growth, a barrier characterized by *brittleness,* the inability to respond to increasing problem sophistication with a complementary sophistication of inference. Thus, the characterization by Giarratano and Riley[7] as "the eternal beginner."

THE DECISION SCIENCES.

There is another discipline, known as *Decision Science (DS)*, interested in problems very similar to that of Intelligent Control. It owes its existance to ancestors in industrial engineering, operations research, econometrics, cybernetics, etc. A close examination of the contributions of Expert Systems to problems of interest in the Decision Sciences community has been made by Sutherland[9]. Because Sutherland's formulation of the decision problem is so congruent to the problem of knowledge-based control, his precepts are briefly reviewed here.

Sutherland first defined the decision problem as being conjunctive. That is, it had two sequential parts, being, first, the recognition of an event (e), and, second, the formulation of a response (r). Both event and response are characterizable in terms of an abstract *state.* Moreover, for an action-oriented problem class, there existed a *function* (i.e., a mapping), f : (e,r), relating response to recognized event. He identified sixteen possible cases, being all (e,r) combinations resulting from four degrees of *uncertainty* in either event or response. The four degrees of uncertainty (from the stochastic control viewpoint, not Sutherland's) ranged from both event and response being purely deterministic to both wholly stochastic.

Sutherland showed that Decision Science and Computer Science (AI) had not (circa 1985) been working on the same problems. Specifically, AI had been focused on either event-recognition or response-formulation, while DS has focused on the conjunction of the two. His bottom-line assessment was that (first-generation) expert systems had not evolved to the point of being able to handle problems which are characterized by a large measure of uncertainty in the data. His reasoning was that the purely logical (arithmetic) algorithms of the first-generation inference engines would have to yield to facilities capable of dealing mathematically with "*graded qualitative valuations.*" He prophesied that AI would have to embrace certain elements of "*fuzzy technology.*"

FUZZY METHODS.

Fuzzy Set Theory was created by Zadeh [10] twenty-five years ago. Within five years, its use in decision theory was being examined [11]. Within nine years, its use in control was being promoted [12]. Later (after 1986), it was implemented in practical control systems[13].

Fuzzy Set Theory is a dual of Bayesian Conditional Probability Theory, in the sense that *events* may be modeled by set functions whose range is the positive reals. Moreover, these functions may be manipulated to create similar positive real functions defined on subsets of the original domain. Therefore, in both cases, the set functions may be used to calculate the solutions of decision problems.

As a case upon which to focus, consider the airplane example. There the real-number domain might be the sensor reading, "Indicated Air Speed," IAS. On this set of positive numbers are defined two events, being "Takeoff" and "Landing." Now, the decision problem is to take an airspeed reading and to make the (hard) decision on whether the aircraft is taking off or landing.

In the Fuzzy context, a priori knowledge about the aircraft is used to create two functions. One is representative of certainty about what airspeeds should be observable during takeoff. The other models landing. These functions are called "*Membership Functions*," according to the idea that the observed airspeed has membership in the subsets supporting the two disparate event definitions. These two functions are unimodal (by design) and are arbitrarily scaled to have unity maximum value. The decision is obtained by choosing the event whose membership function is greatest at the value of the sensed airspeed. See Figure-3, below, for an illustration.

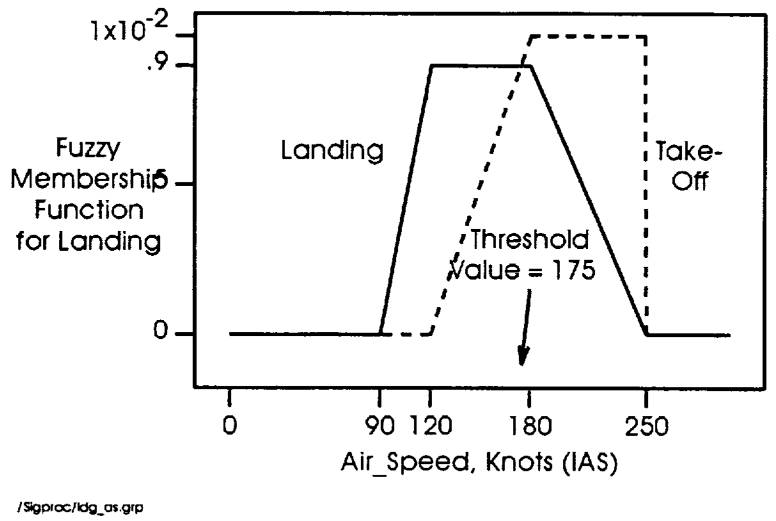


Figure 3. Membership Functions for Takeoff/Landing.

In the Bayesian context, the functions formed are so-called "*a priori conditional probability density functions.*" They are formed in exactly the same manner and have the same shape, as for the fuzzy membership functions. However, they are not arbitrarily scaled, but have unit area. The reason for the unit area requirement is that, to obtain functions for other events defined on the domain, the combining algorithms do not admit arbitrarily scaled functions. The same decision rule is employed, which is called "Maximum-Likelihood Decision Rule."

NOTE: (This note was not in the original paper.) - In a later paper, reproduced below as Chapter-3; Soft Fuzzy Control, The scaling issue is completely dealt with. There, it is shown under what conditions the Membership Functions are scaled to unity maximum value or to unity area.

Now, it is seen that it doesn't matter what the functions are called, so long as the same decision rule is used. Scaling, however, does matter. It can be seen from Figure-3 that while the Bayes decision threshold is 175 knots, the Fuzzy decision threshold is 180. Thus, there is a relative decision bias of 5 knots between the two. For an indicated air speed of 177 knots, the two methods yield different decisions. Figure-3 plots the membership functions with the required Bayes scaling.

Fuzzy Control [14] is decison-directed [15] control. By that is meant that controls implemented from a fuzzy theory basis result as the conjunctive solution of two decision problems. First, based on measured sensor data, it is decided what event has taken place. Second, given that event, it is decided what control to exert.

Fuzzy Control requires two sets of membership functions. The first relates sensor data to event. The second relates event to control action. A joint, two-dimensional membership function may then be synthesized, relating measured sensor data directly to control. The result is then a "*fuzzy set of control*" [14], which must be "*defuzzified*", in order to obtain a unique control effort.

In manipulating the various fuzzy membership functions, there are two different sets of combining algorithms which may be used. The first, and currently most popular, employs the "*conjunction*" and "*disjunction*", characterized by taking pointwise minima and maxima, respectively, of two membership functions. These are described as the "***hard logical AND***" and "***hard logical OR***", respectively. The other set is just the usual "*soft*" logical connectives [11]. It is worthwhile to note that if the soft connectives are used with unit area membership functions, there results just the usual Bayes conditional probability density functions for subsets of the domain set. Subset membership functions derived using the two differing sets of connectives may or may not have the same shape (ignoring scaling).

It is concluded that the general ideas of fuzzy control may be applied to the present problem, even though specific implementations may use the standard (soft) Bayes manipulations. What is important is that there exists support for qualitative inference and control which is essentially geometric, rather than algebraic. If, now, expert systems may admit these geometric approaches to modeling, qualitative inference, and control, a solution to knowledge-based control will be synthesizable therefrom.

EXPERT SYSTEMS: SECOND GENERATION.

The first generation of expert systems was characterized by logic-based implementations. A high-level computer language (PROLOG) was even developed to support such implementations. However, use of that language required that the problem be formulated in the first-order predicate calculus. Thus, the design of expert systems was driven by the programming necessary to embody the paradigm descended from the pioneering efforts of Newell and Simon[6]. The problem definition had necessarily to be shaped into the form required by existing expert system tools. Thus, was born a need for *Knowledge Engineers,* whose task it was to translate the natural problem requirements into forms suitable to programming with the extant tools. Expert system development became characterized by programmers learning enough about the problem domain to develop efficient expert systems.

With the advent of the second generation, the so-called *domain experts* (engineers, in the present context), become the expert system developers, with programming methodology no longer dominating the effort. This is because of a shift in focus on the task of translating problem requirements into programmable form. The formulation is done at a much higher level than previously. Whereas, the problem was abstracted at the programming level, now the problem is abstracted at a level dealing with a generic combination of problem, knowledge representation, and inference strategy[16]. Now, domain experts (in the information and control sciences) may learn enough about symbolic computing to specify the inference engine and knowledge representation down to the (object-oriented) programming level.

Bylander and Chandrasekaran[16] draw attention to the fact that knowledge representation, if unconstrained by a priori programming requirements, is strongly influenced by the combination of problem nature and inference strategy. Thus, there is a knowledge representation which is natural to that combination. "*Natural knowledge representation*" is referred to in [16] as the "*interaction problem.*" There, Minsky[17] is cited as having proposed *frames* as a knowledge representation suitable to the interaction problem. They[16] note that the emphasis in frame representations is on describing the conceptual structure of the domain.

It is intuitive to an engineer in the information and control sciences to characterize a problem by its *functional flow.* That is, the information processing functions being performed are specifically isolated and defined, from the overall problem. Then, the flow of data through these functions is diagrammed, according to the (problem-based) natural sequence of operations. Then, the functionality implied by the diagram is mapped (sometimes directly) into hardware and/or software. In computer science, this is called the "*data-flow*"[18] architecture. As an example, the top-level data-flow diagram for the knowledge-based aircraft management problem is as shown below in Figure-4.

The natural abstraction of the knowledge-based control problem requires the isolation and definition of the functions (or tasks) to be performed by the inference engines. Note that a complete knowledge representation is not obtained without consideration of the inference process.

It has been noted that knowledge-based control is decision-based, requiring a formal dealing with uncertainty. What is needed is definition of a set of inference functions, suitably generic to cover the domain of dynamic systems. Such a set of useful functions has been the subject of much work by Chandrasekaran [19] and his associates [16] [20] during the last several years. The refinement of the generic function approach shall well serve the present purposes.

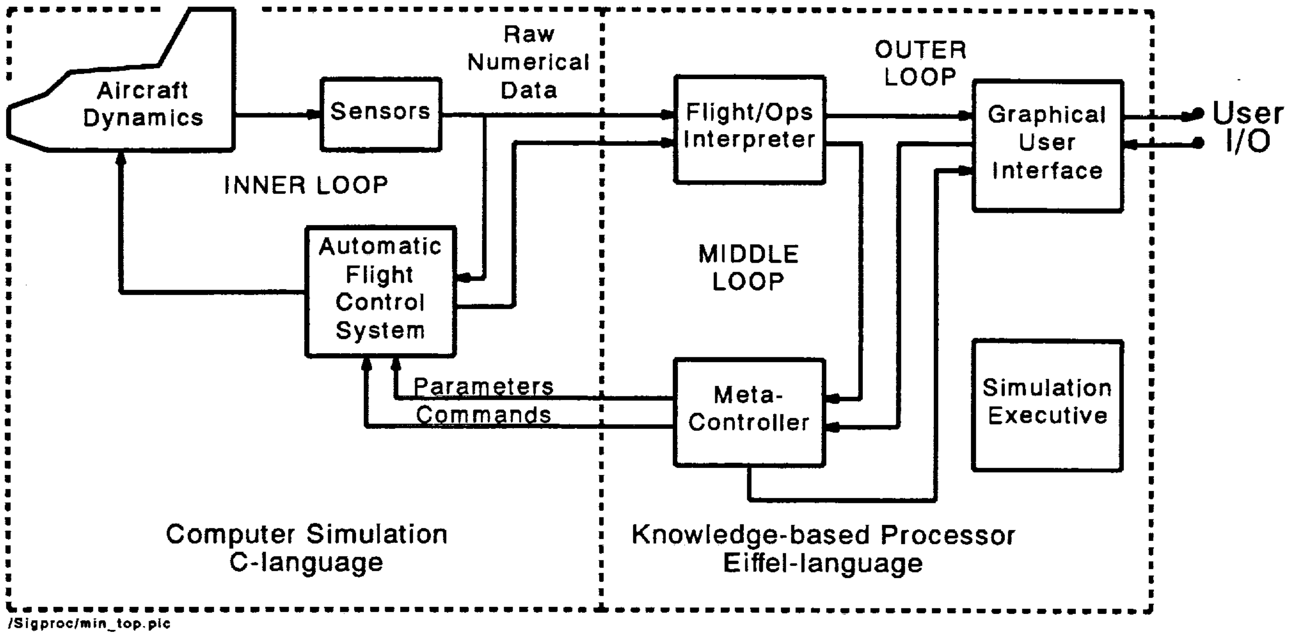


Figure 4. Top-level Data-flow Diagram.

A SPECIFIC APPROACH.

MODEL-BASED KNOWLEDGE REPRESENTATION.

Traditional first-generation expert systems typically consisted of a knowledge base and an inference engine, separately. Two standard approaches to knowledge representation were Production Rules and Frames[21]. Such representations must, in general, provide both syntax and semantics, defining symbols to be used and specifying how meaning is to be attached to the arrangement of symbols[8]. Such a representation may be described as a *semantic net,* comprised of nodes and links. The nodes are data objects and the links are relations, following the nomenclature common to object-oriented modeling and programming[22].

What may be generally viewed as a semantic net may, in the dynamic system domain, better be viewed as a hierarchical inference tree. Therein is defined a hierarchical set of decision hypotheses, with the most general at the top. The most general hypotheses are the most generally defined operating modes. These are defined in terms of linguistic variables. The least general, at the bottom of the tree, are the sensor numerical data corresponding to the linguistic operating modes.

The decision inference is inductive, proceeding up the tree. Control inference is deductive, proceeding down the tree. For a strictly rule-based implementation, these two would correspond to forward-chaining and backward-chaining, respectively.

An instance of such an inference tree, for aircraft, is set forth as Figure-5, below.

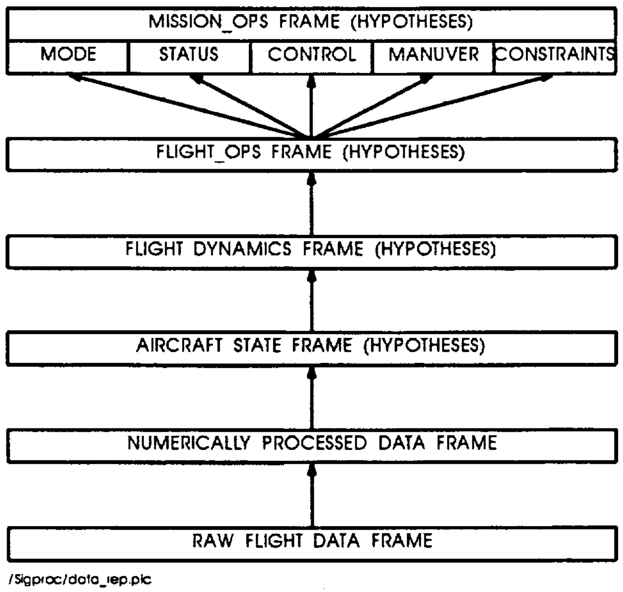

MISSION_OPS FRAME (HYPOTHESES)
MODE
STATUS
CONTROL
MANUVER
CONSTRAINTS
FLIGHT_OPS FRAME (HYPOTHESES)
FLIGHT DYNAMICS FRAME (HYPOTHESES)
AIRCRAFT STATE FRAME (HYPOTHESES)
NUMERICALLY PROCESSED DATA FRAME
RAW FLIGHT DATA FRAME
/Sigproc/data_rep.pic

Figure 5. Inference-Task, Data Representation.

The figure shows four levels of hypotheses. These are compound hypotheses, made up of more elementary hypotheses, having names like "MODE," "STATUS," "CONTROL," "MANUEVER," and "CONSTRAINTS." The highest level concerns the mission of the dynamic system. The next level concerns its operation as a system. Then is a level concerning its dynamics. The next concerns its individual qualitative states. Then is a level of description (which is not a decision hypothesis), being its numerical states, not raw, but after numerical processing. Finally, is the level of raw numerical sensor (and other non-hypothetical) data.

Viewing compound hypotheses as sets, as in decision science, each level of description has the appearance of a direct-product space of linguistic variables. As an analyst thinks of a vector space for numerical variables, a programmer may then think of a data frame for linguistic variables.

Another refinement is added, which further justifies the frames choice. And, that is that each frame (-level) is treated as an object in an object oriented programming environment. Then, procedures (methods) are attached to each slot in each frame, implementing the inference (decision) process from level to level. The inheritance mechanism may be used to advantage, here, to produce efficient, readable code.

For this knowledge representation, a complementary inference procedure is now specified.

ABDUCTIVE INFERENCE.

In fuzzy inference, (Bayes) *Maximum Likelihood* decision is used. This algorithm computes the (soft) product of the various membership functions. The algorithm works properly, provided there is no conflicting evidence present. However, suppose that some part of the dynamic system or operating procedure fails, not badly enough to throw the system into some other mode, but badly enough to yield data in conflict with the actual operating mode.

For example, consider an aircraft on final approach to landing, which is processing sensor measurements of indicated air speed (IAS), altitude (ALT), rate of climb (ROC), flight path angle (FPA), engine power (EPR), FLAPS, and GEAR, in order to make the interpreted decision "MODE_FLT_OPS = APPR_FNL". The variable, GEAR, will enter the computation as P(GEAR|LAND) = 0, where P(*|*) denotes membership function. Thus, the entire ML-product will be reduced to zero, and the computation will fail to yield a decision. Now, it is clear that the airplane is still on final approach, but the gear just hasn't come down (for whatever reason). What is needed is a decision computation which will still yield "MODE_FLT_OPS = APPR_FNL," but will also yield "STATUS_FLT_OPS = ALARM_GEAR."

This example shows that rudimentary (Bayes or Fuzzy) decision won't suffice. What is needed is a modified decision rule, such that the fuzzy foundation is retained, but conflicting evidence is accomodated. A decision framework which admits this modification is that of Abductive Inference, as formulated in the Chandrasekaran group and clearly reported by Punch[20] et. al. Figure-6 shows a functional flow diagram for Abductive Inference.

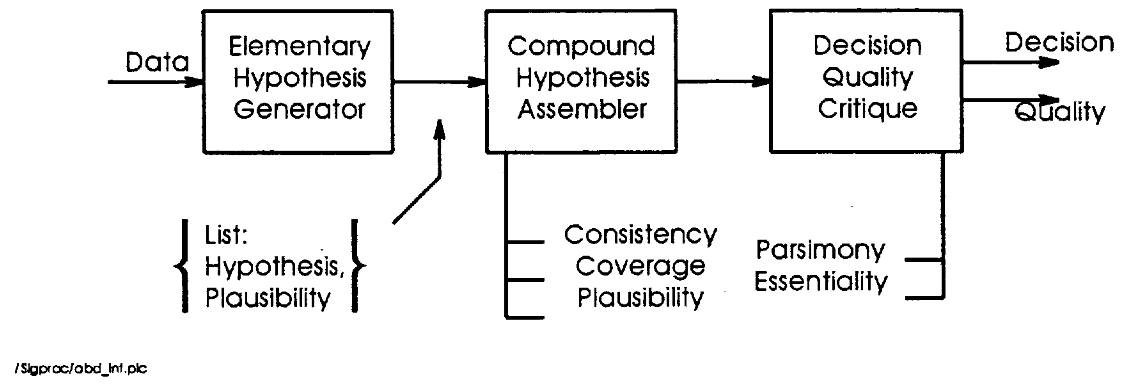

Data
Elementary Hypothesis Generator
Compound Hypothesis Assembler
Decision Quality Critique
Decision
Quality
List: Hypothesis, Plausibility
Consistency
Coverage
Plausibility
Parsimony
Essentiality
/Sigproc/abd_inf.pic

Figure 6. Abductive Inference Functional Diagram.

The first functional element is an Elementary Hypothesis Generator. This produces a list, with each elementary hypothesis tagged with its corresponding *plausibility.* The plausibility is a numerical measure of the likelihood of the elementary hypothesis. It may also be called a *Confidence Factor.* For the airplane, an elementary hypothesis is {LAND|IAS}. A plausible hypothesis is then one whose numerical plausibility value exceeds some predefined threshold.

The next functional element in abductive (decision) inference is a Compound Hypothesis Assembler. Its task is to assemble, from the elementary hypotheses, the *most plausible* compound hypothesis. This is also what Bayes-ML does. However, the Abductive Assembler does this in a constrained way. It satisfies internal requirements of 1). Consistency, 2). Completeness, and 3). Plausibility.

The ***consistency*** requirement is that the elementary hypotheses must be compatible with each other. That is, that, pairwise, they are not mutually exclusive. The ***completeness***, or coverage, requirement is that the compound hypothesis contain all the plausible elementary hypotheses. This definition of *covers* is compatible with the set theoretic definition, considering the compound hypothesis as a direct-product. This means that no plausible elementary hypothesis may be ignored. The last requirement, on ***plausibility***, is to select the compound hypothesis which has the highest plausibility score. If a plausibility threshold is used to remove ***implausible*** elementary hypotheses from the computation, then the Bayes-ML algorithm can be used.

The key to designing the Abductive Inference Engine lies with its internal control. Punch[20] et.al. showed a tree-like goal structure (their Fig.-5) which formed the basis for internal control of the inference engine. Their control was dubbed "Selector-Sponsor," in which a Selector would select from among various Sponsors, each one of which sponsored a particular method, or task. That is, the Selector was a global controller, choosing from among the various subtasks for Abductive Inference. The Sponsor then evaluated its method's appropriateness to be the next task.

The control strategy for the dynamic systems problem is differentiated between the Interpretation task (inductive inference) and the System Control task (deductive inference). The concept of a community of distributed experts is employed, which is a parallel processing concept. Interpretation starts at the bottom of the frame structure (See Fig.-5), with the occurrence of new sensor measurements. Each measurement slot contains its own *expert.* Each expert evaluates its owned membership functions. Each expert then selects its plausible hypotheses. Each expert then passes its data, as messages, to the higher levels of the framework, according to rules. Every slot in the framework functions in this same way.

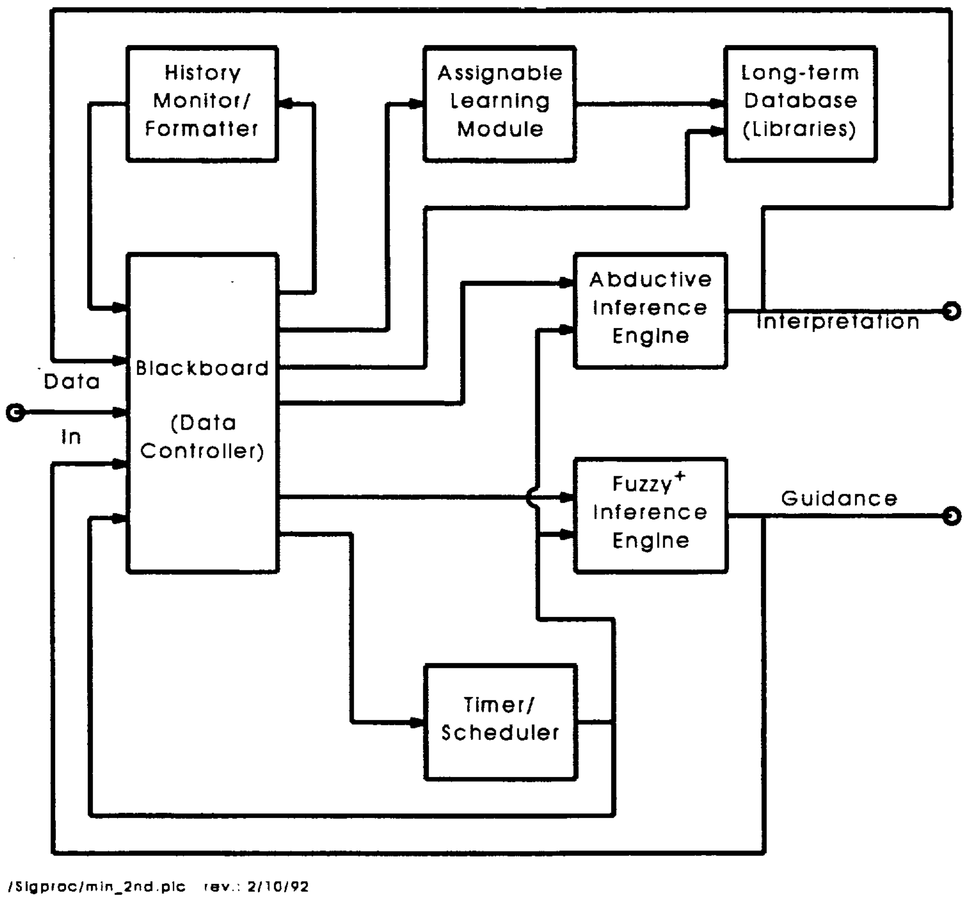

History Monitor/ Formatter
Assignable Learning Module
Long-term Database (Libraries)
Abductive Inference Engine
Interpretation
Data In
Blackboard (Data Controller)
Fuzzy+ Inference Engine
Guidance
Timer/ Scheduler
/Sigproc/min_2nd.pic rev.: 2/10/92

Figure 7. Symbolic Processor.

Chandrasekaran's[19] list of generic functions is now augmented to deal with temporal (dynamic) systems. System operational modes, if ideally defined, are mutually exclusive and sequential. That is, they occur naturally in sequence. Therefore, if the last mode is known, a priori information is available about the present mode. Thus, is needed another generic task, that of *History Formatting and Processing*. This is the dual of exploiting correlation in stochastic decision/estimation (eg. Kalman filtering).

From all of the above, results the second-level data-flow architecture shown in Figure-7, above.

CONCLUSION.

What has been related here, is that Knowledge-based Control (or Management) of (complex) Dynamic Systems may be approached in a way that is very natural to practitioners of the information and control sciences. This approach is a synthesis of results evolving in several heretofore separate disciplines. Modeling and processing methods are favored which support visualization. Dualities are exploited, which exist between Artificial Intelligence and Systems Theory (Communication, Control, and Signal Processing). Based on research in progress, this approach appears to the author to hold great practical promise in applications like aircraft flight management.

CHAPTER-2 BIBLIOGRAPHY

1. Fu, K.S., "Learning Control Systems and Knowledge-based Control Systems: an Intersection of Artificial Intelligence and Automatic Control;" IEEE TRANSACTIONS ON AUTOMATIC CONTROL; vol. AC-16, no. 1, 1971; pp. 70-72.

2. Bellman, R., DYNAMIC PROGRAMMING; Princeton University Press; Princeton, NJ.; 1957.

3. Kalman, R.E., "A New Approach to Linear Filtering and Prediction Problems;" TRANSACTIONS OF THE ASME, JOURNAL OF BASIC ENGINEERING; vol. 82D, March, 1960, pp. 34-45.

4. Schweppe, F.C., UNCERTAIN DYNAMIC SYSTEMS; Prentice-Hall; 1973, pp. 479-489.

5. Ramadge, P.J.G. and Wonham, W.M.; "The Control of Discrete Event Systems," PROCEEDINGS OF THE IEEE; vol. 77, no. 1, Jan. 1989; pp. 81-98.

6. Newell, A. and Simon, H.A.; HUMAN PROBLEM SOLVING; Prentice-Hall, 1972.

7. Giarratano, J. and Riley, G.; EXPERT SYSTEMS, PRINCIPLES AND PROGRAMMING; PWS-KENT Publ. Co.; Boston, 1989; pp. 11-15.

8. Winston, P.H.; ARTIFICIAL INTELLIGENCE Second Edition; Addison-Wesley; 1984; pp. 146-157.

9. Sutherland, John W.; "Assessing the Artificial Intelligence Contribution to Decision Technology;" IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS; vol. SMC-16, #1, Jan/Feb., 1986, pp. 3-20.

10. Zadeh, L.A.; "Fuzzy Sets;" INFORMATION AND CONTROL; no. 8, 1965; pp. 338-353.

11. Bellman, R.E. and Zadeh, L.A.; "Decision-making in a Fuzzy Environment;" MANAGEMENT SCIENCE; no. 4, 1970; pp. 141-164.

12. Zadeh, L.A.; "A Rationale for Fuzzy Control;" TRANSACTIONS OF THE ASME, Series G (USA), no. 94, 1974, pp. 3-4.

13. Astrom, K.J., Anton, J.G., and Arzen, K.E.; "Expert Control;" AUTOMATICA; no. 13, 1986, pp. 277-286.

14. Pedrycz, W.; FUZZY CONTROL AND FUZZY SYSTEMS; John Wiley & Sons, Inc.; 1989; pp. 61-80. (See p. 69 for "fuzzy set of control.")

15. Painter, J.H. and Jones, S.K.; "Results on Discrete-time, Decision-directed, Integrated Detection, Estimation, and Identification," IEEE TRANSACTIONS ON COMMUNICATIONS; vol. COM-25, no. 7, July, 1977.

16. Bylander, Tom and Chandrasekaran, B.; "Generic Tasks for Knowledge-based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition;" INTERNATIONAL JOURNAL OF MAN-MACHINE STUDIES; #26, 1987, pp. 231-243.

17. Minsky, M.; "A Framework for Representing Knowledge;" THE PSYCHOLOGY OF COMPUTER VISION; McGraw-Hill, 1975, pp. 211-277.

18. DeMarco, Tom; STRUCTURED ANALYSIS AND SYSTEM SPECIFICATION; Yourdon Press, 1978.

19. Chandrasekaran, B.; "Generic Tasks in Knowledge-based Reasoning: High-Level Building Blocks for Expert System Design;" IEEE EXPERT; Fall 1986, pp. 23-30.

20. Punch, W.F., Tanner, M.C., Joshephson, J.R., and Smith, J.W.; "Peirce, A Tool for Experimenting With Abduction;" IEEE EXPERT; October, 1990, pp. 34-44.

21. Widman, L.E.; Loparo, K.A.; and Nielsen, N.R.; ARTIFICIAL INTELLIGENCE, SIMULATION, & MODELING; Wiley; 1989, pp. 8-10.

22. Shlaer, S. and Mellor, S.J.; OBJECT-ORIENTED SYSTEMS ANALYSIS; Prentice-Hall (Yourdan Press); 1988.

CHAPTER 3

SOFT FUZZY CONTROL.

by
John H. Painter
Department of Electrical Engineering
Texas A&M University
College Station, Texas 77843-3128
409, 845-7441
painter@zadok.tamu.edu

The following is a paper presented by the author at The Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems, held at College Station, Texas on December 2-4, 1992, and printed in the Proceedings thereof.

ABSTRACT.

This paper interprets fuzzy control in a Bayes context. Using two different sets of logical connectives from the literature, there is shown a clear duality between the fuzzy and Bayes versions of the same control problem. The nature of this duality is explored. Its Bayes implications serve to illuminate the geometrical nature of fuzzy control as opposed to purely algebraic implementations. The results reconcile fuzzy and Bayes viewpoints concerning control.

INTRODUCTION.

For specific fuzzy control methods, this paper provides interpretation, from the useful viewpoint of the Bayes decision, estimation, and control discipline. The resulting interpretations, which are essentially geometric, rest on similarities between fuzzy and Bayes approaches, rather than on differences. This is a paper which is pragmatic in outlook, seeking interpretations which are productive of design and implementation of fuzzy control architectures. This paper seeks to apply that which is useful from Bayes to that which is useful from fuzzy. As such, this paper is one of reconciliation, rather than differentiation.

The paper first sets a context of Intelligent Control, which is the application of Artificial Intelligence (AI) methods to symbolic control based on knowledge. Within this context, Fuzzy Control may be viewed as a particular and highly attractive implementation method. Then follows a short review of Fuzzy Control. Next, the main results of the paper are derived and explained, with a motivation to be clear and simple. The result is a simple fuzzy control algorithm, written in Bayes notation. The interpretation of the algorithm is clearly explained and derivational restrictions stated. Because of the clear relation between the Bayes derivation and the corresponding fuzzy implementation, the Bayes algorithm is called a "*dual*" of the fuzzy algorithm.

INTELLIGENT (KNOWLEDGE-BASED) CONTROL.

Intelligent Control, also known as Knowledge-based Control, comes from the *Systems and Control* discipline, as a branch which is traceable back to a seminal paper by K.S. Fu [1]. Twenty years ago, he recognized that automatic control should benefit from developments in *Artificial Intelligence* (AI), if not vice versa. That these benefits have been so long in coming, is a comment upon the difficulties inherent in AI development and in transfer between two very different technologies.

The difference between traditional *Modern Control* and *Intelligent Control* is that the embedded inference and memoryless control algorithms now contain qualitative, as well as numerical, processing elements. That is, the inferences and resulting controls are focused on qualitative performance measures, as well as numerical. As an adjunct to the usual required numerical state-space modeling of the plant, now there is a required modeling of *qualitative states.* The general pattern in intelligent control of a dynamic system is as follows: First, the usual numerical state-variable modeling of the plant takes place, with a preference given to states which are available as sensor readings, or are, at least, easily derivable (observable) therefrom. Next, a state-space is constructed, which is the *direct product* space. Then, and most importantly, qualitative states are defined as partitions of the direct-product space. These qualitative states are most generally "*operating modes*," as defined by a human operator. Such modes are not directly measurable, but are observed by (human) inference processing of the available sensor measurements. Control is then exerted to maintain or modify these inferred states (modes).

As an example, consider the guidance of the flight of a transport-type aircraft. By *guidance* is meant an automation of the control activities traditionally performed by the pilot. It is assumed that the usual numerical automatic flight control system (AFCS) is present. Thus, the pilot concerns himself with evaluation of flight performance and formulation of inputs to the autopilot, to accomplish the goals of the flight.

The usual sensor readings are assumed to be available, as in Figure-1, below.

SPEED	ALT.	FORW_∠	LAT_∠	HEADING	POWER	RESOURC.	AUX_DEV.
IAS Mach.	Alt.	Pitch	Bank	Mag. Comp. (TH)	EPR EGT N(rpm) Eng-#	Fuel_Qty.	Flaps Slats Gear Spd_Brk

/Sigproc/lst_raw.tbl

Figure 1. Raw Numerical Sensor List.

Based on these available readings, and others derivable from them, qualitative operating modes are defined, as in Figure-2.

MODE	STATUS	CONTROL	MANUVER	CONSTRAINTS
Takeoff	Normal	Pilot in	Pilot	Manuver
Clean_up	Emergency	Command.	Computer/	Power
Climb_out	Alarms:	Air Traffic	Autopilot/	Fuel
Climb_on_course	Systems,	Control	Flight Control	Systems
Cruise	Fuel.	Other	System.	Weather
Descend_on_course				
Vector				
Initial_approach				
Final_approach				
Go_around				
Land				
Unknown				

/Sigproc/lst_sops.tbl

Figure 2. Flight Mode List.

Given the sensor readings, mode is inferred and control exerted, based on knowledge of the aircraft and of the rules of flight. Intelligent control, or guidance, of an aircraft attempts to automate these high-level pilot activities.

The list of operating modes is assumed finite, with *mode_unknown* covering the rest of the possibilities. The required inference is decision, rather than estimation, which is the usual numerical control inference. That is, given a vector of sensor measurements, it is to be decided into which partition of the direct-product space the observed *"point"* fits. Given that decision, and knowledge of the recent history of aircraft state, an input to the AFCS is then formulated, based on known mission goals and characteristics of the aircraft.

Although the above paradigm sounds simple (and it is accomplished routinely by human pilots) the automation thereof is not trivial for several reasons. First, in the definition of operating modes, the partitioning of the state space, is not clean. Although the operating modes may be unique, their modeling in terms of the sensor variables may not be. Second, the decision inference method is not obvious. This is because aircraft operation may generate measured sensor data which conflicts with a mode that is "*essentially true*". A classical example is a landing approach with the landing gear retracted. Although the mode is "landing," the "gear variable" value conflicts with the model definition of the mode. The decision method must accomodate these anomalies. Finally, the method for synthesizing qualitative commands, accompanied by numerical prescriptions, based on qualitative performance interpretations, is also not obvious. Fuzzy Control is one implementation method for Intelligent Control which deals with many of these cited problems.

FUZZY CONTROL.

Fuzzy Set Theory was created by Zadeh [2] twenty-five years ago, in the same time-frame as Fu's postulation of Intelligent Control. Within five years, its use in decision theory was being examined [3]. Within nine years, its use in control was being promoted [4]. Later (after 1986), it was implemented in practical control systems[5].

For use in decision problems, Fuzzy Set Theory is similar to Bayes Conditional Probability Theory, in the sense that *events* may be modeled by set functions whose range is the positive reals. Moreover, these functions may be manipulated to create similar positive real functions defined on subsets of the original domain. Therefore, in both cases, the set functions may be used to calculate the solutions of decision problems. It is for use in control problems where fuzzy methods depart from the usual Bayesian methods.

Figure-3 illustrates a simple case of fuzzy control. Therein, two events are modeled by membership functions, $m_1(x)$ and $m_2(x)$. These functions represent the membership of measured sensor values of input variable, x, in the events E_1, E_2, respectively. A control value, y, is to be formulated according to a goal of driving the sensed value into the quiescent point, $x = 1.375$. This is done by utilizing two control events, with membership functions, $m_1(y)$ and $m_2(y)$ and an "IF-THEN" rule. The rule is that if the input sensed value, x, belongs to event, E_1, then the output control value should be governed by $m_1(y)$, and similarly for event, E_2. Control is obtained by a sequential looping computation employing the IF-THEN rule.

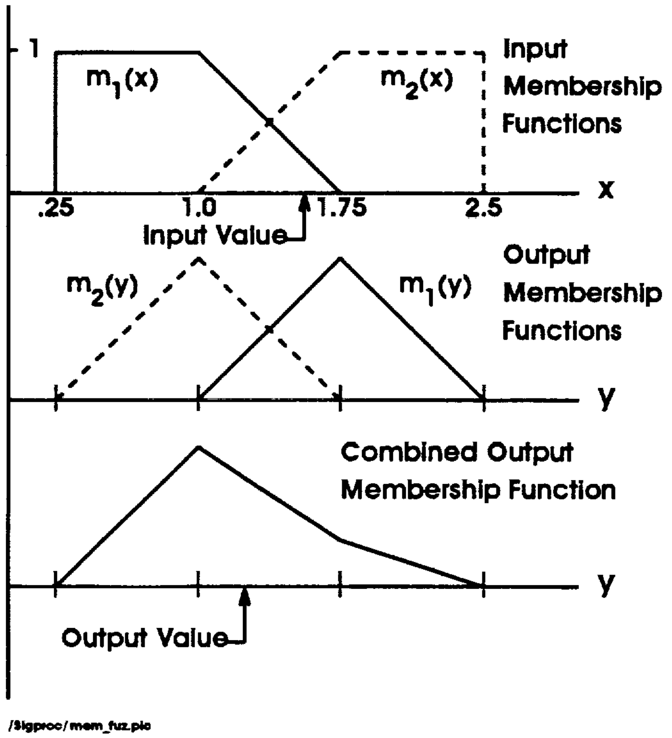


Figure 3. Fuzzy Control Illustration.

A fuzzy algorithm is formulated by using the function values of the input sensed variable, x, to modify the output functions in a process called *inference*. The latter are then combined to produce a single output function in a process called *combination*. Then, a single functional value of the output is obtained by a non-unique transformation called, *defuzzification*. The illustrated case models "correlation-product inference," "additive combination," and "centroid defuzzification," as described in Kosko[6].

The sensed input events, modeled by $m_1(x)$ and $m_2(x)$, are defined by two fuzzy sets, being $\{x \in (.25,1.75)\}$ and $\{x \in (1.0,2.5\}$, respectively. Likewise, the output control events, modeled by $m_1(y)$ and $m_2(y)$, are defined by two fuzzy sets, being $\{y \in (1.0,2.5)\}$ and $\{y \in (.25,1.75)\}$, respectively. The algorithm shown in Figure 3 multiplies $m_1(x)$ times $m_1(y)$ and adds it to the product of $m_2(x)$ times $m_2(y)$. The products represent the *intersection* of the corresponding fuzzy sets. The sum represents *union*.

In fuzzy logic, there are two different representations of *intersection* and *union*. One is given by the arithmetic pointwise product of membership functions. The other is by the pointwise *Min*$\{\cdot\}$ function. Likewise, the *union* is represented by either the arithmetic pointwise sum or pointwise *Max*$\{\cdot\}$ functions. Bellman and Zadeh[3] call the Min/Max pair *hard* logical connectives and the Product/Sum pair *soft*. There are, therefore, four possible implementions of the fuzzy control case shown in Figure-3, being Hard/Hard, Hard/Soft, Soft/Hard, and Soft/Soft. The Figure-3 illustration is that for Soft/Soft, which I shall just call *Soft Fuzzy Control*.

Decision theory also admits the terms, *hard/soft*[7]. In the decision-theory context a hard decision is an immediate one, whereas a soft decision is one which is deferred for further processing of input data. Fuzzy control does not actually make an immediate decision on which of the input events has occurred. The IF-THEN control rule is not implemented in a hard sense. Rather, a sort of average IF-THEN is used, taking from each output function, proportional to the input membership values. In a decision theory context, fuzzy control might be said to employ soft-decision to yield *soft decision-directed control*[8].

From several standpoints, then, the fuzzy control case which is dealt with in this paper may be labeled, *soft*. All fuzzy control is soft from the decision-theoretic viewpoint. The present specific implementation is also soft in terms of the logical connectives used. Now, I will show that it also has a direct Bayes interpretation, or *dual*.

THE BAYES DUAL.

Let us define the input sensor variables, output control variables, and events in the following manner:

z : The numerical control variable, a scalar.

$\underline{x}_N = (x_1, x_2, .., x_N)$: Sensor measurement vector.

$\left\{ A_i : i = 1, 2, .., M \right\}$: Family of Event Sets, defined on $\underline{x}_N$.

The use of a *centroid* method for defuzzification suggests that the output control membership function might profitably be modeled as a probability density function. The usual Bayes approach is to use the density of control variable, conditioned on input data. This control membership function is denoted as $p(z\,|\,\underline{x}_N)$. This is initially transformed, using the usual Bayes transformations, as

$$p(z\,|\,\underline{x}_N) = p(z \cap \underline{x}_N)/p(\underline{x}_N) \tag{1}$$

where $\cap$ denotes the usual logical (soft fuzzy) intersection.

The decision and control events, A_i are brought into the formulation in a standard way, by augmenting the joint density with the *Sure Event*, $\mathcal{S}$, defined in terms of the A_i. These A_i are intentionally designed so as to *cover* all of the vector-space, $\underline{x}_N$, of interest (commonly called domain of interest). These subsets, A_i, are given the decision-theoretic interpretation of *events*, and are named distinctively in English. In the aircraft application, they are called *modes*. Mathematically, then, we have the definition,

$$\bigcup_{i=1}^{M} A_i = \mathcal{S} : \text{sure event; total space} \tag{2}$$

It follows that

$$\begin{aligned} p(z \cap \underline{x}_N) &= p(z \cap \underline{x}_N \cap (\bigcup_{i=1}^{M} A_i)) \\ &= p(\bigcup_{i=1}^{M} (z \cap \underline{x}_N \cap A_i)) \end{aligned} \tag{3}$$

There is a distinction which must now be made between the intrinsic events, A_i, and their modeling as subsets of a vector-space. The events have a meaning all their own, independent of their models. For an aircraft, the event of "final approach" had a meaning of its own, long before state-spaces models came into use. Therefore, we may define the events, A_i, to be ***unique***, or, in Bayes terminology, ***mutually exclusive***, even though their subset models are not disjoint. The fact that unique events are represented by subsets which are not disjoint reflects a *modeling choice*, which injects uncertainty into the model. As designers, we will choose the A_i to be unique and then do our best to model them as vector-space subsets.

Given mutually exclusive A_i, equation-(3) transforms to

$$p(z \cap \underline{x}_N) = \sum_{i=1}^{M} p(z \cap \underline{x}_N \cap A_i); \text{ mutually exclusive } A_i \tag{4}$$

Now, in equation-(4), we invoke the model of the A_i, and interpret the non-empty set, $\{\underline{x}_N \cap A_i\}$, as the event, "$A_i$ occurs." That is, if an input sensor sample vector, $\underline{x}_N$, has a non-empty intersection with A_i, we say A_i occurs. Figure-(4) illustrates this interpretation for $\underline{x}_N$ a 2-vector.

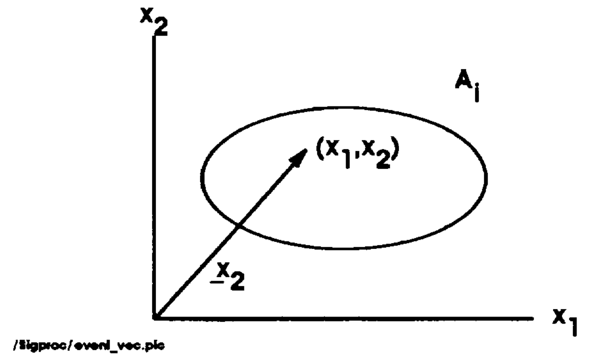


Figure 4. Vector Decision Illustration.

It is the practice in Fuzzy Control to model the input events on single sensor variables, rather than on vectors (N-tuples). These models are the familiar membership functions. Thus, we move now from decision based on N-tuples to decision based on individual vector components. This we do by considering the projection of the subset, A_i, onto its component axes (subspaces). We will *define* subsets, A_{ij}, on each of the x_j axes as

$$A_{ij} = \{x_j \ni (x_1,x_2,..,x_j,..,x_N) \cap A_i \text{ is not empty.}\} \tag{5}$$

These are illustrated in Figure-5.

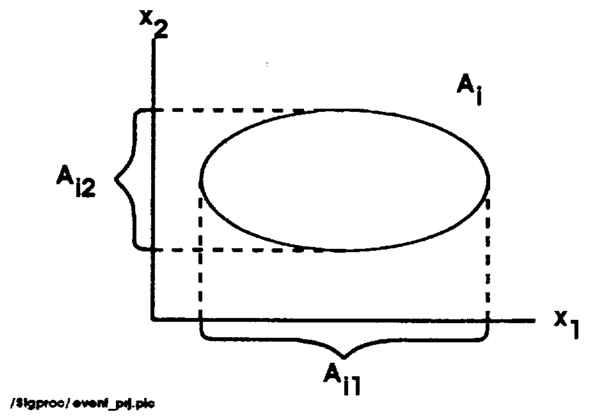


Figure 5. Projection Decision Illustration.

Now, we shift from decision tests for A_i, based on events, $\{\underline{x}_N \cap A_i\}$ to a test based on

$$\left\{\underline{x}_N \in A_i\right\} \equiv \bigcap_{j=1}^{N} \left\{x_j \cap A_{ij}\right\} \tag{6}$$

where the relation, "$\equiv$", is an equivalence, not an identity. This is a sub-optimum test, and it has a performance cost. It can be seen from the 2-space illustration, above, that testing for x_1 in A_{i1} and x_2 in A_{i2}, separately, treats A_i as though it were a rectangle enclosing the actual A_i. (We have actually replaced A_i by the direct sum of its subspace projections.

Continuing on with the arithmetic, we have now,

$$p(z \cap \underline{x}_N) = \sum_{i=1}^{M} p(z \cap (\bigcap_{j=1}^{N} x_j \cap A_{ij}))$$

$$= \sum_{i=1}^{M} \prod_{j=1}^{N} p(z \cap x_j \cap A_{ij}) \text{ ; } x_j \text{ independent } \forall j \qquad (7)$$

where we have restricted the x_j to be *independent*. What this means in application is that the sensed variables, x_j, should be physically independent, and not functional computations of one another. Then, the joint density becomes a product over the modes.

Next, returning to the original conditional formulation of equation-(1), we obtain

$$p(z \mid \underline{x}_N) = \sum_{i=1}^{M} \prod_{j=1}^{N} p(z \mid x_j \cap A_{ij}) \cdot P(A_{ij} \mid x_j) \qquad (8)$$

where $p(\underline{x}_N)$ has canceled out of numerator and denominator.

By way of interpretation, the quantities, $p(z \mid x_j \cap A_{ij})$ are the fuzzy control membership functions, which have the form of conditional probability density functions. They give control density, conditional on the event that sensor variables fall in projections of A_i on corresponding sensor variable axes. We call these projections "sub-events," or "sub-modes." The quantity, $P(A_{ij} \mid x_j)$, is the input (or sensor) membership function, which is the posterior probability of sub-event (sub-mode), given corresponding sensor variable.

Now, we make one last simplification of the notation, to arrive at a final formulation. The quantity, $p(z \mid x_j \cap A_{ij})$ is a control density, conditioned on both event, i, and sensor, j. Although the mathematics supports dependence on j, in practice these control densities are generally a function only of i, the identifier of the predefined mode. That is, control is based only on knowledge of mode, and is not differentiated as to which sensor supplied that knowledge. Thus, we will simplify as

$$p(z \mid x_j \cap A_{ij}) \rightarrow p(z \mid A_i)$$

and there will be only M of these, rather than M·N.

Also, $P(A_{ij} \mid x_j)$ may be notationally changed without changing its meaning, as

$$P(A_{ij} \mid x_j) \rightarrow P(A_i \mid x_j)$$

and there are still M·N of these. Then, equation-(8) becomes

$$\begin{aligned} p(z \mid \underline{x}_N) &= \sum_{i=1}^{M} p(z \mid A_i) \cdot \prod_{j=1}^{N} P(A_i \mid x_j) \\ &= \sum_{i-1}^{M} p(z \mid A_i) \cdot P(A_i \mid \underline{x}_N) \qquad (9) \end{aligned}$$

The two-fold internal product, p(·)·P(·), is the weighting of control membership by the net sensor membership. The N-fold product over the sensors is equivalent to the N-fold intersection over the sensors, for a particular mode-event, where "*product*" is the *soft* version of the *hard* fuzzy intersection, "*min*". The M-fold sum over the events is equivalent to the M-fold union over the events, where "*sum*" is the *soft* version of the *hard* fuzzy union, "*max*"[3].

The algorithm of equation-(8) is equivalent to the fuzzy control method known as "*correlation-product inference, additive combination*"[6]. The control density may be defuzzified by the centroid method (dual to the Bayes, *conditional-mean*), if the density is single-moded. Otherwise, the Bayes *MAP* method[9] may be used (yielding the control value for which the combined control density is maximum).

The quantities, $P(A_i | x_j)$, are the posterior probabilities, which might be used to make a hard decision. The N-fold products of these, over j, can be used to make the hard *MAP* decision on the occurrence of one specific mode-event. However, fuzzy control does not make such a hard decision. Rather, the control densities for the M specific possible mode-events are weighted by the *MAP* decision probabilities and then averaged. This suggests the characterization of fuzzy control as *soft-decision-directed.* An alternate Bayesian view is that fuzzy control is an *averaged control,* where the average is over an ensemble of multiple, uncertain, competing events.

EXAMPLE.

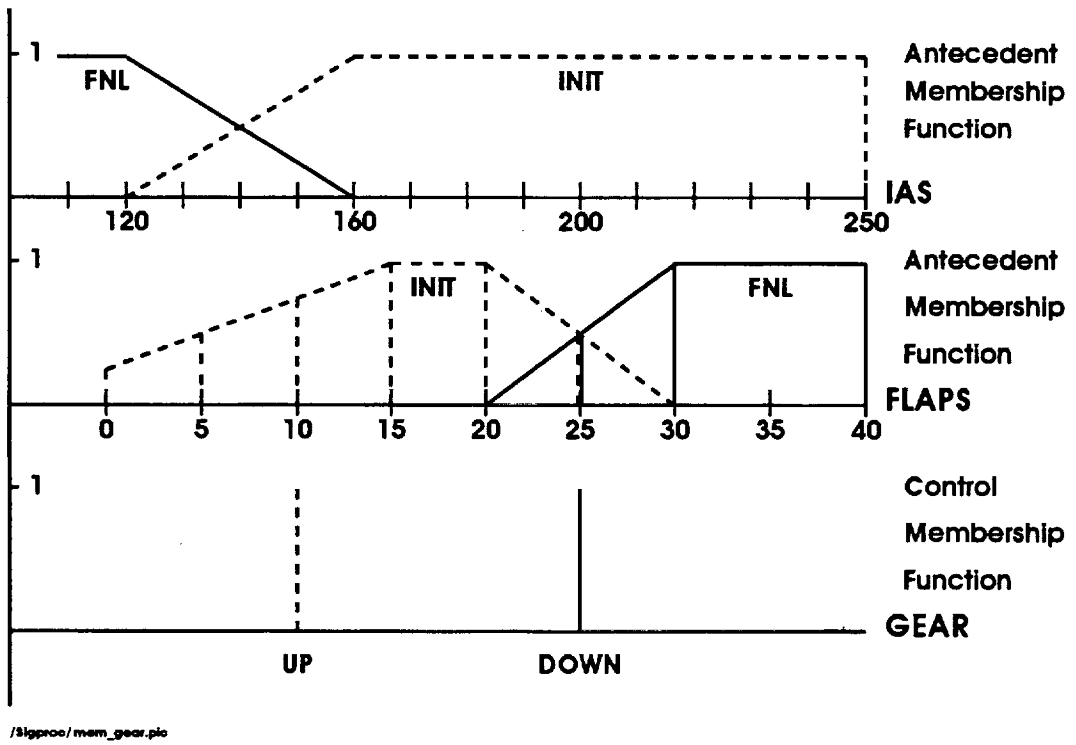


Figure 6. Landing Gear Membership Functions.

Figure-6 shows a realistic example of reduced dimensionality for fuzzy control of aircraft landing gear. The rule is, "extend gear on final approach." Only two possible flight modes are illustrated, which are contiguous. They are Final Approach, APPR_FNL, and Initial Approach, APPR_INIT. Only two sensor variables are illustrated, being Indicated Air Speed, IAS, and Flap Position, FLAPS. The control variable is GEAR.

The antecedent and control membership functions are diagrammed. Note that the functions for FLAPS and GEAR are discrete and the values for GEAR are linguistic, rather than numeric. Dashed lines indicate the functions for Initial Approach, while solid lines indicate Final Approach. Because the control variable is discrete, the method of defuzzification chosen is *MAP*.

IAS	200	170	160	150	140	130	125	120
P(INIT)	1	1	1	0.75	0.5	0.25	0	0
P(FINL)	0	0	0	0.25	0.5	0.75	1	1
FLAPS	0	5	10	15	25	30	40	40
P(INIT)	0.25	0.5	0.75	1	0.5	0	0	0
P(FINL)	0	0	0	0	0.5	1	1	1
Π(INIT)	0.25	0.5	0.75	0.75	0.25	0	0	0
Π(FINL)	0	0	0	0	0.25	0.75	1	1
MODE	INIT	INIT	INIT	INIT	INIT	FINL	FINL	FINL
P(UP)	0.25	0.5	0.75	0.75	0.25	0	0	0
P(DOWN)	0	0	0	0	0.25	0.75	1	1
GEAR	UP	UP	UP	UP	UP	DOWN	DOWN	DOWN

gear_tbl.tbl

Figure 7. Computational Table.

Figure-7 shows a table of the various computations for this example. The flaps are operated according to a specific manuvering-speed table for the Boeing-737. The Table is divided into four blocks for Speed, Flaps, Mode, and Gear, and is read vertically. The membership function values for the two approach modes, INIT and FINL, are shown versus IAS in the first block. The membership function values for the two approach modes are shown versus FLAPS in the second block. In the third block are the values of the membership function value products, taken over each mode and denoted by $\Pi(\cdot)$, corresponding to the first part of equation-(9). From these products the corresponding mode decisions are also shown. In the fourth block are shown the modified values of control membership functions (correlation-product, additive combination) for GEAR_UP and GEAR_DOWN, respectively. Also shown is the final control, defuzzified by $\mathcal{MAP}$.

From the Table, it is seen that the landing gear does indeed cycle from up to down when the mode decision changes from initial_approach to final_approach. In fact, it can be seen that the mode-decision products, $\Pi(\cdot)$ and gear control values, $P(\cdot)$ are identical. This is because the gear control membership function is crisp, rather than fuzzy. In general, the two computations will not yield identical numerical results, although the mode decision and control action should always be commensurate.

CONCLUSION.

The reader will observe that nowhere in this paper have I made reference to any epistemological arguments concerning fundamental differences in the fuzzy or Bayes approaches. No appeals have been made to differences based on uncertainty versus randomness. Rather, my approach has been to find similarities, to look for *correlations.* This paper attempts to interpret the structures of fuzzy control, being its modeling facilities and functional transformations. Since some of these resemble corresponding facilities in the Bayes decision, estimation, and control discipline, my interpretations have been in the Bayes context. What I have concluded is that Bayes is a useful spotlight to shine on fuzzy control.

What this paper shows is that fuzzy control may be easily interpreted, and therefore pragmatically applied, in terms of decision and averaging concepts familiar in the (Bayesian) signal processing and control world. To me, if fuzzy can be said to have *beauty*, it is that it is essentially geometric in concept, rather than algebraic. Application of fuzzy to control reduces and modularizes complexity, rather than increasing it. And, this reduction is due to the available geometric interpretation of fuzzy modeling and transformation. Where discrete-time stochastic control applications become lost in a haze of algebraic complexity, fuzzy control remains refreshingly clear. Because fuzzy is modular control, design and software implementation under the modern object-oriented schema is attractive.

CHAPTER-3 BIBLIOGRAPHY.

1. Fu, K.S.(1971). Learning Control Systems and Knowledge-based Control Systems: an Intersection of Artificial Intelligence and Automatic Control. *IEEE Transactions on Automatic Control.* vol. AC-16, no. 1. pp. 70-72.

2. Zadeh, L.A.(1965). Fuzzy Sets *Information and Control.* no. 8. pp. 338-353.

3. Bellman, R.E. and Zadeh, L.A.(1970). Decision-making in a Fuzzy Environment. *Management Science.* no. 4. pp. 141-164.

4. Zadeh, L.A.(1974). A Rationale for Fuzzy Control. *Transactions of the ASME, Series G (USA).* no. 94. pp. 3-4.

5. Astrom, K.J., Anton, J.G., and Arzen, K.E.(1986). Expert Control. *Automatica.* no. 13. pp. 277-286.

6. Kosko, Bart(1992). *Neural Networks and Fuzzy Systems.* Prentice-Hall. pp. 31, 312.

7. Gibson, Jerry D.(1989). *Principles of Digital and Analog Communications.* Macmillan. Chap. 14.

8. Painter, J.H. and Jones, S.K.(1977). Results on Discrete-time, Decision-directed, Integrated Detection, Estimation, and Identification. *IEEE Transactions on Communications.* vol. COM-25, no. 7, July.

9. Sage, Andrew P. and Melsa, James L.(1971). *Estimation Theory with Applications to Communications and Control.* McGraw-Hill. p. 179.

CHAPTER 4

OBJECT-ORIENTED DESIGN OF DISCRETE FUZZY CONTROL FOR AIRCRAFT FLAPS

by

Emily A. Glass, Ph.D. Candidate
&
Dr. John H. Painter, Professor
Electrical Engineering Department
Texas A&M University
College Station, TX. 77843-3128
409, 845-7441
painter@zadok.tamu.edu
emily@azariah.tamu.edu

The following is a paper presented by the authors at The Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems, held at College Station, Texas on December 2-4, 1992, and printed in the Proceedings thereof.

ABSTRACT

This paper examines a fuzzy control implementation for aircraft wing flaps for a transport-type aircraft. The implementational methods used were correlation-product inference,[1] additive-combination technique,[2] and maximum-membership defuzzification. It was discovered that with fuzzy flaps control, when the instantaneous data does not provide enough information for flight mode interpretation, memory must be introduced. Furthermore, it was discovered that tuning of wing flap extension decision points can be accomplished by a selected set of tuning rules.

INTRODUCTION.

This paper shows the results of an object-oriented implementation of wing flap control for a transport-type aircraft. The research has been supported by a NASA Training Grant, with technical support from the Aircraft Guidance and Control Branch at NASA Langley Research Center in Hampton, VA.

The test-bed was a piece-wise, linear, longitudinal simulation of a Boeing 737-100, written in the language C. The simulation was operated through an interactive graphical user interface, written in EIFFEL by a colleague.[3] The controller was, also, implemented in EIFFEL. EIFFEL is a pure, object-oriented language.[4]

The objective of this study was to examine the practical design aspects of combining correlation-product inference, additive-combination technique, and maximum-membership defuzzification.

SELECTION OF FUZZY CONTROL TECHNIQUES.

In general, fuzzy control utilizes a collection of fuzzy rules, or Fuzzy Associative Memories (FAMs).[1] Each FAM is a transformation that maps a logical combination of *fuzzy input sets* onto a *fuzzy control set* [1] -- i.e., *fuzzy inference*. Each fuzzy rule is part of a collective of rules, called a FAM system, which are fired in parallel. The resulting fuzzy output sets are then combined to form a "combined output set." For most practical applications, the combined fuzzy set must be *defuzzified* to produce a single control value.

Two inference schemes are commonly used in fuzzy control: correlation-minimum inference, and correlation-product inference.[1] Geometrically, they are known as *clipping* and *scaling*. Correlation-minimum inference results in a point-wise minimum of the antecedent result with the control membership function (clipping). Correlation-product inference results in a pointwise product of the antecedent result with the control membership function (scaling). Correlation-product inference preserves the information contained in the shape of the control membership function.[1] It was for this reason that correlation-product inference was selected instead of correlation-minimum inference for this study.

Two popular combination methods are *maximum*-combination technique and *additive*-combination technique.[2] Maximum-combination takes a pointwise maximum between the resulting FAM outputs. Additive-combination forms a pointwise linear combination between the FAM outputs. One problem with the maximum-combination technique is that the combined output set tends to a uniform distribution as the number of non-zero output sets increases.[2] In other words, control sensitivity decreases with increasing number of FAMs fired. According to Kosko, this problem can be avoided by using the additive-combination technique.[2]

The desensitizing effect caused by the maximum-combination technique was negligible with this project, since only a small number of rules (three to five) fired at non-zero values at any given time. However, the additive-combintation technique has an interesting side-effect that is worth studying; that is, rules fired with overlapping control sets produce a cumulative effect, showing more "causes" to support a control decision. For this reason, the additive-combination technique was selected for this study.

There are two popular defuzzification schemes: maximum-membership defuzzification, and fuzzy centroid.[1] Maximum-membership defuzzification selects the value of the independent variable, at which the combined fuzzy output set is maximized. The fuzzy centroid method, on the other hand, uses a "center of mass" computation. For this experiment, we decided to use maximum-membership defuzzification, since the control space for wing flaps consisted of only a few discrete values. Mulitple maximums were resolved with a median.

In summary, the implementational techniques selected for this project were correlation-product inference, additive-combination technique, and maximum-membership defuzzification.

DISCRETE FUZZY FLAP CONTROL.

The fuzzy rules, or FAMs, for this project were derived from a set of operational guidelines located in the *American Airlines 737 Operating Manual.*[5] (Note, the results of this research do not reflect the opinions of American Airlines.) The heuristics for flap management are:

1) "Extend flaps to the next setting before decelerating below the maneuvering speed for the existing flap setting."

2) "While accelerating, retract flaps upon reaching the maneuvering speed for the existing flap setting."

The study focused primarily on the approach and landing phases of flight; however, to round off the control system behavior, rules for flap retraction were also included. Table 1 lists the flap control rules. (These rules were based on "no wind" conditions.)

Membership Functions.

The primary *antecedent* sets were continuous fuzzy sets dependent on acceleration, flight path angle, and indicated airspeed (ias). To prevent erroneous control decisions, the sensors for acceleration, and flight path angle were low-pass filtered before being used. Notice that the ias membership functions were based on the flap maneuvering speeds (Figure 1).

The independent axis for the control membership functions represents the allowable discrete flap settings: 0, 1, 5, 10, 15, 25, 30, 40. There are four basic types of flap membership functions. The first is a crisp single discrete type, like **flaps_1** which has a non-zero value of unity at 1. The second is a crisp multiple discrete rectangular type, like **flaps_0_1_5** that is unity for values 0, 1, and 5, and zero elsewhere. The third and forth are fuzzy multiple discrete types. For example, **flaps_usually_5** is unity for flaps 5, and 0.2 for the other flap settings; and **descend_hysteresis_f0** is unity for flaps 0, and 0.8 for the other flap settings.

Implicit Flight Modes and Memory.

The first observation that can be noted is that fuzzy flaps control includes an inherent *flight mode decision* -- specifically, a decision between "approach" and "climb out". For example, the antecedent subclause of "**decelerating** and **non_pos_fpa**" indicates "approach," while the antecedent subclause of "**accelerating** and **non_neg_fpa**" indicates "climb out".

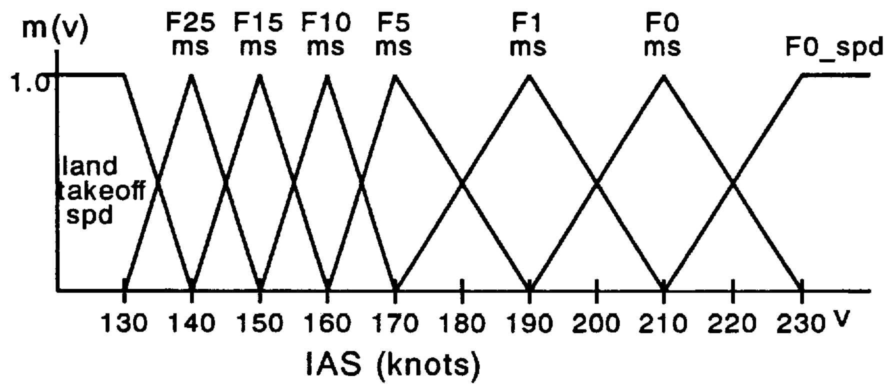


Figure 1. IAS Membership Functions.

During the initial design and test phase, it was observed that under certain conditions -- constant speed, and level flight-- that there was not enough information for an instantaneous flight mode decision. Under these circumstances the controller would not be able to decide if the airplane was climbing out or approaching. One solution was to add a rule telling the controller to leave the flap setting alone, if constant speed was determined (provided ias operating restrictions were not violated). Such a solution involves *remembering* the last flap control decision. This was accomplished by reading the existing flap setting at the time of decision. Rules 36 through 44 are memory rules. (See Table-1, below.) They interact with the airspeed rules at constant speed to accomplish the result.

For example, assume that flaps are currently set at 5, and that the constant speed antecedent fires stronger than the deceleration or acceleration antecedents. Additionally, assume that the indicated airspeed is strongly indicated to be in the flaps 1 maneuvering speed region. Then, rule 9 will fire strongly, indicating that either of flaps 0, 1, or 5 are permitted for this airspeed. Rule 38 will also fire strongly, indicating that the existing flap setting is preferred. The resulting combined output set will have a maximum at flaps 5, causing flaps 5 to be selected. Now, suppose that instead of an existing flap setting 5, we had a flap setting of 10. Then, Rule 39 will fire (instead of rule 38), indicating that flaps 10 is preferred, but any other flap change could be made. However, since flaps 10 is not one of the permitted flap settings for this airspeed (rule 9), the combined output function will have maximums at flaps 0, 1, and 5. The median of these (flaps 1) will be selected.

Tuning Rules.

After adding the memory rules, the control system consisted of rules 1 through 44. As can be easily predicted, flap extension and retraction occurred at predictable ias "crossover" points. For example, when the airplane was decelerating and descending (flaps currently 0), the control system extended to flaps 1 when the ias decelerated past 220 knots. This 220 knots threshold is the crossover point between the antecedent membership functions **f0_spd** and **f0ms**. (ie., The control system decided that the airplane was in "approach" flight mode and was decelerating towards the flaps 0 maneuvering speed. Therefore, flaps needed to be extended to the next setting.)

Now suppose it is desirable to extend the flaps to 1 "later" than 220 knots, but still before dropping below the flaps 1 maneuvering speed. This can be accomplished by adding tuning rules. To test this idea, a few sample tuning rules were added to adjust the extension points for flap settings 0, 1, and 5 (Rules 45, 46, and 47). The shape of the control membership functions for these tuning rules was adjusted empirically. For example, flaps 1 extension was shifted to 116 knots by Rule 45. However, the design was not without problems. Because of the cumulative effect, the acceleration membership functions had to be retuned carefully (this was done empirically, also) to make the "tuning" rules work properly. In other words, the rules firing with the **deceleration** antecedent had to have a greater cumulative effect than the rules firing with the **const_spd** *antecedent.*

CONCLUSION.

The fuzzy control rules were evaluated on an interactive simulation test-bed. The rules were designed with approach and landing phases of flight as the goal environment; "test flights" concentrated on approach and landing scenerios. Acceleration conditions were specified only to complete the FAM system, and were briefly tested to verify that the FAM system was behaving properly.

There is no well-defined performance measure for "flaps behavior." However, the airplane configuration was maintained to stay well within the operating limits of the flaps, and within the American Airlines specified guidelines for flap extension and retraction. (Note: if such a system is to eventually be used in commercial flights, the flap rules would need to be tested and retested by certified pilots under realistic flight conditions.)

This design project examined the aspects implementing fuzzy control in the context of aircraft wing flap control. In particular, the study focused on the practicality of designing with correlation-product inference, additive-combination technique, and maximum-membership defuzzification. It was found that for complex fuzzy control applications the use of memory is often imperative. Furthermore, the usefulness of this implementation scheme was enhanced by the addition of tuning rules. These rules added flexibility to the control system by providing a means for the user to adjust the point at which extension (or retraction) occurs.

During the memory and tuning rule design phases, the design complexity inherent with the additive-combination technique was revealed. If one is not careful, the cumulative effect of this technique can lead to unexpected and undesired control results. For this reason, it is very important that the rules be tested thoroughly, and membership functions carefully tuned. Furthermore, for complex problems, the additive-combination technique's cumulative effect can produce rules and rule combinations whose purpose is not immediately transparent or easily understood by the casual observer. Finally, all of the membership set and rule tuning had to be done empirically.

Thus, the additive-combination technique, while adding design flexibility, may add unnecessary design complexity. For example, if an airplane company was to implement a similar wing flap control scheme, it would be desirable for the rules to be easy to understand and maintain by multiple users. It was discovered that the cumulative effect of the additive-combination technique, can produce complex rule interaction that makes design errors more likely. Comparative research is currently underway to examine the practicality of using maximum-combination technique in place of the additive-combination technique.

CHAPTER-4 BIBLIOGRAPHY.

1. Kosko, Bart(1992). *Neural Networks and Fuzzy Systems.* Prentice-Hall. Ch. 8.
2. Kosko, Bart(1992). *Neural Networks and Fuzzy Systems.* Prentice-Hall. Ch 1. pp. 29-31.
3. Russell, Paul (1992). MS Thesis: *A Graphical User Interface for Knowledge-Based Control.* Texas A&M University, College Station, TX.
4. Meyer, Bertrand(1988). *Object-Oriented Software Construction.* Prentice Hall.
5. American Airlines (1990), *American Airlines 737 Operating Manual.*

Table 1. FAMS for Flaps Control.

FAM	Description
1	If **f0_spd** then **flaps_0**
2	If **f0ms** and **decelerating** and **non_pos_fpa** then **flaps_1**
3	If **f0ms** and **decelerating** and **pos_fpa** then **flaps_0**
4	If **f0ms** and **const_spd** then **flaps_0_1**
5	If **f0ms** and **accelerating** and **non_neg_fpa** then **flaps_0**
6	If **f0ms** and **accelerating** and **neg_fpa** then **flaps_1**
7	If **f1ms** and **decelerating** and **non_pos_fpa** then **flaps_5**
8	If **f1ms** and **decelerating** and **pos_fpa** then **flaps_0**
9	If **f1ms** and **const_spd** then **flaps_0_1_5**
10	If **f1ms** and **accelerating** and **non_neg_fpa** then **flaps_0**
11	If **f1ms** and **accelerating** and **neg_fpa** then **flaps_5**
12	If **f5ms** and **decelerating** and **non_pos_fpa** then **flaps_10**
13	If **f5ms** and **decelerating** and **pos_fpa** then **flaps_1**
14	If **f5ms** and **const_spd** then **flaps_1_5_10**
15	If **f5ms** and **accelerating** and **non_neg_fpa** then **flaps_1**
16	If **f5ms** and **accelerating** and **neg_fpa** then **flaps_10**
17	If **f10ms** and **decelerating** and **non_pos_fpa** then **flaps_15**
18	If **f10ms** and **decelerating** and **pos_fpa** then **flaps_5**
19	If **f10ms** and **const_spd** then **flaps_5_10_15**
20	If **f10ms** and **accelerating** and **non_neg_fpa** then **flaps_5**
21	If **f10ms** and **accelerating** and **neg_fpa** then **flaps_15**
22	If **f15ms** and **decelerating** and **non_pos_fpa** then **flaps_25**
23	If **f15ms** and **decelerating** and **pos_fpa** then **flaps_10**
24	If **f15ms** and **const_spd** then **flaps_10_15_25**
25	If **f15ms** and **accelerating** and **non_neg_fpa** then **flaps_10**
26	If **f15ms** and **accelerating** and **neg_fpa** then **flaps_25**
27	If **f25ms** and **decelerating** and **non_pos_fpa** then **flaps_30**
28	If **f25ms** and **decelerating** and **pos_fpa** then **flaps_15**
29	If **f25ms** and **const_spd** then **flaps_15_25_30**
30	If **f25ms** and **accelerating** and **non_neg_fpa** then **flaps_15**
31	If **f25ms** and **accelerating** and **neg_fpa** then **flaps_30**
32	If **land_takeoff_spd** and **decelerating** then **flaps_30_40**
33	If **land_takeoff_spd** and **const_spd** then **flaps_30_40**
34	If **land_takeoff_spd** and **accelerating** and **non_neg_fpa** then **flaps_15**
35	If **land_takeoff_spd** and **accelerating** and **neg_fpa** then **flaps_40**
36	If **flaps_0** and **const_spd** then **flaps_usually_0**
37	If **flaps_1** and **const_spd** then **flaps_usually_1**
38	If **flaps_5** and **const_spd** then **flaps_usually_5**
39	If **flaps_10** and **const_spd** then **flaps_usually_10**
40	If **flaps_15** and **const_spd** then **flaps_usually_15**
41	If **flaps_25** and **const_spd** then **flaps_usually_25**
42	If **flaps_30** and **const_spd** then **flaps_usually_30**
43	If **flaps_40** and **const_spd** then **flaps_usually_40**
44	If **flaps_40** and **decelerating** then **flaps_usually_40**
45	If **flaps_0** and **decelerating** and **non_pos_fpa** then **descend_hysteresis_f0**
46	If **flaps_1** and **decelerating** and **non_pos_fpa** then **descend_hysteresis_f1**
47	If **flaps_5** and **decelerating** and **non_pos_fpa** then **descend_hysteresis_f5**

CHAPTER 5

EXTENDED FUZZY DECISION-MAKING FOR FLIGHT MODE INTERPRETATION.

by
Gregory T. Economides, M.S. Candidate
&
Dr. John H. Painter
Electrical Engineering Department
Texas A&M University
College Station, TX, 77843-3128
409, 845-7441
econ@tamu.edu
painter@zadok.tamu.edu

The following is a paper presented by the authors at The Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems, held at College Station, Texas on December 2-4, 1992, and printed in the Proceedings thereof.

ABSTRACT.

This paper discusses the implementation of a Flight-Mode Interpreter for a commercial jet airplane, whose purpose is to assist the pilot in managing the complex job of controlling the aircraft. The Interpreter is based on a joining of the disciplines of Bayesian decision theory and Artificial Intelligence. The A.I. contribution is from the areas of Decision Science, Fuzzy Set Theory and Abductive Inference.

INTRODUCTION: Knowledge-based control.

The knowledge-based control problem addresses the need for robust control required in a dynamic system. Classical control techniques work well when the parameters of the system (plant) which is being controlled do not vary far from their nominal values. In order to control a plant whose state may change drastically over time, extremely robust control (beyond that which classical control can offer) is needed. This control, until recent years, often was exerted by a human operator making adjustments to the controls of the plant. The major difference between the human operator and a classical control system is the human's reasoning capability. For example, the human reads the input (from a gauge) and induces the *general* condition of the plant from the *specific* data. The knowledge of that *general* condition, combined with knowledge of how to adjust the plant's *specific* inputs to achieve the desired control (rules and reasoning) allows the human operator to robustly control the plant (a deductive process).

Knowledge-based control seeks to create software that is able to **induce** the state of the plant from sensor data and then **deduce** the proper inputs to the system (using techniques such as Fuzzy Logic and Qualitative Reasoning) to control it. In an airplane cockpit, for example, the amount of information (from computers and sensors) that a pilot must be able to assimilate has increased to the point where it is easy for critical information to be overlooked. A knowledge-based controller could monitor ALL pertinent information in real-time, alert the pilot to unusual conditions and even take action, if that is desired.

This present work is concerned with embedding artificial intelligence into an aircraft guidance and control system. The required architecture breaks down nicely into four major parts, being: Interpreter, Controller, Blackboard and Graphical User Interface (GUI) [Figure 1][1].

The subject of this paper is the Interpreter, whose research is parallel to research efforts on the other parts. The Interpreter's function is to infer the present state of the airplane, along with a measure of "how good" the inference is, from the values provided by sensors in the system. The Meta-controller,

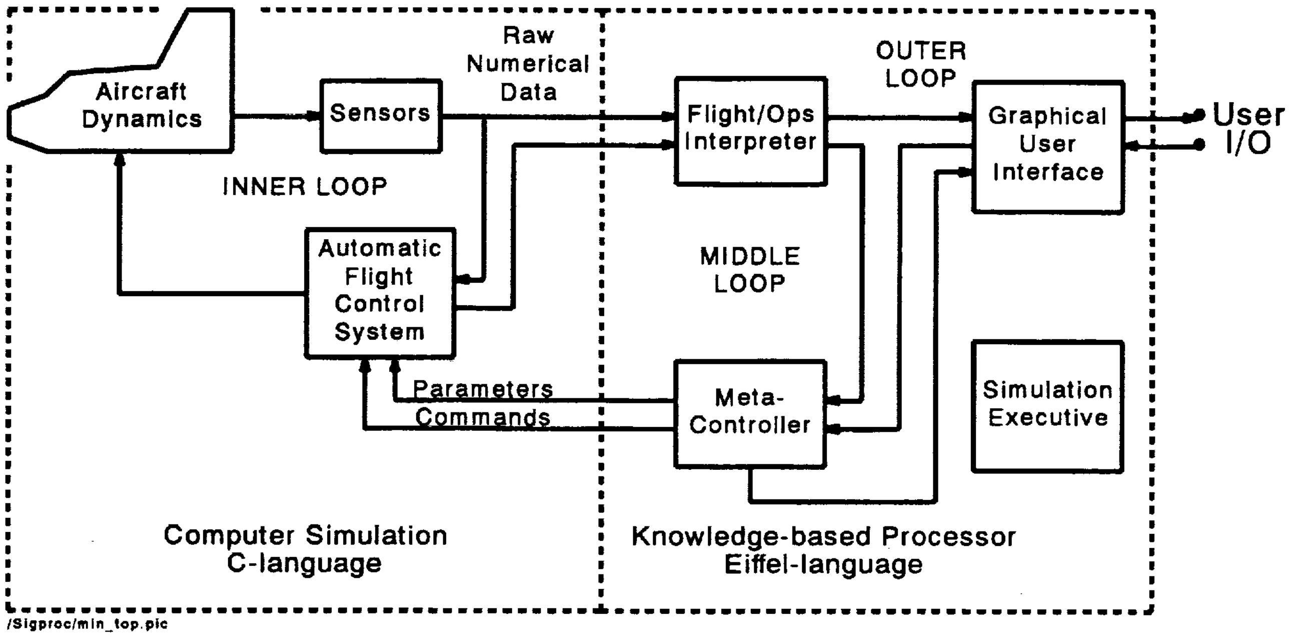


Figure 1. Top-level Architecture.

using sensor information and knowledge of how to control the system, provides the proper inputs to the airplane to bring it to the desired state. The Blackboard provides communication between modules, and is a good architecture for formalizing the inter- module communications interface. It provides for easy integration of additional software modules in the future. The GUI is designed to keep the human operator "in the loop;" allowing that person to monitor the interpretations made and control exerted on the system, and to modify or override them, if necessary.

The Interpreter, completely realized in software, works with a two-dimensional (longitudinal axis only) simulation of a modern, commercial passenger jet. This piece-wise linear numerical simulation produces raw flight data (airspeed, altitude, rate-of-climb, etc.) in response to standard pilot inputs. The linear numerical simulation was developed by Glass [2].

The Interpreter analyzes numerical data from the aircraft sensors. From this data, it identifies which one of a set of pre-defined flight conditions represents what the aircraft is doing. These flight conditions, by nature, are qualitative, "fuzzy" concepts. There is no single set of sensor values that indicates a given state, exclusively. Rather, a human pilot would decide that the plane is landing, for example, because the plane is *near* the runway, the airspeed is *in the vicinity of* landing velocity, the flight-path-angle is *about* right, etc. These italicized words suggest the fuzziness of this decision process. In order for a computer to deal with this type of decision-making, it must be able to make fuzzy, qualitative, "uncrisp" decisions from "crisp" data[3].

A study of the literature available on fuzzy logic and decision-making has revealed that methods for fuzzy decision-making are quite similar to standard Bayes decision-making[4]. Some relaxing of the basic assumptions of Bayes theory were made, to supposedly make the mathematics of Fuzzy Logic a bit more tractable. Unfortunately, this relaxing sacrificed the ability to use Bayes Decision Theory in the fuzzy context. It is proposed that embedding fuzzy decision in a Bayes framework can support the fuzzy decision process, without sacrificing the integrity of Bayesian analysis. We call this "Fuzzy-Bayes Decision."

The basic concepts used in fuzzy decision-making are simple. In the most elementary case, there is one input from which one decision is made. In classical Bayesian decision-making for a case like this, *a priori* density functions (relating a possible decision to an input variable) are used in a simple calculation to give the *posterior* probability that a given decision is the correct one. These density functions reflect prior knowledge of the system being modeled. When these posterior values are calculated for each possible decision, the one with the highest probability is selected as **The Decision**. This is more specifically referred to as the "M.A.P Decision Rule" (Maximum A posteriori Probability).

Fuzzy and Bayes theory give a groundwork for the decision-making process in the interpreter module. However, theoretical extensions are needed in order to make decisions in the presence of incomplete and/or conflicting data. For example, if the airplane were making a landing approach but the pilot had not lowered the landing gear, all of the plane's sensors, except GEAR, would suggest that the flight mode was LANDING. Hence, the Interpreter ought to be able to determine that the pilot is preparing to land the plane but has forgotten to lower the gear. It should then notify the pilot of the situation.

Abductive Inference/Assembly[5] provides one method for making decisions in the presence of incomplete/conflicting data. This technique, combined with Fuzzy-Bayes, "entertains" the postulation that the plane is in a particular state. As more observations of the sensor data are processed, the "belief" in that postulation is either strengthened or weakened. If it appears that an individual reading is in conflict with other readings, further processing is done.

The software for this project was chosen to be implemented using the Object-Oriented (OO) programming paradigm with the language, Eiffel. A project of this nature lends itself to the OO design methodology because it deals with definite, real world objects (airplane, pilot, cockpit controls, etc.). OO methodology allows the design of software modules that represent these objects. This aids in the identification of what modules need to be developed (what their functionality will be), since the modules are designed to "act like" the physical objects they represent.

The OO paradigm also is a great help in working in a software development group. **Information hiding** (disallowing other modules direct access to the internal data structures and functions of a module) requires that the different developers/programmers in the group only be concerned with a specified interface to the other modules of the project. This allows for the decoupling of one developer's work from the rest, freeing him or her from time- consuming familiarity with others' code.

Genericity (objects designed with reuse in mind) and **Inheritance** (sharing common behavior and structure of related objects) are also particularly important to a software project in a group programming environment. Having modules that are "reusable" can reduce the overall development time of a software project.

INTERPRETING FLIGHT MODE FROM MULTIPLE SENSORS.

A mode is defined to be an event. In the Abductive context[5], this event is treated as a "Compound Hypothesis;" a compounding of "sub-events," called "elementary hypotheses." Each sub-event is the coupling of a mode with a particular sensor variable value, such as indicated airspeed (IAS). Thus, for instance the compound event, "APPR_INIT" (initial approach) is determined from sub-events, "APPR_INIT given IAS," "APPR_INIT given FLAPS," etc.

Each sensor variable supports the definition of a number of *unique* events. In the language of Bayes, this means that the events are *mutually exclusive* (in a given instance, no two of the events defined on a sensor can simultaneously occur). This does not mean, for example, that a single value of IAS can not correspond to two different sub-events. The fact that two different sub-events can each be represented by the same IAS value is simply an uncertainty, or imprecision of the modeling of events on IAS.

Decision Confidence Measure: Elementary Hypotheses.

The abductive decision of flight mode is built up piece-wise, using partial decisions based on each sensor. To do this we need a numerical decision measure and a decision strategy. The strategy is to take a sample value of a sensor variable and use it as input to a function which returns the decision measure. The appropriate Bayesian measure is of the form:

$$P(A_i|x) \; ; \; 0 \le P(\cdot) \le 1$$

where $P(\cdot)$ is a measure of Bayes posterior conditional probability. The decision rule is to decide "sub-event m" such that

$$\underset{i=1}{\overset{M}{\mathrm{Max}}} \left\{ P(A_i|x) \right\} = P(A_m|x) \Rightarrow \text{'event - m'}$$

This is the M.A.P. decision rule. The value, $P(A_m|x_0)$, where x_0 is the unique sample value of sensor variable, x, is also a "Confidence Measure" or "Confidence Factor" on the decision.

In the Fuzzy Control context, the $P(A_i|x)$ are the sensor membership functions. As mentioned before, they can be computed from prior knowledge of the application or from models of the a priori probability densities. The latter computation is given by

$$P(A_m | x) = p(x | a_m) / \sum_{i=1}^{M} p(x | A_i)$$

where there are M possible sub-modes. Note that the $p(x | A_i)$ are densities, normalized to unit area, while the $P(A_i | x)$ are probabilities, normalized to unit amplitude. The $p(x | A_i)$ are sometimes easier to model directly than are the $P(A_i | x)$.

Compound Decision: The Extension.

Now, we wish to model the case where we have a collection of possible modes and a collection of sensors by which to decide upon a single mode. What we have to work with may be thought of as an array of membership functions (confidence measures) of the form $P(A_i | x)$, where

$1 \leq i \leq M$: denotes M modes, and
$1 \leq j \leq N$: denotes N sensor variables.

We take a set of measured values, $(x_1^{\circ}, x_2^{\circ}, \ldots, x_N^{\circ})$, which we may think of as a vector, $\underline{x}$. We define the event, A_i, as a subset in the vector space generated by the x_j (Cartesian product space if the x_j are numerical). We test $\underline{x}$ to make the decision on the occurrence of the event, A_i. If $x \cap A_i$ is not empty, then we say "A_i occurs." The decision is not "hard," because we shall create a measure of confidence in the occurrence of A_i from the membership functions.

It is the practice in Fuzzy Control to utilize one-dimensional membership functions (those which depend on just one sensor variable). Thus, we shall have to deal with the event, A_i, one variable at a time. This we do by defining "sub-events," A_{ij}, which are projections of A_i onto its various sensor-variable axes (see [4] for details). Then the decision rule becomes "$\bigcap_{j=1}^{N} (x_j \cap A_{ij})$ not empty implies A_i occurs." This method of defining sub-modes is exemplified in Figure-2 for aircraft, using the IAS and ROC (rate of climb) variables.

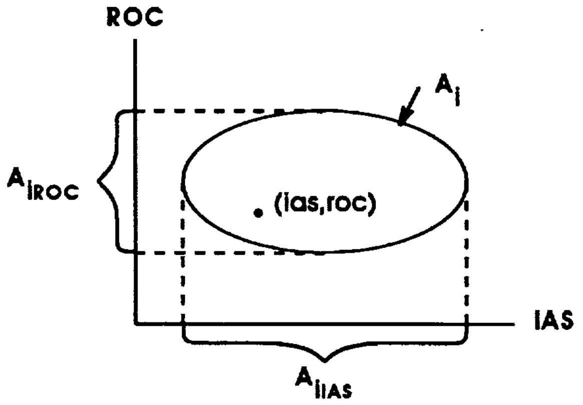


Figure-2. Sub-mode Definition.

The Bayes (or Soft Fuzzy [4]) decision is then made by applying the M.A.P. decision rule to the joint probability density, $p(\bigcap_{j=1}^{N}(x_j \cap A_{ij}))$. Given that the sensor variable, x_j, represents (probabilistically) independent measures of the event, A_i, the same decision will also be obtained by applying the M.A.P. rule to the arithmetic product, $\prod_{j=1}^{N} P(A_{ij} | x_j)$, where the $P(\cdot)$ are the individual membership functions.

Abductive Decision : Conflicting Evidence.

Now, let's see what all this means when we have conflicting evidence and more than two sensor variables. The classic example is a gear-up landing. We'll use it to illuminate our decision strategy.

If we plot membership function values on a MODE-SENSOR plane for a gear-up landing, it might look like Figure-3. IAS, ROC and FLAPS have appropriate values for the LAND mode, but GEAR is zero (gear-up). GEAR looks more appropriate to Initial Approach (APPR_INIT) or Descend on Course (DSC_ON_CRS).

Now, the question is, how can we still decide LANDING correctly, but call out an ALARM_GEAR? If we adopt a "Parallel-Experts" approach (an architecture using multiple "expert objects" that "know" about a particular domain)[6] , the first question is whether to organize the experts as

Sensor Experts or Mode Experts. We have chosen to use Mode Experts. And that goes along with our M.A.P. strategy.

The question now is, what to do about the gear sensor variable. If the "gear-up" membership value (zero) is accepted as valid information, it comes against the IAS, ROC and Flaps information in terms of establishing LANDING as the current mode. Hence, we should expect the confidence of the compound decision, LANDING, to be reduced from what it would be for "gear-down." But, since the GEAR variable is binary, if it is not 1, it is 0, which destroys the M.A.P. product. (A 0-1 membership function is not strictly fuzzy, but "crisp.")

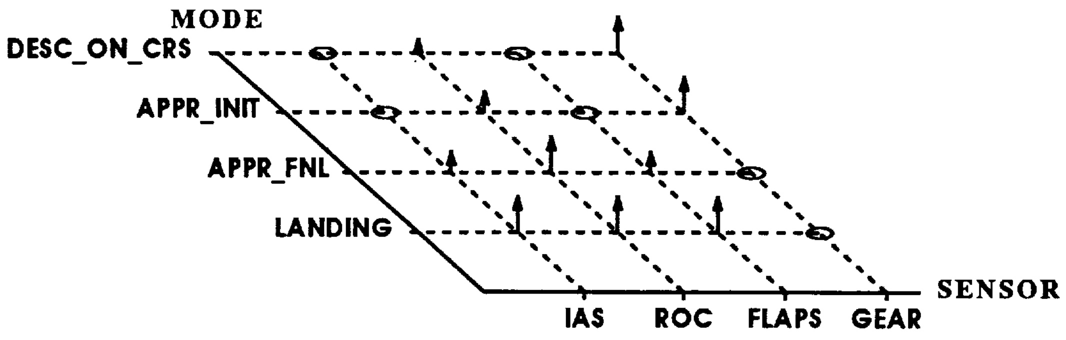


Figure 3. Example: Gear-up Landing.

So, we calculate the confidence value as described above, but exclude the 0 value of the gear membership function, and flag the GEAR variable to indicate that it is out of range for mode LANDING. This is the gear alarm. Then, along with the M.A.P. rule, which chooses the mode with the greatest confidence value, the alarm set on mode LANDING is also taken into account when choosing the "winning" mode. This is the point where abduction is crucial. The act of "the first stating a hypothesis [choosing a mode] and the entertaining of it, whether as a simple interrogation or with any degree of confidence" is defined to be abduction.^{5} And that is exactly what is being done here.

RESULTS.

The first-pass results of the Object-Oriented, Fuzzy-Bayes, Abductive Interpreter have been very encouraging. Under normal flight conditions (no alarms) the Interpreter selects the proper flight mode with confidence values often reaching the maximum value of 1.00. The most critical step in creating the Interpreter was the creation of the one-dimensional, fuzzy membership functions. A flight manual for the airplane being modeled was obtained which provided the a priori information needed to construct them. The most time-consuming work done thus far has been the "tuning" of these to this application. That is, adjusting the shape and value of the membership functions to truly reflect the relationships between sensors and modes.

CONCLUSION.

The value of the research, to this point, is in its potential to aid in the control of systems of growing complexity. The human operators of these systems are being supplied with an increasing amount of information which must be assimilated in order to control the plant (an airplane, in this instance), increasing the probability of an error being made. In a cockpit, a working Interpreter provides a system which can discover potential problems (e.g.: no-gear landing) and warn the pilot before they become life-threatening ones.

CHAPTER-5 BIBLIOGRAPHY

1. Painter, John H., (1992). Knowledge-based Control, *Proceedings of the 1992 IEEE Symposium on Computer Aided Control Systems Design*, pp. 138-147.

2. Glass, E., and Painter, J.H., (1992). Discrete Fuzzy Control for Aircraft Flaps, *Proceedings of the Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems*, College Station, TX.

3. Bellman, R. E., and Zadeh, L. A., (1970). Decision-Making in a Fuzzy Environment, *Management Science*, Vol. 17, No. 4, pp. 141-164.

4. Painter, J.H., (1992). Soft Fuzzy Control, *Proceedings of the Second International Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems*, College Station, TX.

5. Punch, W.F. III, Tanner, M.C., et al, (1990). Pierce: A Tool for Experimenting with Abduction, *IEEE Expert*, pp. 34-44.

6. Jowers, S.J., (1988). Symbolic Diagnosis for Intelligent Control of Real-Time Systems, *M.S. Thesis*, Dept. of Elec. Eng., Texas A&M University, College Station, TX. 77843. Available from Xerox University Microfilms, 300 N. Zeeb Rd., Ann Arbor, MI. 46106.

CHAPTER 6

A GRAPHICAL USER INTERFACE FOR KNOWLEDGE-BASED CONTROL.

by
Paul J. Russell, M.S. Candidate
&
Dr. John H. Painter
Electrical Engineering Department
Texas A&M University
College Station, TX, 77843-3128
409, 845-7441
painter@zadok.tamu.edu

The following chapter is extracted from a Thesis submitted to Texas A&M University by Paul J. Russell in partial fulfillment of the requirements for the Master of Science Degree. The necessary editing to produce this shortened version was done by Dr. Painter.

ABSTRACT.

This thesis reports the results of research to create a Graphical User Interface to support a Knowledge-Based controller. The objective is to investigate the environment of object-oriented programming techniques. The application is a graphical user interface for a simulated Boeing-737 flight management system. All research uses the Eiffel programming language. This research was supported, in part, by the National Aeronautics and Space Administration, Langley Research Center.

THE GUI AS A SYSTEM COMPONENT.

A knowledge-based controller may be subdivided into four entities: a symbolic controller, a symbol interpreter, a user interface, and an internal blackboard. The symbolic controller (meta-controller) exerts control on the numerical controller to produce the desired goals. The symbolic interpreter (interpreter) gathers raw numerical data from the dynamic system and decides what mode the system is in. The user interface acts as a symbolic input/output device for the operator and makes the system useable and practical (Figure 1). The blackboard is an internal data controller that coordinates communication between the subsystems.

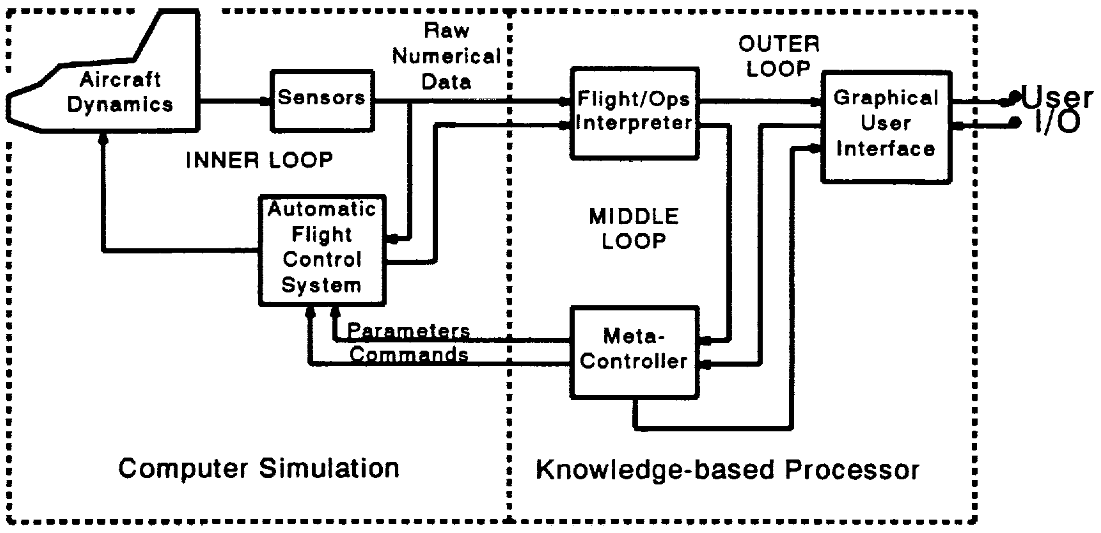


Figure 1. Architecture

The application described in this thesis is a graphical user interface (GUI) for a knowledge-based controller. The objective of the research effort is to produce an object-oriented GUI capable of managing the input/output flow of data relative to an aircraft flight management system. The goals of the flight management system are to control the aspects of normal flight cruising, final approach, and landing for a simulated Boeing-737 transport aircraft.

A GUI's role in a dynamic system is to be an "impedance-matching" device for information exchanged between the dynamic system and the human user. A GUI must be designed so that only a minimal amount of information is lost or misinterpreted when a user accesses the system. The user must be able to grasp all relevant information and manipulate inputs to the system in a timely manner. Also, it is widely accepted that a good user interface will not give the user an option of choosing illegal actions or a combination of actions[1].

The designer of an aircraft GUI should keep in mind the type of individual who will be using the system. Bearing in mind that the user is assumed to be a qualified pilot, an aircraft GUI should contain all the necessary information to fly an aircraft within a recognizable, graphical format. A standard aircraft instrument cluster and command inputs are essential.

Here, the GUI contains a standard instrument cluster, directional controls for the autopilot, and an interface for the flight management system. Consistent with object-oriented software properties, this application implements a basic GUI architecture useable for more than just an aircraft application. This multi-use structure is applied to internal components such as communication routines, animation techniques, and functional loops. Object-oriented programming assists with the emphasis on the reusability of the GUI.

A successful implementation of object-oriented programming is another goal of this research. Object-oriented programming is a utilization of software engineering to produce higher quality software, which is gaining momentum within the software engineering community. Object-oriented techniques aim to produce software that is reusable, flexible, and easily modified.

Based on this defined problem statement, this chapter contains a detailed description of the GUI, its functions, and its structure. In addition, the theories of object-oriented programming are evaluated and justified for this application.

FUNCTIONALITY ISSUES.

INTRODUCTION.

Layout and functionality are important design considerations for any GUI and are legitimate research topics themselves. Concerning this GUI, design effort was placed on functionality but was not a central theme for this project. This was decided because there did not exist a large mass of information that needed to be exchanged in some sort of multiplexed or condensed format. Also, the user is expected to be a pilot who is considered an expert in aviation. A pilot does not require explanations of most of the information given; he or she needs only a reasonably fast, simple interface.

The layout of the GUI is divided into three sections: the output, input, and guidance sections. The first section is the output section containing the flight instruments and other sensors. The flight instruments are designed to be visually realistic, functional, and simple. The second section contains the autopilot input buttons that manually fly the simulation. The autopilot input buttons were designed to be uncluttered and simple. The last section is the guidance section, which handles all input/output for the meta-controller and the interpreter. The guidance section is designed to be useful and versatile.

OUTPUT SECTION.

The flight instruments (gauges) in the GUI are designed to maximize the information transferred to the pilot by use of instruments that would be found in a comparable Boeing-737, series-200, aircraft. There exist six standard instruments on modern aircraft: airspeed, vertical speed indicator, altimeter, artificial horizon, heading indicator, and turn coordinator ("needle and ball").

The 737 simulation is linearized (piece-wise), being an integrated chain of linearized models of predicted, normal flight conditions (called trim-points). The simulation contains models for flying modes ranging from cruising to landing. However, in the interest of simulation simplicity, no models were designed to handle turning of the aircraft. Because the simulation manipulates the aircraft only in the vertical plane with no turning, there is no need for a heading indicator or a turn coordinator. In their places (in Figure 2) were added an engine pressure ratio (EPR) gauge and pilot direction indicator (PDI), which indicates aircraft position, relative to the glideslope (vertical landing path).

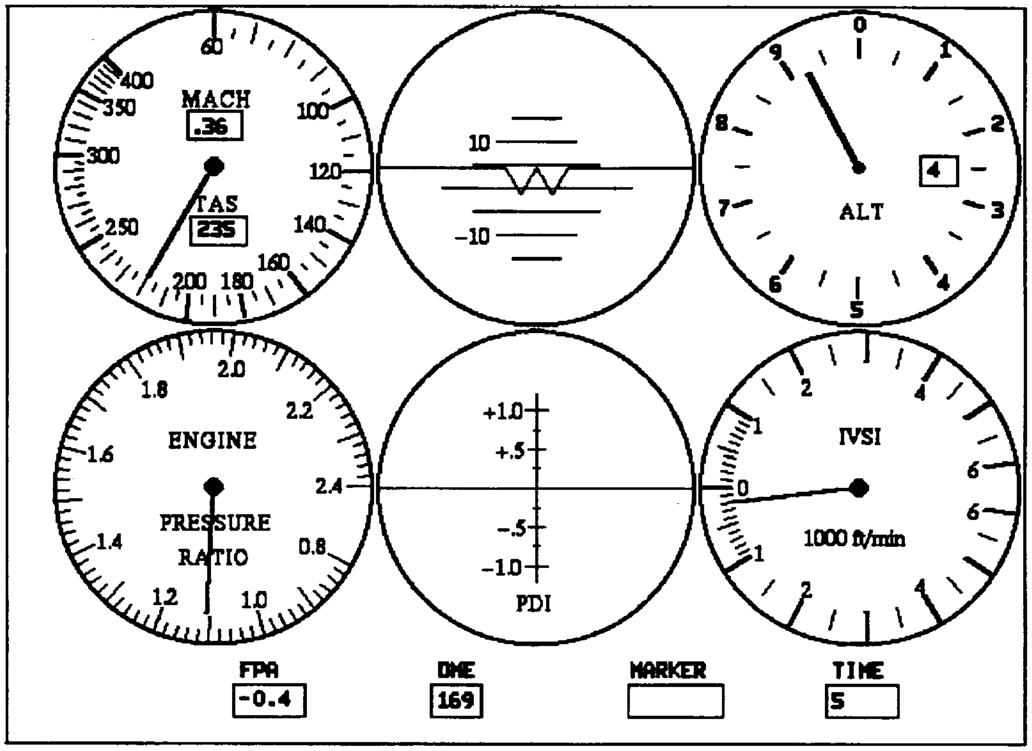


Figure 2. Output section.

The airspeed gauge (upper left) was the first created, serving as a model for the other gauges. It has a non-linear presentation, indicated in knots by the rotary needle. It also indicates true airspeed and Mach number as numerics. Note that indicated airspeed is the value of speed produced by air pressure measured in a Pitot tube. True airspeed is then computed from indicated airspeed and altitude.

Engine pressure ratio denotes the pressure differential between the intake and exhaust of the main engines. It is a measure of how much thrust a turbo-jet engine is producing. The EPR gauge (lower left) has a linear range of 0.8 (which is an engine idle condition) to 2.4 (which is a full thrust condition). The simulation does not permit direct control of engine throttle settings. It allows only for air speed settings, with the autopilot adjusting the engine thrust to achieve the desired airspeed.

The artificial horizon gauge (upper middle) emulates a gyroscopic gauge that maintains a constant, steady horizon. The artificial horizon indicates roll and pitch of the aircraft as a reference for the pilot. Roll is defined as the rotation of the aircraft about the longitudinal axis (parallel to the direction of motion). Pitch is defined as the rotation of the aircraft about a lateral axis (up and down rotation)[2]. Because the simulation can change only altitude (no turning), the artificial horizon in the GUI indicates only changes in pitch (Figure 3). As the simulation changes flight path angle or angle of attack, the artificial horizon displays changes in degrees off the horizon. Maximum range is +/- 15 degrees.

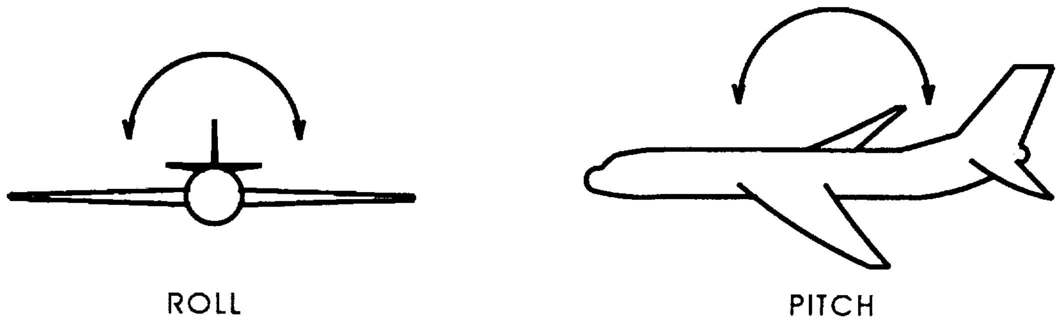


Figure 3. Roll & pitch diagram.

Flight path angle (FPA) is the angle between the direction an aircraft is travelling (velocity-vector) and the local horizontal. Angle of attack is the angle between the velocity-vector and the longitudinal axis of the aircraft. Angle of attack is directly proportional to aerodynamic lift produced by the wings. In a landing scenario, for example, a pilot uses a large angle of attack to produce extra lift at low air speeds (flaring).

A single-handed altimeter (upper right) with a numeric window was designed specifically for this application. The numeric window indicates thousands of feet while the rotating needle indicates hundreds of feet. The altimeter was designed as a simplified model compared with the three-handed altimeter found in most cockpits that resembles an analog clock. This was necessary to maintain a reasonable level of computational speed for the program as a whole. There generally exists an inverse relation between computational speed of the software and the amount of animated "mass" that must be manipulated during every iteration of the GUI.

The vertical speed indicator (VSI) displays rate of climb or descent in thousands of feet per minute. The VSI gauge (lower right) is also non-linear and closely mimics the one found in a true Boeing 737. However, the GUI's VSI is different because the information is calculated and displayed instantly. In a true Boeing 737, the VSI gauge integrates samples of changes in air pressure. This results in a five-second delay of the readings in an actual gauge.

The instrument landing system (ILS) is an important aid to the pilot when making reduced-visual or instrument landings. The ILS consists of an array of radio-frequency beacons, aircraft sensors, and instruments. Its purpose is to provide distance references and to assist the pilot in locating the aircraft within a standardized glide slope of 3 degrees with respect to the horizon. This GUI utilizes three ILS instruments: PDI, distance measurement equipment (DME), and marker beacons (Figure 4).

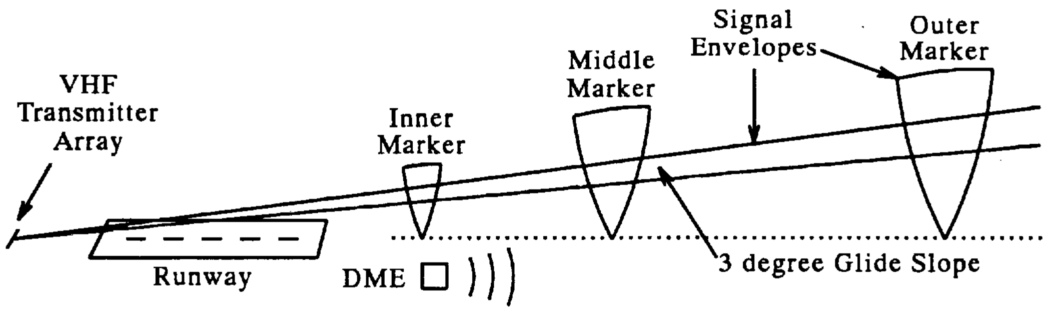


Figure 4. Typical ILS System.

The PDI gauge (lower middle) indicates glide slope deviation relative to the 3 degree standard. This instrument lies dormant until the glide slope radio signal becomes strong enough to receive and interpret (usually at 30 nautical miles from the runway). Then a horizontal bar appears and slides up or down the scale (a similar action to the artificial horizon gauge), indicating a deviation of +/- 1.0 degree.

The DME display indicates the distance in nautical miles between the aircraft and a beacon located 1000 feet in front of the runway threshold (Figure 4). This gives the pilot long-range information during approaches to an airport. The theoretical range of the DME is approximately 199 nautical miles; however, the simulation provides DME information at any distance to the runway.

Marker is another ILS distance indicator that displays when the aircraft is flying over three radio beacons located close to the runway. The beacons are called outer, middle, and inner by the Federal Aviation Administration (FAA). Their distances to the runway are 4.5 nautical miles, 2800 feet, and 1000 feet, respectively. The marker's purpose is to give another reference to the pilot during final approach and landing scenarios.

The time display shows the simulated flying time (in seconds) that has passed since the beginning of a simulation run. The FPA window (below the EPR gauge) always indicates the current and true flight path angle the aircraft is flying, regardless of what is being commanded by the autopilot. FPA is not a standard aircraft measurement or display and is presented here for user information.

INPUT SECTION.

The input section consists of all controls necessary to manipulate the autopilot (i.e., manually fly the simulation). Input from the pilot to the GUI consists of an array of buttons manipulated by a mouse. When the arrow icon on the screen enters a button, the button is highlighted. The button blinks when the mouse is clicked, and the respective routine is executed.

Different schemes for entering information were considered, but the most practical are the buttons. Using the array of buttons is simple because it requires only a computer mouse and not a keyboard (Figure 5). There is the possibility that the mouse and button system could be substituted by a touch screen display in a real aircraft since the use of a keyboard might be unrealistic.

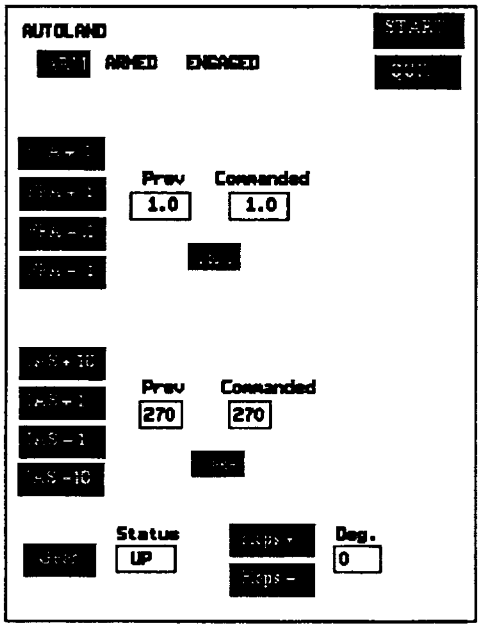


Figure 5. Input Section.

The START and QUIT buttons at the top of the screen fulfill their obvious functions of starting and stopping a simulation run. The AUTOLAND button arms the landing mode of the autopilot. The autoland takes full control of the aircraft and makes proper adjustments to the FPA, to track the 3 degree glide slope signal. The autoland can be "armed" at any time during a flight, but it will only "engage" at approximately a DME of 30 nautical miles and a deviation of less than +/- 0.1 degree off the true glide slope as shown on the PDI gauge. The ARMED and ENGAGED words are normally hidden on the screen and appear when necessary.

FPA is adjusted by the array of positive and negative increment buttons. The pilot "dials in" the requested FPA value, which is displayed in the PREV (preview) window. Once this is achieved, the TAKE button is pushed and the preview FPA value is communicated to the blackboard. Once at the blackboard, the data is routed to the autopilot and possibly the meta-controller. For confirmation purposes, the new FPA is also sent from the blackboard to the GUI and appears in the COMMANDED window. The range of increments for the FPA is +/- 0.1 and +/- 1.0 degree. Changes to indicated air speed (IAS) operate in the exact same manner as FPA, with the exception that the increments are +/- 1 and +/- 10 knots.

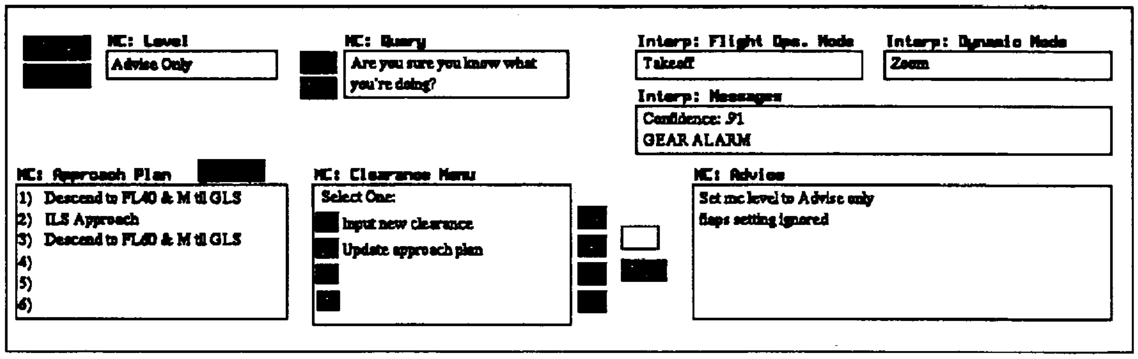
The GEAR button toggles the landing gear of the simulation. The display window next to it will indicate only UP or DOWN. The FLAPS+/FLAPS- buttons set the aircraft flaps to predetermined values. The flaps are aerodynamic devices on the trailing edge of the wings. They are used to produce extra lift for low-speed landings and takeoffs, yet have the disadvantage of producing added aerodynamic drag. The amount of lift produced by adding flaps is proportional to the angle of the flaps setting. The absolute flaps pre-sets available in the simulation are 0,1,5,10,15,25,30, and 40 degrees.

GUIDANCE SECTION.

The guidance section is the most complex input/output data flow manager of the GUI. It is also unique to this GUI because it is specific to the functions of the interpreter and the meta-controller. The interpreter function comprises three output windows; the meta-controller has 5 output windows and an array of input buttons (Figure 6).

1. Interpreter Interface.

There are three interpreter windows: dynamic mode, flight operations mode (flight ops), and interpreter messages (Figure 6). The dynamic mode window displays a mode such as "climb" or "dive," which is an instantaneous description of what the plane is doing. The interpreter decides this mode, based on information from the airplane's sensors and fuzzy rules (membership functions) which are pre-defined. The flight ops mode window displays a mode such as "Go Around," "Approach," or "Cruise." This is a higher level, less instantaneous description of the airplane's mode. The decision processing for this mode is based on sensor input in a way similar to the dynamic mode decision. In addition, it also includes some memory of previous flight ops modes and the current dynamic mode. The message window is used for the interpreter to communicate to the pilot about any aircraft or operation abnormalities or anomalies detected by the decision process. For example, if sensors indicate the airplane is about to land and the landing gear is not down, the pilot would be notified about the situation.


MC: Level
Advise Only

MC: Query
Are you sure you know what you're doing?

Interp: Flight Ops. Mode
Takeoff

Interp: Dynamic Mode
Zoom

Interp: Messages
Confidence: .91
GEAR ALARM

MC: Approach Plan
1) Descend to FL40 & M til GLS
2) ILS Approach
3) Descend to FL60 & M til GLS
4)
5)
6)

MC: Clearance Menu
Select One:
Input new clearance
Update approach plan

MC: Advice
Set mc level to Advise only
flaps setting ignored

Figure 6. Guidance Section.

2. Meta-Controller Interface.

The meta-controller functions are the most complex functions of the GUI because of the unique requirements for communication with the pilot. There are control levels, output windows, and air traffic control (ATC) clearance information exchange. Specifically, there are five output windows: level, approach plan, query, advice, and clearance menu (Figure 6). The advice window is the only true, pure output window with no corresponding input button for pilot feedback.

The meta-controller operating level (MC: Level) denotes the overall level of control being exerted by the meta-controller. The pilot has control of this level and there are four options available: Advise Only, Emergency Prevention Only, Pilot Assistance, and Terminal Control. The pilot may select the level of control at any time during a flight. This design leaves the pilot "in the loop" as the final arbiter of the amount of guidance provided by the meta-controller.

Advise Only mode represents what would be thought of as manual control mode. The pilot is in complete control of the autopilot, and the meta-controller will give only warnings of improper commands and will recommend flap, gear, and autoland settings. In Emergency Prevention Only mode, the meta-controller passes all valid commands to the autopilot and will make flap, gear, and autoland command recommendations to the pilot. The definition of "passing a valid command" is the meta-controller determining the appropriateness of a flight command relative to the current flying conditions. Pilot Assistance mode enables the meta-controller to pass all valid commands to the autopilot, plus plan and execute flap, gear, and autoland commands automatically. In Terminal Control mode the meta-controller plans and issues commands to the autopilot to take the airplane from cruise to landing, under a pilot-specified approach plan. This approach plan is the only way the pilot may specify the behavior of the airplane during Terminal Control mode. The Terminal Control mode is the most complex aspect of the meta-controller and is given the most research effort.

The approach plan (MC:Approach Plan) contains a numbered list (max 6) of the ATC clearances that the pilot specifies for the meta-controller to follow during Terminal Control Mode. The clearances use standard FAA abbreviations to condense information. Once these clearances are approved by ATC, the pilot presses the associated PROCEED button, which directs the meta-controller to recognize the clearances.

The clearance menu (MC:Clearance Menu) is a device for the creation and change of the approach plan, and it can enter new clearances. New clearances are changes to the approach plan that are requested by ATC and must be acted upon immediately (i.e., rapidly changing weather conditions or impending collision). There are seven different menus that the pilot must page through in order to specify all the information that is necessary. This is the only instance where a menuing system is used in the GUI. Menus tend to be slower to use, but they exchange a greater amount of information. Normally, pilots request clearances sequentially from ATC during a flight. This menu and the approach plan will now allow the pilot to plan all necessary clearances and communicate this information electronically with ATC.

The query window (MC:Query) represents a section of the meta-controller that constantly monitors pilot commands and makes decisions as to the appropriateness of any given command. If it witnesses a command it deems abnormal for the given circumstances, it asks the pilot to confirm the input with the corresponding YES/NO buttons.

The advice window (MC:Advice) displays any general comments the meta-controller wishes to provide the pilot. It gives recommendations for proper flying techniques. This window also contains any future output requirements, if necessary.

IMPLEMENTATION ISSUES.

SOFTWARE ENGINEERING.

The universal aim of software engineering is to generate quality software. Yet, it is difficult to produce satisfactory results when experts in the field agree that vast inefficiencies exist within the industry. The typical engineering lifecycle for software consists of 7 stages: requirements definition, design, implementation, testing, installation, and maintenance[3]. It is the last step in the process, maintenance, that consumes the majority of costs.

It has been determined that the software industry consumes 70% of its resources on maintenance (debugging, rewriting, or writing newer versions)[4]. Although a portion of the workload resembles creative thought, the vast majority of this pursuit constitutes rehash of previous labors of co-workers. Often one programmer may not be responsible for a whole task over the software's lifetime, requiring coordination between a succession of programmers. Thus, the productivity of human resources in the software construction industry is relatively low.

One solution to these problems lies in the tools or languages computer specialists use today. The classical languages (i.e., FORTRAN, Pascal, C, Cobal) promote programming environments such that functions control data, with these functions having a specific, well-ordered layout within the body of the program. This application-specific set-up works well for small, low-level jobs. However, for jobs with increasing complexity or dynamic conditions, the difficulty of changing a system of structured functions can increase geometrically with the size of the original system. This is because a modification in one section of the program produces a "rippling effect" that disturbs other sections severely. Also, complex flows of commands within the source code become a temptation for programmers to produce quick fixes to normal programming obstacles (i.e., "spaghetti" programming).

THE OBJECT-ORIENTED APPROACH.

A better system is organized when functionality and program control are modularized. This creates a decentralized architecture in which programming problems are localized and more easily identified. Modifications disturb smaller portions of the system and thus are made easier. This decentralized design theme falls under the domain of object-oriented programming.

Object-oriented programming was conceived from computer science's software engineering subdiscipline as a means of symbolic representation of data. Object-oriented design has developed into its own respected field of software engineering. Object-oriented programming strengths have been highlighted recently for an ability to endure an increased measure of maintainability and reusability. Another useful property is a proficiency for supporting artificially intelligent environments while simultaneously handling classical programming problems.

This present research project is embedded in object-oriented programming. The term "object-oriented" means that software is organized around discrete objects that represent data and functions. This is different from conventional programming where the relationship between data and functions is not emphasized. There is a measure of uncertainty between software experts as to what exactly defines object-oriented programming; however, there are generally four characteristics: identity, classification, polymorphism, and inheritance[5].

GUI DEVELOPMENT REQUIREMENTS.

Many of the beneficial properties of object-oriented programming suit the needs of a potential GUI for knowledge-based control, especially in terms of system construction. Many GUIs are designed to have a separate function from the rest of the system. The bottom-up, decentralized nature of object-oriented programming fits this design image because changes based on changing (dynamic) system specifications are simple. Also, the method of geometric modeling of objects is useful in system development. There is a direct correspondence between objects and the icons/images the user sees. This results in a clean and uncluttered system design[6]. The reusability aspects of object-oriented software can be employed during construction when completed graphical objects are copied and modified for another use. Any repetitious patterns in a GUI can take advantage of this property.

Performing graphical programming in classical languages can be quite complex, especially when attempting movement or animation. In an object-oriented environment creation of images is simplified by an extensive graphics library. The programmer can call upon many classes capable of producing the necessary graphical objects. In addition, objects can be combined in a process called composition. This allows several objects to be combined into a single, complex object, which can be treated as a unit. Animation or movement is made easier in an object-oriented environment because only a simple position adjustment is needed. This is especially useful when dealing with complex objects.

EIFFEL, THE SELECTED LANGUAGE.

1. Introduction.

There is a number of object-oriented programming languages available on the market today (C++, Ada, SIMULA). In 1986 the Eiffel programming language was released and publicized as one of the purist environments for object-oriented software construction. Eiffel is a language and environment intended for the design and implementation of quality software in production environments. The language is based on the latest principles of object-oriented design, with a special emphasis on producing reliable software from industrial components. Eiffel promotes a method of software construction by combination of self-contained and flexible modules[7]. The supporting environment includes a library of reusable classes and several useful development tools. For these reasons, in 1989, Eiffel was chosen by predecessors in knowledge-based control as the programming environment for future research, including this present project.

Eiffel was written and developed by Bertrand Meyer and his company, Interactive Software Engineering (ISE) in Santa Barbara, CA. Most of the central design themes of Eiffel can be traced to Meyer's work with Simula, another object-oriented language. He also added important concepts from his academic work on software verification and computer language definition. Eiffel has evolved continuously since its first introduction in 1986[8]. Eiffel is one the few software development systems that offers an academic textbook written by Meyer, explaining its software theories.

The programming environment that Meyer created compels programmers to produce quality software in Eiffel's object-oriented image. This "ideal" software image that Meyer is striving for encompasses what he calls five external and five internal properties.

The external properties represent top-level characteristics whose presence or absence may be detected by the users. The internal properties are perceptible traits that only computer professionals recognize. It is important to note that the external factors take precedence in the end because the user does not care how internal mechanisms are organized but, rather, how well the software works for him or her. However, the internal factors are the key to ensuring that the external factors are satisfied[9].

PROGRAM DESIGN AND APPLICATION DETAILS.

DESCRIPTION OF SYSTEM ARCHITECTURE.

1. Flow Charts.

When describing object-oriented system architecture, it is practical to show relations between classes graphically. This conveniently uses a human's symbolic reasoning abilities. The conventional programming world calls this a flow chart. In a conventional flow chart, the flow of data comprises the lines that link the sections of the program (nodes). The nodes in the chart are the functions or processes that transform the data.

In an object-oriented system, the flow of data is not very obvious and is sometimes confusing to follow. A proper graphical representation of the system should show only the relationships between classes (nodes). The only two real relationships between classes are inheritance and client/supplier. The flow of data between classes is not apparent. Data flow is considered unimportant during the analysis of an object-oriented system because it is so highly regulated in the source code and yet unknown as to its exact location during execution.

2. Graphical Representation.

The following figure shows the top-level architecture of the knowledge-based controller, using Eiffel's GOOD facility. The class, EXECUTIVE, is the "root" class where the main program control loop resides. Classes GUI, META_CONTROLLER, INTERPRETER, and BLACKBOARD are the four major modules of the system. The double-lined arrows indicate client/supplier relationship. A single-line arrow (not shown) indicates an inheritance relationship. Control of each module by the EXECUTIVE is exerted through the client/supplier relationship. Communication between modules and the BLACKBOARD is also accomplished by the client/supplier relationships (Figure 7). Actually, communication between the GUI and BLACKBOARD classes is accomplished with C routines because of the fact the GUI is a separate UNIX process (a design oversight of Eiffel graphics).

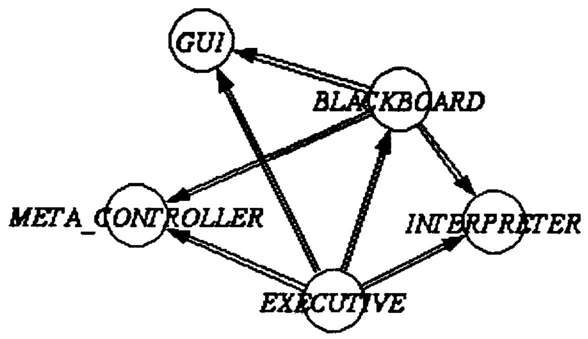


Figure 7. The EXECUTIVE and Related Classes.

Not shown in Figure-7 are inheritance relationships, because this would clutter the figure with over 100 classes. These inherited classes are support classes that come from the Eiffel libraries. They produce the base of the functional infrastructure of the system and supply operations ranging from basic math functions to complex searches of multi-dimensional arrays.

The next layer of analysis of the GUI is shown in Figure 8 . Above are all classes inherited by the GUI, and below are the suppliers and clients (EXECUTIVE,BLACKBOARD) of the GUI. The suppliers directly represent the gauges and output windows seen on the GUI. For simplicity, not shown in Figure 8 is the supplier relationships of all the buttons, which would clutter the figure with thirty classes. Of the inherited classes; BASIC and SINGLE_MATH provide simple math functions from the Libraries, GUI_GLOBAL_OBJECT provides public or "global" information to each of the classes in the GUI, and PROJ_CONST shares information with the meta-controller.

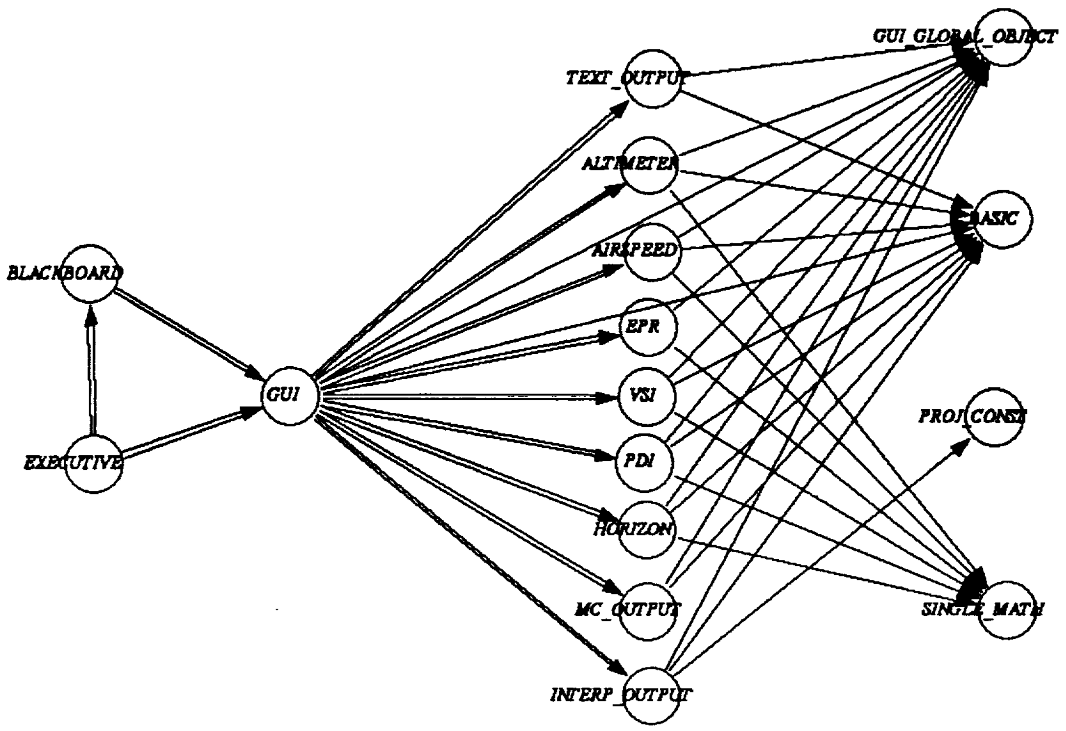


Figure 8. The GUI and Related Classes.

USING THE GUI.

1. Flying.

To use the GUI is an entirely simple enterprise. However, to fly the simulation requires the knowledge of a pilot to properly land the aircraft. This requires one to have working knowledge of the proper settings of the aircraft controls for every conceivable flying condition. Under most conditions this meta-knowledge consists of knowing what altitude and airspeed should be maintained during cruising and normal descent to the runway. Also, the pilot should be familiar with the final approach sequence and proper positioning of the aircraft before engaging the auto-landing system. In theory, any qualified 737 pilot could use the GUI for the first time and successfully complete a landing. However, through repeated attempts, an unqualified user is capable of acquiring the "feel" of the aircraft and performing successful landings.

As stated before, only cruising and landing scenarios were programmed with the simulation. A take-off scenario is possible, but was not programmed. Simulation models for take-off and ascent were not created because academic resources were focused on the landing paradigm. Pilots agree that airport approach and landing are the most difficult aspects of flight and are therefore the most taxing on meta-controller ability.

2. Initialization.

To run the system, a user types the name of the executable file and hits return. The UNIX filename of the executable is "executive." The program asks the user for the initial states for the aircraft simulation and other information. The simulation requires an initial airspeed, altitude, and distance to the runway. The simulation polling frequency is also requested. This is the rate that the simulation is sampled in time. For realistic results that match the rate the GUI can iterate, a value of two seconds is suggested. Last, the initial meta-controller operating level is requested. A recommended starting point for a simulation run would be a cruising state of 25,000 feet altitude, 300 knots IAS, a DME of 100 nautical miles, and the meta-controller in Advise Only mode (Figure 9).

The default values are presented in the brackets [ ]. The user has the option to save any new set of initial values. The software then generates an X-Window application window and fills it with recognizable graphical objects of the main display of the GUI. There is a six-second delay while

```
EXECUTIVE: Start-up file environment variable AFCS_STARTUP not set.
Simulation poll freq. (sec)    [2]: 3
IAS (knots)                    [250]: 300
Altitude (feet)                [10000]: 25000
DME (nautical miles)           [90]: 100
Meta-controller operating level:
        0) Advise Only
        1) Emerg. Prevention Only
        2) Pilot Assistance
        3) Terminal Control
Meta Controller Level      [0]: 0
Save the new values?       [n]: n
Simulation:                Setup complete.
```

Figure 9. Initialization of the EXECUTIVE.

the remaining background programs are started. To begin a simulated flight the user first clicks the START button with the mouse. Next, the user must wait for the autopilot to fly from the center of a linearized model towards the specified starting point, known as Start-Up Mode. After this is accomplished, full-control is transferred to the user.

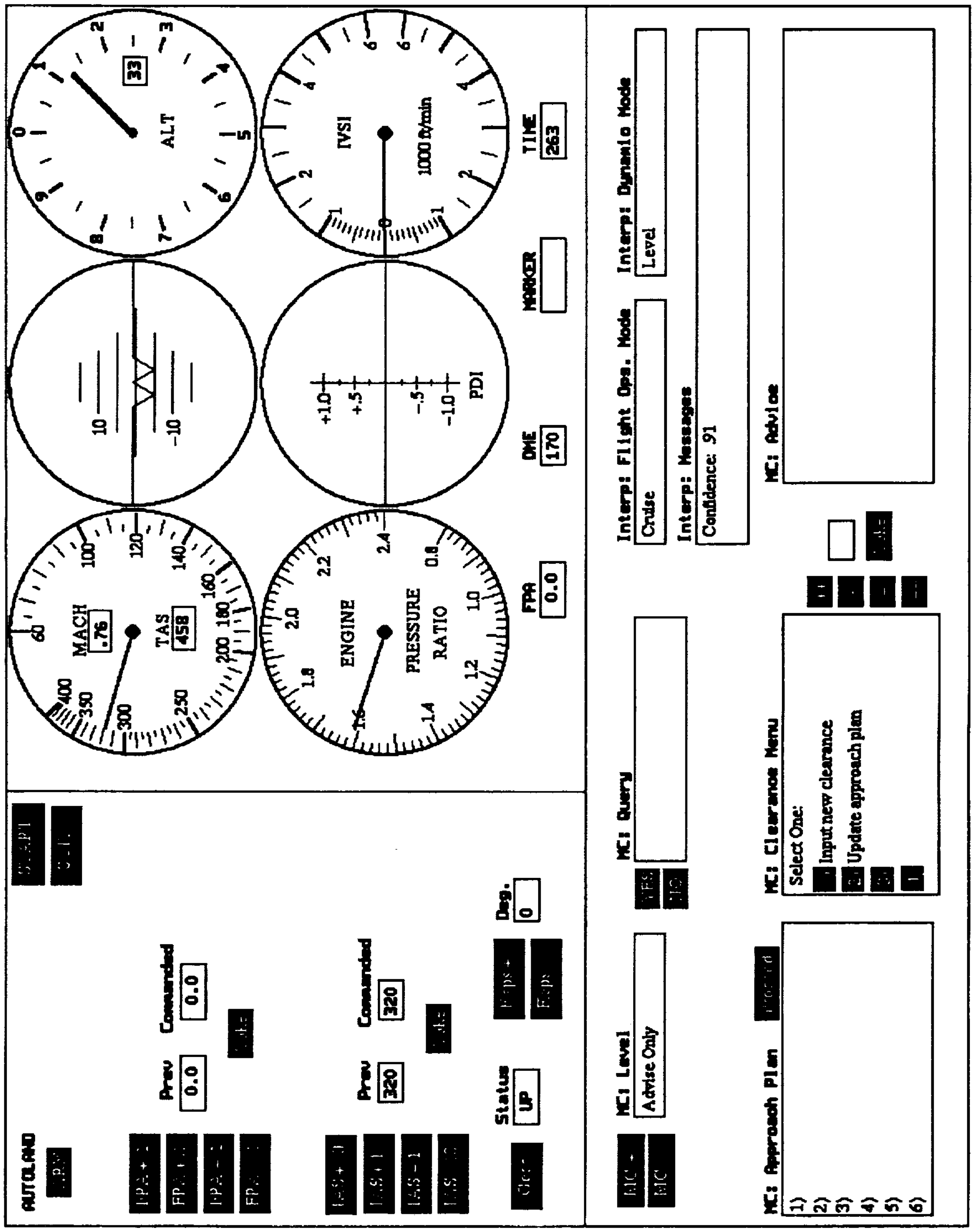
If the pilot wishes to use manual control of the aircraft, he or she "flies" the simulation by selecting inputs to the autopilot. There are no "stick & rudder" controls that one would expect in most aircraft simulations. The scope of the research grant was to explore knowledge-based control schemes for flight management and not to produce the most realistic flight simulation. The computer simulation is manipulated by an inner-loop, numerical autopilot. Only input is needed from FPA, IAS, landing gear, flaps position, and autolanding system.

FPA and IAS commands are the most important flight controls. Note that changes in FPA and CAS cause changes in the throttle settings, as well as elevator movement. For FPA changes there is initial elevator movement to pitch the aircraft to the new FPA, but, in the long term, FPA changes cause primarily throttle changes. For CAS changes, there is both long term throttle and elevator changes. The elevator change is primarily a re-trimming movement. There is no direct control over the throttle settings, only airspeed.

Flaps and gear commands are the simplest to use and understand. They produce a corresponding direct action on the aircraft when used. The AUTOLAND button activates a glideslope error numerical control law (algorithm) for automatic control of the pitch axis in landing the aircraft. Autoland provides two functions: it automatically tracks the glide slope down to the runway and it flares the aircraft just before touch down. Once the AUTOLAND button is pressed, the system is armed ("ARMED" appears on the GUI). The aircraft then is flown to the 3 degree glide slope within a tolerance of +/- 0.25 degree of deviation as indicated on the PDI gauge. Within this tolerance the autoland is engaged and will guide the aircraft down the glide slope to the runway ("ENGAGED" appears on the GUI). Just before touch down, the aircraft flares, increasing the angle of attack, thus providing larger amounts of aerodynamic lift at slower airspeeds, necessary for landing.

A typical scene from a high altitude cruising state is shown in Figure 10. Note that the PDI gauge and MARKER window are not needed and lie dormant. The simulation contains many complex models for almost all normal flying maneuvers. However, in the event the pilot flies the aircraft into a state where no appropriate aerodynamic models exist, the GUI will display the words "MODEL OUT" between the FPA and IAS controls. For example, flying out of model could constitute deploying the landing gear while cruising at 25,000 feet or flying too slowly without the correct flaps setting. The simulation will still run, but unpredictable results may occur. The pilot should try to restore a more "normal" flight regime.

With the meta-controller in Advise Only mode, it is the responsibility of the pilot to make an approach and landing. A good approach consists of reducing airspeed and altitude to around 180 knots and 3000 feet to intercept



MACH .76
TAS 458
ALT 33
FPA 0.0
DME 170
TIME 263

FPA Prev 0.0 Commanded 0.0
IAS Prev 320 Commanded 320
Status UP
Deg. 0

MC: Level
Advise Only

Interp: Flight Ops. Mode
Cruise

Interp: Dynamic Mode
Level

Interp: Messages
Confidence: 91

MC: Clearance Menu
Select One:
Input new clearance
Update approach plan

MC: Approach Plan
1)
2)
3)
4)
5)
6)

Figure 10. Cruising Scenario.

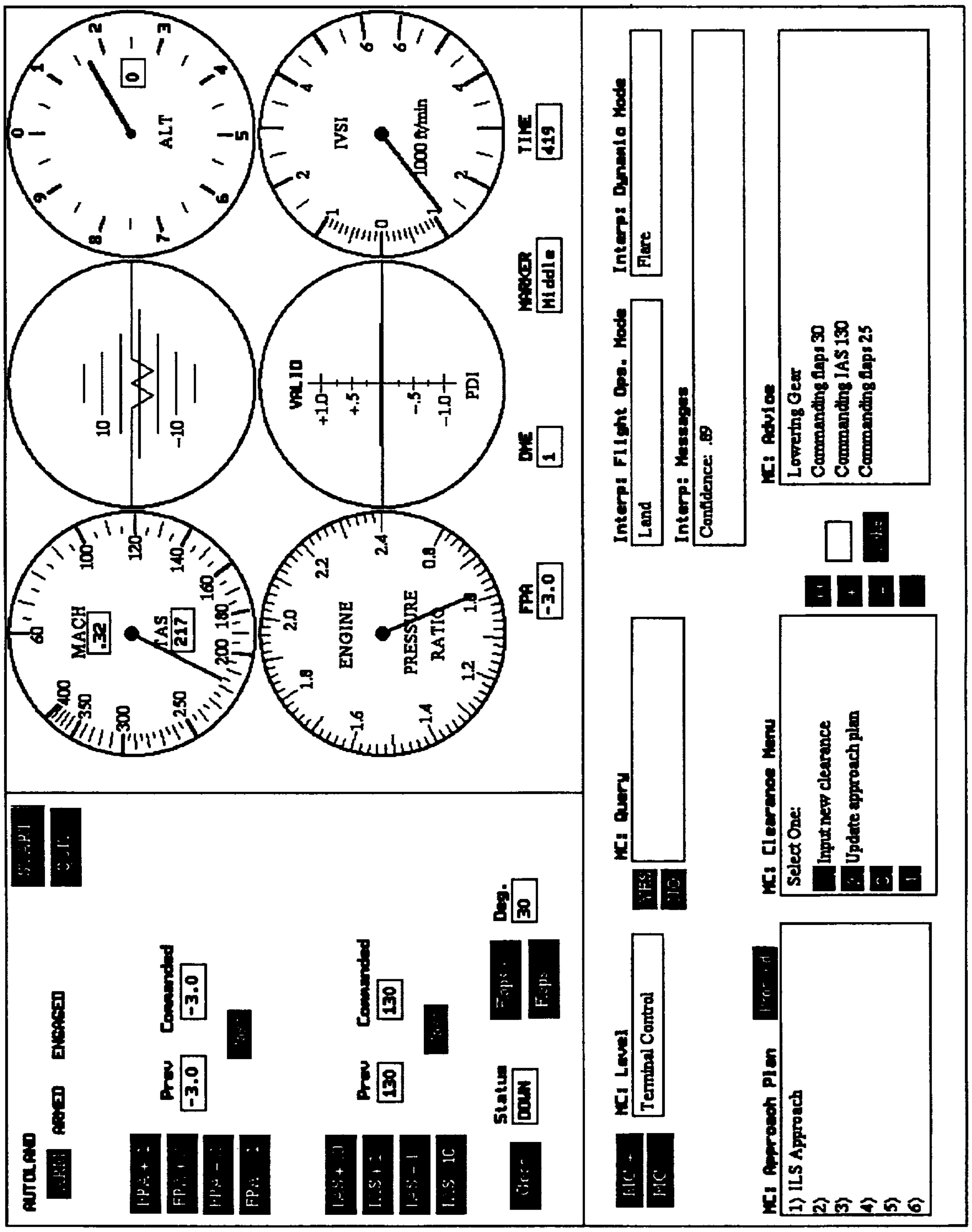


Figure 11. Landing Scenario.

the glide slope with the autoland armed. Thus, the autoland will be engaged and make the necessary adjustments to the FPA to track the glide slope. The pilot is then left to incrementally reduce airspeed and add degrees of flaps to end with a landing IAS of 120 knots and 30 or 40 degrees of flaps. The landing gear should be deployed around an altitude of 500 feet. Figure 11 shows a scene from a landing sequence; note the marker window is activated.

CONCLUSIONS

PROGRAMMING PERSPECTIVES.

The previous sections described an application of knowledge-based control. With an application to transport aviation and a medium of object-oriented programming, this research straddles the disciplines of electrical engineering and software engineering. The electrical engineering aspects encompass real-time, advanced control systems and interfaces. The scope of the software engineering properties includes object-oriented programming, system planning, and co-development of differing subsystems. Knowledge-based control is the marriage of two disciplines. There are no tools or techniques within the domain of electrical or software engineering that are off limits in the pursuit of future applications.

It is becoming apparent that a critical mass of interest is building up around object-oriented programming. Individuals in the software community foresee it as the new tool for the future. While now a rallying point, proper object-oriented techniques are in danger of losing focus. C++ is currently the most popular object-oriented language in use. It offers the programmer an environment in which C and C++ co-exist, thus reducing the culture shock of a pure object-oriented language. Therefore, a C programmer stands on familiar ground when exploring the domain of object-oriented programming.

The danger lies in the fact that C++ may be fool's gold. C++ delivers an environment with modular design themes, but it lacks the mandatory compliance to object-oriented communication policies that are found in Eiffel[10]. It is possible to produce quality software in the C++ environment. However, undisciplined programmers are then capable of producing unreliable software with hidden "time-bombs." This would effectively weaken the trust required to propagate a library of reusable classes[11]. In the fall of 1991, AT&T's long distance network crashed in the New York area; the problem was traced to one single line of code in the control software. The repercussions of software reliability will become more evident in the future as more of the infrastructures we depend on fall under software control.

Eiffel does offer a haven for software production. While not promising the moon, it can be successfully defended in the software arena as a viable choice. The future of Eiffel has been unclear to this point. International popularity and ISE's ability to produce improved versions continue to push the language along, grabbing the attention of software experts along the way. Eiffel will always remain a complex software entity. To those who can tackle the aspects of data abstraction and foresee long-term software needs, this investigator believes Eiffel will be the best language to use.

HINDSIGHT.

Given an opportunity to do things differently or to rework the project, I would like to have used a different version of Eiffel, if possible. Version 3.0 is reported to contain many improvements over the current version 2.3. The press has reported that 50% of the changes to version 3.0 are within the graphics libraries. The compiler was highly modified to improve development time. Also, the small bugs in the libraries that have given Eiffel the reputation of a language with teething problems have been fixed[12].

Given another chance, I would have spent more time practicing and perfecting object-oriented techniques before I began programming a complex system as large as the GUI. Major decisions about the program architecture were made at the beginning of the project. Without a sound understanding of object-oriented techniques, I found myself restarting the project several times to accomplish my goals and to streamline intermodule communication.

GUI CRITIQUE.

The GUI performs all the goals and necessary functions demanded by the knowledge-based controller. There still exist criticisms in the design of the program and in the operational performance. One could criticize the fact that there is no view displayed of a far-off runway as if looking through a cockpit window. Although this was not a goal or request of the GUI, a simplistic graphical representation of a runway could be calculated and displayed. However, such a processing algorithm would certainly have robbed valuable CPU attention from a system already highly taxed.

The GUI relays almost no navigational information with the exception of the DME display and navigational clearance data. There are no computerized map displays or inertial navigating system interfaces. Such systems are found in present aircraft and would be a research project beneficial to future aircraft.

CLOSING COMMENTS.

This research was part of an overall attempt to obtain an increased understanding of knowledge-based control. In this process, object-oriented software techniques and tools were explored as a means to improve and implement applications of the theory. This research was also prompted by the desire for "hands-on" experimentation with a pure object-oriented language like Eiffel. Because the applications of knowledge-based control have become so complex, there now is a requirement for sophisticated interfaces. All of these predetermined requirements and desires were fulfilled in the development of the GUI.

CHAPTER-6 BIBLIOGRAPHY

1. L. Bass and J. Coutaz, *Developing Software for the User Interface*, Addison-Wesley Pub. Co. Inc., New York, 1991, pp. 26-34.

2. *Private Pilot Manual*, Jeppesen Sanderson, Inc., Englewood, CA., 1991, pp. 25-27.

3. Op. Cit., L. Bass and J. Coutaz, pp. 2-3.

4. A. von Mayrhauser, *Software Engineering, Methods and Management,* Academic Press, Inc., San Diego, CA., 1990, p. 89.

5. J. Rumbaugh, *Object-Oriented Modeling and Design,* Prentice Hall, Englewood Cliffs, NJ, 1991, pp. 2-3.

6. E. Fiume, "Active Objects in the Construction of Graphical User Interfaces," in *Computers and Graphics*, Vol. 13, No. 3, 1989, p. 1.

7. B. Meyer, *Eiffel: The Language,* Interactive Software Engineering, Goleta, CA, 1989, p. 226.

8. R. Howard, "Eiffel FAQ," in *Internet News Group*, 1992, pp. 2-3.

9. Op. Cit., Meyer, pp. 3-23.

10. K. Ring, "Eiffel is the EC's Object-Oriented Business language of choice, Should It Be Yours," in *Software Futures,* APT Data Services Ltd., London, UK, Mar. 1992, pp. 5-6.

11. I. Joyner, "A C++ Critique," in *Internet News Group*, Apr., 1992, pp. 3-4.

12. H. Rock, *Eiffel Outlook,* Rock Solid Software, Austin, TX, Apr., 1991, pp. 4-5.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE October 1994	3. REPORT TYPE AND DATES COVERED Contractor Report

4. TITLE AND SUBTITLE Knowledge-Based Processing for Aircraft Flight Control	5. FUNDING NUMBERS G NAG1-1066 WU 505-64-52-01
6. AUTHOR(S) John H. Painter, Emily Glass, Gregory Economides, and Paul Russell	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Texas A&M University Dept. of Electrical Engineering College Station, TX 77843-3128	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23681-0001	10. SPONSORING / MONITORING AGENCY REPORT NUMBER NASA CR-194976

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Richard M. Hueschen

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unclassified - Unlimited Subject Category 63	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This Contractor Report documents research in Intelligent Control using knowledge-based processing in a manner dual to methods found in the classic stochastic decision, estimation, and control discipline. Such knowledge-based control has also been called Declarative, and Hybid. Software architectures were sought, employing the parallelism inherent in modern object-oriented modeling and programming. The viewpoint adopted was that Intelligent Control employs a class of domain-specific software architectures having features common over a broad variety of implementations, such as management of aircraft flight, power distribution, etc. As much attention was paid to software engineering issues as to artificial intelligence and control issues. This research considered that particular processing methods from the stochastic and knowledge-based worlds are duals, that is, similar in a broad context. These dualities are especially useful in implementations which naturally fall into the fuzzy control category. They provide architectural design concepts which serve as bridges between the disparate disciplines of decision, estimation, control and artificial intelligence. This research was applied to the control of a subsonic transport aircraft in the airport terminal area.

14. SUBJECT TERMS Knowledge-Based Controls, Intelligent Control, Blackboards, Fuzzy Control, Object-Oriented Programming, Graphic User Interface	15. NUMBER OF PAGES 96
	16. PRICE CODE A05

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

a-2.