AN UNMANNED AERIAL VEHICLE CONTROLLER BASED ON A LEARNING FUZZY CLASSIFIER SYSTEM

by

YAN QU

(Under the Direction of Walter D. Potter)

ABSTRACT

Autonomous Unmanned Aerial Vehicle (UAVs) have been increasingly employed by researchers, commercial organizations and the military to perform a variety of missions. This thesis discusses the design of an autonomous controller using a Learning Fuzzy Classifier System (LFCS) to store and evolve fuzzy rules and fuzzy membership functions. The controller executes the fuzzy inference process and assigns credit to the population during a flight simulation. This framework is useful in evolving a sophisticated set of rules for the controller of a UAV, which deals with uncertainty in both its internal state and external environment. A flight simulation is implemented in Matlab/Simulink providing the opportunity to assess the accuracy of the control rules. The simulation results show that this approach is able to develop a controller that achieves high effectiveness in both lateral and longitudinal control.

INDEX WORDS: Unmanned Aerial Vehicle, Learning Fuzzy Classifier System, Fuzzy Control

AN UNMANNED AERIAL VEHICLE CONTROLLER BASED ON A LEARNING FUZZY CLASSIFIER SYSTEM

by

YAN QU

B.Eng., The University of Electronic Science and Technology of China, China, 2005

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2011

AN UNMANNED AERIAL VEHICLE CONTROLLER BASED ON A LEARNING FUZZY CLASSIFIER SYSTEM

by

YAN QU

DEDICATION

To my parents in China and my wife for their love and support.

ACKNOWLEDGEMENTS

I would like thank my major professor Dr.Potter for his support and instruction during my study at the Institute for Artificial Intelligence at the University of Georgia. I also want to thank my committee members: Dr.Rasheed, who is one of the greatest professors I have ever met, as well as Dr.Bhandarkar for his cooperation during my thesis work.

I would like to thank my parents for their encouragement and trust in me. Last but not least, I want to thank my lovely wife since nothing can happen without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 UAV OVERVIEW

Unmanned Aerial Vehicles (UAVs) are defined as powered aerial vehicles which do not require an on-board pilot to operate but are rather controlled autonomously or remotely. A typical UAV system is comprised of three major components: the aircraft, the ground control station and the operator. Since UAVs can fly without a human pilot on board, they are helpful in missions that do not necessarily need a human's direct oversight.

Based on UAVs' capabilities, they were first adopted for military and intelligence missions including deception operations, route and landing reconnaissance, and battle damage assessment. Recently, a large number of UAV applications have also emerged in civil markets. In [1], a camera-equipped mini UAV is used to support wilderness search and rescue. In [2], a large number of swarming UAVs are organized to establish an airborne communication relay. An application of UAV cooperative control is seen in [3], where multiple UAVs are collaborating for map building tasks and in [4] several mini UAVs function as a single unit for surveillance.

As a UAV can be autonomously controlled, a powerful controller plays a crucial role in a UAV's development. One of the traditional methods of designing a UAV controller is based on the proportional-integral-derivative (PID) control theory which tunes out an "error" value between a measured state and a desired state of the UAV by adjusting its throttle and control surfaces such as ailerons, elevator, and rudder. Three separate parameters define the PID controller calculation: the proportional value P, the integral value I, and the derivative value D.

These values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D as a prediction of future errors. This method shows descent performance in short term control by gradually damping the control of the UAV. But a PID controller is only a reactive system which depends on feedback and pre-defined constant parameters and thus it has no direct knowledge of the control process.

1.2 FUZZY LOGIC AND FUZZY CONTROL

The theory of fuzzy sets [5] extends standard set theory by defining a Boolean membership value of a real value. A fuzzy value could be a linguistic term such as "small" or "big" which imitates the non-precise approach to describe conditions used in everyday life. A membership function u(x) represents a curve that takes a value x and maps it to a "membership degree" falling in the interval [0,1] of a fuzzy value. Fuzzy sets help depict classes that people find hard to describe with real values. It is also possible to define partially overlapping classes that generate a gradual classification of elements.

Fuzzy Control is a control strategy that uses fuzzy sets to define its input/output variables, instead of a mathematical description. A fuzzy controller consists of three stages: an input stage, a process stage, and an output stage. In the input stage, inputs from sensors or other devices are mapped into the proper fuzzy membership degrees. Most membership functions usually have triangular shapes, although trapezoidal and other bell curves, such as Gaussian or Sigmoidal curves, are applicable as well, to cover the input range, or the "universe of discourse". In the process stage, the fuzzy inference is executed based on a collection of fuzzy rules whose antecedent and consequent in an IF-THEN template are in linguistic terms, which allows imprecise information to be used in inference. A fuzzy rule can have several antecedents that are joined together using fuzzy operators like "AND", "OR", or "NOT". In the most common definition of these operators, AND simply implies the minimum fuzzy membership value of all the antecedents, OR uses the maximum value and NOT subtracts the membership value from 1 to give the complementary value. There are different ways to determine the result of a rule [6] such as the "max-min" inference which outputs the fuzzy degree by the given premise. Finally, in the output stage, results of all the triggered rules are "defuzzified" to a crisp value by available methods such as centroid area, bisector or mean of maximum [7]. Fuzzy Control has shown great success in many areas such as vehicle positioning [8] and mobile robotics [9].

1.3 EVOLUTIONARY ALGORITHM IN CONTROL ENGINEERING

1.3.1 PROBLEM DOMAIN

The automatic control problem first appeared over two thousand years ago. Control theory made significant progress after modern mathematical techniques were invented, especially in complex dynamic systems. These techniques mainly include optimal control, stochastic control and adaptive control theories where most analysis is executed in a real time domain using differential equations. Evolutionary Algorithm (EA) methods are global, parallel, search and optimization methods originated from the concept of natural selection and genetics. At present, many researchers are applying EA methods [10], as in computer automated design, to control problems where previous methodologies failed to show adequate accomplishment. These problems are usually poorly understood and mathematically difficult to formalize involving multiple performance criteria or non-measurable variables.

1.3.2 GENETIC ALGORITHM

The genetic algorithm is one of the most successful techniques among Evolutionary Algorithms. Its flow chart is shown below in Figure 1. It simulates the search process of natural evolution in a computer system, and is especially suitable for searching irregular spaces. Inspired by the underlying principle of the genetic process of evolution, a set of candidate solutions to a given optimization problem is considered as a population in a generation being evolved. Each chromosome has a fitness value assigned to represent its optimality in the environment. A genetic algorithm then applies several genetic operators like selection, crossover and mutation to evolve the current population to the next generation until an optimal solution is reached.

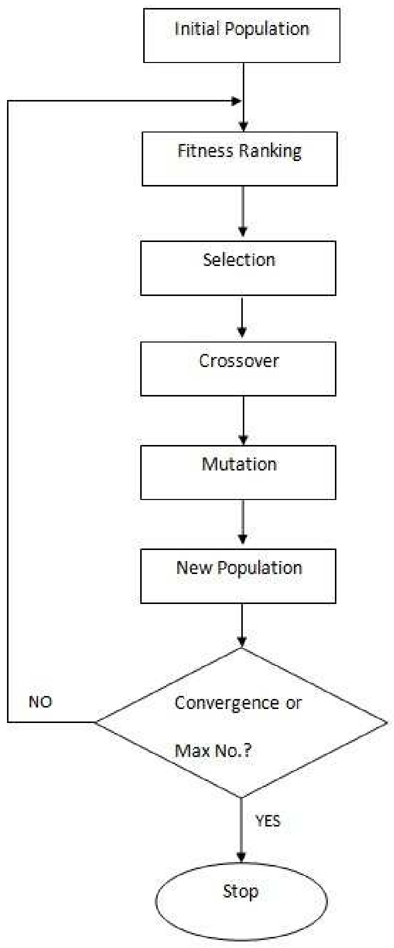


Figure 1. Flow Chart for a Genetic Algorithm

Now, let us briefly discuss several main operators used in genetic algorithms. Selection operators choose the parental chromosomes from the current population based on their fitness to produce the next generation. There are several strategies on choosing the most fitting chromosomes to mate. For instance, the standard roulette wheel selection operator picks each parental chromosome stochastically. But individuals with a higher fitness value will have a higher share of the wheel and therefore a better chance to be selected. This process assumes that stronger individuals in this generation will have a higher probability of survival. This selection method is biased since for most times genes contained in chromosomes with higher fitness are favored and might spread across a population over time, which leads to a phenomenon called genetic drift. There are also other selection operators such as tournament selection [11] that chooses the chromosome with better fitness in a random tournament as well as stochastic universal sampling [12] which uses a single random value to sample all of the solutions by choosing them at evenly spaced intervals.

Crossover, or combination, is an operator that exchanges random segments of the parental chromosomes. The motivation of this operator is to utilize potential sequences of genes in parental chromosomes. Below is an example of a one-point crossover on two binary chromosomes:

Parent1	***10001 | 010101***
Parent2	***00101 | 111010***
Child1	***10001 | 111010***
Child2	***00101 | 010101***

Figure 2. A crossover sample

Mutation is a genetic operator that allows one or more gene values to be altered from their initial values. By this means, a new gene value can be introduced into the chromosome, thus mutation can effectively prevent the search process from falling to local optima. The user sets a mutation rate defining the probability of mutation during evolution. A variety of customized mutations can be applied to solve a specific problem. The most basic approach is Bit-flip mutation that simply inverts the binary allele value. Below is a one-point bit-flip mutation example:

Parent1 *10001* ***0*** *10101*

Child1 *10001* ***1*** *10101*

Figure 3. A bit-flip mutation sample

Besides operators, there is the fitness function, or objective function, that generates a numerical value indicating the quality of a solution in the environment. The genetic algorithm attempts to maximize the fitness values of an entire population. This fitness is also the measure of solution quality used in replacing an old chromosome in the population with a new one.

Each genetic algorithm has its own configuration, namely chromosome representation, genetic operators and fitness function. Genetic algorithms may perform very differently under different situations so an appropriately configured GA is the key solving any particular search problem.

1.3.3 LEARNING CLASSIFIER SYSTEM

A Learning Classifier System (LCS) is a kind of rule-based system that adopts a genetic algorithm to search the space of possible rules and a reinforcement learning technique to assign utility to existing rules. Classifiers, or "condition-action rules", encode their population of rules as bit strings and evolve them intermittently based on the stimuli and reinforcement from its environment.

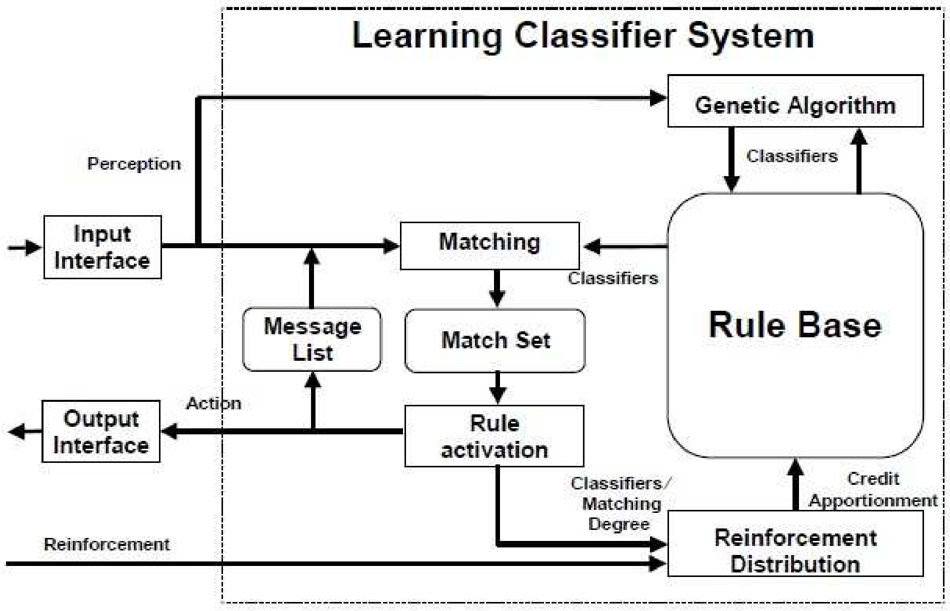


Figure 4. A generic architecture of a Learning Classifier System

A classifier system prototype as shown in Figure 4 consists of several components:

- An interface between the internal states and external environment that enables the classifier system to detect information from outside and impact the environment;

- A message system that is responsible for receiving input and transmitting output as well as maintaining an internal message list for rule matching;

- A rule system that stores the population of classifiers;

- A credit assignment mechanism that distributes credits or penalties according to the result of the last action;
- A genetic procedure that refines classifiers.

Two types of standard LCSs are defined on the basis of their rule base representation styles [13]. The Michigan approach encodes each rule and assigns each rule a bid. It activates classifiers through bidding, and the bucket brigade algorithm can update classifiers' bids. However, in the Pittsburgh approach, the entire rule base is represented in one string and utilizes global genetic operations on the string.

1.3.4 EA APPLICATIONS IN CONTROL ENGINEERING

A wide range of control engineering problems can be solved after a process of tuning EA parameters to the specific domain problem. Compared to conventional control methods, the lack of dependency on domain specific heuristics makes EA methods attractive as EA methods are universally robust when applied to a majority of control problems. Therefore, EAs can typically outperform conventional methods in non-linear and stochastic systems, which are difficult to formalize.

Most successful applications of EAs in control engineering appeared in off-line design or training. These applications cover controller design, robotic navigation, collision avoidance, path-planning, and fault diagnosis [14].

One promising field is that of using EAs to develop an intelligent controller. Off-line training evolves a controller's rule base to fit a certain application field. Moreover, a genetic controller may also include features like fuzzy logic or neural network to form an intelligent control mechanism. An application in robotic motion control [15] shows a highly evolved knowledge base under a messy genetic algorithm scheme. Some evolved controllers employ both binary and integral representations and display great performance in a truck backing system [16] and a ship rudder controller [17].

1.4 PROJECT GOALS AND CHALLENGES

In this thesis, we attempt to use a Learning Fuzzy Classifier System (LFCS) controller to replace the traditional PID tuning method as an "autopilot" for a UAV since the PID controller has certain limits as stated previously. Research [18] also shows that a fine tuned PID controller gives small overshoot in the target error and needs longer settling time while a fuzzy controller provides no overshoot and smaller settling time.

A Learning Fuzzy Classifier System is a Learning Classifier System whose symbols in the rule clauses are associated with fuzzy sets so that it is possible to build a fuzzy logic knowledge base. This LFCS employs a knowledge-based controller with a genetic algorithm to evolve its rules. A learning mechanism for mapping fuzzy input and output as the membership function is necessary in this LFCS as well. This could be viewed as a sort of structural learning of the membership shape generally defined by a center position of the membership function and its coverage. Therefore, the genetic algorithm in our controller is customized for the UAV control problem so that it can evolve fuzzy rules and fuzzy membership functions in the knowledge base simultaneously.

To conduct a simulation-based analysis on the UAV controller, an aerodynamics model and a controller model are built in Matlab/Simulink as discussed in the following sections. A particular visualization module is also adopted to show the entire training process and operation quality in animation with a flight simulation platform called FlightGear.

Several major challenges have to be tackled during the controller development. The first challenge is designing the aerodynamics package which accurately follows the UAV's characteristics in the real world and facilitates the control flow. Chapter 2 will show the technical details of this package.

The second challenge is developing the control strategy since a simple flying maneuver might engage a variety of information on aircraft status, action planning and decision making. An efficient and robust control flow needs to be set up so that an action could be generated based on correct and sufficient information.

The third challenge deals with an appropriate genetic algorithm configuration. A genetic algorithm will perform satisfactorily only if genetic operators and parameters are correctly designed and configured for this problem domain. In addition, the genetic algorithm has certain limits dealing with real-time control, as it needs additional time to evolve rules during the control process. How to combine a genetic algorithm into solving a real-time control problem is also a crucial concern.

The last challenge is developing and tuning the autopilot in a simulation. This task was accomplished by modulization of each control unit for lateral and longitudinal control. The control units were developed and evolved independently from each other for the basic scenarios first and then integrated together to function as a full autopilot.

CHAPTER 2

THE UNMANNED AERIAL VEHICLE SYSTEM

2.1 FLIGHT CONTROL OVERVIEW

This section explains the flight control principles of a UAV that we employ to design a controller which adequately stabilizes and controls a UAV. To characterize the aerodynamics of the UAV is the first step we ought to achieve for designing the controller. Our aerodynamics is based on a mini-UAV Test platform that was adopted in [19] with a light mass and light wing loading airframe. The control of a UAV can be viewed as two dynamics: lateral dynamics and longitudinal dynamics as shown in Figure 5.

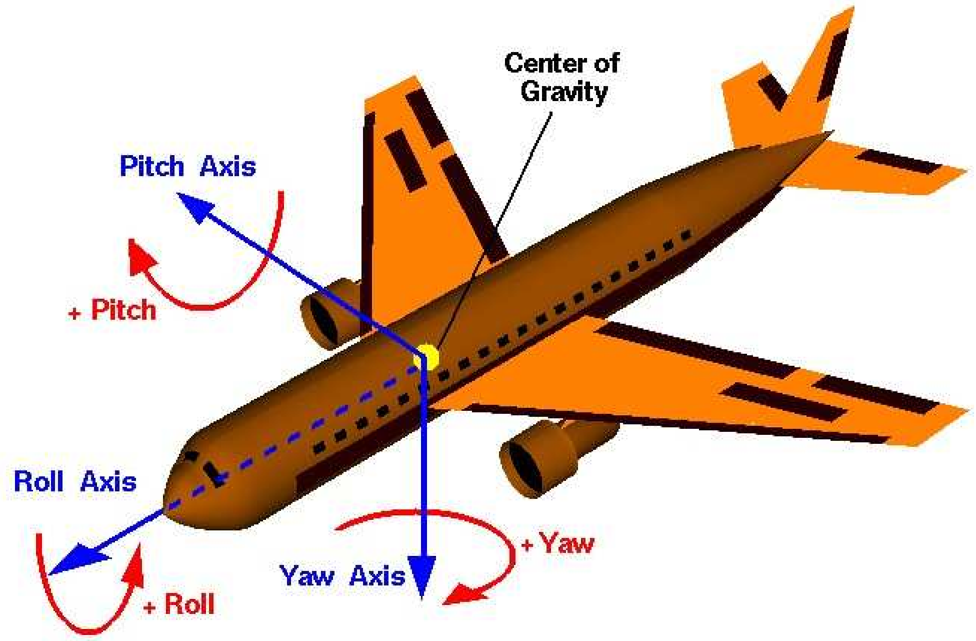


Figure 5. Aircraft flight dynamics

Lateral dynamic controls the UAV's roll and yaw axes. The roll axis is excited with ailerons and the yaw axis is excited with the rudder. Longitudinal dynamic controls the UAV's velocity and pitch angle, which respond to the throttle and the elevator respectively.

The two dynamics are independent as each of them generates responses in its own dimension. As for autonomous control of a UAV, the controller should be capable of adjusting the heading, altitude and airspeed smoothly. To accomplish this task, we will consider how control deflection is determined along each axis.

2.1.1 LATERAL CONTROL

In the lateral control mode, the aileron value is determined based on a UAV’s roll error (desired roll - current roll), heading error (desired heading - current heading), and current roll rate which indicates how fast the UAV is rotating on the roll axis. To obtain this information, a desired roll can be generated based on desired heading. Roll error can be calculated by subtracting desired roll from the current roll. Roll rate is measured by a roll gyro.

2.1.2 LONGITUDINAL CONTROL

In the longitudinal control mode, the throttle controls the air speed and pitch angle is controlled by the elevator. An adjustment on the elevator should account for the UAV’s pitch error (desired pitch - current pitch), altitude error (desired altitude - current altitude), and current pitch rate which is measured by a rate gyro.

For the speed change, the throttle deflection is based on current speed error.

2.1.3 PID CONTROL METHOD

The proportional-integral-derivative (PID) Control is one of the traditional methods of controlling a UAV as described below. The control gain Gc at time s is determined by the proportional gain Kp, the integral action time Ti and Td as the derivative action time.

$$\mathrm{Gc(s)} = \mathrm{Kp}\ (1 + \frac{1}{\mathrm{Ti} * \mathrm{S}} + \mathrm{Td} * \mathrm{s})$$

A PID controller depends on an input representing the error between the desired state and the current measured state. The PID controller is applicable for the majority of the control system and provides a robust and reliable performance if the parameters are fine-tuned. However, there are several shortcomings of a PID controller. A PID controller is an error adjustment method, therefore before the parameters are perfectly tuned, it might cause constant overshoot and long settling time until the error approaches to zero. Another weakness of PID control is that it is linear and the performance of PID controllers in non-linear systems is undecided [20]. Finally yet importantly, the PID controller is a purely mathematical tuning device that lacks a direct knowledge of the control process, and therefore we want to replace it with an artificial intelligence controller.

2.2 SIMULATION OF THE UAV MODEL IN SIMULINK

The development of a Learning Fuzzy Classifier System is accomplished in a simulation environment since we cannot afford possible damage to real UAV hardware in the real world. Besides, a simulation provides us a platform to test and tune the algorithm and observe the learning process of the UAV controller under different scenarios.

Matlab/Simulink is a popular scientific computing software environment for multipurpose simulation. Engineers can design, implement and test a customized set of models. Inside each model, functionalities are provided by a set of blocks which are function units with graphic appearance in the model-based simulation, including mathematical formula, control signals, and video processing. We used Matlab/Simulink and its aerospace toolbox, which collects numerous practical pre-defined blocks in aircraft simulation, aerospace standard, and propulsion systems, to construct the aerodynamic, controller and visualization model for the UAV. It also allows us to define self-customized blocks with either customized mathematical expressions or embedded code.

The original model of the UAV shown in Figure 6 was implemented in Matlab/Simulink by several former graduate students from the Institute for Artificial Intelligence at the University of Georgia [21]. The simulation contains three major models as their inner blocks described below:

1. AirFrame model
    a. Air frame: characterizes the UAV aerodynamics and feedback from environment
    b. Actuator: models actuators and output control signals to environment
2. Controller model
    a. Lateral Control: control ailerons of the UAV
    b. Longitudinal Control: control throttle and elevator of the UAV
3. Visualization model
    a. FlightGear: provide 3D animation of the test flight
    b. Scopes: display the UAV status data and simulation parameters in real time

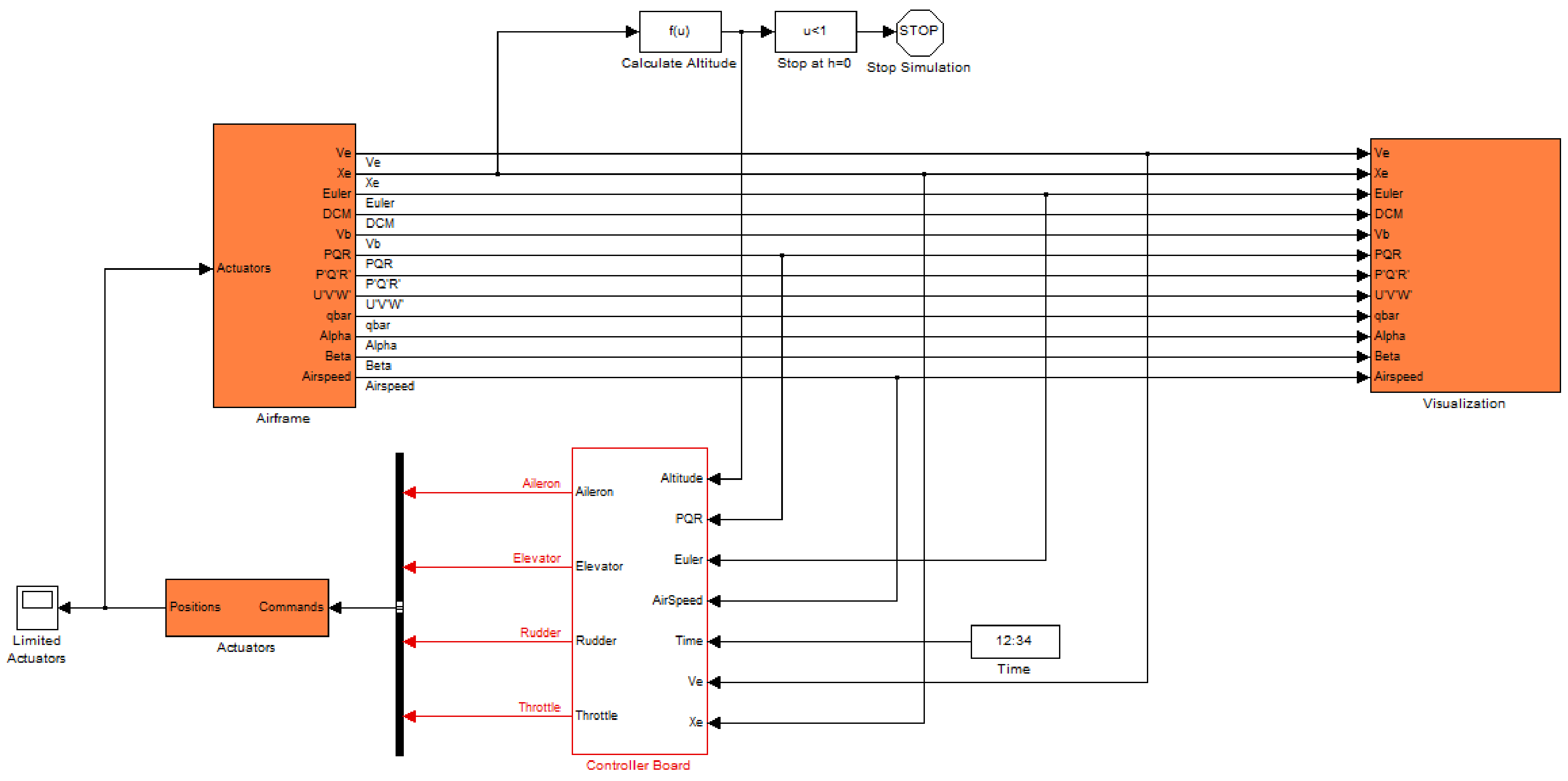


Figure 6. The UAV Model

2.2.1 AIRFRAME MODEL

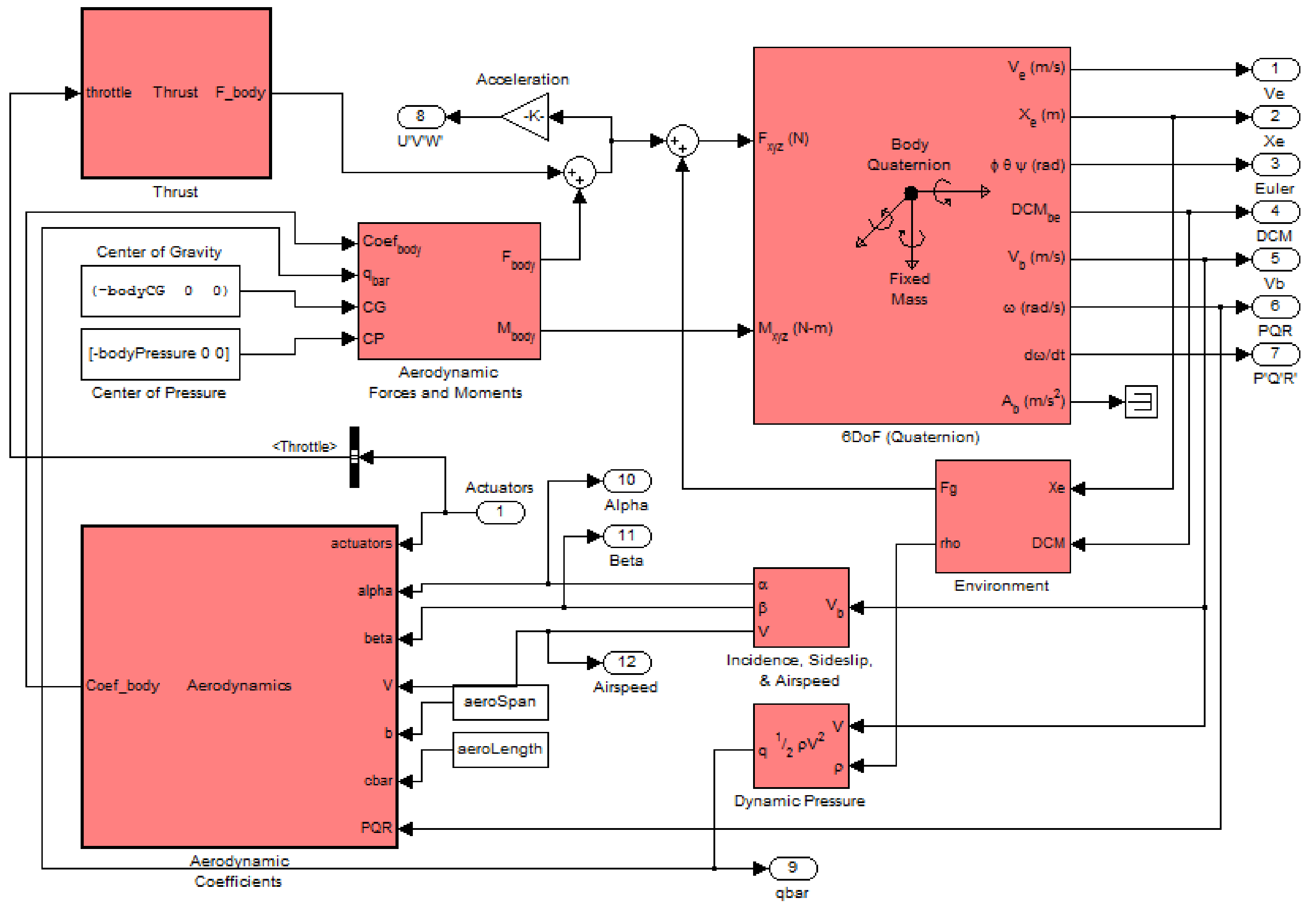


Figure 7. The AirFrame Model

The AirFrame model as shown above in Figure 7 characterizes the aerodynamics of the UAV using a 6 degree-of-freedom (6DOF) Matlab simulation. Actuators are modeled to as Second Order Nonlinear Actuators in Figure 8 that output actual actuator positions based on the input demanded actuator position and actuator deflection or rate limits.

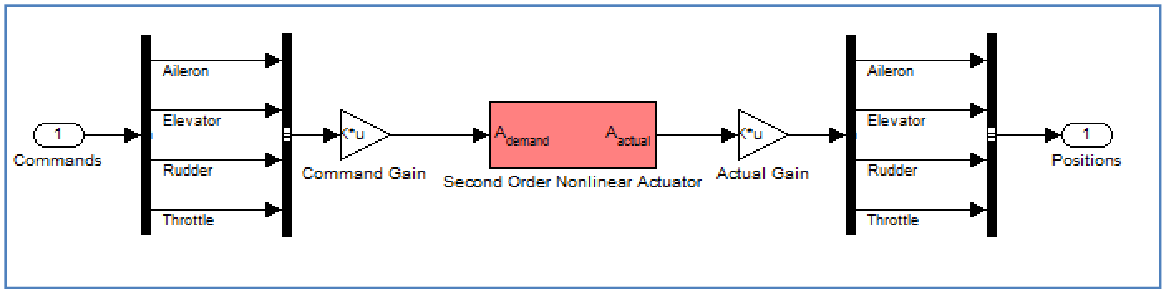


Figure 8. The Actuators block

A sophisticated set of equations stored in the Aerodynamics block is employed to calculate various rigid-body parameters in modeling the 6DOF flight dynamics of the UAV. The most accurate rigid-body dynamics equations available have been adopted at the expense of increased computational complexity, but this ensures high accuracy of the aerodynamics modeling. The Aerodynamics block takes the actuator outputs and feeds the aerospace coefficients into the aerodynamics forces and momentum block to generate the resultant force and momentum on the aircraft's body. Finally, the 6DOF characteristics will be calculated using the 6DOF block, taking the resultant force and momentum in. This process of generating 6DOF data from Actuators is shown in Figure 9.

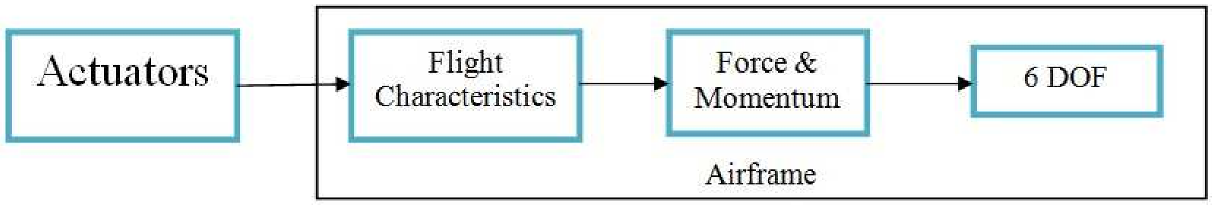


Figure 9. The 6 degree-of-freedom data flow

In the Environment block, there resides an International Standard Atmosphere (ISA) block computing atmospheric data from the current altitude using a lapse rate method and a gravity model generating gravity g value based on current altitude and GPS coordinates.

2.2.2 CONTROLLER MODEL

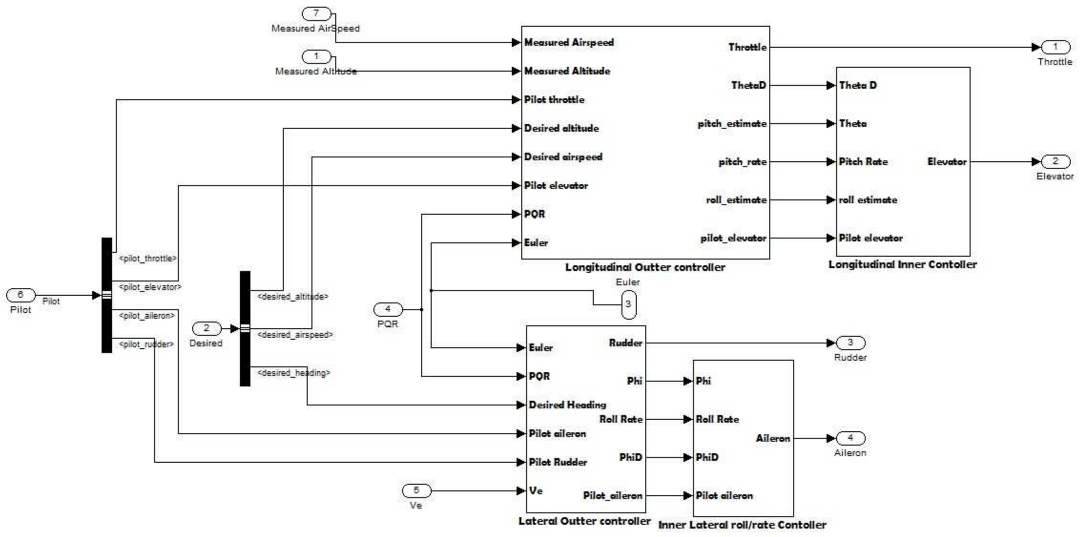


Figure 10. The Controller Model

The controller model is the unit implementing the flight planning, air navigation, and controlling of the UAV. Based on the incoming information regarding the UAV's status, the controller aligns deflections to control surfaces for the desired state described using three variables: desired altitude, desired airspeed, and desired heading.

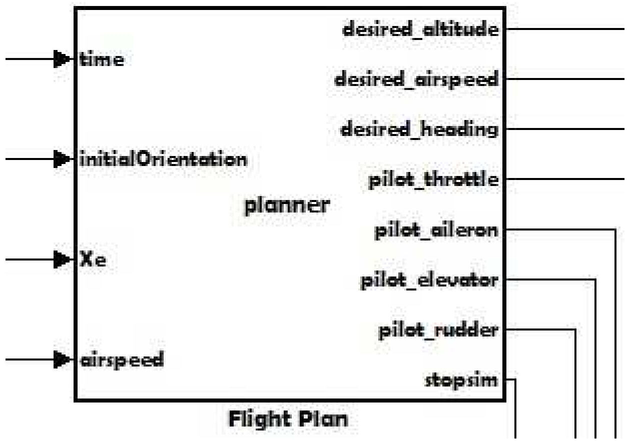


Figure 11.The Flight planner block

The first mission of the controller is deciding where the UAV should go next. Inside the Controller model, a FlightPlanner block as shown in Figure 11 is used to navigate the UAV's behavior based on a pre-defined flight plan. A designed flight plan specifies each waypoint with desired altitude, airspeed and heading and is input to the FlightPlanner block before the simulation starts. Once the simulation starts, the FlightPlanner parses every waypoint’s requirement and generates a sequence of desired states for the UAV. It also reads the current actuator positions, defined as pilot throttle, pilot aileron, pilot elevator, and pilot rudder, from the AirFrame model and transmits them to the Controller block.

The actual controller blocks we implemented in Simulink are divided into two independent sub controllers: lateral controller and longitudinal controller. This design follows the philosophy of independent control along each dimension and facilitates the UAV development process.

1. Lateral Controller

As stated before, the lateral controller manages the rudder and ailerons of the UAV. It could be further divided into two layers as shown in Figure 10: an outer controller that calculates and wraps up status information needed before the decision making and an inner controller which consists of the actual controller function in terms of LFCS and a Saturation block. The Saturation block here is utilized to limit the actuator force/angular rate in its minimum/maximum range; otherwise, excessive oscillations may lead to degraded flight performance or control failure. Our UAV does not have a rudder so in this model it is a fixed fake value assuming the rudder is always at its neutral position.

2. Longitudinal Controller

The design of the longitudinal controller follows the same fashion of the lateral controller. The longitudinal controller is mainly in charge of the elevator control and the throttle control that is an independent adjustment based on the airspeed error only.

The Controller model produces control surface outputs and throttle change, after all the fuzzy inference, to the Actuators block and then the 6DOF block and Environment blocks update the 6DOF data. By this means, a closed control/feedback loop is formed among the AirFrame, Controller and Actuators models.

2.2.3 VISUALIZATION MODEL

During the simulation, a Visualization model as shown in Figure 12 presenting a 3D animation of the UAV enables us to better observe, build and troubleshoot the learning process. The Visualization model utilizes the Aerospace Toolbox in Simulink to generate scripts on a 6DOF of the UAV. Then we can drive the position and altitude script coded in longitude, latitude, altitude, roll, pitch and yaw values, to a flight simulation platform called FlightGear so that FlightGear can display the UAV's flying process in a 3D framework.

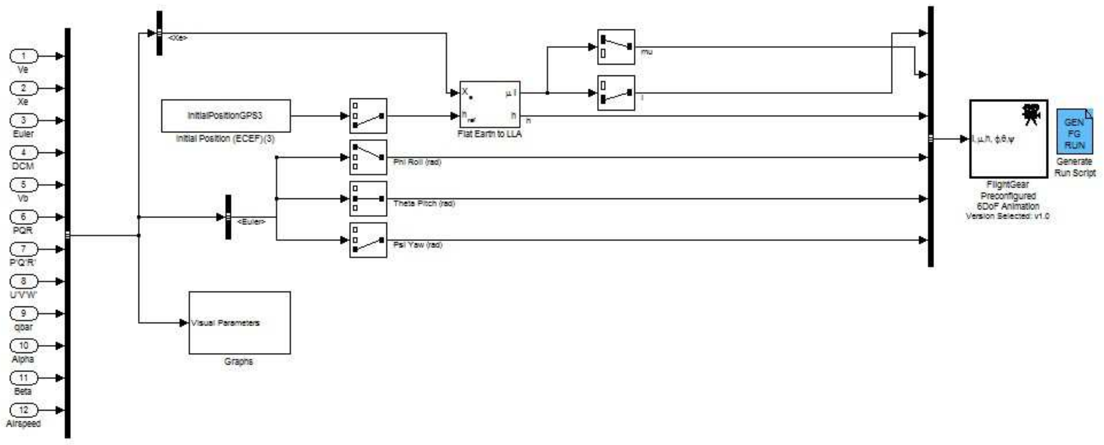


Figure 12. The Visualization model

CHAPTER 3

A UAV CONTROLLER BASED ON A LEARNING FUZZY CLASSIFIER SYSTEM

3.1 LEARNING FUZZY CLASSIFIER SYSTEM ARCHITECTURE

A learning fuzzy classifier system (LFCS) inherits the fundamental components of a learning classifier system including input/output interface, rule matching mechanism, message list, reinforcement, and genetic algorithm. However, as a LFCS seeks to learn fuzzy knowledge, it adopts additional components to evaluate fuzzy membership, compose fuzzy inputs and defuzzify output values. Figure 13 shows the architecture of a typical LFCS system and this chapter will discuss its key components in detail.

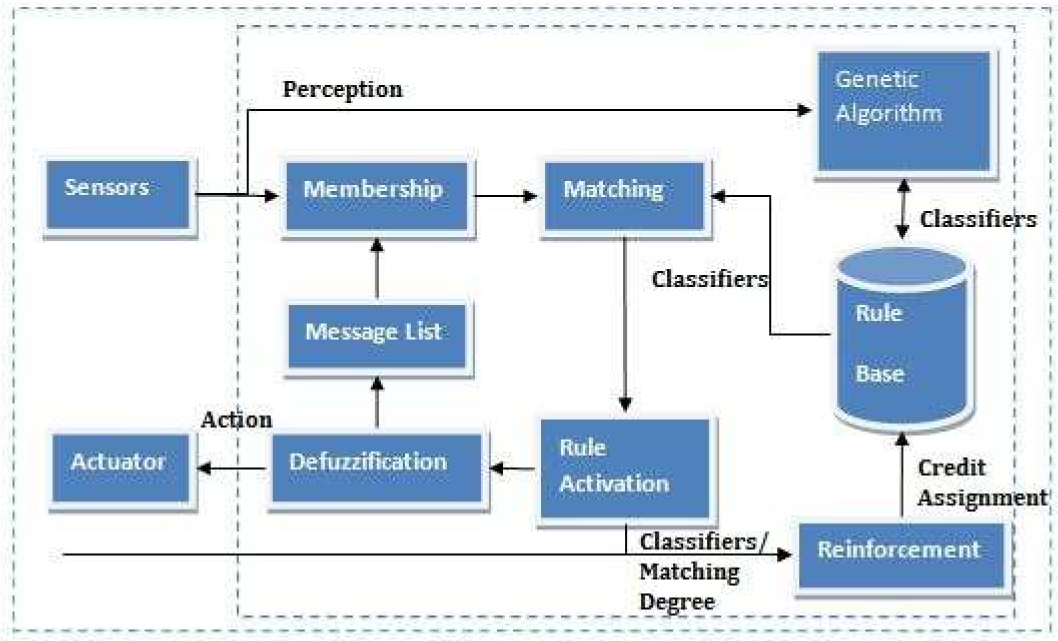


Figure 13. The UAV Learning Fuzzy Classifier System Controller Architecture

3.2 FUZZY RULES AND FUZZY MEMBERSHIP FUNCTIONS

Fuzzy rules in a LFCS follow the typical "IF-THEN" rule template but use linguistic values defined by fuzzy sets, such as "High" or "Slow", to measure its antecedent and consequent. Consider a fuzzy rule for a speed change in UAV control:

IF (Speed is "Slow") THEN (Throttle is "High")

In this fuzzy rule, it uses the input value of "Speed" which is a membership value of "Slow" to produce an output on "Throttle" which is a membership value of "High".

In our system, fuzzy rules are categorized into different rule families depending on which sub controller, either the lateral or longitudinal, will call them so that each rule family can be managed and trained independently in later development phases. Each rule family has a set of available variables as its antecedents or consequents. Below is a table of rule families and their antecedents/consequents.

Table 1. Rule family conditions and actions

Rule Family	**Antecedents**	**Consequents**
Lateral Control	Roll Error, Roll Rate, Heading Error	Aileron
Longitudinal Control	Pitch Error, Speed Error, Pitch Rate	Elevator, Throttle

For sensor input and control output values, the system uses fuzzy membership functions to evaluate their fuzzy membership degrees. Thus, a real input value is mapped into a fuzzy value representing one aspect of the UAV's states. As we stated previously, membership functions may have different shapes including triangular, bell-shape, trapezoid and so on. However, the selection of the membership curve is truly arbitrary to the designer and the only condition a membership function must really satisfy is that it must vary between 0 and 1. In our design, we employ a Gaussian curve membership function expressed below which is one of the most popular membership functions in robotic and ship control as it provides smooth transitions in membership value [22].

$$f(x;\sigma,c) = e^{\frac{-(x-c)^2}{2\sigma^2}}$$

In this Gaussian function, x is the input real value; σ defines the shape of the curve and is set at a constant 0.4 by default so that most points in the curve have a membership value larger than 0.3 which is an arbitrary threshold value for activating a fuzzy rule; c defines the center of the curve. Figure 14 is an example of a Gaussian membership function for the UAV's roll rate input variable ranging from -1.5 to 1.5 radian/second with parameter σ equal to 0.4 and c equal to 0.

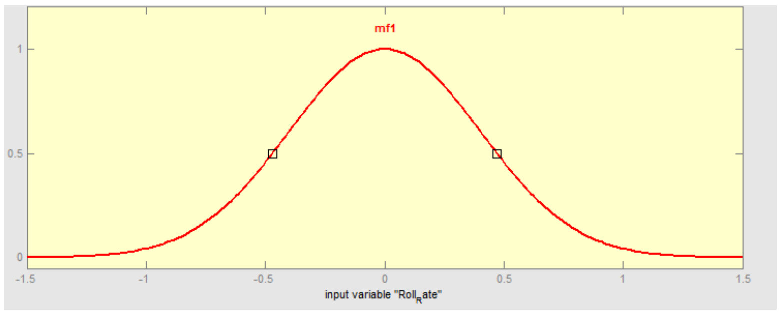


Figure 14. A sample Gaussian membership function[1]

For all input/output variables, we define their numerical ranges according to the UAV's aerodynamics model. Table 2 shows numerical ranges for each input/output variable and their respective units. Each fuzzy variable has seven possible fuzzy degree outputs:

[1] The y axis is the membership degree from 0 to 1.

"Negative Big", "Negative Medium", "Negative Small", "Zero", "Positive Small", "Positive Medium", "Positive Big" and initially we distribute the seven fuzzy degrees evenly over the range. In Figure 15, there is a sample of an evenly distributed fuzzy membership function for the roll rate variable.

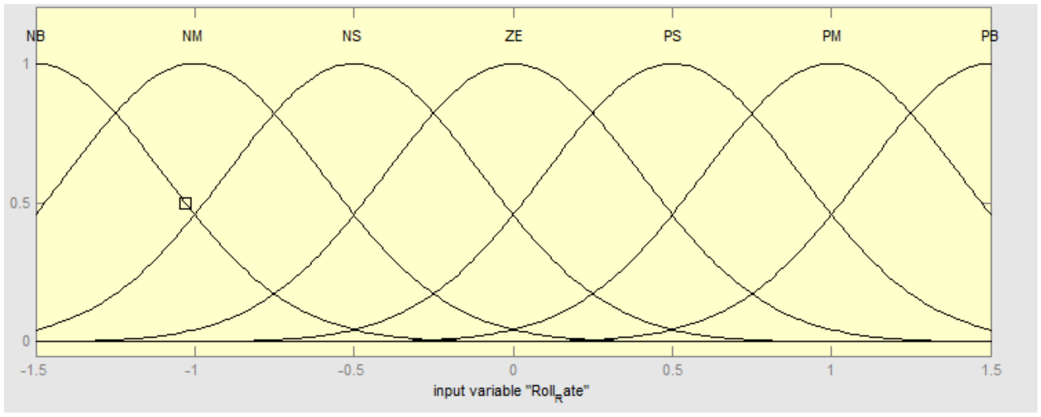


Figure 15. An evenly distributed membership function for roll rate[2]

Table 2. Input values and their ranges

Variable	Range	Unit
Roll rate	[-1.5, 1.5]	Radian/second
Roll error	[-3, 3]	Radian
Heading error	(-360, 360]	Radian
Speed error	[0, 180]	Knot
Pitch rate	[-2, 2]	Radian/second
Pitch error	[-90, 90]	Radian

[2] In this figure, NB = Negative Big, NM = Negative Medium, NS = Negative Small, ZE = Zero, PS = Positive Small, PM = Positive Medium, PB = Positive Big

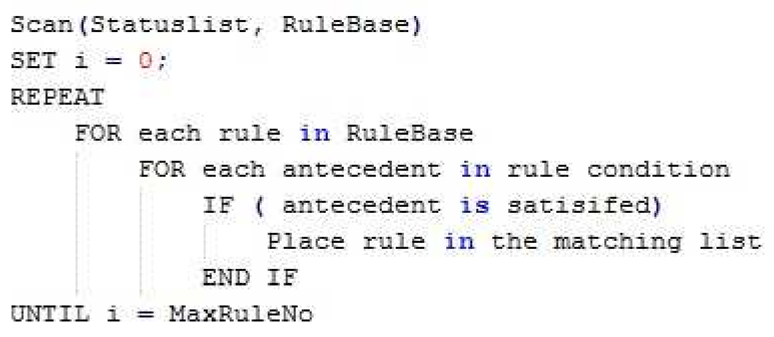
States with fuzzy degrees can be joined in the condition portion of an IF-THEN rule. The result of each successful conditional rule represents an assignment of desired amount of change in the actuators of the UAV from the current state in order to achieve the desired state of the UAV. A defuzzification function as stated later will then compose the fuzzy output into a crisp value and transmit it to actuators to perform the desired action.

3.3 RULE MATCHING

In the rule matching step, the system needs to determine classifiers to be triggered under the current situation. This process maintains a blackboard-like message list. When inputs are transmitted into the system, an internal message list is constructed denoting the current status of the UAV. In this list, each value represents an individual state variable of the UAV as shown below:

$$Status = [RollRate, RollError, HeadingError, \ldots., ThrottleError]$$

A scan algorithm as its pseudocode shown in Figure 16 will scan the entire rule base and determine if certain rule's antecedent is satisfied by the current inputs. Here, we assume that a rule is activated when its conditional implication result has a membership value larger than 0.3 determined by the min/max inference method. If successful, an activated rule will then be placed into the message list.



```
Scan(Statuslist, RuleBase)
SET i = 0;
REPEAT
    FOR each rule in RuleBase
        FOR each antecedent in rule condition
            IF ( antecedent is satisifed)
                Place rule in the matching list
            END IF
UNTIL i = MaxRuleNo
```

Figure 16. Pseudocode for Scan algorithm

3.4 RULE ACTIVATION

After the matching process, rules are selected based on current inputs and put into the message list. Fuzzy outputs from these activated fuzzy rules, need to be aggregated and converted into scalar output quantities so that they can be transmitted to actuators and align deflections on control surfaces. This process of converting fuzzy output is called defuzzification.

As discussed before, there are about twenty typical methods of defuzzification, but generally the maxima methods are good candidates for fuzzy reasoning systems while the distribution methods and the area methods exhibit the property of continuity that makes them suitable for fuzzy controllers [23]. Thus, in this system, we choose the centroid method, one of the area methods, to determine how much force ought to be applied. The centroid method can be expressed as

$$Output(x) = \frac{\int \mu_i(x)xdx}{\int \mu_i(x)dx}$$

The output will be determined by the aggregated membership function output $\mu_i(x)$ with a union operator (max), i the number of rules, and x the output variable.

3.5 REINFORCEMENT LEARNING

Reinforcement learning in a Learning Fuzzy Classifier System aims to study the rule base through trial and error via the reception of a numerical reward obtained from the environment. The system assigns a credit to each fuzzy rule and attempts to maximize future rewards. There are different strategies to define a rule credit. In a ZCS [24], a rule credit is a payoff value from the last action performed by the learner and this type of credit is referred to as rule strength. In a XCS [25], a credit, also called a rule accuracy, is based on the accuracy of predictions in action payoff. Reward or penalty from the environment is typically delayed which means a reward is received only after several actions have been applied to the environment so that the learner can better observe the consequence of its previous decisions.

Given that our system is trying to solve a control problem and control rules are considering error in each dimension as one of its antecedents, we derive the reinforcement from the error evaluation of an action. Typically, after one action is executed, the system will observe if the target error value such as the heading error, altitude error or speed error reduces accordingly as the controller intended. For instance, suppose a rule in Lateral Controller activates a "Negative Big" deflection on the aileron and the heading error diminishes after the action is applied. The system then ought to give a positive credit to this rule as it helps reducing the control error. In the opposite case, if the error increases, it tells that this action might be a wrong choice under such a situation and so the rule receives a negative credit.

Besides, the implication result of a fuzzy rule by min/max is taken into account again as the degree of an output decides how much effort the UAV makes in this action. Therefore, we use the following equation to calculate the credit:

$$credit = \mu_A(x) * \Delta error$$

where x is the very last action applied by the controller, $\mu_A(x)$ denotes the aggregated output value by union from fuzzy rules and $\Delta error$ is the error change in the control dimension.

After an action being taken, the system receives a feedback from the environment and calculates the reinforcement based on this equation and updates the rule base. This credit value will also be used later in the genetic algorithm for selecting parental rules to evolve.

3.6 GENETIC ALGORITHM

A genetic algorithm in a LFCS usually works in conjunction with a reinforcement learning techniques as an adaptive generation of rules. Rule credits updated by the reinforcement learning are the criteria for selecting parental individuals in genetic algorithms. The rule base of a LFCS is considered as an evolving ecology of rules and the genetic algorithm will be employed when the current rule base does not succeed to obtain satisfactory reinforcement from the environment.

3.6.1 REPRESENTATION OF THE FUZZY KNOWLEDGE BASE

When designing a genetic fuzzy controller, two main representation methods are available for different problem domains. The first method is tuning the parameter of a fuzzy controller that requires a certain amount of prior knowledge from experts to finish this task. In this approach, parameters in a fuzzy controller such as fuzzy membership function shape or scaling factors are encoded into a chromosome and a GA will optimize them to fulfill a given performance criteria function. This method, referred to as knowledge base tuning, is useful in those fields where the control process is dynamic or rule conditions are changing from time to time.

Another approach is that a fuzzy controller encodes the entire knowledge base, which is also our preference in the system. This approach would maintain both fuzzy rules and fuzzy membership functions in one population. It does not require prior knowledge for generating the initial population, since it concerns more on an automatic derivation of the knowledge base [26]. The genetic search process aims to target the best set of a knowledge base, which is more difficult than the first alternative method.

In the Michigan style LCS that encodes its rules one by one, there are two ways of evolving them. It could be either evolving its fuzzy rules with fixed membership functions or evolving both of them simultaneously. For the first one, a membership function, in Gaussian or triangular shape, is pre-defined by the designer to cover the complete input/output space.

However, learning fuzzy rules and fuzzy membership functions at the same time requires a coding scheme for both of them. In [27], fuzzy membership functions are encoded with their centers and widths of triangular fuzzy sets for each input and output.

The crossover operator randomly exchanges fuzzy sets between two fuzzy membership functions and the mutation operator adopts the creep mutation to change the width of a membership function.

3.6.2 THE MATRIX REPRESENTATION FOR FUZZY RULES

A fuzzy rule base or fuzzy sets usually use a non-binary representation since they have multiple available values on each allele. Suppose we have a two-input and one-output rule base and each input/output has 7 possible values. Obviously, in a non-binary representation, a rule base can be represented by a 7x7 table for the entire rule base and literally 49 genes are created. For a binary representation, it needs at least 343 genes to express all rules.

A sample knowledge base in integral matrix representation is shown in Table 3 where each number in a table entry represents an output fuzzy variable. We use integers from 1 to 7 to represent possible actions from "Negative Big" to "Positive Big" in order. For example, a chromosome represented as “(NB, ZE) = 5" denotes the following rule:

IF "Condition1 is Negative Big" AND "Condition2 is Zero" THEN "Action is Positive Small"

Table 3. A sample knowledge base matrix using non-binary representation

	NS	NM	NG	ZE	PS	PM	PB
NS	2	6	5	6	7	6	5
NM	3	4	1	2	2	2	2
NB	4	5	7	5	6	5	4
ZE	4	3	5	6	7	1	6
PS	5	5	6	6	5	3	1
PM	7	6	4	7	6	3	7
PB	1	1	1	6	5	5	4

In the system, for the lateral controller, the rule base is a 7x7x7 matrix since we have 7 possible fuzzy values for each input value and lateral control rules take three antecedents: roll rate, roll error, and heading error in their conditions. For longitudinal control, the controller has a 7x7 matrix rule base for the elevator as it needs two inputs: pitch rate, and pitch error. For the speed control, since the adjustment only depends on the speed difference, we simply keep a separate PID control to reduce system complexity. For each fuzzy rule base, the system keeps a corresponding credit matrix to record rule credits.

3.6.3 THE STRING REPRESENTATION FOR FUZZY MEMBERSHIP FUNCTIONS

For fuzzy membership functions, we define the curve shape as a Gaussian function whose parameter σ is set to 0.4 so that membership curves vary in center position but in the same shape for consistency. A string of real value numbers defining fuzzy membership centers represents membership functions of one input or output variable. Each value in this string denotes a corresponding center value of a fuzzy membership curve from fuzzy variable "Negative Big" to "Positive Big". For example, membership functions of input variable roll rate can be encoded as in

MF_Roll_Rate =

NB	NM	NS	ZE	PB	PM	PB
-1.5	-1.21	-0.25	0.1	0.62	0.97	1.5

where each digit denotes a center value of a membership curve.

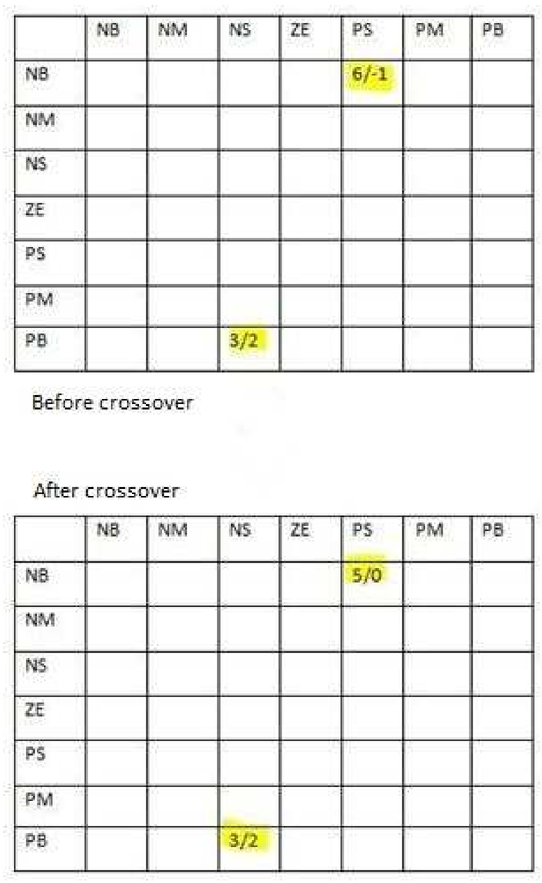
3.6.4 GENETIC OPERATORS

Before explaining details of genetic operators, there are two assumptions in genetic algorithm dealing with uncertain situations. First, selection is operated based on rule credits as stated in Chapter One. During the initialization, each rule should be treated fairly and assigned a credit of zero. Secondly, if antecedents of two rules are similar, we assume that consequences of these two rules should also be similar. This motivation has proved effective in several control problems [28].

For the crossover in the population, only fuzzy rules will be crossed-over since fuzzy membership functions in our system cover different numerical ranges for different fuzzy inputs. Fuzzy rules are organized into matrices, and hence a typical crossover that exchanges random sequences of pairs of rules might not be applicable here. However, the motivation of a crossover is to make use of the potential good genes among existing rules. Therefore, we define a point-radius crossover operator that replaces negative genes with opposite actions of their positive center-symmetric genes in the matrix for fuzzy rules arranged in matrices.

By observation, we can predict that a rule matrix in two-dimensions has a characteristic of "centre-anti-symmetric" which means a proper action under a certain input condition could be the opposite action under the center-symmetrical input in a matrix. For example, suppose an action of "Positive Small" on the aileron is appropriate for the condition "Roll rate is Positive Big" AND "Roll error is Positive Small" then it happens that an action of "Negative Small" is the right action for "Roll rate is Negative Big" AND "Roll error is Negative Small ".

When the GA carries out the crossover, it first starts scanning each two-dimensional matrix rule base and selects the lowest 10 percent of rules whose credits are less than 0. A chosen rule may not perform correctly under its condition. If its center-symmetric rule has a credit larger than 0, we just alter the action of this chosen rule to the opposite action of the center-symmetric rule and give the chosen rule a new credit of 0. Nevertheless, if the center-symmetric rule also has a negative credit, the operator will seek the best rule (with highest positive credit) adjacent to the center-symmetric one and place the best one's action into the chosen rule's action. This strategy is inspired by our second assumption that similar rules should have similar actions. Eventually, if there is no other action with positive credit near the center-symmetric rule, the crossover operator will skip this chosen rule and leave it for mutation. Figure 17 demonstrates a process of a typical crossover, and for display convenience, each matrix entry shows its "action/credit" pair. Suppose an action under "NB, PS" has a value “6” as in “Positive Medium” with a rule credit -1 while its "center-symmetric" rule, "PB, NS", has an action "3" as in “Negative Small” with a rule credit 2. In this case, the operator will change the action of "NB, PS" rule to “5” as “Positive Small” which is the opposite of "Negative Small".



	NB	NM	NS	ZE	PS	PM	PB
NB					6/-1		
NM							
NS							
ZE							
PS							
PM							
PB			3/2				

Before crossover

After crossover

	NB	NM	NS	ZE	PS	PM	PB
NB					5/0		
NM							
NS							
ZE							
PS							
PM							
PB			3/2				

Figure 17. A “center-anti-symmetric” crossover sample

Mutation can happen to both fuzzy rules and fuzzy membership functions. We first discuss how mutation happens on fuzzy rules. For rules with credit less than zero or if they somehow fail to crossover, then a mutation will change the action of this rule to a random one and reset its negative credit to a zero. Mutation of a fuzzy membership function will add or subtract the center values with a relatively small random number drawn from a Gaussian distribution with a mean of zero.

A Repair operator is employed here to fix the mutated center values ensuring that the mutation is within the input range and that every value in the sensor input range are covered by at least one fuzzy variable.

Below in Table 4 is a short summary of the genetic algorithm in the system.

Table 4. Genetic algorithm summary

GA configuration	**Fuzzy Rules**	**Fuzzy Membership Functions**
Representation	Multi-dimension matrices	String
Population size	392	9
Selection	Worst 10 negative rules	100%
Crossover	Point-radius crossover	N/A
Mutation	Random	Non-uniform mutation
Mutation rate	Ones with negative credit	100%
Initialization	Random	Memberships evenly distributed

3.6.5 FITNESS FUNCTION

The fitness function evaluates the optimality of the existing population and the genetic algorithm ought to maximize it. In the system, since we have two separate sub controllers, we set up one fitness function for each. The fitness is measured for a sub controller's knowledge base, not a single rule, since in flight maneuvers, it is difficult to account for just one or two rules for the entire performance during the control process. At this point, we consider the entire rule population as one unit, which is more like in a Pittsburgh style LCS.

The goal of the controller, no matter which dynamics it is managing, is to adjust the current state to the desired state with minimum control error and effort. Therefore, to develop a performance criterion, the fitness considers both the cumulative control error and control demand during certain control periods.

For the lateral control, the fitness function is as follow:

$$F = \frac{T}{\int_0^T (he^2 + re^2) + e} dt$$

For the longitudinal control, the fitness function is:

$$F' = \frac{T}{\int_0^T (se^2 + pe^2) \ + \ e} dt$$

In these two equations, t is time, he and re denote heading error and roll error; se and pe mean speed error and pitch error respectively; and e is a relatively positive small number.

After the evolutionary operations, given the same amount of simulation time t, the system evaluates the fitness value of the entire population based on the aggregated error during the simulation period. If a generation outperforms the best-so-far generation, then it replaces the old generation and becomes the new best population. Otherwise, the best-so-far remains in place and another series of evolution will be performed until a better generation arises.

CHAPTER 4

UAV CONTROLLER SIMULATION AND TEST RESULTS

4.1 REAL-TIME SIMULATION AND RELATED COMPONENTS

Before starting a description of the simulation environment, it is necessary to talk about the software components used and how we assemble them.

In the system, the implementation of LFCS is coded as an embedded Matlab function inside the controller model as a customized block.

FlightGear, an open-source, multi-platform and cooperative flight simulation platform maintained by open community over the internet, is a popular visualization tool for vehicle motion research. It aims to create a sophisticated flight simulator framework for use in research or academic environments. There are several existing flight simulation platforms such as the Microsoft Flight Simulator series, X-plane and Flight Pro Sim but we choose FlightGear for the following features.

1. Freeware: FlightGear is an open-source software and its source code is available to download, edit for researchers' own needs and re-distribute under the GNU General Public License.

2. Moderate System requirements: Unlike other advanced simulation games, FlightGear only requires a typical moderate Windows computer to run a simulation smoothly. This is due to the flexibility of the software configuration. The basic version of FlightGear does not automatically include any heavy cost unit such as an advanced graphic engine or a global scenery database but users can expand its database to 90%

coverage of the global area in high-resolution scenery texture and improve the graphic effect by upgrading with available service packages.

3. Compatibility with Matlab: In FlightGear, each aircraft has an aerodynamic configuration file defining its characteristics. Users can also replace its original model file with their own dynamic models written in Matlab or any available aerodynamic modeling tool. For our needs, FlightGear serves just as a “snapshot window” showing the UAV’s flying attitude.

Each sub controller first develops independently through a unit test to produce a mature knowledge base for its own dimension. For instance, the lateral flight plan involves shallow, medium and heavy turning scenarios. The simulation repeats until a certain generation of rules can perform acceptable flight maneuvers.

After training on each dimension, a regression test is set up to coordinate the UAV's knowledge base for more extensive flight operations. A lazy eight flight pattern as shown in Figure18 is preferred as a training prototype since it involves various flight controls and requires perfect coordination of controls through a wide range of airspeeds, headings, and altitudes.

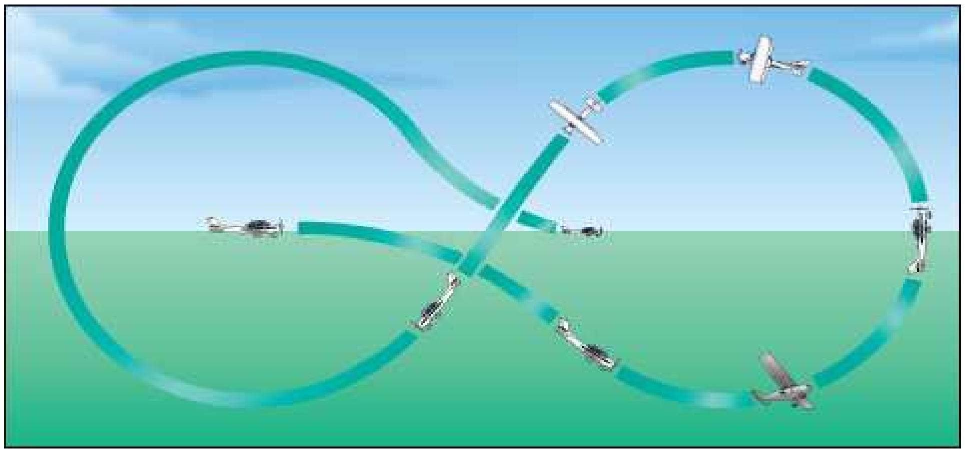


Figure 18. The Figure Eight Maneuver

The simulation runs in a real-time environment, which provides great convenience for designers to watch the controller flying the UAV in real-time and helps troubleshooting problems in either programming or knowledge base design. In addition, the plots as in Figure 19 from Matlab/Simulink display data updates and analysis on how well the controller is performing. However, it may also bring problems in simulation speed since everything in the simulation takes a large amount of system resources (CPU time, memory, and hard drive storage).

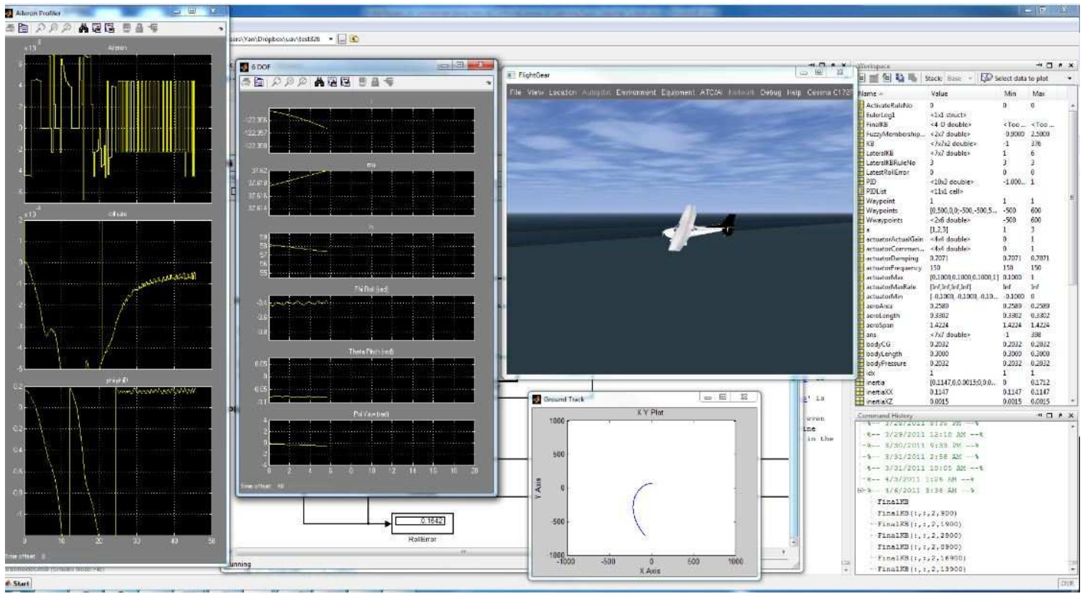


Figure 19. Simulation in real time

In order to tackle this system issue, we introduce several optimization steps:

1. Reduce the global variable used in the simulation

Due to the system requirement, the controller keeps a set of global variables including generation number, latest control error, latest activated rule, etc. These global variables in the simulation can affect performance heavily as they are stored permanently in memory and passed around blocks. Thus, our solution is to encode all of them into a global public storage array and no other memory is allowed to be opened as global variables.

2. Change embedded functions into blocks

One of the most time consuming factors in simulation are the Matlab embedded functions. Users can define them when there is no such defined block to satisfy users' specific requirements. These user-developed codes are in an embedded function that will have to be generated every time the simulation calls it, which slows down the speed dramatically. Our solution is to diminish the number of embedded functions and replace them with existing functional blocks and logic connectors if possible.

3. Stabilize a command

This idea comes from the reaction time difference between a single control loop and a state response from the UAV aerodynamics. To run through a single control loop, it requires only a tiny amount of time that approximately equals 10 milliseconds. However, for a response to occur on the UAV and have an effect on the current state (heading, speed, altitude), it takes much longer for the aerodynamics and environment blocks to respond. Thus, there is a need to "stabilize" the last command until it really works on the UAV.

To the simulation implementation, we add a small step in the controller determining if the last action has truly taken effect on the UAV so that the next action should be delivered. If not, the controller will hold on the last action output until the UAV’s state begins to change. This process exactly mimics how a real pilot operates an airplane. In the real world, a pilot would also maintain the position of his flight yoke or throttle for a while until the aircraft starts to change its heading or speed correspondingly. Surprisingly, a certain amount of time in control delay turned out to improve the simulation speed greatly too because the simulation now need not go through the entire decision making process in every single control loop as it waits for the UAV’s response, which saves a lot of computation cost.

.

4.2 KNOWLEDGE BASE DEVELOPMENT

4.2.1 FUZZY RULE DEVELOPMENT THROUGH REINFORCEMENT

In the unit test of the lateral controller, a flight plan designed to train the ability of controlling the aileron to different desired headings with consecutive right/left turns is loaded into the simulation. The UAV ought to make three right turns and then three left turns in different degrees of intensity while its airspeed and altitude are constant at 50 knots and 30 feet respectively.

During the simulation, we can observe, at the beginning, the UAV randomly chooses an action to perform. If the action leads to a worse state, another action is tested. Otherwise, it keeps applying this command until it receives a negative credit or an unmatched condition appears. This random acting process can be viewed in Figure 20 that shows the aileron value in the top is stochastic at the beginning until an action which generates 0 roll error (represented as phi-phD) is performed. The UAV stabilizes that action for a short amount of time to decrease the roll error effectively.

At that point, positive credits are given to the activated rule that is generating helpful action for this condition. This process can be viewed as a Trial and Error method and it successfully creates correct rules.

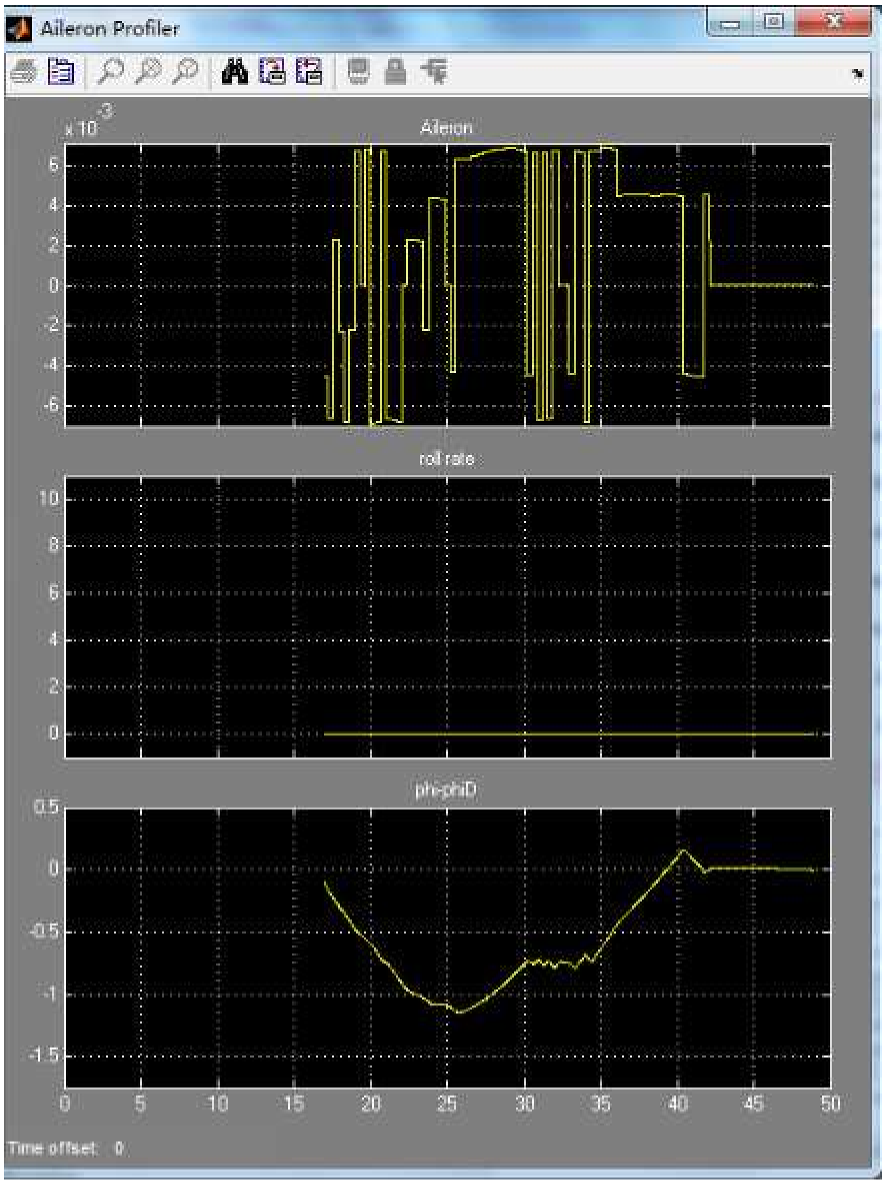
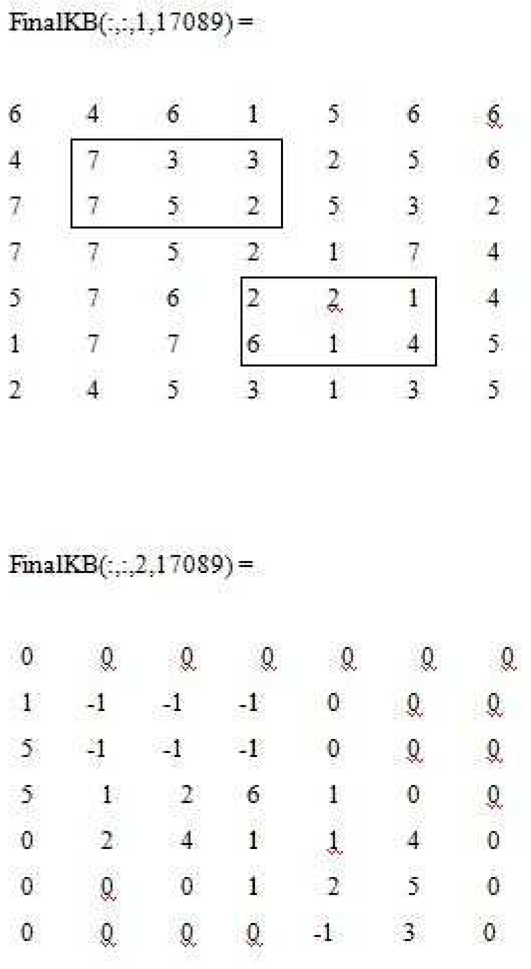


Figure 20. Initial trials in lateral control

4.2.2 FUZZY RULE DEVELOPMENT THROUGH GENETIC ALGORITHM

When the simulation time reaches a threshold value for invoking the genetic algorithm and if the current controller performance needs to be improved, it means that the previous Trial and Error learning does not produce rules robust enough for scenarios encountered.

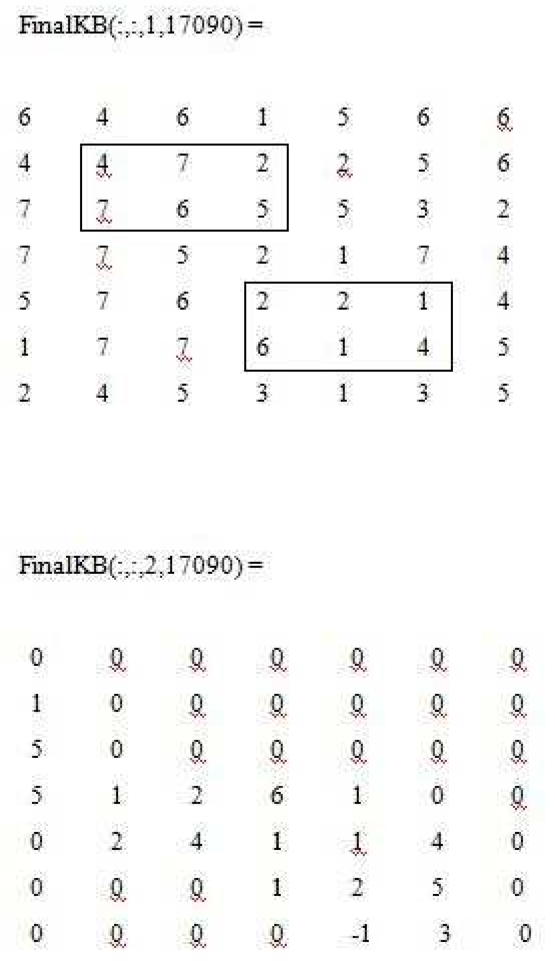


```
FinalKB(:,:,1,17089) =

6    4    6    1    5    6    6
4    7    3    3    2    5    6
7    7    5    2    5    3    2
7    7    5    2    1    7    4
5    7    6    2    2    1    4
1    7    7    6    1    4    5
2    4    5    3    1    3    5


FinalKB(:,:,2,17089) =

0    0    0    0    0    0    0
1   -1   -1   -1    0    0    0
5   -1   -1   -1    0    0    0
5    1    2    6    1    0    0
0    2    4    1    1    4    0
0    0    0    1    2    5    0
0    0    0    0   -1    3    0
```

Figure 21. A rule base at simulation loop 17,089

Figure 21 is a rule base generated at simulation loop 17,089 that did not outperform the best performance so far and called for an evolutionary development. The above matrix is a lateral knowledge base and the bottom one is its corresponding credit matrix. We can see that most rules at the upper left corner received negative credits while rules at the lower right and center have better credits. Therefore, a genetic algorithm executes the crossover which picks up worst rules among the upper left corner as candidates.

The crossover operator changes the actions of these rules to the opposite action of their center-symmetrical rules. A new rule base after crossover and mutation is displayed in Figure 22 which utilizes the assumption we made about the "center-anti-symmetric" characteristics of the rule base.

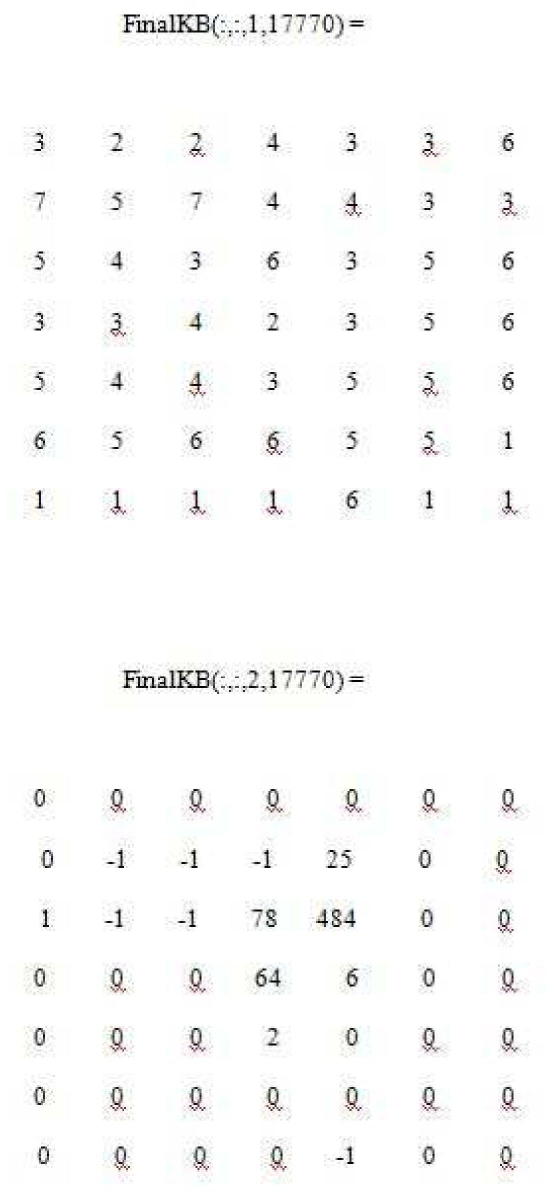


```
FinalKB(:,:,1,17090) =

6    4    6    1    5    6    6
4    4    7    2    2    5    6
7    7    6    5    5    3    2
7    7    5    2    1    7    4
5    7    6    2    2    1    4
1    7    7    6    1    4    5
2    4    5    3    1    3    5


FinalKB(:,:,2,17090) =

0    0    0    0    0    0    0
1    0    0    0    0    0    0
5    0    0    0    0    0    0
5    1    2    6    1    0    0
0    2    4    1    1    4    0
0    0    0    1    2    5    0
0    0    0    0   -1    3    0
```

Figure 22. A rule base at simulation loop 17090

4.2.3 FUZZY MEMBERSHIP FUNCTION DEVELOPMENT THROUGH GENETIC ALGORITHM

Fuzzy membership functions are tools that map a real value to a fuzzy variable in the simulation. Its partition decides how accurate the controller can estimate the UAV status and output to the control surfaces and throttle. Therefore, when the simulation is carried out, the system will observe the distribution of activated rules in the rule base and measure if fuzzy membership functions are accurate. As shown in Figure 23, what we can see from the credit matrix at the bottom is that most rules activated so far were located around the second and third row. It means that the fuzzy membership function for the row input classifies most of the sensor inputs as “Negative Small” or “Negative Medium” and hence only rules that include these two fuzzy variables as antecedents had been activated and therefore lead to inappropriate actions and poor performance.

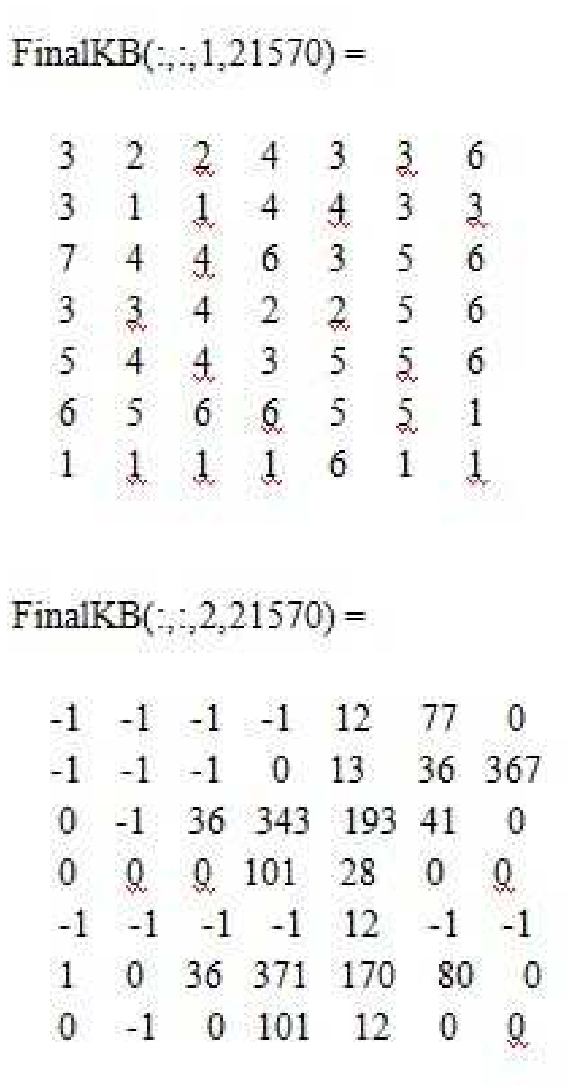


```
FinalKB(:,:,1,17770) =

3   2   2   4   3   3   6
7   5   7   4   4   3   3
5   4   3   6   3   5   6
3   3   4   2   3   5   6
5   4   4   3   5   5   6
6   5   6   6   5   5   1
1   1   1   1   6   1   1


FinalKB(:,:,2,17770) =

0   0   0   0   0   0   0
0  -1  -1  -1  25   0   0
1  -1  -1  78  484  0   0
0   0   0  64   6   0   0
0   0   0   2   0   0   0
0   0   0   0   0   0   0
0   0   0   0  -1   0   0
```

Figure 23. A rule base at simulation loop 17,770

The mutation operator in the genetic algorithm should try to expand or shrink the coverage of fuzzy membership functions so that a real value input can accurately fall into its optimal fuzzy degree category.

After generations of mutation, an evolved rule base as shown in Figure 24 displays an almost average distribution of rule credits. In this new rule base, most rules were activated and received credits from the environment. The new fuzzy membership functions had better capability of determining to which fuzzy degree an input variable belongs.



```
FinalKB(:,:,1,21570) =

 3   2   2   4   3   3   6
 3   1   1   4   4   3   3
 7   4   4   6   3   5   6
 3   3   4   2   2   5   6
 5   4   4   3   5   5   6
 6   5   6   6   5   5   1
 1   1   1   1   6   1   1

FinalKB(:,:,2,21570) =

-1  -1  -1  -1   12   77    0
-1  -1  -1   0   13   36  367
 0  -1  36  343  193  41    0
 0   0   0  101   28   0    0
-1  -1  -1  -1   12   -1   -1
 1   0  36  371  170   80    0
 0  -1   0  101   12    0    0
```

Figure 24. A rule base at simulation loop 21,570

4.3 CONTROLLER OVERALL PERFORMANCE

After continuously running the simulation for over 12 days, the evolved controller finally achieved an acceptable level of controlling the UAV. The successful controller is able to complete the figure eight flying pattern within reasonable deviation ranges of ground track, altitude, and speed change.

Figure 25 shows the ground track of a successful figure eight completion. The controller succeeds to follow the flight plan that includes over 12 waypoints and 7 turnings with different heading changes.

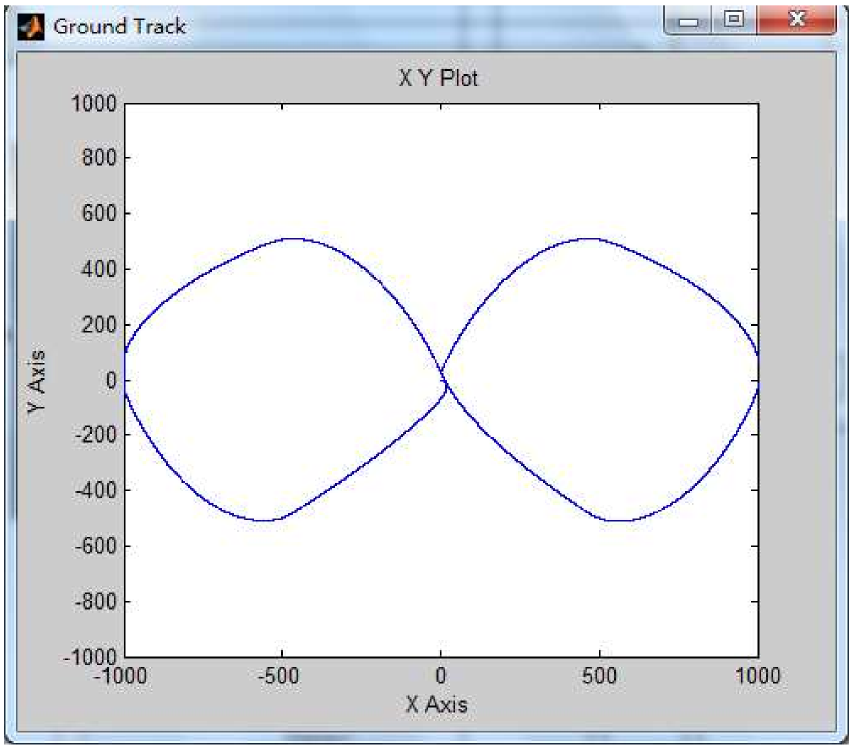


Figure 25. Ground track of a successful completion of figure eight flight pattern

Figure 26 shows the altitude level changes during the figure eight test where the top chart is the desired altitude ranging from 60 to 90 feet and the bottom chart is the UAV's actual altitude starting from 0. As we can see, the controller manages to make required altitude level adjustments within a short period.

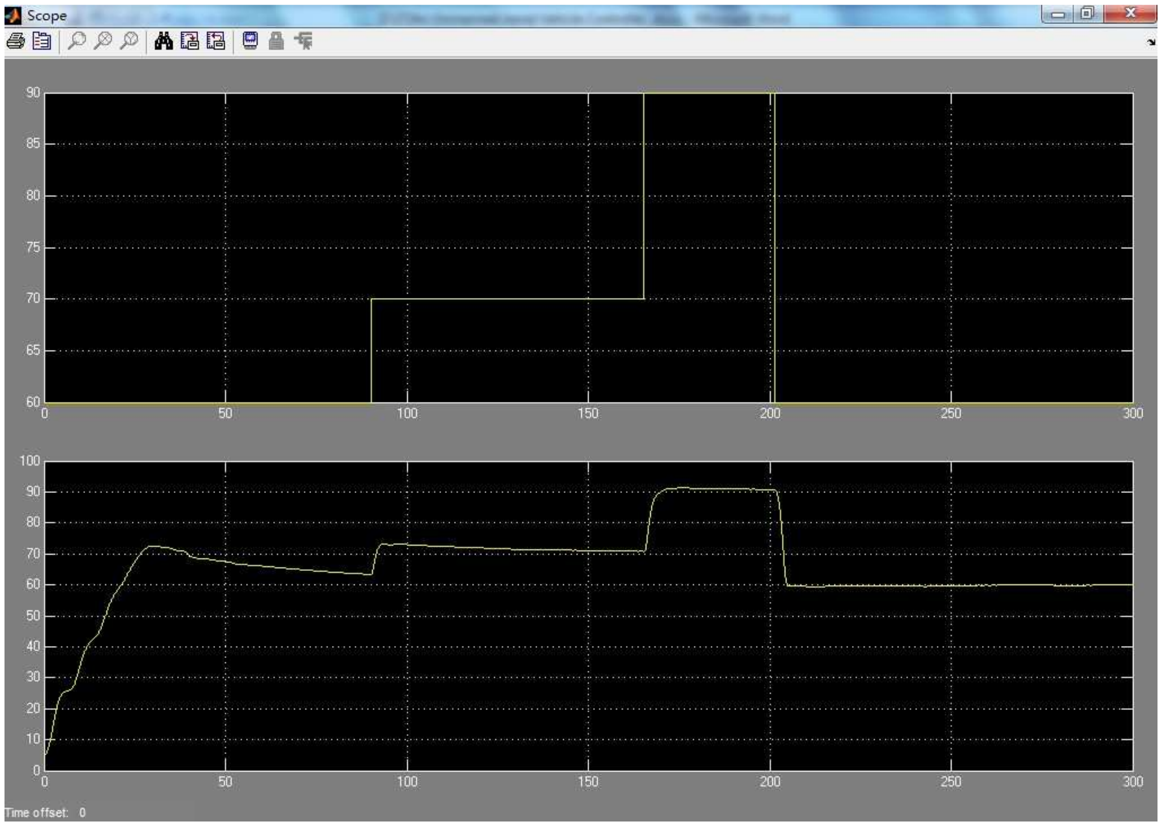


Figure 26. Altitude change of a successful completion of a figure eight flight pattern

CHAPTER 5

CONCLUSIONS AND FUTURE WORKS

The Learning Fuzzy Classifier System is an exciting approach to design an autonomous controller for an unmanned aerial vehicle. The fuzzy logic embedded in the rule base enables the system to deal with uncertainty in various flying scenarios. A reinforcement module drives the learner to correct its existing knowledge based on feedback from the external environment. A genetic algorithm refines the rule population by evolutionary heuristics. Given those powerful tools, a learning fuzzy classifier system has proven to be a positive method in solving control engineering problems.

The aerodynamics and controller simulation in Matlab/simulink provides a useful method to develop and tune an algorithm without any risk of physical damage to the UAV's hardware. However, in order to model the UAV's performance successfully, the aerodynamics model needs to be highly accurate and thus requires more development effort and cost. In this thesis, we compromise the computation cost in modeling the aerodynamics to ensure the model is able to simulate the real response on the airframe.

The broader scope of this LFCS methodology is to design an autonomous controller that can perform not only the basic flight control but other advanced tasks such as avoid obstacles, track objects and take pictures. These tasks will need a more delicate rule base design and higher-level decision-making mechanism in behavior. A further goal would be developing a human-like language for flight instructions which only specify high-level behaviors for the UAV to perform such as Take-off, Cruise, and Land. These behavior instructions then will be parsed, analyzed and interpreted into Matlab or assembly code on a real UAV to guide its missions.

BIBLIOGRAPHY

[1] Goodrich, A., Morse, S., Gerhardt, D., Cooper, L., Quigley, M., Adams, A. & Humphrey, C. (2008), Supporting Wilderness Search and Rescue using a Camera-equipped mini UAV. *Journal of Field Robotics, 25*: 89–110.

[2] Palat, C., Annamalau, A., & Reed, R. (2005). Cooperative Relaying for Ad-hoc Ground Networks using Swarm UAVs.*Proceedings of IEEE Military Communications Conference 2005, Vol.3*, 1588-1594.

[3] Yang, Y., Minai, A. (2005). Evidential Map-Building Approaches for Multi-UAV. *Proceedings of American Control Conference 2005*, 116-121.

[4] Girard, R., Howell, S., &Hedrick, K. (2004). Border Patrol and Surveillance Missions using Multiple Unmanned Air Vehicles. *43rd IEEE Conference on Decision Control, Vol.1*, 620-625.

[5] Zadeh, L. (1965). A.: *Fuzzy Sets Information and Control* 8(3): 338-353.

[6] Nakamura, K., Sakashita, N., Nitta, Y., Shimomura, K., & Tokuda, T. (1993) Fuzzy Inference and Fuzzy Inference Processor, *IEEE Micro*, *Vol.13*, No.5, pp. 37-48.

[7] Hellendoorn, H., & Thomas, C. (1993). Defuzzification in Fuzzy Controller. *Journal of Intelligent and Fuzzy Systems*, *Vol.1 (2)*, 109-123.

[8] Huang, L. J. & Tomizuka, M. (1990).A Self-paced Fuzzy Tracking Controller for Two-dimensional Motion Control. *IEEE Transaction on Systems Man Cybernet.* 205, pp. 1115–1124.

[9] Carse, B., Fogarty, T., & Munro, A. (1995). Evolving Fuzzy Rule Based Controllers using Genetic Algorithm. *Fuzzy Sets and System, Vol. 90, Issue 3*, 273-293.

[10] Fleming, J., & Purshouse, C. (2002). Evolutionary Algorithms in Control Systems Engineering: A Survey. *Control Engineering Practice, Vol.10*, Issue 11.

[11] Goldberg, D., & Deb, K. (1991). A Comparative Analysis of Selection Schemes used in Genetic Algorithms. *Foundation of Genetic Algorithms*, 1, 69-93.

[12] Thierens, D., & Goldberg, D. (1994). Convergence Models of Genetic Algorithm Selection Schemes. *Lecture Notes in Computer Science, Vol. 866*, 119-129.

[13] Bonarini, A. (2000). An Introduction to Learning Fuzzy Classifier Systems. *Lecture Notes in Computer Science*, *Spreinger Berlin*, *Vol.1813*, pp.83-104.

[14] Cordon, O., Herrera, F., Gomide, F., Hoffmann, F., & Magdalena, L. (2001). Ten Years of Genetic Fuzzy Systems: Current Framework and New Trends. *IFSA World Congress and 20th NAFIPS International Conference, Vol.3*, 1241-1246.

[15] Hoffman, F., & Pfister, G. (1997). Evolutionary Design of a Fuzzy Knowledge Base for a Mobile Robot. *International Journal of Approximate Reasoning, Vol. 17*, 447-469

[16] Homaifar, A., & McCormick, E. (1995). Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers using Genetic Algorithm. *IEEE Transactions on Fuzzy Systems, Vol.3*, 129-139

[17] Sutton, R ., & Marsden, G. (1997). A Fuzzy Autopilot Optimized Using a Genetic Algorithm. *The Journal of Navigation*, 120-131.

[18] Vaishnav, S.R., & Khan, Z.J. (2007). Design and Performance of PID and Fuzzy Logic Controller with Smaller Rule Set for Higher Order System. *Proceedings of the World Congress on Engineering and Computer Science 2007*, 855-858.

[19] Christiansen, R. S. (2004) Design of an Autopilot for Small Unmanned Aerial Vehicles. Master Thesis in Electrical and Computer Engineering at Brigham Young University.

[20] Carvajal, J., Chen, G., Ogmen, H. Fuzzy PID Controller: Design, performance evaluation, and stability analysis. (2000), *Information Sciences, Vol. 123*, Issue 3-4.

[21] Ceren, R., Durden, M., Lakshmanan, R., Pandhiti, S., & Thayasivam, U. (2008) UAV Project Guide, Institute for Artificial Intelligence at the University of Georgia Technical Report.

[22] Casillas, J., Cordon, O., Jesus, M., & Herrera, F. (2005). Genetic Tuning of Fuzzy Rule Deep Structures Preserving Interpretability and its Interaction with Fuzzy Rule Set Reduction. *IEEE Transactions on Fuzzy Systems, Vol. 13, No.11*,13-29 .

[23] Yager. R., & Filev, D. (1993). On the issue of Defuzzification and Selection based on a Fuzzy Set. *Fuzzy Sets and Systems, Vol.55*, Issue 3, 255-271.

[24] Bull, L., & Hurst, J. (2001) ZCS: Theory and Practice. UWE Learning Classifier System Group Technical Report 01-001.

[25] Butz, M., & Wilson, S. (2001) An Algorithmic Description of XCS. *Lecture Notes in Computer Science, Vol.1996*, 267-274.

[26] Hoffmann, F. & Pfister, G. (1996). G. Genetic Evolutionary Learning of a Fuzzy control Rule Base for an Autonomous Vehicle. *In Proceedings of the Fifth International conference on Information Processing and Management of Uncertainty in Knowledge-Based System (IPMU-96)*: 659-664.

[27] Velagic, J., Vukic, Z., & Omerdic, E. (2003). Adaptive Fuzzy Ship Autopilot for Tracking-keeping. *Control Engineering Practice, Vol. 11, Issue 4*, 433-443.

[28] Wang, M., Yu, Y., & Wei, L. (2009). Adaptive Neural-Based Fuzzy Inference System Approach Applied to Steering Control. *Proceedings of the 6th International Symposium on Neural Networks: Advances in Neural Networks*, 1189-1196.

[29] Shu, L. & Schaffer, J. (1991). HCS: Adding hierarchies to classifier systems. *In Proceedings of the Fourth International Conference on Genetic Algorithms,* Los Altos, CA , pp. 339–345.

[30] Tan, K., Lee, T. & Lee, L. (2001). A Messy Genetic Algorithm for the Vehicle Routing Problem with Time Window Constraints. *In Proceedings of the 2001 Congress on Evolutionary Computation (CEC' 2001)*: 679-686.

[31] Van Veldhuizen, D.A. (2000). Genetic Multi-objective Optimization with Messy Genetic Algorithms. *In Proceedings of the Fifteenth ACM Symposium on Applied computing (SAC'2000)*: 470-476.

[32] Riley, J. & Ciesielski, V. (2002). Evolving Fuzzy Rules for Reactive Agents in Dynamic Environments. *In Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, Singapore, November 2002, pp124-130.

[33] Kinzel, J., Klawonn, F., & Kruse, R. (1994). Modifications of Genetic Algorithm for Designing and Optimizing Fuzzy Controllers. *Proceedings of the First IEEE on Computational Intelligence, Vol.1*, 28-33.

[34] Motta, D., Polati, A., & Kawakami, R. (2009). Model-Based-Design of an Aircraft Auto-pilot Controller. *Third CTA-DLR Workshop on Data Analysis and Flight Control*.

[35] Chao, H., Cao, Y., & Chen, Y. (2010). Autopilots for Small Unmanned Aerial Vehicles: A survey. *International Journal of Control, Automation, and Systems(2010) 8(1)*: 36-44

[36] Stevens, B.L. & Lewis, F.L. (2003). *Aircraft Control and Simulation*. John Wiley & Sons, Inc., 2nd edition.

[37] Sorton, E., & Hammaker, S. (2005). Simulated Flight Testing of an Autonomous Unmanned Aerial Vehicle Using FlightGear. *Institute for Scientific Research, Fairmont,* West Virginia. September 2005.