University of Cincinnati

Date: 3/27/2018

I, Nicklas O Stockton, hereby submit this original work as part of the requirements for the degree of Master of Science in Aerospace Engineering.

It is entitled:

Hybrid Genetic Fuzzy Systems for Control of Dynamic Systems

Student's name: **Nicklas O Stockton**

This work and its defense approved by:

Committee chair: Kelly Cohen, Ph.D.

Committee member: Manish Kumar, Ph.D.

Committee member: George T. Black, M.S.

UNIVERSITY OF Cincinnati

30350

Hybrid Genetic Fuzzy Systems for Control of Dynamic Systems

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree
Master of Science in Aerospace Engineering and Engineering Mechanics

By

Nicklas Stockton, B.S. Aerospace Engineering

Graduate Program in Department of
Aerospace Engineering and Engineering Mechanics

2018

Master's Examination Committee:

Dr. Kelly Cohen, Advisor

Dr. Manish Kumar

Prof. George Black

Abstract

Aerospace applications are composed of many dynamic systems which are coupled, nonlinear, and difficult to control. Fuzzy logic (FL) systems provides a means by which to encode expert knowledge into a set of rules which can produce highly nonlinear control signals; this is possible because FL, like many other soft computational methods is a universal approximator. While FL systems alone excel at encapsulating expert knowledge bases, when coupled with genetic algorithms (GA), they can learn the knowledge base from evolutionary repetition. It is the goal of this work to present the efficacy of hybrid genetic fuzzy systems (GFS) in a variety of applications. This will be achieved through exploring three specific use cases.

First, a variation of a benchmark problem presented at the 1990 American Control Conference is used to demonstrate the robustness of FL control as well as the utility of GAs in the learning process. The results are a controller that is far more resistant to even large changes in the plant dynamics compared to a linear controller and a process by which a class of controllers may be quickly tuned for changes to the plant system.

The next problem applies the same approach to an elevator actuator for pitch control of an F-4 Phantom. This controller is tuned for a nominal case and ten subjected to the same plant with degraded aerodynamic coefficients. It is compared to a well-tuned PID controller.

The effort culminates in a practical application of a FL system to guide a small unmanned aerial system (sUAS) to a precision landing on a target platform moving with uncertain velocity. This was accomplished using custom developed Python software for GFS control in conjunction with Robot Operating System (ROS) and a simulation environment called Gazebo. Heavy emphasis was placed on using only software components which can be easily implemented on popular hardware platforms. ROS was critical to meeting this goal, as well as the open source flight controller project PX4. A controller is presented which is capable of exerting the control necessary to guide the vehicle to a successful landing on both a static and moving platform.

To my wife, Lori, for her unending support, motivation and loving guidance. I simply would not have been able to complete this work without her by my side.

I owe a great debt to my advisor, Dr. Kelly Cohen. Ever the motivator and champion, he has done more to build the scaffold for my success than he knows. I also thank Dr. Manish Kumar for his example, intellect, guidance, and support. I would be remiss without acknowledging and sincerely thanking my friends and colleagues: Bryan Brown, for being my sounding board and making sure I always had the tools I needed (both literally and figuratively) to accomplish my work; Nathaniel Richards for listening to my endless ramblings as I work out ideas; Justin Ouwerkerk for lighting the fire of competition under me; and Anthony Lamping for letting me know just how bad my code is, and for helping me make it better. All have helped me both directly and indirectly in the completion of this thesis. Finally, I owe an unpayable debt to my wife, Lori. She has sustained me in every way imaginable through this process.

Table of Contents

Page

List of Tables

List of Figures

Chapter 1: Introduction

Fuzzy logic (FL) and genetic algorithms are members of a family of so-called soft computing methods, along with neural nets (NN) and probabilistic reasoning[1]. Each of these methods has particular strengths in specific domains, but all are employed in applications which require computation based on imprecise, vague, or uncertain data. In recent years, NNs have risen greatly in popularity as improvements in compute power and algorithm development have allowed their application to a wide variety of problems via their role in deep learning[2]. While FL has seen much less application in deep learning applications, it has long been shown to be particularly useful in feedback control applications of nonlinear systems[3]. FL excels at encoding expert knowledge of a human operator in machine-interpretable logic. In many cases, however, it is difficult or impossible to encode a knowledge base which is sufficient to control a system satisfactorily by hand due to the complexity of the system; in these cases, a GA may be applied to tune, or enable a FL system to learn to accomplish the task.

1.1 Fuzzy Logic

The basic component of a FL system consists of fuzzy membership sets[4], which are an extension of classical binary set theory. The advantage of fuzzy set theory is that an element may share membership across many sets simultaneously, in contrast

to traditional set logic (see figures 1.1 to 1.2). This formal encapsulation of "fuzziness" provides a method for machine instructions to operate on imprecise values. Fuzzy sets constitute the most basic element of a FL system; they are made useful in controls application when combined with a rule base (RB).

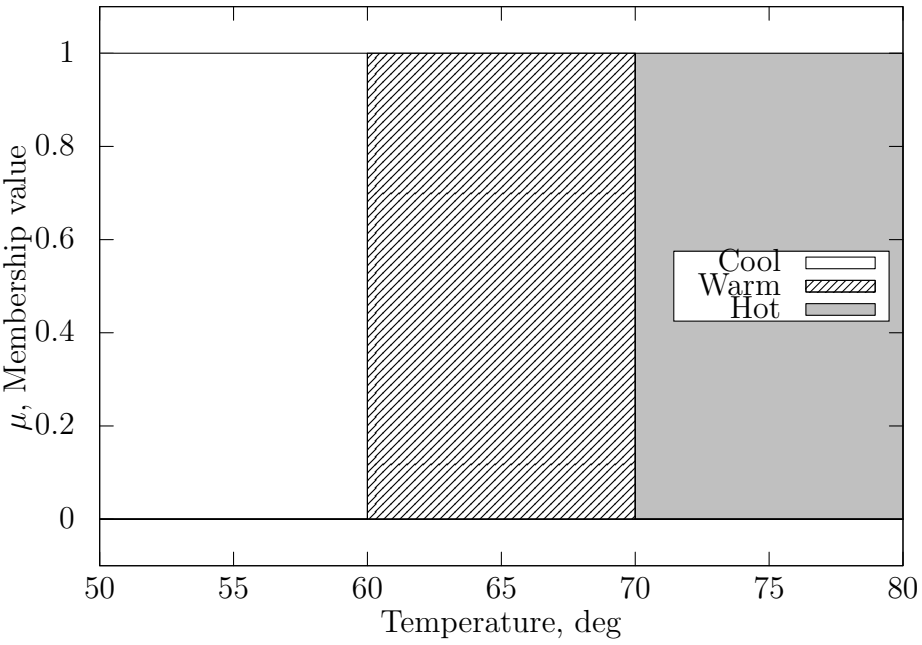


Figure 1.1: Crisp (classical) logic set transitions are discrete and abrupt

A fuzzy inference system (FIS) is a control system built on the basic principles of fuzzy logic[5][6]. It can take an arbitrary number of analog inputs and map them to a set of logical variables ranging from 0 to 1. This mapping is performed by membership functions (MF), which determine the degree of membership each input has to a function. Each input typically has multiple MFs, allowing the input value (a so-called "crisp value" to be given membership in multiple sets simultaneously. The

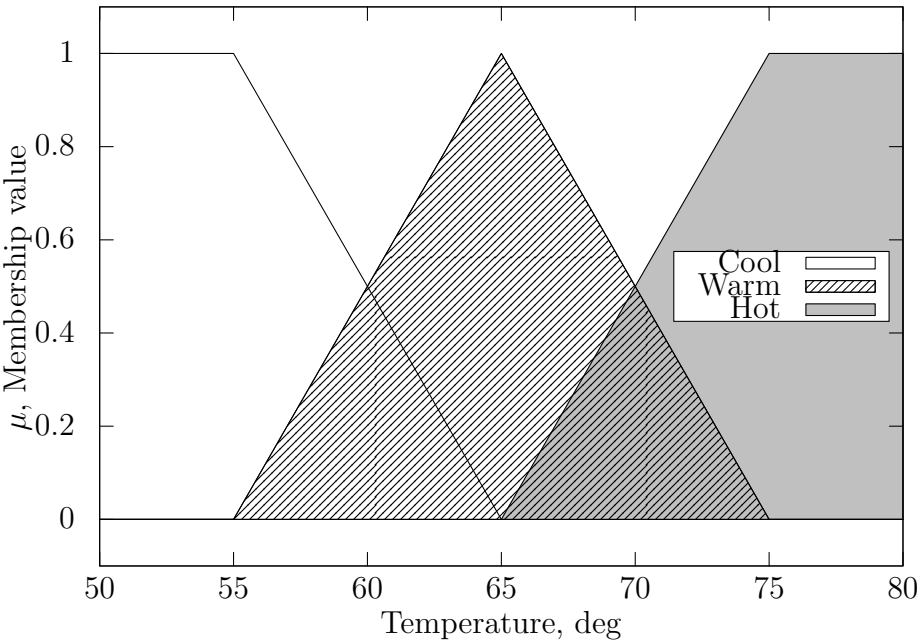


Figure 1.2: Fuzzy sets allow for multiple membership, so a value can smoothly transition from being a member of one set to another.

memberships are used to make an inference about the input according to linguistic rules. The RB is composed of IF-THEN statements which determine the crisp output variable. For example, if the membership sets in figure 1.2 were to be the input to a fan controller, and the output membership functions were as they are in figure 1.3, the rules could be described as:

IF	*temperature* is	COOL,	THEN	*fan speed* is	OFF
IF	*temperature* is	WARM,	THEN	*fan speed* is	LOW
IF	*temperature* is	HOT,	THEN	*fan speed* is	HIGH

These rules can be succinctly placed in a table as shown in table 1.1. This representation becomes more useful still as another input is added to the controller. By using the set of rules and the MFs, the FIS is able to determine an appropriate

crisp output given a set of crisp inputs. The FIS representation is in essence a transfer function, but a transfer function which can be arbitrarily complex. The common representation of a FIS shown in figure 1.4 shows the components of a FIS.

Table 1.1: Example rule base for a fictitious fan speed controller

Temperature	Cool	Warm	Hot
Fan Speed	Off	Low	High

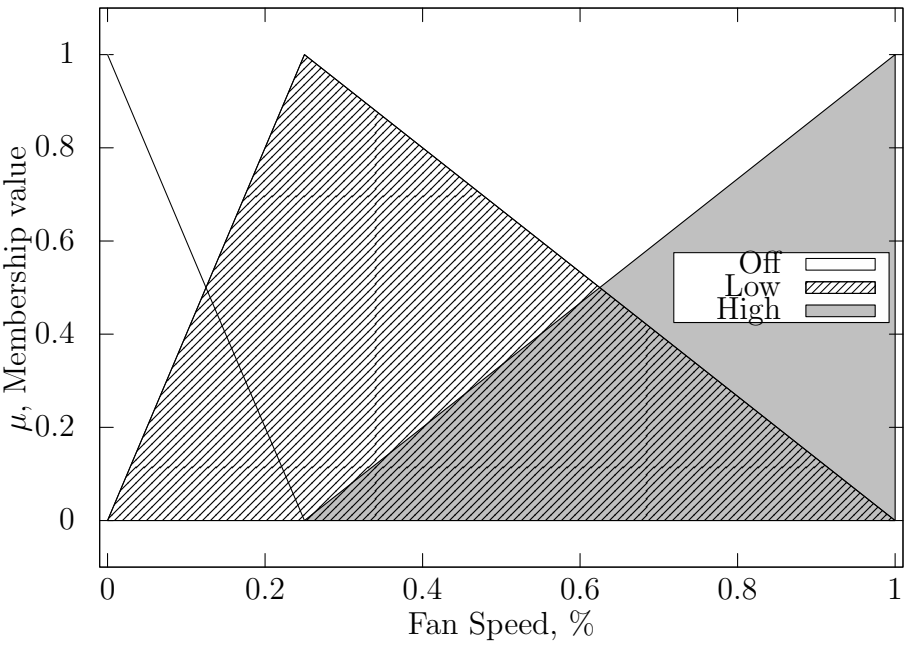


Figure 1.3: Possible output membership functions for a fan controller with figure 1.2 as the input.

One limiting constraint of FL systems is manifested in a type of state explosion. As the number of inputs, specifically the number of input MFs, increases, the number

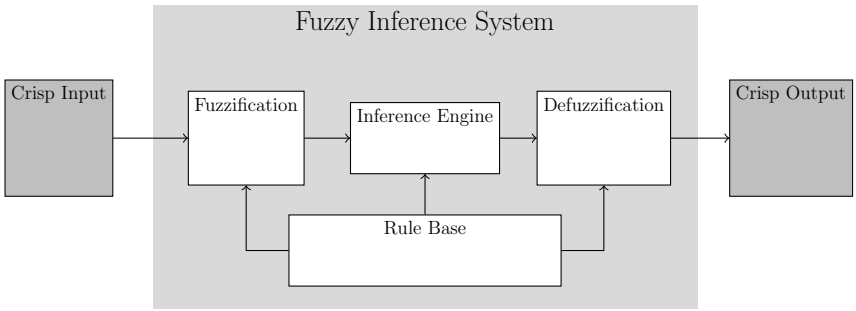


Figure 1.4: Visual representation of a fuzzy inference system

of rules needed to cover every case increases as the product of the number of MFs across all inputs. There have been numerous approaches to overcome this problem such as cascading small FISs together[7], developing more robust methods to guide convergence[8] or using approximate fuzzy rule bases[1]. The approach taken in this work, however, has been to employ GA techniques to tune the MFs or even learn the entire knowledge base. The dynamic systems being controlled in this research are adequately small such that no special state reduction was deemed necessary in most cases.

1.2 Genetic Algorithms

Genetic algorithms are an evolutionary computing strategy commonly used in optimization and search problems [9]. Their effectiveness comes from the combined ability to explore and exploit. The exploration of even large search spaces is made possible with large populations of candidate solutions which are stochastically sampled across the whole space. Candidates are ranked according to a fitness function and recombined together to create the next generation. The algorithm exploits learning in the selection process, favoring recombination of candidates which ranked well according

to the fitness function. In order to maintain gene pool diversity, random mutation is introduced into each generation as it is created. This process is illustrated in figure 1.5. Due to the stochastic nature of the algorithm, it is not guaranteed to provide an optimal solution and may converge on a local optimum, but careful selection of hyper parameters such as population size, mutation rates, recombination methods, and random candidate injection can somewhat circumvent these deficiencies. As the goal of a GA is to maximize some definition of a reward, they are commonly employed in reinforcement learning applications[10].

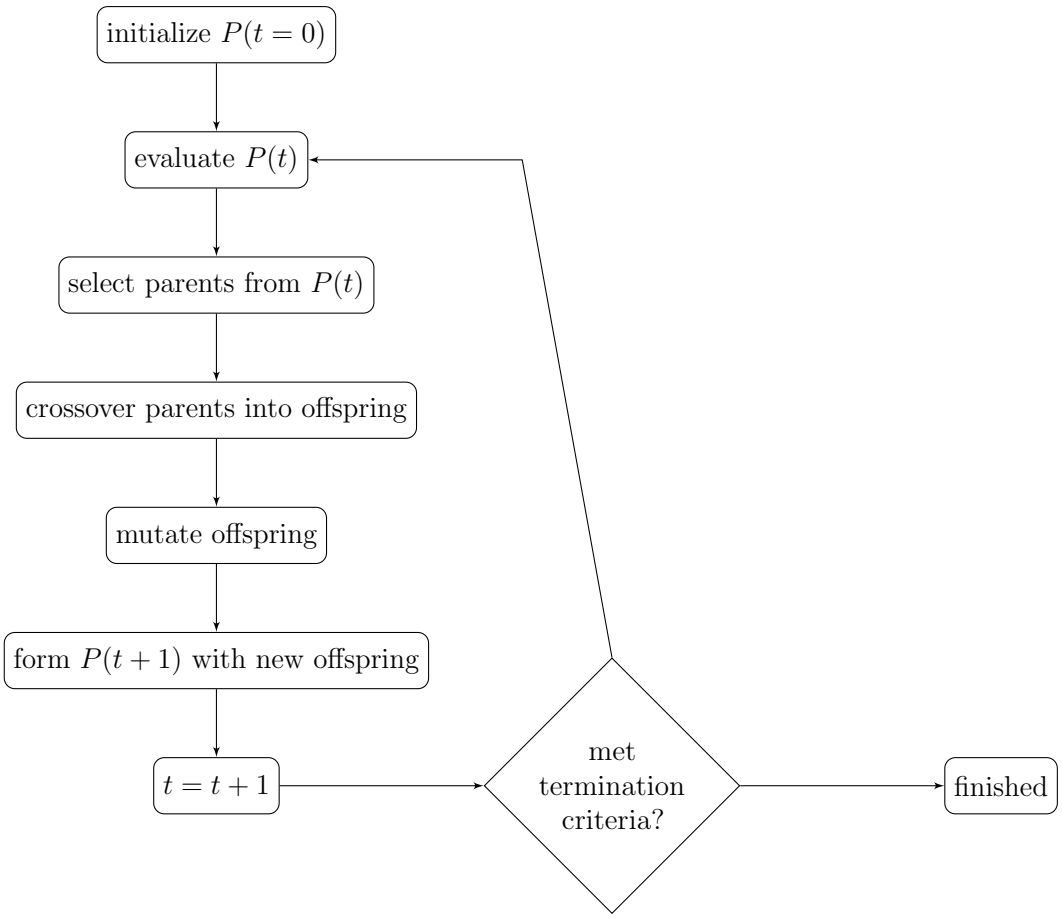


Figure 1.5: Block diagram describing genetic algorithm

Since the fitness function is the sole driver of the algorithm and the solution it provides, great care must be taken in formulating a fitness function. In robotics applications in particular, formulating a fitness which adequately encapsulates complex outcomes can be a difficult task[11]. This issue will be revisited a number of times throughout the course of this work.

Another major consideration when utilizing a GA is how to represent or encode a candidate solution for optimization. A particularly simple method is to use a binary encoded format[1], but this increases the distance between the candidate in its useful form (phenotype) and its encoded form (genotype) for many real-values problems[12]. A genetic encoding of a FIS is an inherently heterogeneous structure with real values describing the MFs and discrete classes describing the RB. The genetic operator chosen for a specific genotype will depend on its representation; ideally, the mapping between genotype and phenotype will be one-to-one such that the combination of two genes will produce a candidate which performs similarly to its parents. This allows the GA to properly exploit its inherent learning and converge on a solution.

1.2.1 Genetic Operators

The GA advances by applying operations to the chromosome candidates in its population. The genetic operations employed in this work are crossover, mutation, and randomization. Crossover is the operation which allows the genes from two chromosomes to mix and create new children. If the chromosomes are binary encoded or discrete valued, the crossover method is generally single-, double-, or n-point crossover. This involves randomly selecting a point in the encoding string to "cut" the

chromosomes and swap the tails for single-point, or sections for n-point. Figure 1.6 demonstrates this process.

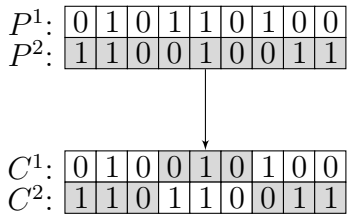


Figure 1.6: Double point crossover for binary or discrete valued chromosome.

For real-valued chromosome strings, the crossover methods can be much more complex and more closely related to solution space. Throughout this work, variations on flat crossover will be employed as the main crossover operation for real-valued portions of the chromosomes[1]. Using this method, a new gene is created from its parents by drawing a real value from the interval between the parent genes using a uniform distribution. In many situations, it becomes simpler to write code which always produces two children from a pair of parents; in these cases, the other gene from a parent interval is selected to be the complement of the first. Given parents, $P^1 = \{p_1^1, p_2^1, \ldots, p_n^1\}$, $P^2 = \{p_1^2, p_2^2, \ldots, p_n^2\}$, children $C^1 = \{c_1^1, c_2^1, \ldots, c_n^1\}$, $C^2 = \{c_1^2, c_2^2, \ldots, c_n^2\}$ are created such that:

$$c_i^1 = \lambda p_i^{min} + (1 - \lambda) p_i^{max} \tag{1.1}$$

$$c_i^2 = (1 - \lambda) p_i^{min} + \lambda p_i^{max} \tag{1.2}$$

where $p_i^{max} = \max(p_i^1, p_i^2)$ and $p_i^{min} = \min(p_i^1, p_i^2)$. An extension, *BLX-*$\alpha$ crossover, is used to expand the selection region beyond the interval between the parents to allow the crossover function to be more exploratory[1]. This method is described in more detail in chapter 2.

Mutation for discrete-valued chromosomes consists of randomly choosing a value from the set of possible values for a number of genes in a chromosome. For this work, mutation is applied to the chromosomes after crossover is applied. The number of genes in each chromosome to mutate is defined as a hyperparameter. Mutation for real-valued chromosomes is done with a non-uniform mutation rate which decreases as the generations progress. This process is detailed in chapter 2.

For each new generation of chromosomes, a small number of the best performers are preserved untouched from the previous generation to allow the best genetic material to survive. These chromosomes are called elite. All chromosomes are ranked according to their fitness to the task and used in crossover proportional to their fitness. In other words, parents are selected for mutation by drawing chromosomes from the ranked list of chromosomes using a triangular distribution. Figure 1.7 shows a distribution laid on top of a sorted population from which parents would be selected for recombination. Parents are replaced after selection so that they may reproduce many times in one generation.

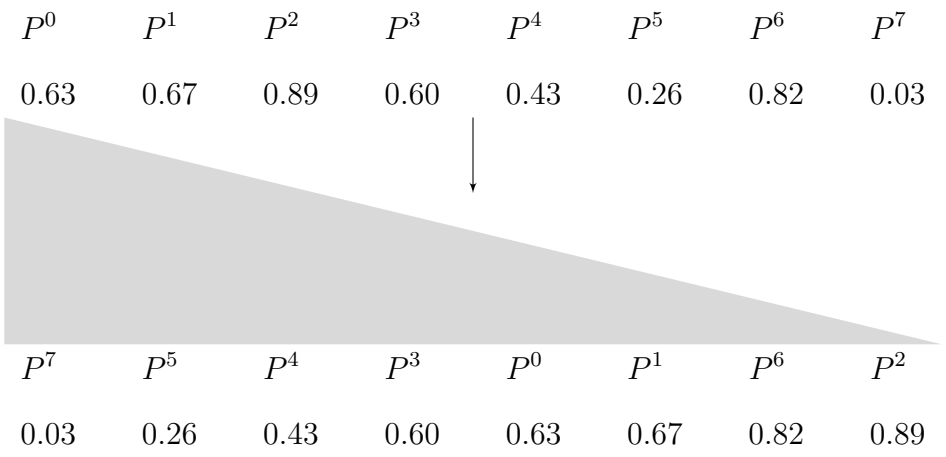


Figure 1.7: Population sorting and selection distribution overlay

1.3 Genetic Fuzzy Systems

Genetic Fuzzy Systems (GFS) here describe FL systems which have been tuned or learned using GA strategies. Tuning a GFS implies that the rule base is static, while the GA is allowed to operate on only the MF values or input/output scaling. Learning, on the other hand, implies that the RB itself is allowed to vary. For this research, except where otherwise stated, GFSs are allowed to learn the RB from scratch. In order to effectively use GAs to interact with FISs, some simplifying constraints are commonly imposed on the GFS to ease the use of genetic operators. One such constraint that is frequently used in this research is to mandate strict triangular or trapezoidal MFs. This allows the entire collection of MFs across all of the inputs to be described by either 3- or 4-tuples of sorted real values. As the GA is allowed to operate on the MFs, checks are placed to make sure that like MFs are recombined and that the children contain valid MFs. All of this is eased by constraining a GFS

to only contain triangular or trapezoidal MFs. Additionally, it is assumed that the RB completely covers the input combination possibilities. This is similar to a fully connected layer of neurons in a NN. This forces the number of rules to be equal to the product of the number of MFs across all inputs; as a result of this constraint, the number of rules can quickly explode as inputs are added to the system or inputs are granularized with additional MFs. To prevent RB explosion, FISs in this work are kept minimal. Knowing the number of MFs, the RB can be reduced to a single string of integers, where each integer is equal to the index of one of the output MFs. Thus in this way, we can fully describe a FIS with one heterogeneous string of values. For instance, the FIS illustrated in figures 1.2 to 1.3 with the RB in table 1.1 would be described completely by the list:

$$(0, 0, 55, 65)\,(55, 65, 75)\,(65, 75, 100, 100)\,|\,(0, 0, 0.25)\,(0, 0.25, 1)\,(0.25, 1, 1)\,[0, 1, 2]$$

where each MF tuple is grouped in (), input is separated from output with |, and the RB is in [ ]. The genetic operations on MF parameters are any operations which can work with real values, and the RB is constrained to n-point crossover.

1.4 Motivation and Problem Statement

This work explores the utility of using the GFS approach to exert control on dynamic systems. Because of the ability of a GFS to deal with imprecision and uncertainty, it can be an adaptive controller which can respond to changes in plant dynamics. Also, if the GFS is kept small, the trained system can be interpreted by a human operator with linguistic rules. With the rise in popularity of black box decision-making frameworks in recent years, interpretability has become a topic of

fervent research[13, 14, 15, 16]. Learning provides the benefit of exhibiting emergent behavior, but usually at the cost of interpretability. Genetic fuzzy systems may provide a middle ground; starting with a hand-crafted template, and then allowing a GA to learn behavior, the final model can still be assigned linguistic variables and given meaning.

This thesis is composed of a handful of problems which were approached using a GFS.

1. **Two-cart flexible system**. First, the problem of controlling a two-cart system is addressed. The system is allowed to move freely along a track with the objective of meeting, but not exceeding, a goal. This system exhibits two modes of behavior: over large distances, it behaves similar to a rigid structure; when finer control is needed in the terminal phase, the flexible body vibrations dominate the behavior. A controller is tuned to handle this system in chapter 2.

2. **F-4 Phantom Pitch Controller**. Secondly, a pitch attitude controller for the F-4 fighter jet is designed. This system is allowed to train on only a nominal flight condition, but then subjected to large degradations in plant dynamics and evaluated. The performance is shown to exceed that of a PID controller in chapter 3.

3. **sUAS Precision Landing** Finally, a controller for a small quadrotor is designed to guide the vehicle to a moving target using only visual feedback from an on-board camera. This problem is constrained by physical systems, so simulation environments, controllers, computations, and sensors are chosen such that the system will be more readily physically realized. The use of the Robot Operating

System with the Gazebo simulation environment are also discussed at length. This discussion comprises chapter 4.

Chapter 2: Two-cart Flexible System

2.1 Introduction

The 1992 American Control Conference published a set of benchmark problems to explore the capabilities of modern control techniques[17]. The second stated problem solicited solutions to the stabilization of a two-body spring mass system which was robust to uncertain masses and spring constants. This benchmark problem focuses on disturbance rejection in minimum time. Stengel[18] surveyed the best performing control strategies and quantified their stability and performance robustness using stochastic robustness analysis. Cohen et. al [19] independently proposed a solution using fuzzy logic which has more robust stability, performance and uses less control effort.

This problem extends the benchmark problem with the addition of automatically tuned fuzzy controllers for various plant models via a GA. The goal of the controller is to move the dual-mass system from a stationary initial condition to a stationary end goal in minimum time. Simplifications to the problem include a collocated sensor and actuator, removed plant disturbance, and removed sensor noise. The model dynamics are similar to those experienced by a robotic arm moving from one control point to another. The system exhibits both rigid and flexible body modes and thus provides

a good test case for a nonlinear fuzzy controller. The controller is first developed by hand and then automatic tuning strategies are employed using a GA. The plant model masses are then changed significantly and the controller is retrained to test the efficacy of the GA tuning process and to demonstrate the utility of having an automated tuning method. The result is a system which can be quickly trained to accommodate new plant dynamics within this class of models. The performance of the hand- and GA-tuned controllers are compared.

2.2 Coupled Spring-Mass Simulation Model

A simulated environment is necessary to test the efficacy of the proposed FIS. The simulation consists of two cars connected by a spring; this system is expected to traverse a given distance as quickly as possible without exceeding a certain boundary represented by a wall. The system is propelled by a force on the leading car which represents the actuating output of the controller. At each time instant, the controller must determine how much force to exert on the carts, and in which direction, to get as close as possible to the wall in minimum time. The carts are represented by point masses which may occupy the same position in space. The constants for the model are the weight of each car, the spring constant, and the distance which must be traversed to the wall. A diagram of the simulation is shown in figure 2.1.

$$m_1 = 1\,\text{kg}, \quad m_2 = 2\,\text{kg}, \quad K = 250\,\frac{\text{N}}{\text{m}}, \quad L = 100\,\text{m}$$

The modeled system contains displacement and velocity sensors on each cart; therefore, the four inputs to the FIS are the distances traveled and the velocities of

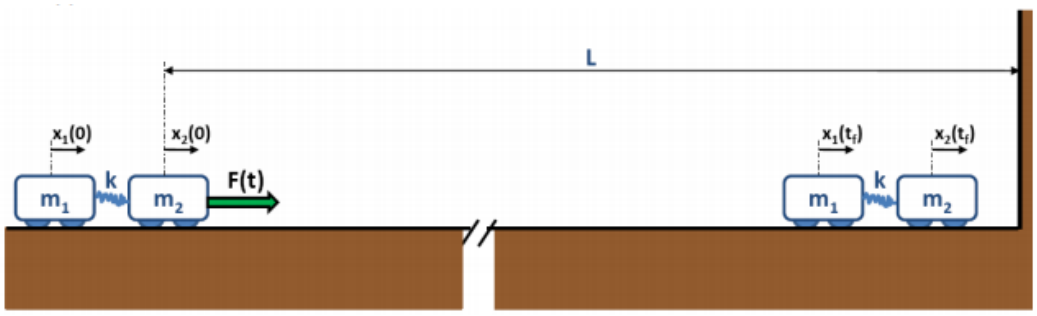


Figure 2.1: Diagram of two rigid bodies connected by a spring traversing distance L in minimum time.

both carts. These inputs represent the state vector of the system. The output of the controller is the force, $F(t)$, applied to cart 2, limited to $\pm 1\,\mathrm{N}$. At each time step of the simulation, the FIS uses the state of cart 2 to determine the force which must be exerted according to a fuzzy rule base.

$$\text{Input}: \quad \vec{y}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix}$$

$$\text{Output}: \quad |F(t)| \leq 1\,\mathrm{N}$$

At time $t = 0$, both carts are at rest at position $x = 0$. The maximum allowed run time of the simulation is 500 s. The system requirement for the final condition is that both carts come to rest between 99 m to 100 m with minimal oscillation. Neither cart is allowed at any point in the simulation to exceed 100 m.

$$\vec{y}_0 = \begin{bmatrix} 0\,\mathrm{m} \\ 0\,\mathrm{m} \\ 0\,\frac{\mathrm{m}}{\mathrm{s}} \\ 0\,\frac{\mathrm{m}}{\mathrm{s}} \end{bmatrix}, \quad \vec{y}(500) = \begin{bmatrix} 99\,\mathrm{m} < x_1 < 100\,\mathrm{m} \\ 99\,\mathrm{m} < x_2 < 100\,\mathrm{m} \\ 0\,\frac{\mathrm{m}}{\mathrm{s}} \\ 0\,\frac{\mathrm{m}}{\mathrm{s}} \end{bmatrix}$$

The acceleration of cart 1 is determined by the displacement between the two carts, the spring constant, and the mass of the cart. The acceleration of cart 2 is also a function of these parameters as well as the control force. The equations of motion for the system are represented by simple second-order differential equations.

$$\text{Cart1}: \quad \ddot{x}_1 = \frac{K}{m_1}(x_2 - x_1) \tag{2.1}$$

$$\text{Cart2}: \quad \ddot{x}_2 = \frac{K}{m_2}(x_1 - x_2) + \frac{F}{m_2} \tag{2.2}$$

2.2.1 System Performance Cost

The efficiency of a FIS's control of the system is based upon the amount of time it expends traversing the distance to the wall and how close the carts are to the wall when they settle. Any breach of the wall results in immediate system failure. A cost function, J, is defined by the settling time t_f, the time taken to settle within 1 m of the wall and the steady state error, the distance between the leading cart and the wall. The control system that produces the lowest J value will be proven to be the most fit solution for the simulation. This cost function in particular was provided in order to compare to results previously obtained[20, 21, 22, 23].

$$t_f = |L - \bar{x}(t_f)| \leq 1\,\text{m} \tag{2.3}$$

$$J = \frac{t_f}{100} + 2[L - x_2(500)] \tag{2.4}$$

where the constants 100 and 2 are scaling factors. In order to provide a frame of reference for the performance of any control system, the theoretical limits of the

simulation were calculated to provide a lower bound for the value of equation (2.4). Assuming a single rigid body assembly with no dynamic coupling, the model is greatly simplified to a single equation where the acceleration of the body is a function of only the control force. Since no energy is lost to a spring, the optimal solution to this system is assumed to be a lower bound on the flexible system with losses.

$$\ddot{x} = \frac{F}{M} \tag{2.5}$$

where M is the total mass of the system. The total mass of the system is $3\,\mathrm{kg}$, whereas the maximum input force is limited to $1\,\mathrm{N}$, rendering equation (2.5)

$$\ddot{x} = \frac{1\,\mathrm{N}}{3\,\mathrm{m}} = \frac{1}{3}\,\frac{\mathrm{m}}{\mathrm{s}^2}$$

as the maximum acceleration. Given this acceleration and the distance to be traversed to the wall, the minimum time to complete the trip can be calculated. There are, however, two scenarios to consider.

1. Applying maximum force over the entire distance and instantaneously stopping the carts at (but not touching) the wall provides an absolute, if unfeasible, optimum simulation completed in minimum time. Given an initial velocity of zero and a constant accelerating force of $1\,\mathrm{N}$, the traversal time is calculated. Note that once the carts have breached $99\,\mathrm{m}$, the system may come to rest and be considered settled.

$$x(t) = \dot{x}_0 t + \frac{1}{2}\ddot{x}t^2, \quad \dot{x}_0 = 0$$

$$x(t) = \frac{1}{2}\ddot{x}t^2$$

Letting $x(t) = 99\,\mathrm{m}$

$$t = \sqrt{\frac{2x(t)}{\ddot{x}}} = \sqrt{\frac{2 \cdot 99\,\mathrm{m}}{\frac{1}{3}\,\frac{\mathrm{m}}{\mathrm{s}^2}}} = 24.37\,\mathrm{s}$$

2. Applying maximum force over half of the distance and then applying maximum negative force in the second half to slow the velocities of the carts to zero at the wall position.

   First half:

$$t = \sqrt{\frac{2x(t)}{\ddot{x}}} = \sqrt{\frac{2 \cdot 50\,\mathrm{m}}{\frac{1}{3}\,\frac{\mathrm{m}}{\mathrm{s}^2}}} = 17.23\,\mathrm{s}$$

   Traversing the second half and stopping at the wall takes the same amount of time, therefore the total time expended in reaching 100 m is 34.46 s; however, the interest lies in the time taken to breach the 99 m mark. The time taken to travel the last meter is 2.45 s, so the best possible time to reach the 99 m position is :

$$t = 32.19\,\mathrm{s}$$

   As this is a much more realistic scenario, this is the limit used as the benchmark in this research. Using this time to evaluate equation (2.4)

$$J = \frac{32.19}{100} + 2[100 - 100] = 0.3219$$

   results in the minimum possible cost. The addition of harmonic oscillation and non-linear dynamics ensures that this limit will not be reached, but merely provides a standard against which a controller may be measured.

2.3 Fuzzy Inference System

The FIS built during this project uses two measured inputs: the position and velocity of cart 2. The output of the FIS is the force exerted on cart 2 at a given point in the simulation.

2.3.1 Membership Functions and Rule Base

Position

The first input variable, position, is composed of three membership functions to which it can map. The position of the car, from 0 m to 100 m, is described in human-understandable language as far away, close to, or very close to the wall. If the simulation has just begun and the cart is as far away from the wall as it can be, the "Far" membership function will map to 1 and the "Close" and "VeryClose" functions will evaluate to 0. Conversely, at the end of the traverse, as the car approaches the wall, "VeryClose" will map to 1 and "Far" to 0. "Close" may evaluate to some value in between. The membership functions are shown in figure 2.2. As the system predominately behaves as a rigid body on a large scale, the membership functions mirror the ideal simulation of a rigid body for the majority of the carts' travel. Each function is represented by a vector of values expressing the points at which the function switches from 0 to 1 or 1 to 0. For the FIS used in this research, trapezoidal- and triangular-shaped membership functions are utilized. Trapezoids are represented by a four-element vector as they start at 0, rise to 1, remain at 1, and finally fall to 0. Likewise, triangular functions are expressed as three-element vectors. The parameters of the position membership functions shown in figure 2.2 are:

$$\text{Far}: \begin{bmatrix} 0\,\text{m} \\ 0\,\text{m} \\ 49.8\,\text{m} \\ 50.1\,\text{m} \end{bmatrix}, \quad \text{Close}: \begin{bmatrix} 49.8\,\text{m} \\ 50.1\,\text{m} \\ 99.9\,\text{m} \\ 100\,\text{m} \end{bmatrix},$$

$$\text{VeryClose}: \begin{bmatrix} 99.9\,\text{m} \\ 100\,\text{m} \\ 100.1\,\text{m} \end{bmatrix}$$

Velocity

The second input variable, velocity, is composed of three membership functions. The velocity of the cart, within a range from $-6\,\frac{\mathrm{m}}{\mathrm{s}}$ to $6\,\frac{\mathrm{m}}{\mathrm{s}}$, can either be classified as "Negative", "Zero", or "Positive". When the simulation starts, the carts are at rest and the degree of membership to "Zero" velocity will be 1 whereas "Negative" and "Positive" will be 0. For the majority of the simulation, the carts are moving forward with a fast pace; therefore, the "Negative" membership function does not come into play until close to the very end of the simulation when the oscillatory effects dominate the motion of the cart system. It is at this point that the true dynamic nature of the system is exhibited and the controller does the most calculation in an attempt to damp the oscillations. The membership functions for $\ddot{x}_2$ are shown in figure 2.3.

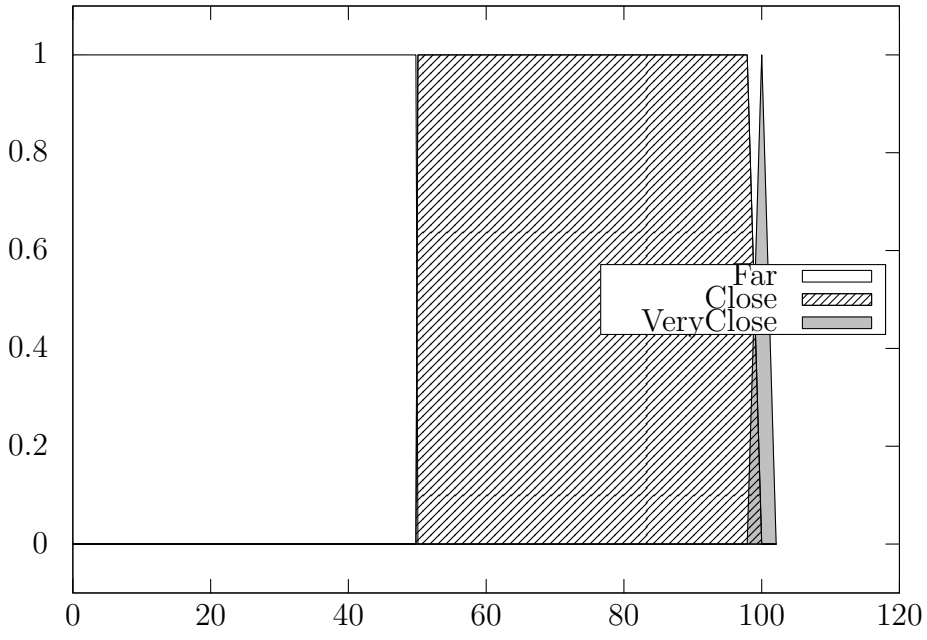


Figure 2.2: x_2 membership functions

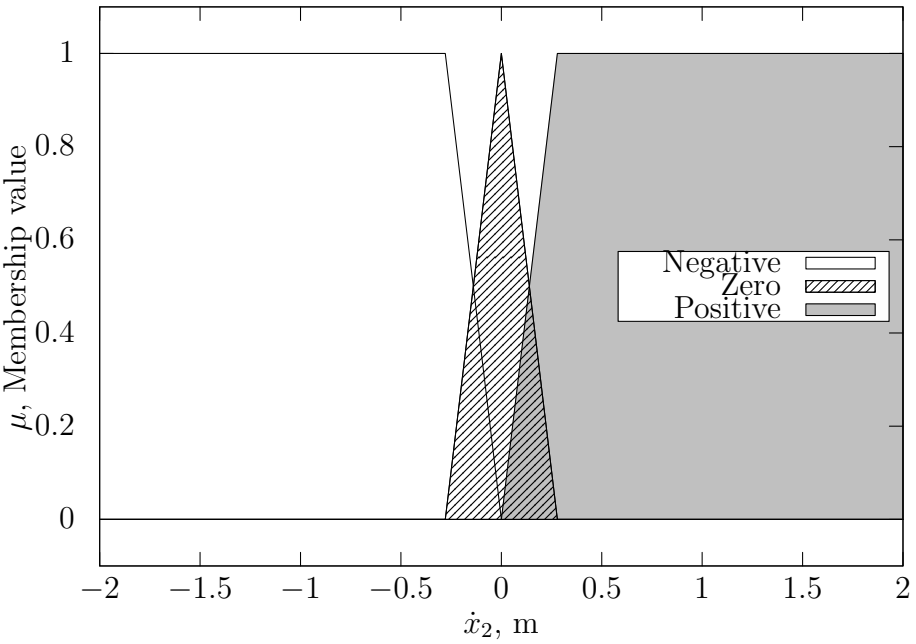


Figure 2.3: $\dot{x}_2$ membership functions

The velocity membership function parameters are:

$$\text{Negative}: \begin{bmatrix} -6\,\frac{\text{m}}{\text{s}} \\ -6\,\frac{\text{m}}{\text{s}} \\ -0.2792\,\frac{\text{m}}{\text{s}} \\ 0\,\frac{\text{m}}{\text{s}} \end{bmatrix}, \quad \text{Zero}: \begin{bmatrix} -0.2792\,\frac{\text{m}}{\text{s}} \\ 0\,\frac{\text{m}}{\text{s}} \\ 0.2792\,\frac{\text{m}}{\text{s}} \end{bmatrix},$$

$$\text{Positive}: \begin{bmatrix} 0\,\frac{\text{m}}{\text{s}} \\ 0.2792\,\frac{\text{m}}{\text{s}} \\ 6\,\frac{\text{m}}{\text{s}} \\ 6\,\frac{\text{m}}{\text{s}} \end{bmatrix}$$

Rule Base

As previously discussed, the rule base of an FIS is a series of IF-THEN (antecedent-consequent) conditions that use the membership functions of all the inputs in order to determine the output variable. There are also several membership functions for the output variable that the rule base maps to. Each of the rules has an influence on

what the output of the system should be. The weight of these influences again varies from 0 to 1 and the final output value is determined by calculating the centroid of all of the rules. For instance, one rule dictates that if the car is far away then the output force should be positive and large so that the car will move towards the wall. Another rule will say that when the car is close to the wall the output force should be negative in order to slow the car down. All of these rules have influence over all input ranges determined by how the input variables map to the given membership functions in that specific rule.

The antecedent of each statement contains memberships of both input variables and the consequent maps to the output membership functions. The rules for this FIS were developed based upon intuitive decision making. The rules developed for our FIS are shown in table 2.1.

Table 2.1: Rule Base of the Fuzzy Inference System

Position Measurement	Velocity Measurement: **Negative**	**Zero**	**Positive**
Far	Positive		
Close	Positive	Zero	Negative
VeryClose	Positive	Zero	Negative

Control Force

Control Force

The output variable, force, is also composed of three membership functions. The output force is bounded from -1 N to 1 N thus the membership functions allow for the output to be "Negative", "Zero", or "Positive". As the centroid of the area beneath

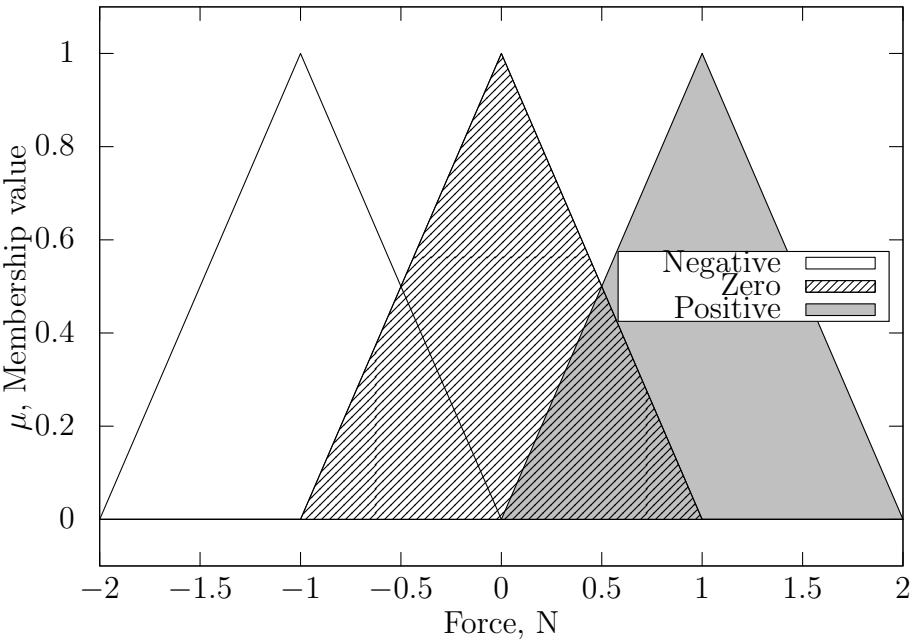


Figure 2.4: Control force membership functions.

each function is used to evaluate the control force, the upper and lower bounds are centered over 1 and -1 respectively. These functions represent the "defuzzification" stage of the FIS and these outputs are determined by evaluating each rule in the rule base (see section 2.3.1) in parallel[24]. The controller emulates bang-bang control for the majority of the simulation lifetime, sharply transitioning from full acceleration to full deceleration. As the cart system nears the wall, the oscillation damping rules will dictate the force to apply on the system. Typically, the output force will be directed opposite to the velocity of cart 2. The membership functions for the control force are shown in figure 2.4.

$$\text{Negative}: \begin{bmatrix} -2\,\text{N} \\ -1\,\text{N} \\ 0\,\text{N} \end{bmatrix}, \quad \text{Zero}: \begin{bmatrix} -1\,\text{N} \\ 0\,\text{N} \\ 1\,\text{N} \end{bmatrix}$$

$$\text{Positive}: \quad \begin{bmatrix} 0\,\text{N} \\ 1\,\text{N} \\ 2\,\text{N} \end{bmatrix}$$

2.3.2 Simulation Performance

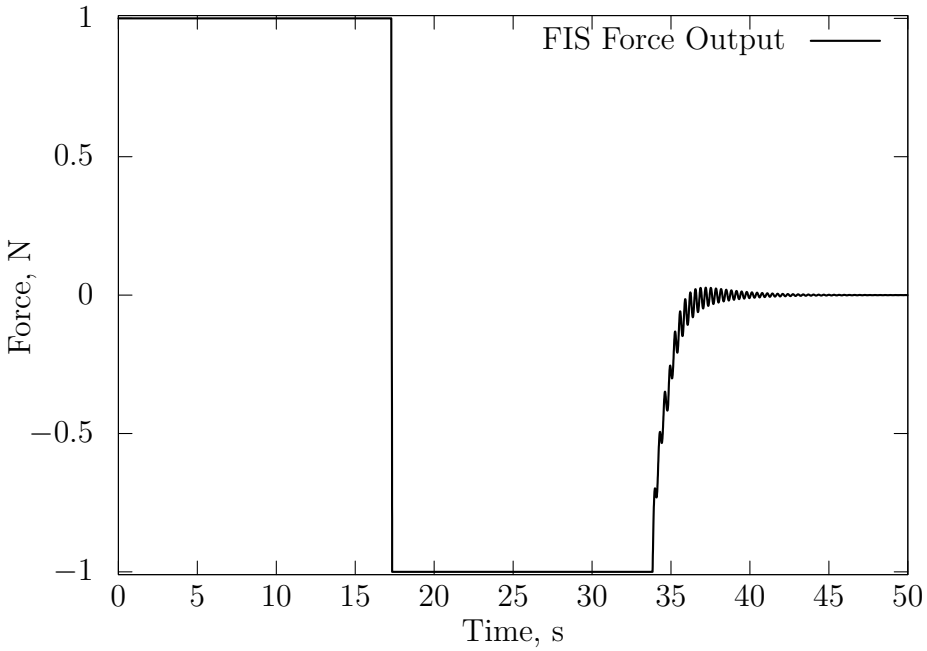


Figure 2.5: System control force over time.

The efficiency of this FIS was determined by using it as the control mechanism in the simulation. The goal was to approach the realistic theoretical limits calculated above and reach this goal with minimum force. Table 2.2 shows the settling time, final position of cart 2 and the calculated J value for the controller's performance. Figure 2.5 shows the controller's output force over time for each step of the simulation.

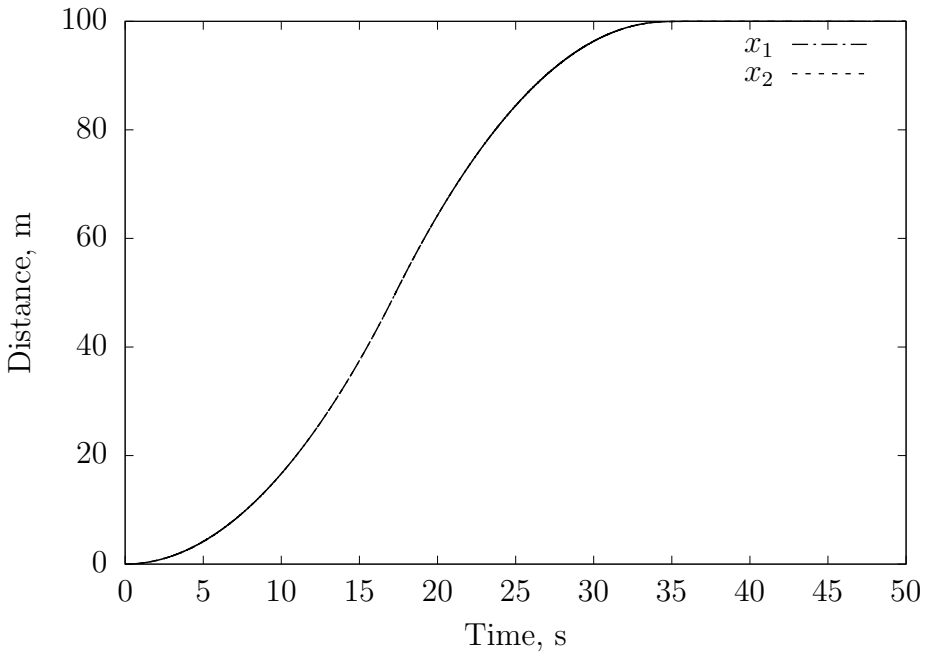


Figure 2.6: Cart position time history. Note that the displacement between the carts is negligible.

It is clear that the controller expends less energy than a traditional bang-bang type controller would in the oscillation damping process. It can be seen that the FIS responds quickly to control needs and applies the needed force with low-latency.

Much work was done to hand-tune this controller to perform optimally, thus the work was undertaken to develop a genetic algorithm to produce similar results autonomously. Automating the tuning process ultimately produces a controller which is as good as the hand-tuned controller, but requires little to no effort from the programmer.

Table 2.2: Results from hand-tuned FIS.

t_f	$x_2(500)$	J
32.303 s	99.999 821 m	0.32328

2.4 Genetic Algorithm

The performance of the FIS depends directly on the value of each parameter of the membership functions. These values were hand-tuned by a time-consuming process of trial and error. To quicken this process, a genetic algorithm was utilized to autonomously tune the membership functions and approach an optimal solution.

As previously discussed, a genetic algorithm is a computational mechanism which imitates evolutionary behavior to achieve optimality. It consists of a population of individuals which undergoes a process similar to natural selection, reproduction, and mutation over the course of a number of generations[1]. Selection is attained by evaluating the fitness of each individual FIS according to equation (2.4). The individual which most effectively minimizes the cost function is considered most fit for the control environment.

In order to manipulate the FIS structure in an algorithmic manner, it is necessary to represent it as a genetic individual. Since the values of the parameters of the membership functions of the FIS have significant impact on the control performance, it was decided to manipulate only these parameters with the algorithm; however, of the thirty-one parameters which comprise this FIS model, many are trivial to the overall performance. It is desirable to reduce the number of parameters to facilitate the optimization process.

2.4.1 State Reduction

To simplify the genetic tuning of the parameters, the symmetry of the system was exploited. The parameters were reduced from thirty-one to seven.

The position membership functions were simplified to only three parameters by defining a center point (center) between far and close functions, a distance from the center point at which the far and close functions will be valued at 0 and 1 respectively (iTrap1), and half of the base of the triangular membership function which decides when the car is very close to the wall (iTriBase1). The velocity membership function parameters were reduced to two parameters. One parameter describes the distance from 0 to the point at which each of the velocities reach 1 (iTrap2). The other parameter is defined as half of the base of the zero velocity triangular membership function (iTriBase2). The output force membership functions were reduced similarly by allowing the negative and positive membership functions to become trapezoidal (oTrap and oTriBase).

- Position Membership Function Parameter

$$\text{Far}: \begin{bmatrix} 0 \\ 0 \\ (center - iTrap1) \\ (center + iTrap1) \end{bmatrix}, \quad \text{Close}: \begin{bmatrix} (center - iTrap1) \\ (center + iTrap1) \\ 99.9 \\ 100 \end{bmatrix},$$

$$\text{VeryClose}: \begin{bmatrix} (100 - iTriBase1) \\ 100 \\ (100 + iTriBase1) \end{bmatrix}$$

- Velocity Membership Function Parameters

$$\text{Negative}: \begin{bmatrix} -6 \\ -6 \\ (0) - iTrap2) \\ 0 \end{bmatrix}, \quad \text{Zero}: \begin{bmatrix} (0 - iTriBase2) \\ 0 \\ (0 + iTriBase2) \end{bmatrix},$$

$$\text{Positive}: \quad \begin{bmatrix} 0 \\ (0 + iTrap2) \\ 6 \\ 6 \end{bmatrix}$$

- Control Force Membership Function Parameters

$$\text{Negative}: \quad \begin{bmatrix} -2 \\ (-1 - oTrap) \\ (-1 + oTrap) \\ 0 \end{bmatrix}, \quad \text{Zero}: \quad \begin{bmatrix} (0 - oTriBase) \\ 0 \\ (1 + oTriBase) \end{bmatrix},$$

$$\text{Positive}: \quad \begin{bmatrix} 0 \\ (1 - oTrap) \\ (1 + oTrap) \\ 2 \end{bmatrix}$$

These state reductions allow an individual to be defined by a single vector of seven variables.

- Individual Definition

$$\begin{bmatrix} iTrap1 \\ center \\ iTriBase1 \\ iTrap2 \\ iTriBase2 \\ oTrap \\ oTriBase \end{bmatrix}$$

2.4.2 Population Initialization

An initial population is generated by assigning random values to each of the individual parameters within given ranges. iTrap1, iTriBase1, and oTriBase are allowed to vary between 0.05 and 1. iTrap2 and iTriBase2 are allowed to vary between 0.05 and 2. Center values fall between 45 and 55, and oTrap between 0.05 and 0.95. Twenty individuals comprise a population. Each individual is evaluated for fitness and brought up for selection to produce a new generation.

2.4.3 Parent Selection and Reproduction

A new generation consists of three individuals which remain unchanged from the previous generation, called elite individuals, ten children which are created from recombination of two parents each, five individuals which are created from mutating recombined children, and two individuals randomly defined from the previously defined ranges. The parents for this population consist of the three elite individuals as well as seven more individuals. Each of the seven is selected by randomly choosing three individuals from the previous population, selecting the most fit, and returning the other two. This tournament style of selection is repeated until all parents are selected.

Reproduction occurs by blended crossover process with an α modification (BLX-α), by selecting a new parameter x_i' from the range $[x_{min} - I\alpha, x_{max} + I\alpha]$, where

$$x_{min} = min(x_i^1, x_i^2) \quad \text{and} \quad x_{max} = max(x_i^1, x_i^1)$$

Parents are defined as

$$C^1 = (x_1^1, x_2^1, \cdots, x_7^1) \quad \text{and} \quad C^2 = (x_1^2, x_2^2, \cdots, x_7^2)$$

$$I = \frac{x_{max} - xmin}{b_i - a_i}$$

$$\alpha = min(d^1, d^2)$$

$$d^1 = x_{min} - a_i \quad \text{and} \quad d^2 = b_i - x_{max}$$

The interval $[a_i, b_i]$ is the parameter-specific range. This mechanism allows the algorithm to create a child from two parents which is a blend of both parents, while still expanding the search space. As a population generally converges on a solution, so too do the children of the population.

2.4.4 Mutation

Five of the recombined children are selected by random sampling and then two randomly sampled parameters are selected from each of these child for mutation. Mutation is defined to be nonuniform such that the mutation has a smaller effect in later generations as follows:

$$x_i' = \begin{cases} a_i + \Delta(t, x_i - a_i), & \text{if } \tau = 0 \\ b_i - \Delta(t, b_i - x_i), & \text{if } \tau = 1 \end{cases}$$

where τ represents a coin flip such that $P(\tau = 1) = P(\tau = 0) = 0.5$

$$\Delta(t, x) = x(1 - \lambda(1 - \frac{t}{t_{max}})^b)$$

where t is the current generation, and t_{max} is the maximum number of generations. The variable λ is a random value from the interval $[0, 1]$. The function Δ computes a value in the range $[0, x]$ such that the probability of returning a zero increases as the algorithm advances. The value of b determines the impact of the time on the probability distribution of Δ. The value of b is set to 1.5 for the algorithms used in this research.

2.5 Results

Running the algorithm for 50 generations yields a FIS which performs as well as the hand-tuned FIS from section 2.3.2. This result converges out of the evolution process quickly as can be seen in figure 2.7. Though the algorithm finds a near optimal solution quickly, it continues to search similar solutions, selecting the best individuals each time to produce progressively better fit children each generation. Figure 2.8 shows the average fitness of generation. It is clear from this plot that the algorithm

Table 2.3: Final Results from algorithm-generated FIS

t_f	$x_2(500)$	J
32.308 s	99.999 999 m	0.32311

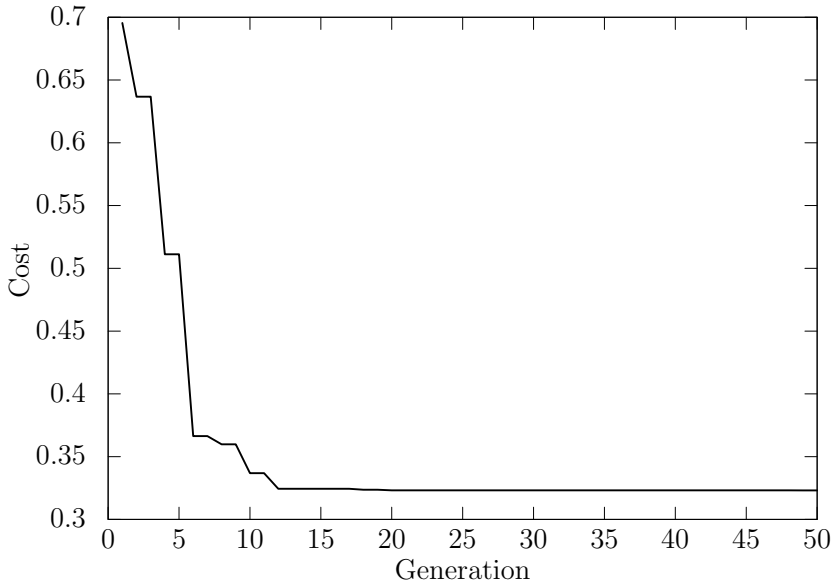


Figure 2.7: Best fit individual by generation.

produces many unfit children in its search for optimality. This satisfies the need of a good algorithm to expand the search area to eliminate premature convergence.

The results of the performance of the most fit individual produced by the genetic algorithm are shown in table 2.3. It is shown in figure 2.9 that the output force of the GA-tuned controller is very similar to the hand-tuned version. Examining figure 2.10 shows that the GA has changed the membership functions appreciably within the allotted envelopes while still yielding a near-optimal result.

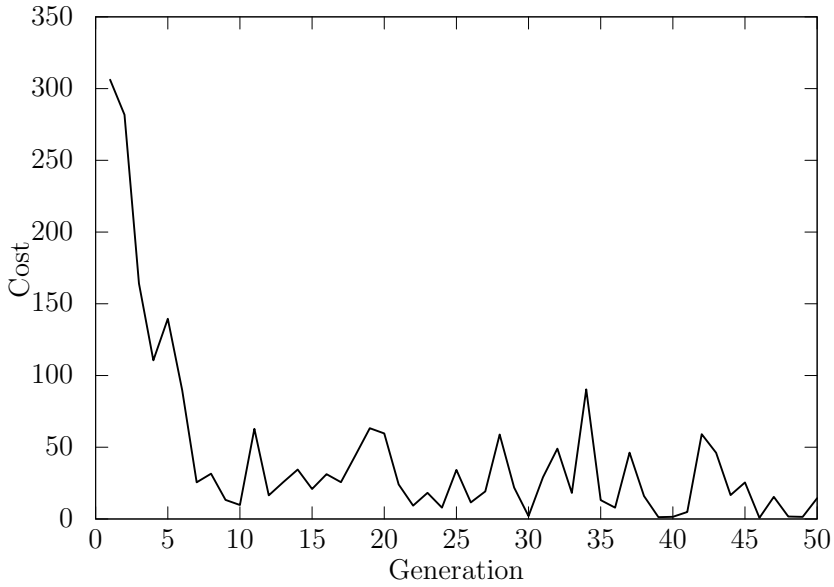


Figure 2.8: Individual fitness average by generation.

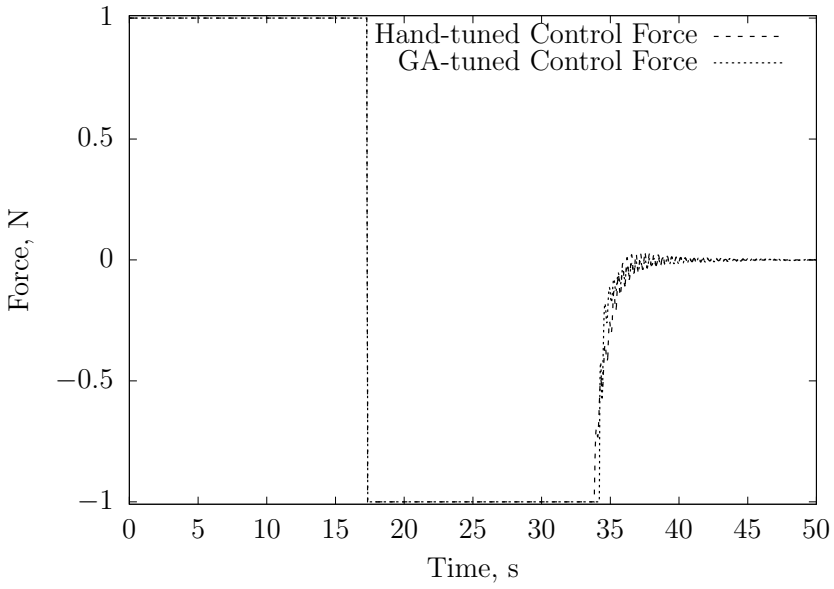
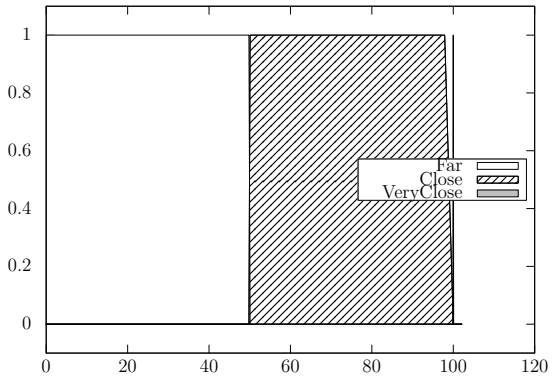
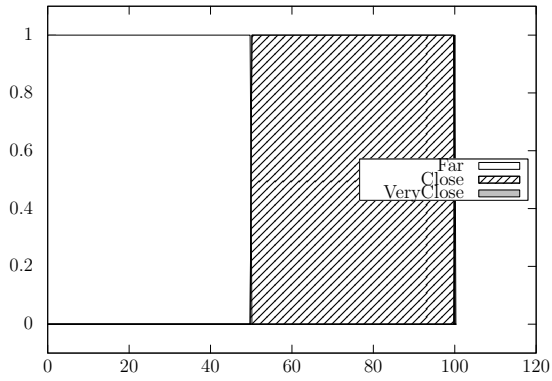


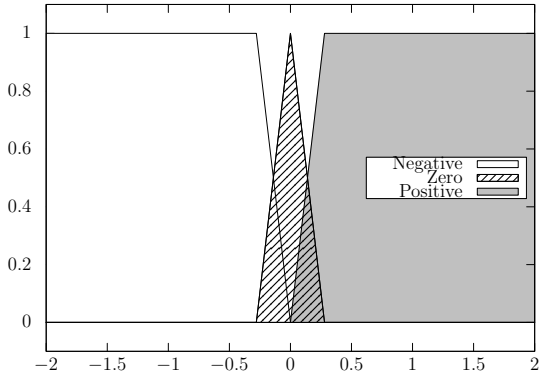
Figure 2.9: The force signals over time from the hand- and GA-tuned controllers.



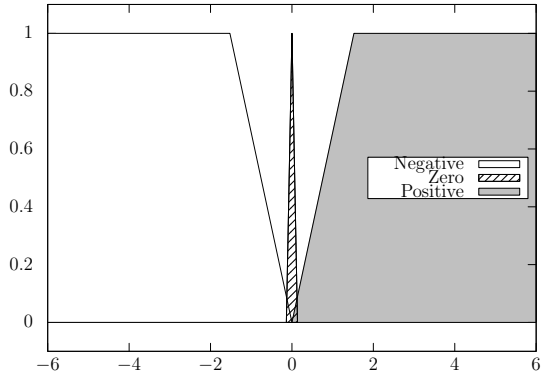
(a) x_2 input partition before GA tuning



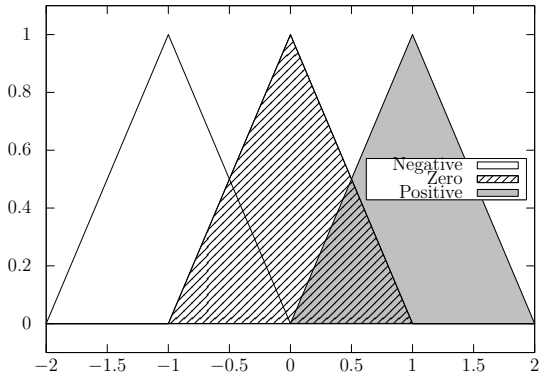
(b) x_2 input partition after GA tuning



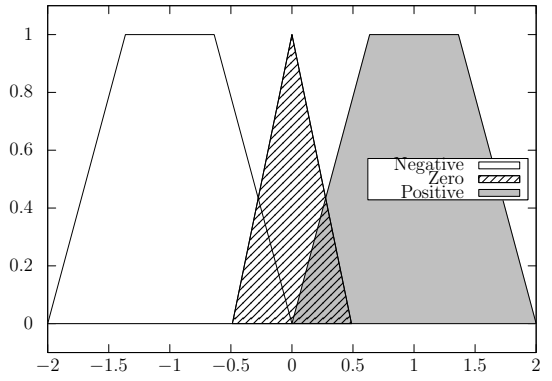
(c) $\dot{x}_2$ input partition before GA tuning



(d) $\dot{x}_2$ input partition after GA tuning



(e) Force output partition before GA tuning



(f) Force output partition after GA tuning

Figure 2.10: FIS comparison between hand- and GA-tuned membership functions

Table 2.4: Genetic algorithm FIS performance compared to the hand-tuned FIS and rigid body limit.

	Mass 1 (kg), Mass 2 (kg)			
	1 kg, 2 kg	2 kg, 4 kg	4 kg, 8 kg	4 kg, 16 kg
Theoretical Limit	0.3191	0.4553	0.6438	0.8312
GA FIS	0.3231	0.4562	0.6579	0.8434
Hand-tuned FIS	0.3233	0.6125	5.3072	3.3240
GA Error	1.3%	0.2%	2.2%	1.5%
Hand-tuned Error	1.3%	34.5%	724.4%	299.9%

2.5.1 Genetic Adaptability

These results demonstrate the ability of the genetic algorithm to tune a FIS to near-optimal performance. All tuning and development heretofore was done with an unchanging system setup of known masses connected by a known spring. Although a robust controller[19], introducing changes to the masses of each car significantly alters the performance of the FIS as it was carefully tuned to only a certain envelope; however, utilizing the genetic algorithm to generate an optimum FIS for each new system setup is an efficient method of developing good active controllers. This is demonstrated by changing the masses m_1 and m_2 to 2 kg and 4 kg respectively. They are again changed to 4 kg and 8 kg, and finally 4 kg and 16 kg. The algorithm was deployed for each case to optimize a controller for that envelope. After fifty generations of evolution, the genetically optimized FIS performed within 3% of the theoretical rigid body limit in all four cases. These results are displayed in table 2.4.

It is easily seen in these results that the use of the genetic algorithm is advantageous in the autonomous development of near optimal FIS controllers. Given a generic

control architecture, the genetic algorithm is able to tune a FIS rapidly and accurately for a varied set of circumstances.

2.6 Conclusions

Fuzzy logic provides a robust framework for control. It has been demonstrated that proper fuzzy control is efficient and computationally inexpensive. The inherent vagueness of set membership and linguistic operation of fuzzy logic allows the controller to mimic expert human control. This superior control, however, comes with a steep cost in FIS development. Hand-tuning a FIS is time-consuming and tedious.

The use of the genetic algorithm facilitates FIS development. Once a FIS has been developed for a general type of control situation, it is relatively simple to define the FIS as a genetic element and automate the tuning through the evolutionary process. These results imply that if a general fuzzy controller is developed for a family of control situations, then a genetic algorithm can be implemented to tune each FIS to its specific task. The tuning, therefore, can be accomplished by someone with no expertise in the control of the situation. As the computation is quick, efficient control could be widely distributed due also to the low-cost of development.

Chapter 3: F-4 Pitch Attitude Control

3.1 Introduction

This chapter outlines a fuzzy PID system for the approach condition of an F-4 fighter jet. The F4 stands in for more modern jets with fly-by-wire systems, and was chosen because the equations of motion were available. The task at hand is to design a pitch attitude hold controller for the F4 that is robust enough at be used across the entire flight envelope without modification. The controller was designed for the approach condition due to its complexity.

The main challenge of this problem lies in the fact that the flight conditions that a fighter jet experiences are wide and varied. It is not feasible to design and implement an optimal controller for each one individually. A common approach to this situation would be to derive a gain schedule which allows the controller to automatically adjust the linear gains to meet the control needs at any time. However, the approach explored in this chapter is to train a GFS to determine a set of PID gains based on state feedback. This allows the controller to adapt to new flight situations similarly to a PID gain schedule, but without the need to explicitly tune a PID controller for each case.

3.2 Fuzzy PID

While seeking to design a controller robust enough to be used across the entire flight envelope, the approach condition was chosen for its challenging dynamics. Once the controller is tuned for the approach condition , it is tested across many off-design cases to determine its robustness. The cases chosen for robustness analysis are the approach condition with a 50% reduction in aerodynamic coefficients , subsonic cruise condition , and the supersonic cruise condition (see table 3.1). Each of these scenarios presents a challenging dynamic control problem with poles very close to the right half of the plane. Fuzzy logic is well-suited to this task in that it will allow the control gains to move around in a way which may keep the poles in the left half of the plane. The system dynamics studied here were used by Bossert and Cohen[25] to design both a PID and a Fuzzy controller, which will be used as benchmarks. The equations for the four scenarios

One large downside to fuzzy systems are that they require a significant expense of effort to tune. This task becomes even more cumbersome with an increase of membership functions, which, in turn, increase the number of rules (possibly exponentially). The FIS designer is then relegated to either drawing on a store of heuristic knowledge or experience (perhaps from an expert), or to trial-and-error iteration to incrementally improve the controller. The method used here is a GA which plays the part of automating and guiding the trial-and-error process (see next section). This method effectively circumvents the difficulty of tuning a FIS, and makes the job of controlling this near-unstable system more feasible.

Table 3.1: Transfer functions for various flight conditions of the F-4 fighter jet

Flight Condition	**Transfer Function**
Approach	$\frac{3361s^2 + 1357s + 102.2}{230.6s^5 + 2508s^4 + 2161s^2 + 63.04s + 32.01}$
Approach 50% deg	$\frac{3361s^2 + 1372s + 105}{230.6s^5 + 2472s^4 + 1731s^3 + 744.8s^2 + 36.92s + 16}$
Subsonic Cruise	$\frac{9.99\text{e}4s^2 + 5.105\text{e}4s + 623.4}{877.6s^5 + 9884s^4 + 1.82\text{e}4s^3 + 7.114\text{e}4s^2 - 61.19s - 114.1}$
Supersonic Cruise	$\frac{1.3\text{e}5s^2 + 2.183\text{e}4s - 31.29}{1742s^5 + 1.83\text{e}4s^4 + 3.553\text{e}4s^3 + 2.677\text{e}5s^2 + 1247s + 123.9}$

3.3 System Controller Design Methodology

At first, an attempt was made to utilize Simulink® and MATLAB® to design a genetic algorithm (GA) which would simulate the plant transfer functions with a fuzzy-PID controller in the feedback loop. The GA, written in MATLAB, would call the Simulink model and evaluate the performance of the controller. This methodology presented a number of problems to the designer. The Simulink Fuzzy Logic Block seemed to slow down the execution of the control loop siginificantly, making a single simulation run up to an order of magnitude slower than one with only PID control. This shortcoming can be circumvented by using a lookup table in place of the fuzzy controller, but the overhead time needed to generate the lookup table negates the simulation speed gains.

In light of the issues with using a Simulink GA, the GA was implemented in traditional MATLAB using `ODE45` as the ordinary differential equation solver; however, due to the adaptive timestep of the solver, it is prohibitively difficult to accurately obtain error derivative and integral terms for use in the PID controller. This led the author to eventually write a simple ordinary differential equation solver in MATLAB which gives access to all errors at each time step, but the execution of a single simulation was expectedly and prohibitively slow.

As a result, a genetic fuzzy system was implemented using the C programming language[26]. This GA uses the comparable rk8pd (Runge-Kutta Prince-Dormand (8,9)) stepping function from the Gnu Scientific Library (GSL) to solve the ordinary differential equation. This method allows the user to specify some basic parameters which are desired in the final FIS, a cost function to be applied to each solution, and some basic parameters for the GA, at which point the GA starts generating and evaluating FIS's until a suitable stop condition has been met.

To avoid the need to tune three individual FISs, a multi input multi output FIS was created which outputs the three gains K_p, K_i, and K_d. The FIS accepts as its inputs the absolute error in elevator angle, the error integral, and the error derivative, e, e_i, and e_d respectively. To determine the quality of a controller, a number of metrics with respect to its response to a step input are gathered as a means for comparison. These metrics are the settling time, T_s, rise time, T_r, time to peak value, T_p, maximum value, M_p and final value, FV.

The cost function used to describe the desired controller behavior plays an integral role in the success or failure of any evolutionary strategy. The problem of designing this fuzzy PID controller presents a nice case study in which to observe undesired

side effects of an overly simplistic cost function. The first cost function which is implemented is simply to minimize the settling time of the system. This has the side effect of increasing the overshoot of the elevator pitch angle. The cost function is then adjusted to take into account the overshoot and yields a more amenable response behavior.

3.4 Results for Nominal and Other Flight Conditions

Figure 3.1 shows the responses of the aircraft to the fuzzy PID controller as trained by a GA using the simple minimal settling time cost function. This cost function yields a significant overshoot as an outcome. To lessen the magnitude of this effect, the GA was given a new cost function which places a higher premium on overshoot, yielding a more amenable result, as shown in figure 3.2.

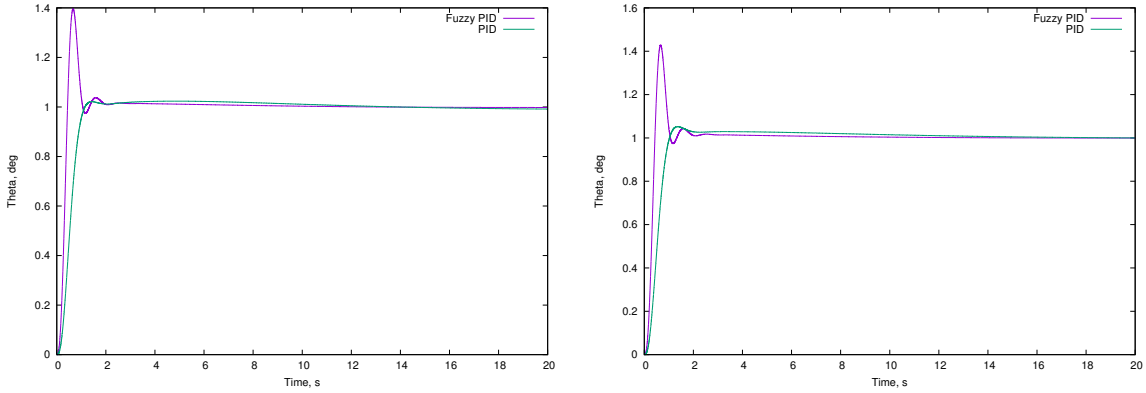
As can be seen from tables 3.2 to 3.3, the settling time of the step response is significantly slower when training a controller using the new cost function. However, the overall response is arguably better due to the reduced overshoot. Additionally, though not nearly as good for the nominal and degraded flight conditions as was Bossert and Cohen's solution, the response for the subsonic and supersonic cruise conditions show arguably better performance (see table 3.4).

Table 3.2: Comparison table of current results with those of Bossert and Cohen

Case	T_s (s)	T_r (s)	T_p (s)	M_p (deg)	FV (deg)
Root Locus F4 Approach (Design Condition)	23.6	1.09	3.14	1.09	0.62
PID F4 Approach (Design Condition)	7.08	1.27	4.98	1.023	1.00
Fuzzy F4 Approach (Design Condition)	1.65	1.68	1.74	1.013	1.00
Fuzzy PID Approach (Design Condition)	1.86	0.26	0.66	1.396	1.00
Root Locus 50% Red in $C_{m\alpha}$ and C_{mq}	25.8	1.16	3.57	1.42	0.77
PID 50% Red in $C_{m\alpha}$ and C_{mq}	8.0	1.03	1.35	1.045	1.01
Fuzzy 50% Red in $C_{m\alpha}$ and C_{mq}	1.66	1.69	1.75	1.014	1.00
Fuzzy PID 50% Red in $C_{m\alpha}$ and C_{mq}	1.87	0.25	0.66	1.428	1.00

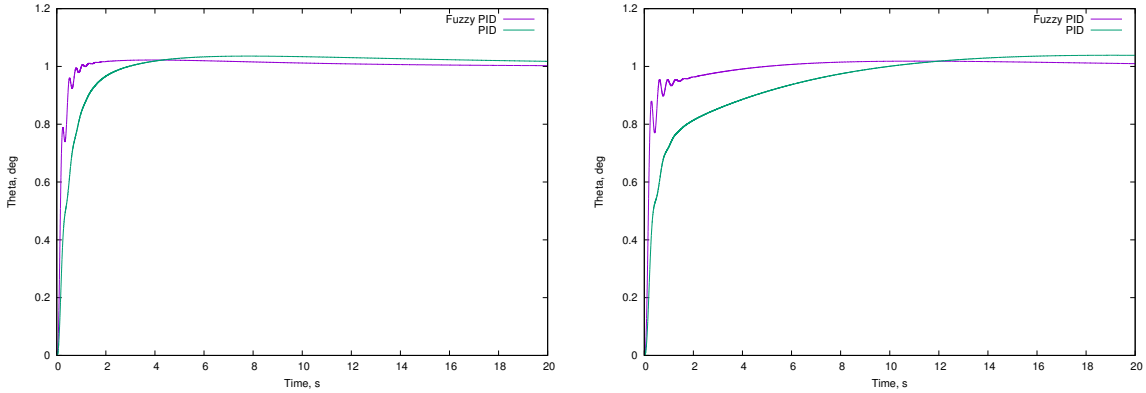
Table 3.3: Resulting FIS response from GA with modified cost function

Case	T_s (s)	T_r (s)	T_p (s)	M_p (deg)	FV (deg)
Fuzzy PID Approach (Design Condition)	4.96	0.34	1.43	1.031	1.00
Fuzzy PID 50% Red in $C_{m\alpha}$ and C_{mq}	4.47	0.33	0.66	1.043	1.00



(a) Nominal approach condition

(b) Degraded aerodynamic derivative approach condition



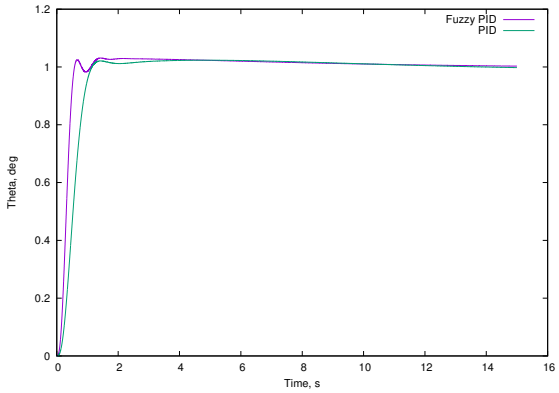
(c) Subsonic cruise condition

(d) Supersonic cruise condition

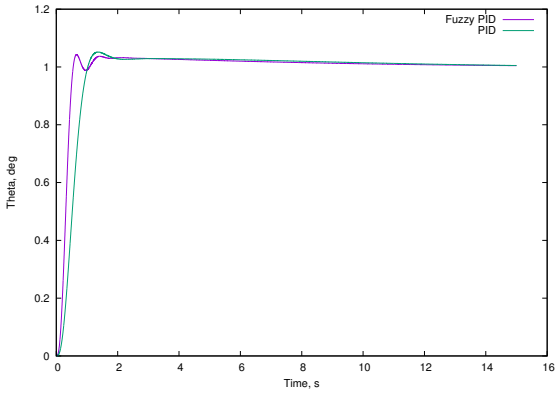
Figure 3.1: Step response for various flight conditions. Note the increased overshoot for nominal conditions from the Fuzzy-PID controller.

Table 3.4: Comparison of response times for subsonic and supersonic cruise conditions for original and modified cost function. Modifying the cost function showed no improvements for this set of conditions, rather only proved to slow the response time with no gain in overshoot.

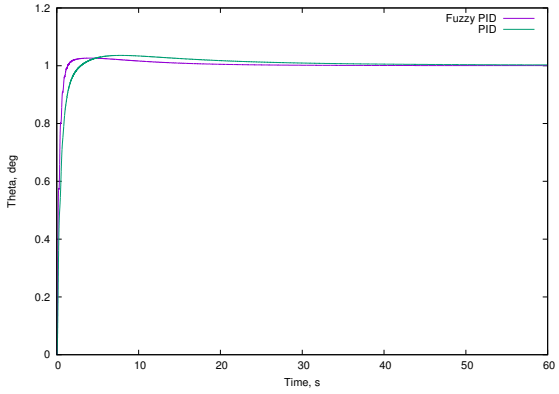
Case	T_s (s)	T_r (s)	T_p (s)	M_p (deg)	FV (deg)
Subsonic Cruise (Orig. Cost)	0.93	0.38	3.91	1.022	1.00
Supersonic Cruise (Orig. Cost)	3.84	0.73	11.10	1.018	1.01
Subsonic Cruise (Mod. Cost)	7.97	0.59	4.09	1.026	1.00
Supersonic Cruise (Mod. Cost)	15.53	0.73	11.74	1.020	1.00



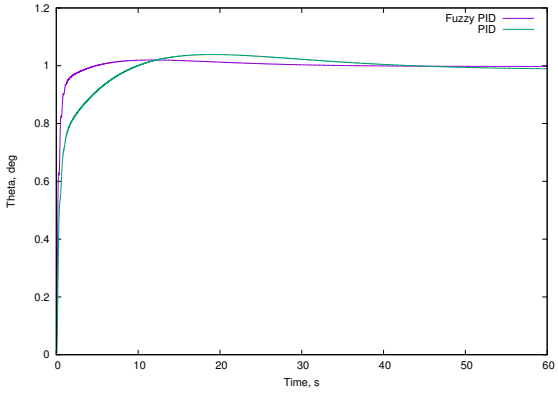
(a) Nominal approach condition



(b) Degraded aerodynamic derivative approach condition



(c) Subsonic cruise condition



(d) Supersonic cruise condition

Figure 3.2: Step response for various flight conditions using the controller created with modified cost function. Note the significantly decreased overshoot for nominal and degraded flight conditions.

3.5 Conclusion

In order to demonstrate the flexibility of a fuzzy PID controller, a controller was designed to meet the demands of the approach condition of an F4 fighter. It was shown that this controller exhibited good behavior across a wider domain than that in which it was trained. This demonstrates that an adaptive fuzzy PID controller can be an effective approach to problems with a wide variety of scenarios.

Future will work include assessing the response of the aircraft to fuzzy PID control according to the C* criterion, Gibson's drop back criterion, and other flying qualities standards[27]. This will provide a richer assessment of the controller's performance.

Chapter 4: Fuzzy Landing

4.1 Introduction

With the ever-increasing proliferation of small unmanned air vehicles (sUAVs) and their use in commercial and emergency response applications, there is a growing need for intelligent, reliable control methodologies to safely manage their navigation, especially in possibly congested areas such as disaster areas or urban centers. Commercial delivery companies are moving towards an automated model with reduced human operator intervention to increase the efficiency of their deliveries. One such model consists of a vehicle-based sUAV which departs from the delivery vehicle to make a delivery to a remote residence. Upon completing the delivery, the sUAV returns to the vehicle and docks to receive additional packages. Considering the small target size of a landing platform affixed to the potentially moving vehicle, and the highly dynamic conditions in which deliveries may be accomplished, the control effort must be accurate and robust in the face of disturbances.

Fuzzy control is able to accommodate nonlinearities in the dynamic system such as are found in the situation of an air vehicle to ground transport rendezvous[28]. Current approaches for developing trajectory paths have focused on time-optimality[29][30] and not necessarily on lightweight, on-board controllers. In contrast, this work optimizes

for reduced control effort, as well as computational simplicity and efficiency. Accuracy is paramount, as the target platform is nearly equally-sized to the sUAV.

4.2 System Architecture

The research setup consists of a quadrotor aircraft of size 450 mm on the diagonal and a mobile rover robot with a 255 mm radius landing platform affixed to it (as shown in figure 4.1). The quadrotor is controlled by a Pixhawk flight controller which uses the PX4 flight control firmware. This flight controller allows for an on-board computer to take over control of the aircraft via a serial wire connection. A small Linux-based computer is placed on the quadrotor which sends velocity setpoints to the flight controller. All control logic is written in Python using a collection of softwares called Robot Operating System (ROS). ROS allows for easy integration of sensors and control actuation due to a distributed computation framework. As a highly event-driven, publish-subscribe model, ROS maintains an accurate, up-to-date view of internal states which are then exposed to any connected nodes.

An assumption is made that GPS (or some other global positioning) data is available for the simulation; however, this positioning has a margin of error which is far too large to be used exclusively for precision landing. For this reason, an on-board camera is utilized to detect and locate the target. Using the characteristics of the camera focal length and distortion coefficients, an accurate positional error can be obtained for the feedback control loop.

4.2.1 Simulation

Gazebo is a 3D simulation software which uses open source dynamics engines Bullet, Dart, and ODE to model its components. While Gazebo has very high fidelity

Figure 4.1: The test sUAV with the mobile target platform.

simulation capabilities for robots (its initial purpose), the complexities of aerodynamics in general, and multicopter physics in particular, can only be modeled with many simplifications. Even with the simplified dynamics, the simulation environment that Gazebo provides is very useful for high-level controller development. Gazebo allows for the simulation of many sensor types and nicely integrates with ROS and the PX4 flight controller firmware. The simulation provides a real time interface for tuning the controller with visual feedback. This is the process which was used to tune the fuzzy controller.

The most important aspect of sensing for the system is the image sensor. A camera sensor is simulated from the underside of the quadrotor to test the efficacy and efficiency of the computer vision algorithms. Care was taken to accurately represent the field of view and pixel noise of the physical camera sensor to be used.

4.2.2 Onboard Software

ROS is an open source framework developed specifically to ease the development of software for robotics and robotics control[31]. Using ROS, it becomes a simple

task to distribute computational loads across a computational graph of separate nodes. Additionally, ROS has a rich library of packages which are useful for both low-level processing of sensor information or managing hardware interfaces, and also high-level behavioral control or localization schemes. This project use a number of such packages to perform tasks in the areas of visual odometry[32], kalman filtering [33], and flight control[34, 35]. ROS allows for a system designer to aggregate any number of processing units, called nodes, into a complex computational graph. An example of the graph structure created to complete this work is shown in figure 4.2 and provides a visualization of this complex structure. Each node in the graph represents a unit of computation which consumes information from another node in the graph. Each edge in the graph represents a direct message passing pipeline through which information is transmitted. Each node runs continuously and in parallel, facilitating the creation of reactive, concurrent systems.

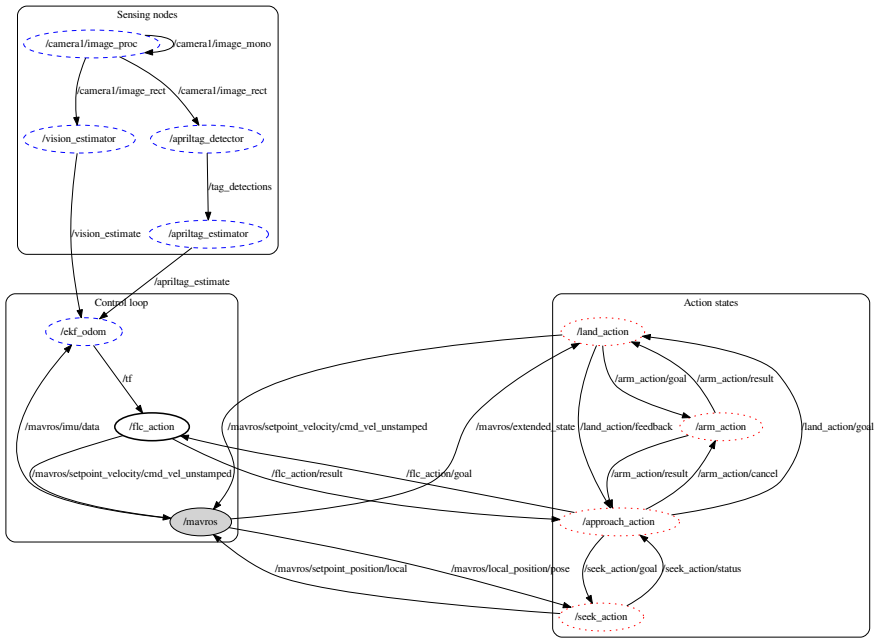


Figure 4.2: Computational node graph for a typical landing simulation. Note that this graph may change over time as node may be dynamically started and stopped. Likewise, message passing pipelines may be opened or closed at any time.

The nodes can be broadly categorized into three groups: Sensing nodes, Action states, and Control loop nodes. The Sensing nodes consume image sensor output and pre-process it for Kalman filtering. The Action states each represent a state in the state machine. These represent basic actions which the vehicle can perform and can be aggregated to construct more complex behavior. The nodes in the Control loop represent the plant (`/mavros`), the sensor (`/ekf_odom`), and the compensator (`/flc_action`). In this way, it becomes easy to visualize the canonical form of a feedback control loop. As can be seen in figure 4.2, the structure allows a roboticist to build and manage highly complex systems in a maintainable way.

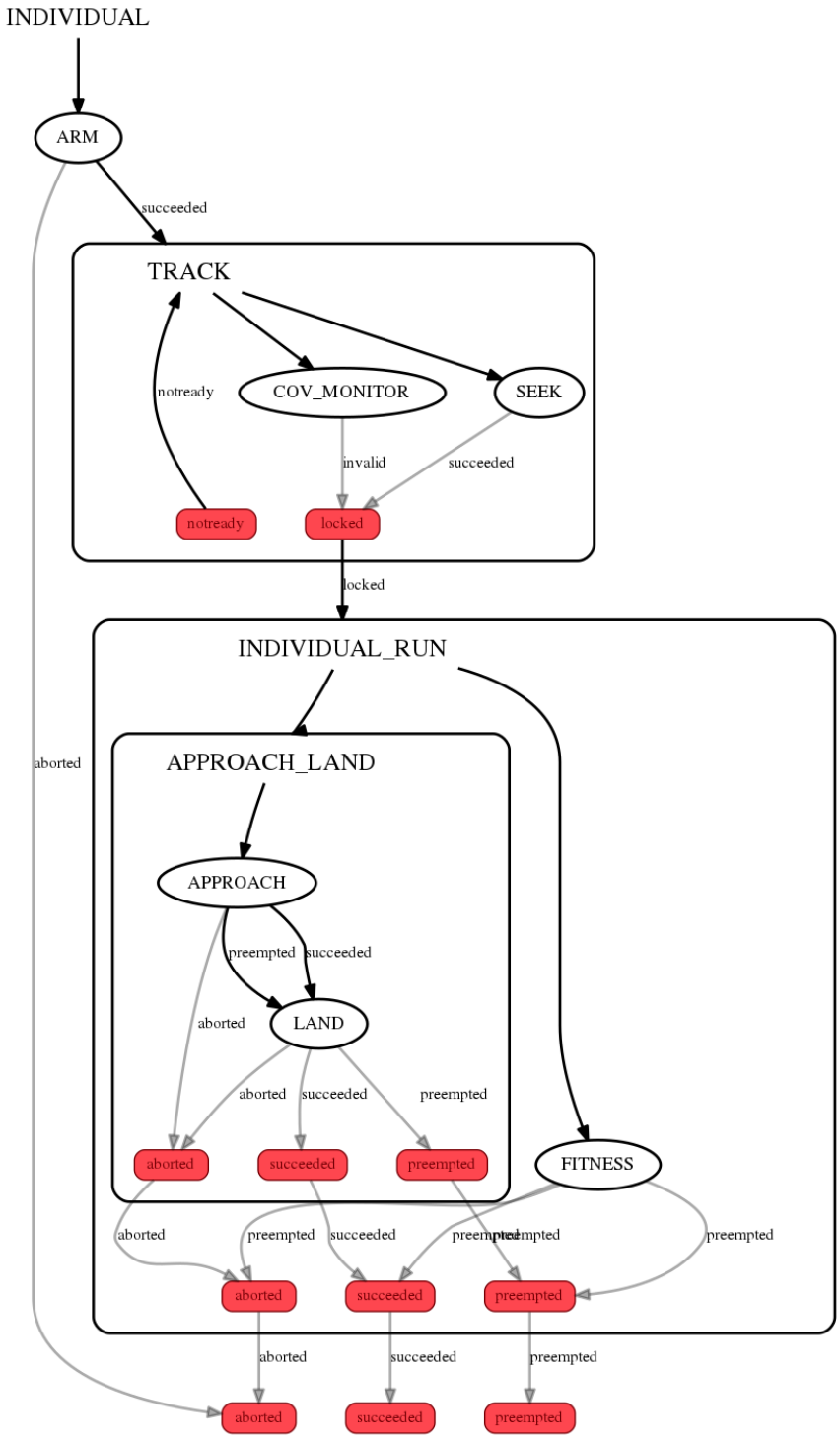


Figure 4.3: State machine of robotic lander

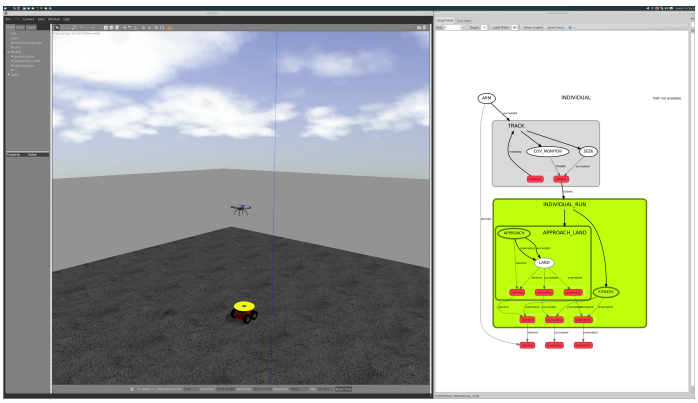


Figure 4.4: Image of the simulation and live state diagram while vehicle is approaching the platform.

Control of the quadcopter is handled in discrete stages based on vehicle state. It is assumed that the vehicle will have a rough estimate of the target location given to it so that it can travel to the appropriate region and find the target in the field of view of the camera sensor. Vehicle motion from arming until target location is handled by sending waypoints to the flight controller. Once the target is located in the image, the vehicle gives control over to a set of FISs. The controller is described in more detail in section 4.3. The vehicle behavior over an entire mission is handled using a state machine[36]. Using a state machine allows the control to be handled in well-defined domains and ensures that transitions between states are handled smoothly. The states comprising a full mission from takeoff to landing are shown in figure 4.3. A link to the video showing a full landing mission can be found in the references section[37]. An annotated image of the simulation at each state can be found in appendix A (example in figure 4.4).

The state of the vehicle is managed by fusing together the positional estimate from the camera sensor and the AprilTag estimate (see section 4.2.3), as well as orientation

information from the onboard IMU using an extended kalman filter (EKF). This estimate is only valid when the vehicle has a visual track on the target platform; otherwise, it is assumed that the vehicle is still in transit from its launch location, or it has landed.

As can be seen in figure 4.3, a mission starts in the "ARM" by arming the vehicle and immediately sending it a waypoint which is near the target's location. Once the vehicle has received the waypoint, it enters the "TRACK" state and is en route to the target location. While in this state, it monitors the quality of its visual estimation of the target location by evaluating the norm of the covariance matrix computed by the EKF. Only when the covariance is sufficiently small does the vehicle transition to the next state.

The transition to the "APPROACH_LAND" state signals the transfer of control from the flight controller's waypoint manager to the FISs. The "APPROACH_LAND" state is simply a composition of two substates, "APPROACH" and "LAND". During the "APPROACH" substate, the EKF is still running, sending estimates to the fuzzy logic controller. The fuzzy logic controller is sending velocity setpoints to the onboard flight controller, thus closing the feedback loop. The desired outcome of this state is to get the vehicle in a position above the platform such that we can initiate the landing sequence. When the vehicle meets a proximity threshold, it transitions to the "LAND" substate and puts down onto the landing pad. The details of the simulation, control, vision estimation, and development process are discussed at length in the following sections.

4.2.3 Computer Vision

Image processing is handled by a node in the ROS computation graph that is constantly processing image outputs from the onboard camera. Once it senses the platform, it then publishes a state estimation to the rest of the nodes in the ROS graph. Much emphasis is put on the sensing algorithms to be computationally efficient to decrease the load on the on-board computer. For this purpose, only a small number of image processes are required to detect and locate the target. As a first pass, the image is brought into the Hue-Saturation-Value (HSV) color space. This has been shown to be a robust space in which to perform color detection and segmentation in uncontrolled and unpredictable lighting conditions[38]. A simple thresholding is performed on the image to isolate a sufficiently wide band of yellows to match the color of the target and dilate this to a binary blob. From this binary image, the image moments are calculated by

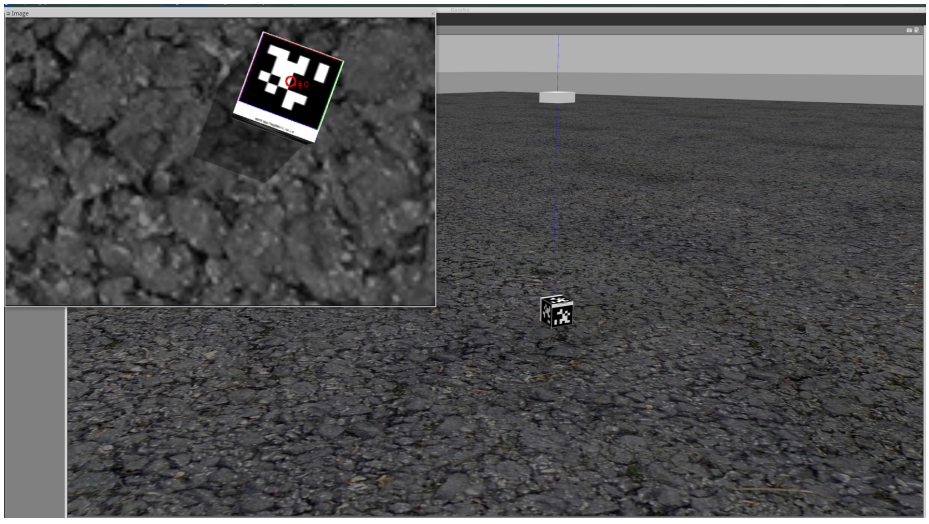


Figure 4.5: Simulated image sensor detection of AprilTag marker.

$$m_{ij} = \sum_{x,y} x^i y^j I_{xy} \tag{4.1}$$

where I_{xy} is the pixel intensity value for each pixel (x, y) (equal to 1 for this binary blob) and $i, j = 0, 1, 2$. From this it can be seen that m_{00} describes the area and $\frac{m_{10}}{m_{00}}$ and $\frac{m_{01}}{m_{00}}$ describe the centroids $\overline{x}_p$ and $\overline{y}_p$ in terms of pixels[39]. The blob is assumed to be circular and hence a diameter is extracted from the pixel area. Using the focal length of the sensor, these image points are then projected onto the ground plane using the known diameter of the landing pad and a vertical offset estimate is obtained as is shown in equation (4.2).

$$d_z = \frac{d \cdot f}{m \cdot d_p} \tag{4.2}$$

where f is the focal length of the camera in units of length, d is the known diameter of the landing pad, d_p is the estimated diameter in pixels, and m represents a scaling factor in units of $\frac{\text{pixel}}{\text{mm}}$ (unity in the simulation). Assuming that the image plane and the ground plane are parallel, the center of the image can be assumed to point directly below the vehicle and the horizontal offsets to the landing pad can then be calculated as

$$d_x = d_z \cdot \frac{m\overline{x}_p}{f} \tag{4.3}$$

$$d_y = d_z \cdot \frac{m\overline{y}_p}{f} \tag{4.4}$$

Unless the camera is mounted to a perfect gimbal, the image plane cannot be assumed to be parallel to the ground plane, thus invalidating equations (4.3) to (4.4). To overcome this, the rotation sequence needed to transform the image into body-relative coordinates is found and applied to the image data. The camera is assumed to

be rigidly affixed to the body of the vehicle, thus this is a simple static transformation. Then, the vehicle body frame is related to the inertial frame by another rotation. The landing pad is assumed to be on level ground and the distance between the vehicle and pad is estimated from equation (4.2). Note that although there will be image skew when the vehicle is under some rotation, the error in estimated distance this introduces is small compared to the actual distance of the vehicle from the target. As the vehicle approaches the target, the estimate improves in quality due to a crisper image, level flight, and the addition of AprilTag estimation.

In order to account for vehicle attitude, a rotation sequence is applied to the projected target location. Let $R\left(\phi,\theta,\psi\right)$ be the rotation matrix defined as

$$R(\phi,\theta,\psi) = \begin{bmatrix} c_\theta c_\psi & -c_\theta s_\phi & s_\theta \\ c_\phi s_\psi + c_\psi s_\phi s_\theta & c_\phi c_\psi - c_\psi s_\phi s_\theta & -c_\theta s_\phi \\ s_\phi s_\psi - c_\phi c_\psi s_\theta & c_\psi s_\phi + c_\phi s_\theta s_\psi & c_\phi c_\theta \end{bmatrix} \tag{4.5}$$

where ϕ, θ, and ψ are vehicle roll, pitch and yaw respectively. c_α and s_α represent $\cos\alpha$ and $\sin\alpha$ respectively. Let R_x^y represent the rotation from frame x to y, then $R_x^y R_y^z$ is the rotation from frame x to frame z with respect to the stationary frame x. Now assuming the vehicle can estimate its orientation from the onboard inertial measurement unit, the following rotation matrices can be defined.

$$R_{cam}^{body} = R(\pi,0,0) \tag{4.6}$$

$$R_{body}^{inert} = R(\phi,\theta,\psi) \tag{4.7}$$

where R_x^y represents the rotation from frame x to frame y. These rotations are then prefixed to the projected point found in equations (4.2) to (4.4) to find the point in

world units with respect to the vehicle.

$$P_i = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \tag{4.8}$$

$$P_r = R_{body}^{inert} R_{cam}^{body} P_i \tag{4.9}$$

where P_i is the projected point of the target in the plane parallel to the image plane, and P_r is the rotated point of the target with respect to the vehicle.

As the vehicle approaches the landing pad, the image field of view is overtaken by the landing pad itself and the former segmentation no longer becomes effective. For this purpose, an AprilTag[32] (see figure 4.5) is placed on the center of the target. In figure 4.6, a landing approach is illustrated from the viewpoint of the image sensor. This figure shows three different stages of the image processing pipeline: raw image capture (figures 4.6(a) to 4.6(d)), AprilTag detection (figures 4.6(e) to 4.6(h)), and platform detection (figures 4.6(i) to 4.6(l)). Individual estimates from both the AprilTag detection and platform detection are then fused using an EKF (section 4.3.2).

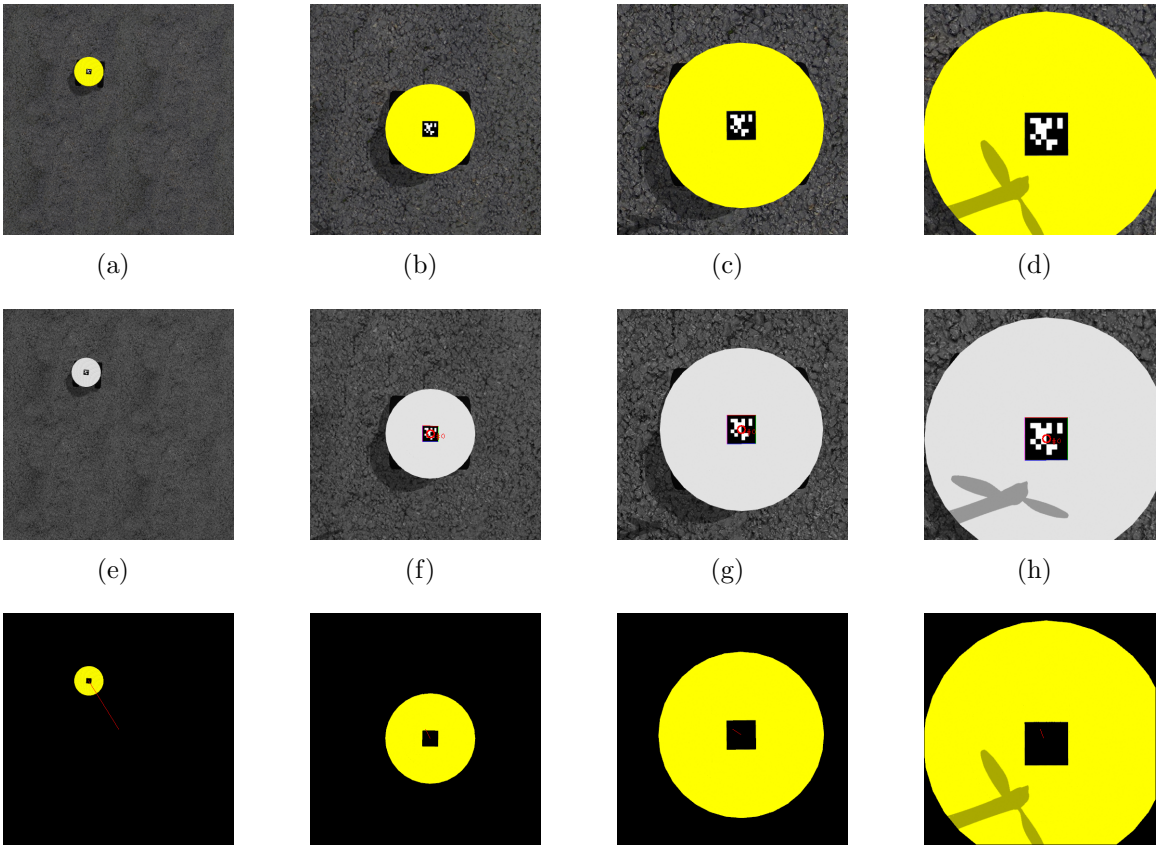


Figure 4.6: A time series of images taken during the landing maneuver.

4.3 Controller

Control is actuated on the vehicle by providing velocity setpoints to the flight controller, v_x, v_y, v_z, and yaw rate, $\dot{\psi}$. Ideally either linear acceleration setpoints or forces would be the control output, but limitations in the flight controller make this impossible at this time. Setting velocities allows the controller to cope with step changes to position offset well, but have difficulties tracking ramps. The yaw angle, ψ of the vehicle at landing is irrelevant for this project, but the controller is able to control that axis regardless.

4.3.1 PID Controller

A PID controller was created which uses error estimates and sends velocity setpoints to the flight controller. Due to the lack of a mathematical model of the system, the PID gains were found by iterating the simulation with various position setpoints and dynamically adjusting the gains while observing the response visually. This closely mimics the method by which PID gains are attained in the tuning of an actual quadrotor by flying a series of test flights and adjusting gain values by feel. In this way, a reasonable response and landing sequence was achieved for first a static target and then repeated for a dynamic target moving with constant velocity. The results are presented in figures 4.7 to 4.8 in which the normed horizontal offset from target is shown in conjunction with the vertical distance to the platform. This view of the data gives an overall sens of how the controller performs by showing how smooth the trajectory of the vehicle is as it approaches the platform. Viewing the charts, it is apparent that as the vehicle approaches the target, it tends to accumulate horizontal errors and must correct more frequently, particularly in the dynamic case. In both

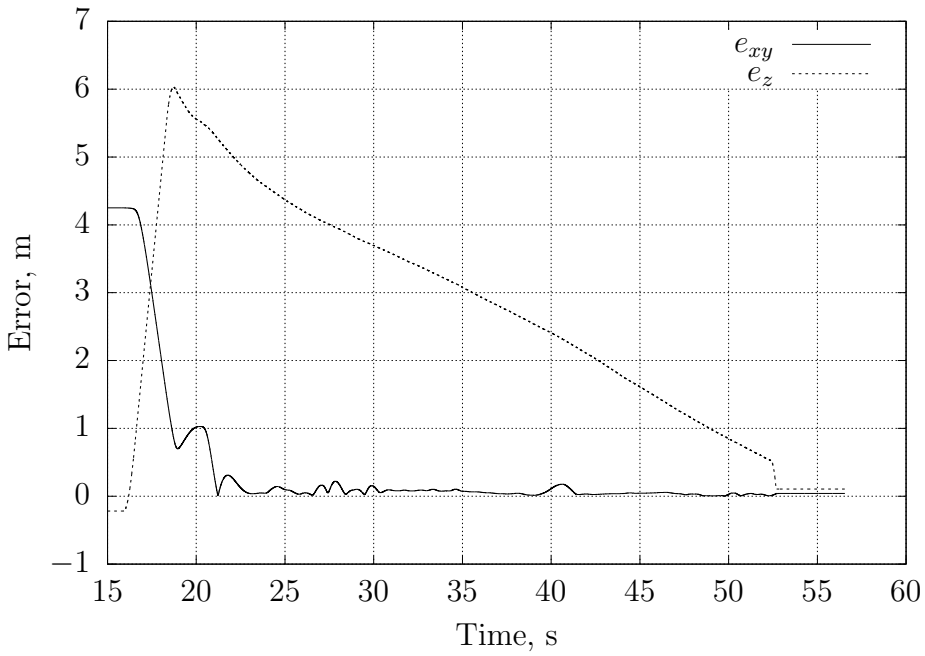


Figure 4.7: $(kp, ki, kd) = (2.1, 0.015, 0.2)$ for static target interception.

cases, the quadrotor managed to successfully land on the target. In the static case, the final offset from the center of the target was less than 5 cm. For the dynamic case, the error was 7 cm which nearly displaced it from the target surface. In both cases, the sink rate and yaw rates were controlled by simple PD controllers. It can be seen that the controller took a significantly longer time to intercept and land on the moving target as it repeatedly had trouble responding to the motion and would temporarily lose its visual track. This is apparent in the large spiking errors in figure 4.8.

4.3.2 Kalman Filter

In order to reduce the deleterious effect of instantaneous error accumulation when losing a visual track, a kalman filter is introduced. The filter provides a continuous,

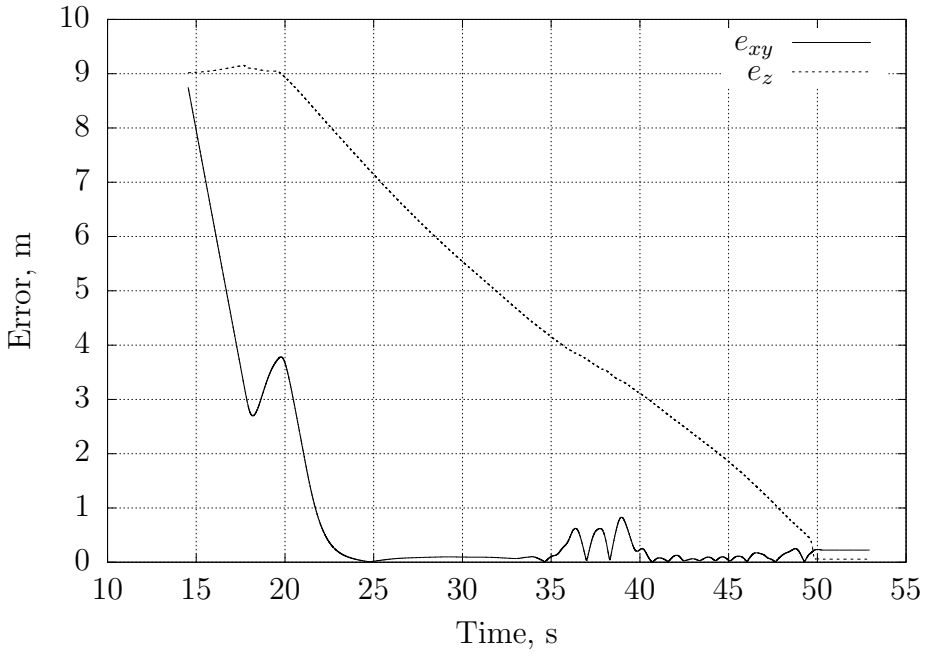


Figure 4.8: $(k_p, k_i, k_d) = (1.8, 0.012, 0.02)$ for dynamic target interception. The target is moving with a velocity of $0.1\,\frac{\text{m}}{\text{s}}$ in a straight line.

fused estimate of the vehicle's position relative to the platform whenever there is at least visual sensing of the platform. This is done using the `robot_localization` package from ROS[33]. This package is an implementation of an extended kalman filter (EKF). The EKF provides a means by which to predict the vehicle state even in the absence of current measurement[40]. The prediction is then corrected when new measurement values are obtained. The measurements used to update the filter state are 1) the estimate from the computer vision algorithm to update the vehicle position, 2) the estimate from the AprilTag[32] routine to update both position and orientation, and 3) the IMU to update angular rates and linear accelerations. These measurements are fused together to build an internal model estimate of the dynamics of the vehicle. It is this model which provides the predictions between measurements. An example of the EKF measurement is shown in figure 4.9. It can be seen in this data that though the visual estimate is very noisy, the EKF fusion follows the truth signal closely even between the infrequent update periods of the accurate AprilTag measurement.

Appendix B contains annotated screenshots (example in figure 4.10) of a video of a simulated vehicle landing on a moving platform. This video also includes a visualization of the EKF state over time and various estimates which are fused together. A link to this video is found in the references section[41].

4.3.3 Fuzzy Controller

Four fuzzy controllers of similar architecture were created as shown in figure 4.11. Each input fuzzy partition was multiplied by a scaling factor to bring it into the range of the controller. Likewise, the outputs were then again scaled to match actuation limits. The rule base was developed using common intuition about the system dynamics. A

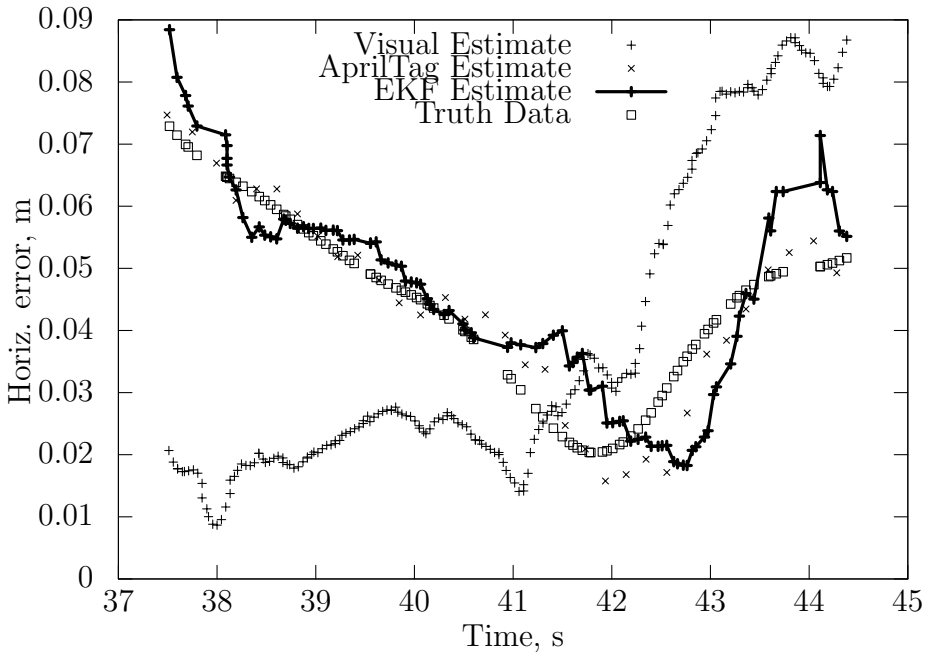


Figure 4.9: EKF estimate with input estimates over short time sample.

full rule matrix was defined to fully cover the system possibilities. This rule matrix is shown in table 4.1.

These rules provide a process by which to lend the controller a decision-making system with a foundation in human reasoning. The tuning process of the fuzzy controller then becomes the task of defining the membership functions which decide how much of each rule should be activated for certain inputs. Triangular membership functions are used exclusively for their simplicity in definition and tuning[42] while the aggregation of rules is the popular min-max method put forth by Mamdani[3]. The membership functions shown in figure 4.11 are the result of a number of iterative tuning steps and were found to be the most effective of the configurations attempted.

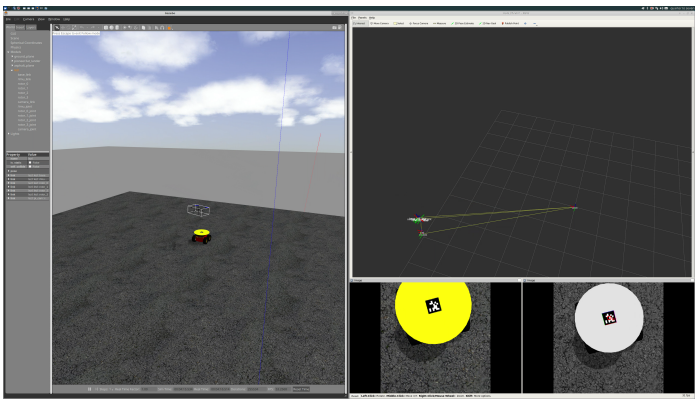


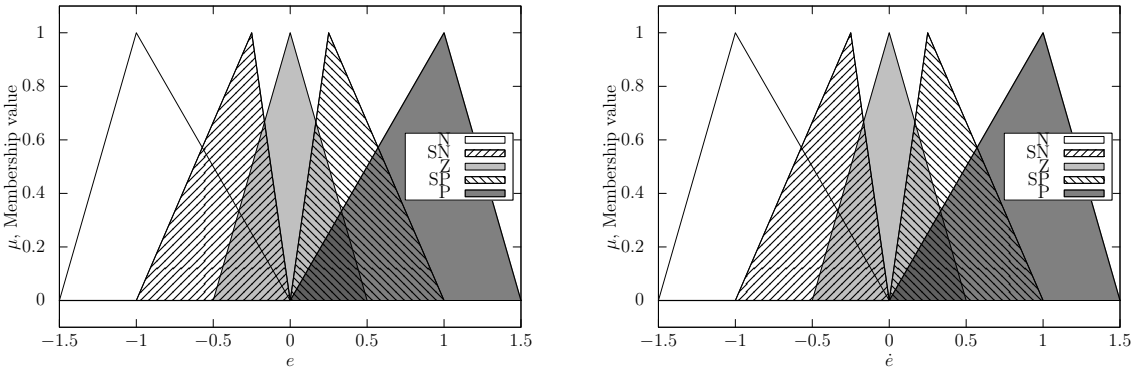
Figure 4.10: Image of the simulation environment and along with estimate visualization and camera feeds

Table 4.1: Fuzzy rule base. The error and error rate membership sets correspond to velocity output membership sets according to this table. N-negative, SN-small negative, Z-zero, SP-small positive, P-positive

re intuitive and easy to understand and pr

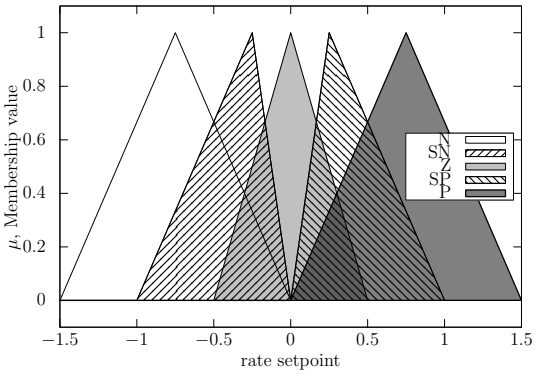
		error				
		N	SN	Z	SP	P
error rate	N	P	P	SP	SP	Z
	SN	P	SP	SP	Z	SN
	Z	SP	SP	Z	SN	SN
	SP	SP	Z	SN	SN	N
	P	Z	SN	SN	N	N

The result of this controller was sufficient to land the quadrotor on both the static and constant linear velocity dynamic platforms. Figures 4.12 to 4.13 show the results of the static and dynamic landings using the fuzzy controllers. These figures begin at takeoff of the vehicle, and it is clearly seen where the controller locks onto the visual image and initiates the landing state. It is also clearly seen that with the



(a) Error input

(b) Error rate input



(c) Rate command output

Figure 4.11: Membership function definitions for fuzzy logic controller.

implementation of both the kalman filter and the fuzzy controller, the errors resulting from image loss are eliminated, providing a smoother landing approach.

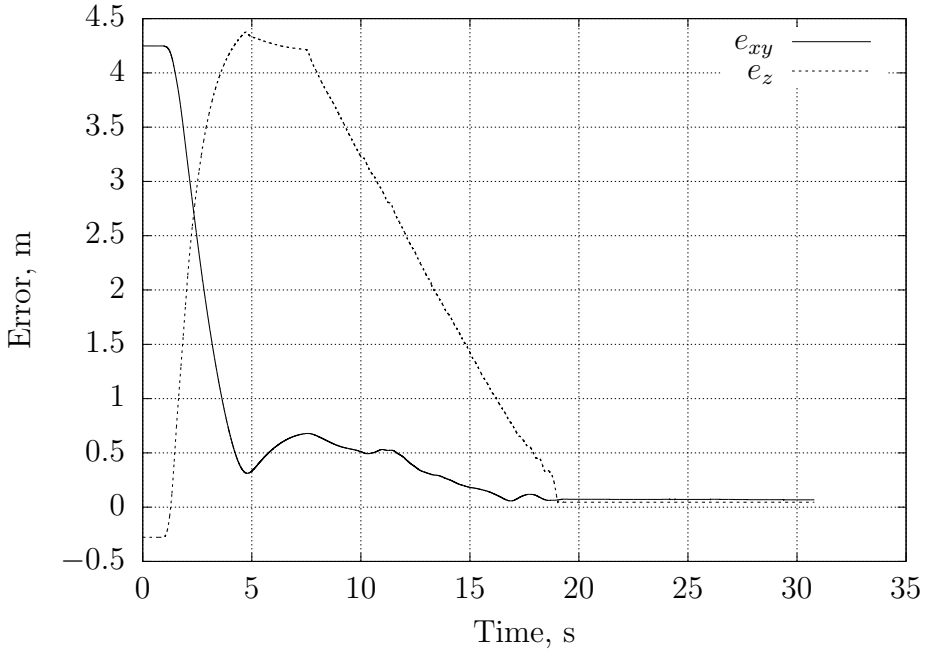


Figure 4.12: Fuzzy-controlled static target interception.

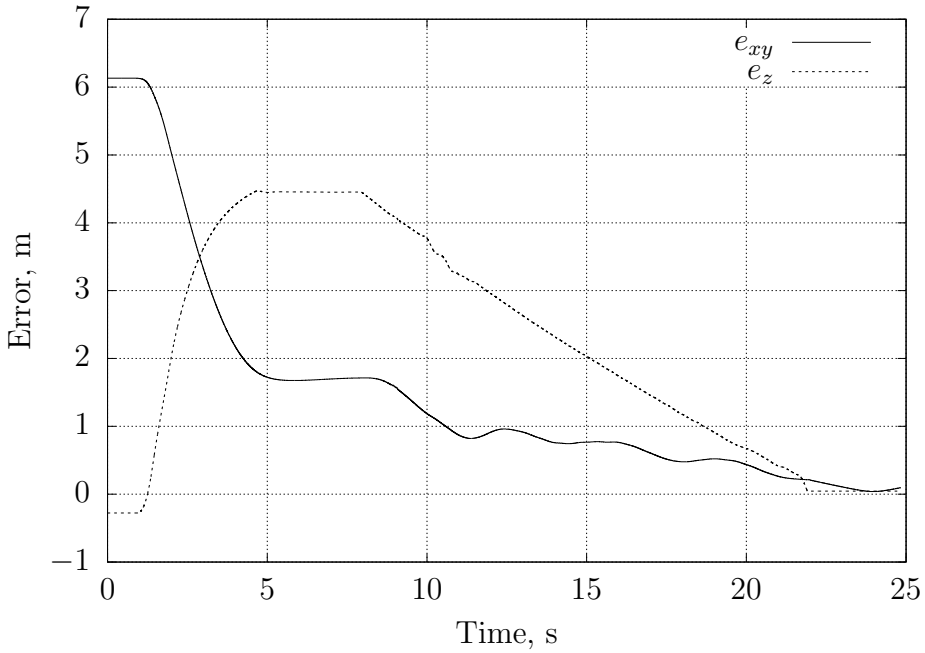


Figure 4.13: Fuzzy-controlled dynamic target interception. The target is moving in a straight line with a velocity of $0.1\,\frac{\mathrm{m}}{\mathrm{s}}$.

4.4 Conclusion

It has been shown that a pure FLS controller is capable of controlling the position of a multirotor vehicle in the task of precision landing on a small target. The small size of the platform presents a challenging landing target; the FLS proved capable enough to overcome this challenge easily. The kalman filtering was an integral component in providing the controller with a steady estimate of its error state. The components chosen to accomplish this task are readily available in university laboratory environments and the physical realization of this system was kept in the forefront of the design process.

Chapter 5: Conclusions

5.1 Summary

Genetic Fuzzy Systems are not a silver bullet solution to every problem; however, this can be said of any architecture, paradigm, or approach. It becomes the job of an engineer to know the breadth of tools at their disposal and be able to chose the best for a job. It is the proposition of this thesis that FL systems should be considered as first-class citizens in the tool chest of any controls engineer. They are more than capable of performing the low-level control tasks required in dynamics problems and are remarkably well-suited to make decisions at higher levels as well. The ability of FLS to translate the intuition of expert controls designers into machine-executable signals is a unique benefit that is worth exploring for a wide array of problems we face every day.

It was shown in this work that the combination of manual construction and genetic tuning of a FLS produces a controller which performs near-optimally in a highly non-linear system. It was also shown that a FLS which is learned almost solely via a GA can attain results which meet any number of physically realizable specifications. The expert intuition in these cases emerges in the development of a fitness (or cost) function which accurately describes the desired behavior. Finally, it was shown that a purely hand-tuned system can perform remarkably well. This system is perhaps far

from optimal, but has the distinction of being implementable on affordable hardware in real time.

In conclusion, fuzzy logic systems and genetic fuzzy systems are valuable tools in the controls engineering field and should be considered as viable alternatives when working with many systems. Indeed, there are many applications in which FL control provides a nearly optimal controller and does so with minimal computational overhead and little control effort.

5.2 Future Work

This thesis opens the door to many opportunities for future work. First, there is still much work which could be done with regards to the components of the controller itself. There are many corollaries to the controller which remain unexplored. The computer vision algorithm is simplified and could be expanded greatly to produce a much more robust and versatile application. Also, the EKF is minimally tuned, more work could be done to produce a cleaner estimate signal.

Second, this thesis showed the effectiveness of landing a FL-controlled multirotor aircraft on a moving platform. It is the author's belief that layering a GA-automated tuning process to the simulation would result in even better performance. Some preliminary work has been done in developing a cost function which will aid the development of evolutionary approaches in the future.

Third, it is the author's hope that the work will be continued in the laboratory and eventually be applied to hardware. The use of ROS to implement the control architecture allows this control architecture to operate in physical hardware, as has been shown in other projects in the laboratory environment in recent projects.

Finally, integration of this work with other projects within the UAV Master Labs at the University of Cincinnati would facilitate this project in being used in real applications. The controller could rather easily be ported into the FlyMASTER framework as a controller module. This would ease the integration of this work with other projects in the future and work towards the goal of unifying many projects in the lab towards a greater whole.

Bibliography

[1] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic Fuzzy Systems, Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. World Scientific Publishing Co., MA, 2001.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] E.H. Mamdani and S. Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1):1 – 13, 1975.

[4] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[5] Bart Kosko. *Fuzzy Thinking, The New Sciencs of Fuzzy Logic*. Hyperion, NY, 1994.

[6] B. Kosko and S. Isaka. Fuzzy logic. *Scientific American*, 269(1):76–81, 1993.

[7] Nicholas Ernest, Kelly Cohen, Elad Kivelevitch, Corey Schumacher, and David Casbeer. Genetic fuzzy trees and their application towards autonomous training and control of a squadron of unmanned combat aerial vehicles. *Unmanned Systems*, 3(03):185–204, 2015.

[8] Nikolaus Hansen and Andreas Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The (=. *Eufit*, 97:650–654, 1997.

[9] Ulrich Bodenhofer and Francisco Herrera. Ten lectures on genetic fuzzy systems. *Preprints of the International Summer School: Advanced Control-Fuzzy, Neural, Genetic*, pages 1–69, 1997.

[10] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[11] Mohammad Divband Soorati and Heiko Hamann. The effect of fitness function design on performance in evolutionary robotics: The influence of a priori knowledge. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 153–160. ACM, 2015.

[12] UK Chakraborty and DG Dastidar. Chromosomal encoding in genetic adaptive search. In *Proc. Intl Conf. on Signals, Data and Systems, AMSE*, volume 2, pages 191–195, 1991.

[13] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[14] Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

[15] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[16] Yinpeng Dong, Hang Su, Jun Zhu, and Bo Zhang. Improving interpretability of deep neural networks with semantic information. *arXiv preprint arXiv:1703.04096*, 2017.

[17] Bong Wie and Dennis S Bernstein. Benchmark problems for robust control design. *Journal of Guidance, Control, and Dynamics*, 15(5):1057–1059, 1992.

[18] Robert F Stengel and Christopher I Marrison. Robustness of solutions to a benchmark control problem. *Journal of Guidance, Control, and Dynamics*, 15(5):1060–1067, 1992.

[19] K. Cohen, T. Weller, and J.Z. Ben-Asher. Control of linear second-order systems by fuzzy logic-based algorithm. *Journal of Guidance, Control, and Dynamics*, 24(3):494–501, 2001.

[20] A. Walker. untitled. Class project, feb 2013.

[21] T. Vick. untitled. Class project, feb 2013.

[22] S. Mitchell. untitled. Class project, feb 2013.

[23] A. Stimetz. untitled. Class project, feb 2013.

[24] The MathWorks Inc, Natick, Massachusetts, United States. *MATLAB and Fuzzy Logic Toolbox Release 2012b*, 2012.

[25] David Bossert and Kelly Cohen. Pid and fuzzy logic pitch attitude hold systems for a fighter jet. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4646, 2002.

[26] Nicklas Stockton. *C Library for Genetic Fuzzy Systems*, 2018.

[27] Military Standard. Flying qualities of piloted aircraft. *US Dept. of Defense MIL-STD-1797A*, 1990.

[28] S. Ionita. A fuzzy approach on guiding model for interception flight. In *Fuzzy Systems Engineering*, pages 85–111. Springer Science & Business Media, Jul 2005.

[29] Jerrod C. Adams. An interpolation approach to optimal trajectory planning for helicopter unmanned aerial vehicles. Master's thesis, Naval Postgraduate School, Monterey, California, 6 2012.

[30] Markus Hehn and Raffaello D'Andrea. Real-time trajectory generation for interception maneuvers with quadrocopters. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Institute of Electrical & Electronics Engineers (IEEE), oct 2012.

[31] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[32] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407. IEEE, May 2011.

[33] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014.

[34] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ros). *Studies Comp.Intelligence Volume Number:625*, The Complete Reference (Volume 1)(978-3-319-26052-5):Chapter 23, 2016. ISBN:978-3-319-26052-5.

[35] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 6235–6240. IEEE, 2015.

[36] Jonathan Bohren and Steve Cousins. The smach high-level executive [ros news]. *IEEE Robotics & Automation Magazine*, 17(4):18–20, 2010.

[37] UAV Master Labs. *Static target interception with state machine visualization.* Youtube, 2018. `https://youtu.be/cbB5jMtYAvw`.

[38] Ming Zhao, Jiajun Bu, and Chun Chen. Robust background subtraction in hsv color space. In *ITCom 2002: The Convergence of Information Technologies and Communications*, pages 325–332. International Society for Optics and Photonics, 2002.

[39] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE transactions on information theory*, 8(2):179–187, 1962.

[40] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[41] UAV Master Labs. *Interception of a moving target with RViz demonstration.* Youtube, 2018. `https://youtu.be/mv-3zwVzMWc`.

[42] SK Mishra, IG Sarma, and KN Swamy. Performance evaluation of two fuzzy-logic-based homing guidance schemes. *Journal of Guidance, Control, and Dynamics*, 17(6):1389–1391, 1994.

Appendix A: Static landing sequence state machine video description

The following screen shots come from a video of a simulation using the kalman filter and fuzzy controller, landing the vehicle on a stationary platform (figures A.1 to A.6). The video shows the simulation environment on the left of the screen, and tracks progress of the state machine on the right of the screen. Any active state in the state diagram is highlighted in green. Multiple states are commonly active at the same time. As a state is completed, its color reverts. In the screenshots, you can clearly see the vehicle taking off, traveling to the communicated waypoint, locating and locking the location of the platform via image recognition, and initiating the approach and land states.

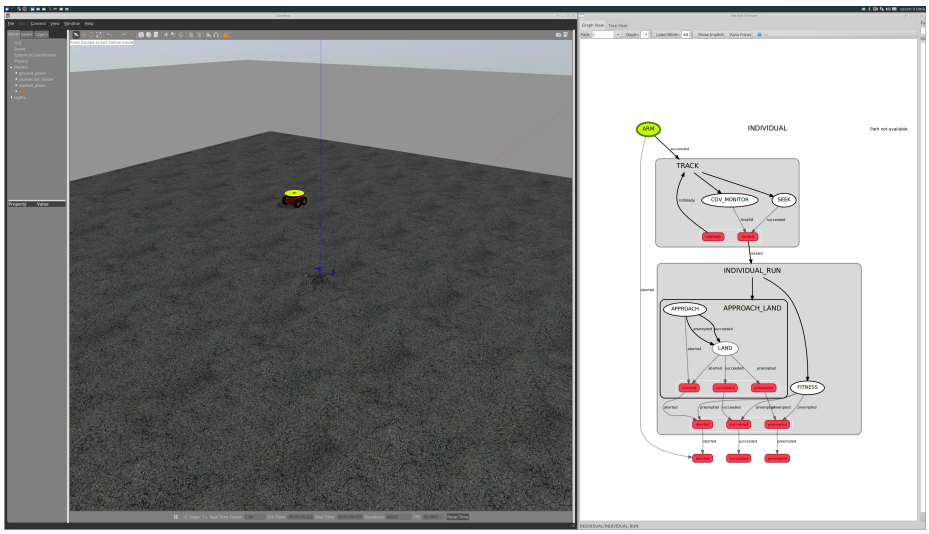


Figure A.1: The "ARM" state as the vehicle is arming its motors.

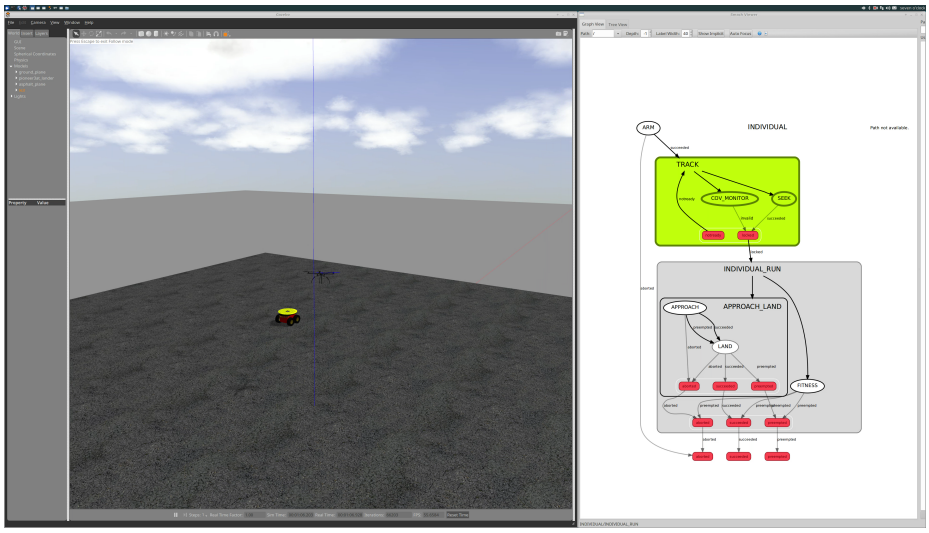


Figure A.2: The transition to "TRACK" sends the command to go to a waypoint via the "SEEK" substate. The vehicle immediately takes off and navigates to the commanded position. During this state, the covariance is monitored to trigger the transition to the next state ("COV_MONITOR").

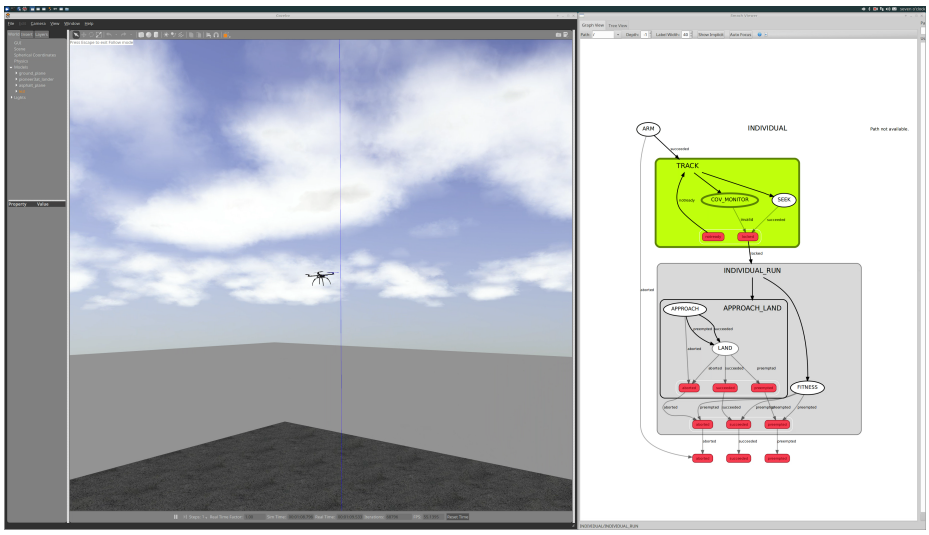


Figure A.3: The "SEEK" substate has completed and the state machine is waiting for "COV_MONITOR" to verify that the EKF estimate covariance is sufficiently small to signal that the vehicle has a visual track.

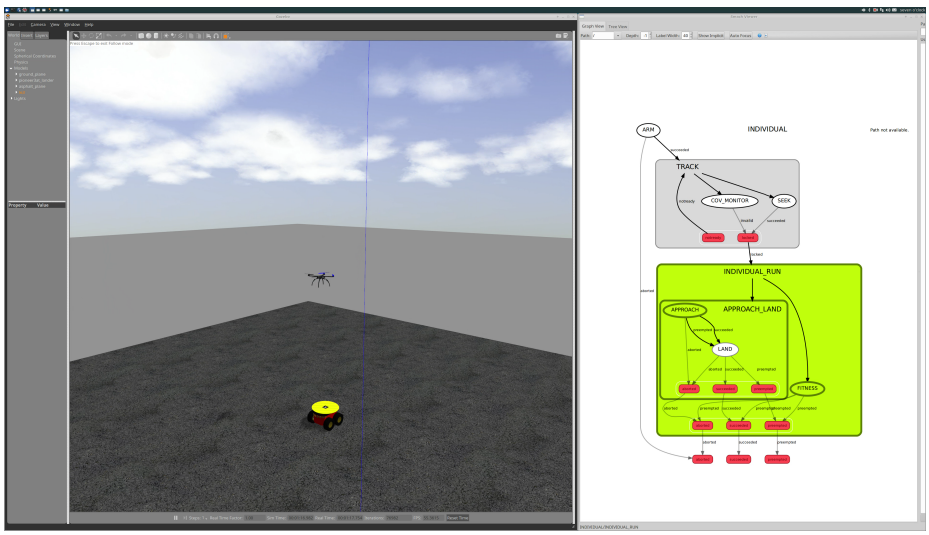


Figure A.4: The vehicle has transitioned now to the landing approach. At this point, the FL controller is in command of the vehicle and all pose estimation is based on visual sensor feedback. The "APPROACH" state will continue to apply FL control until either the vehicle is sufficiently close to the platform to transition to the "LAND" state or loses track of the platform which will force the vehicle to abort it's approach

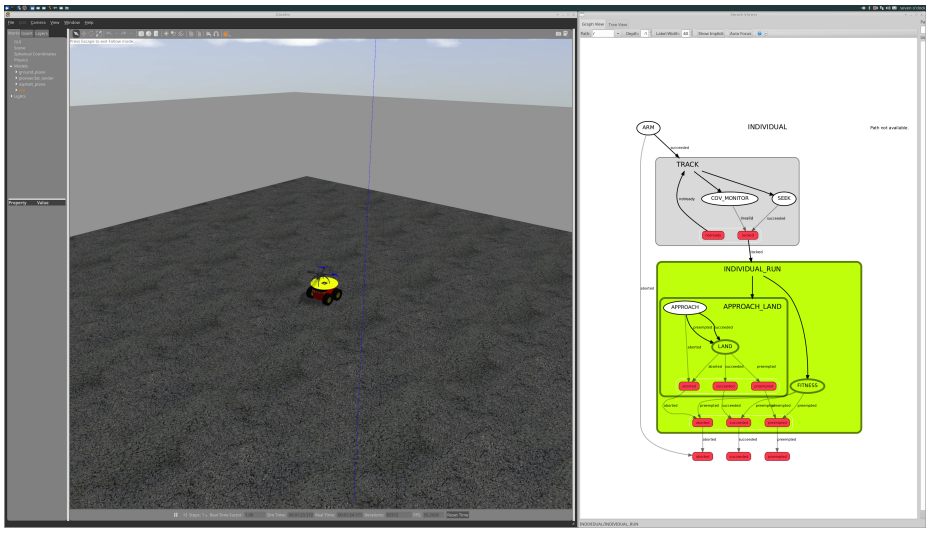


Figure A.5: The vehicle has successfully approached the platform and started the landing sequence.

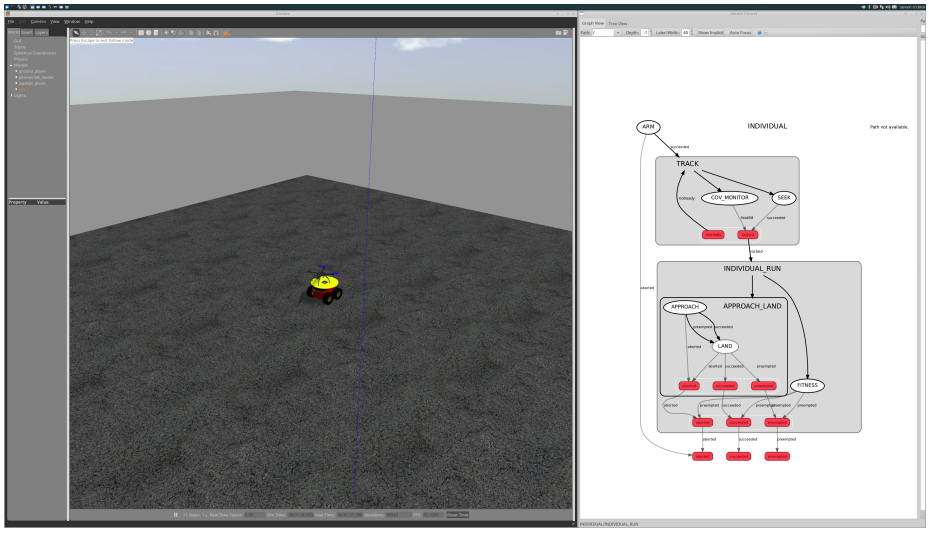


Figure A.6: The vehicle is landed on the platform and motors are disarmed at mission success.

Appendix B: Moving target platform landing simulation video description

In this video, the simulation appears on the left of the screen. The upper right window shows a coordinate frame representation of the environment. It is in this that various frame transformations are. A frame transformation is represented by a small unit coordinate system showing orientation, and a line to some parent frame denoting translation. There will be four frames representing the vehicle. These four frames are the three estimates (visual, AprilTag, EKF) and simulation truth. The images show that before the on-board camera can see the platform, the EKF estimate is invalid as it has no data. Once the platform appears in the image, an EKF estimate starts to converge. When the vehicle is close enough to the platform to resolve the AprilTag, the estimate accuracy increases and continues to converge on truth until the vehicle has landed.

Also shown in the images are the raw camera feed (below and to the left of the coordinate frames) and the AprilTag detection images (to the right of the raw feed).

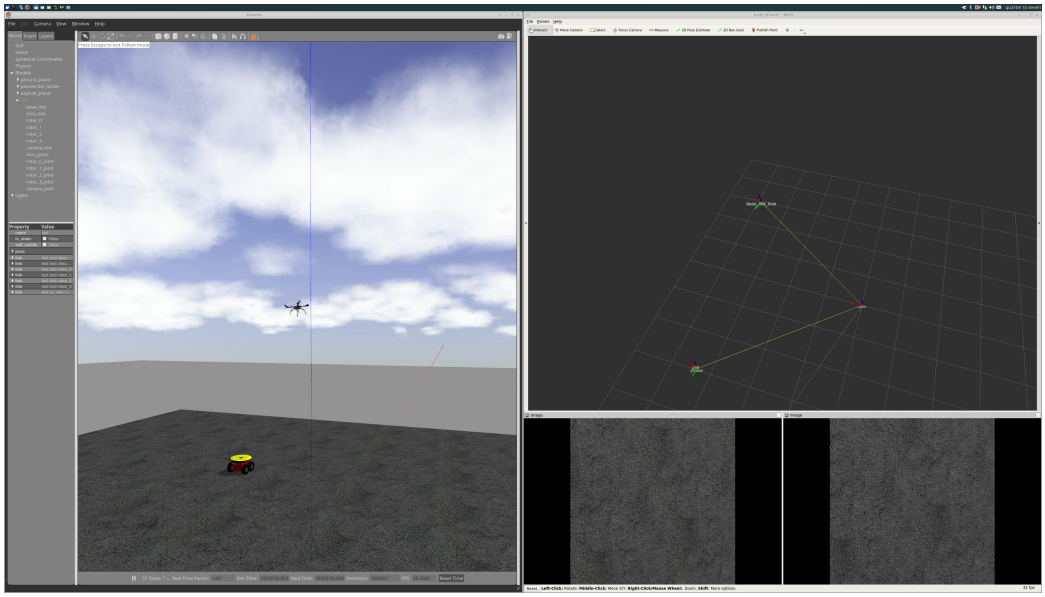


Figure B.1: The vehicle before its camera has had an opportunity to image the target. Truth data is the only frame depicted in the coordinate transformation space.

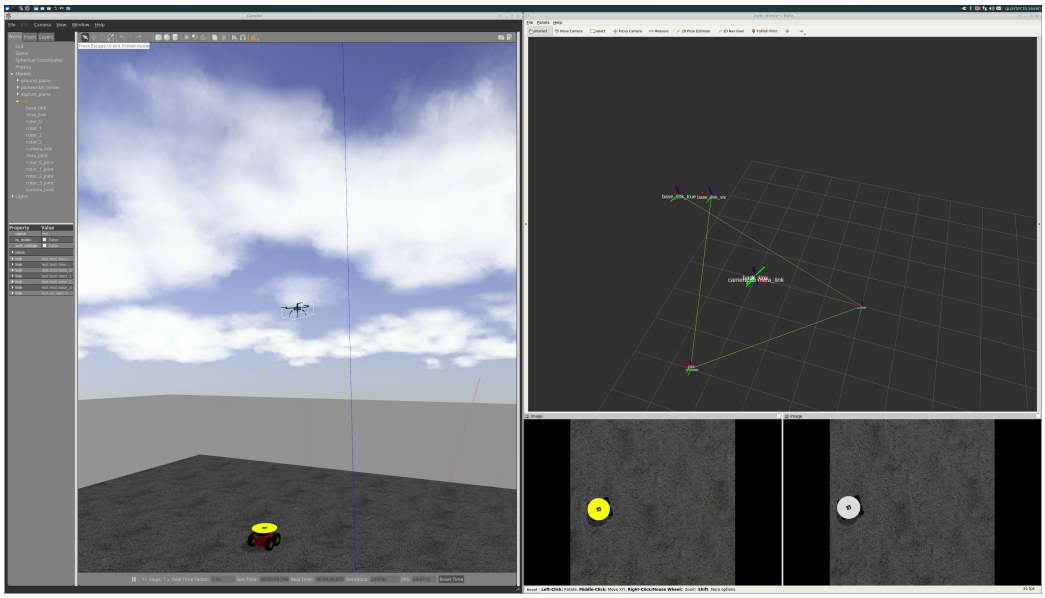


Figure B.2: The state of the simulation shortly after the platform has come into view. Note that the platform is shown in the camera's image feed, but there is no AprilTag detection as the vehicle is too far away to resolve the tag. The EKF coordinate frame is now visible, but not yet stable. Note that the visual estimate frame is a child of the platform frame.

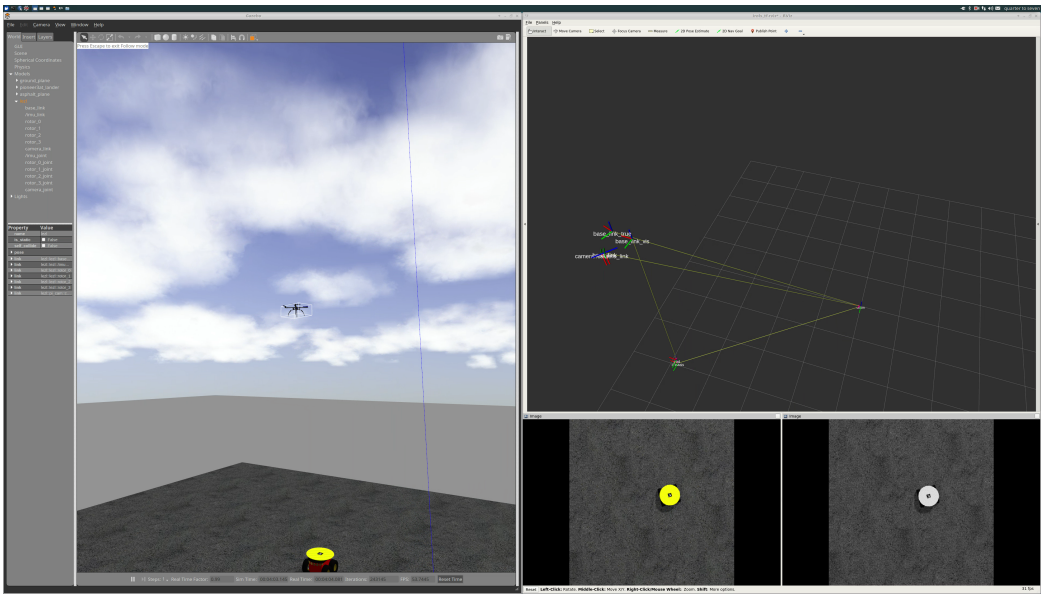


Figure B.3: After some small amount of time, the EKF estimate has now started to converge onto truth data, but still shows large error as the visual estimate from the camera is the only input as of yet.

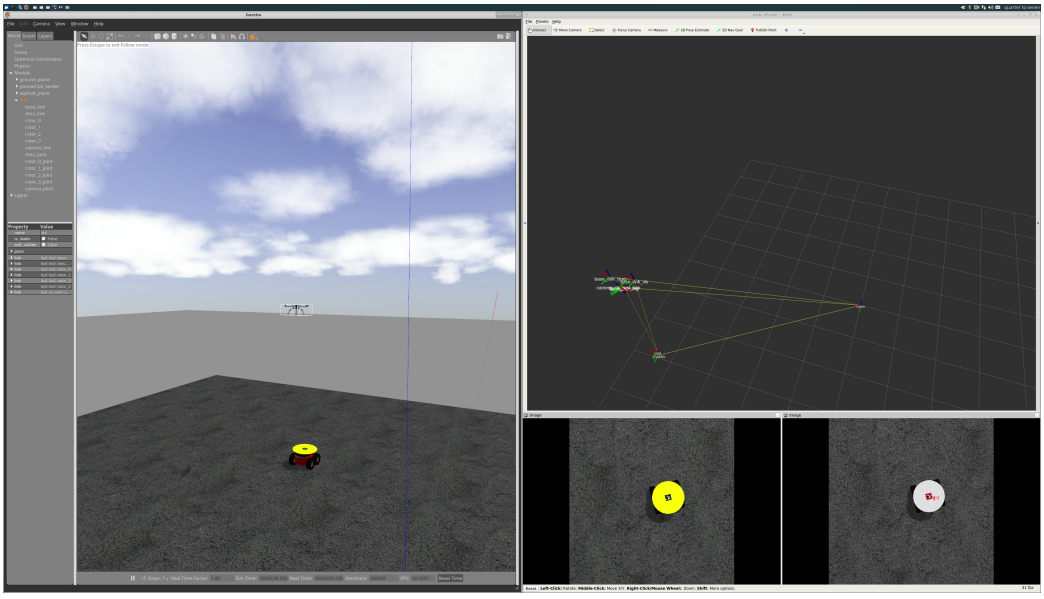


Figure B.4: The AprilTag has now returned a detection and the EKF estimate is likewise converging towards truth with two estimate to fuse together.

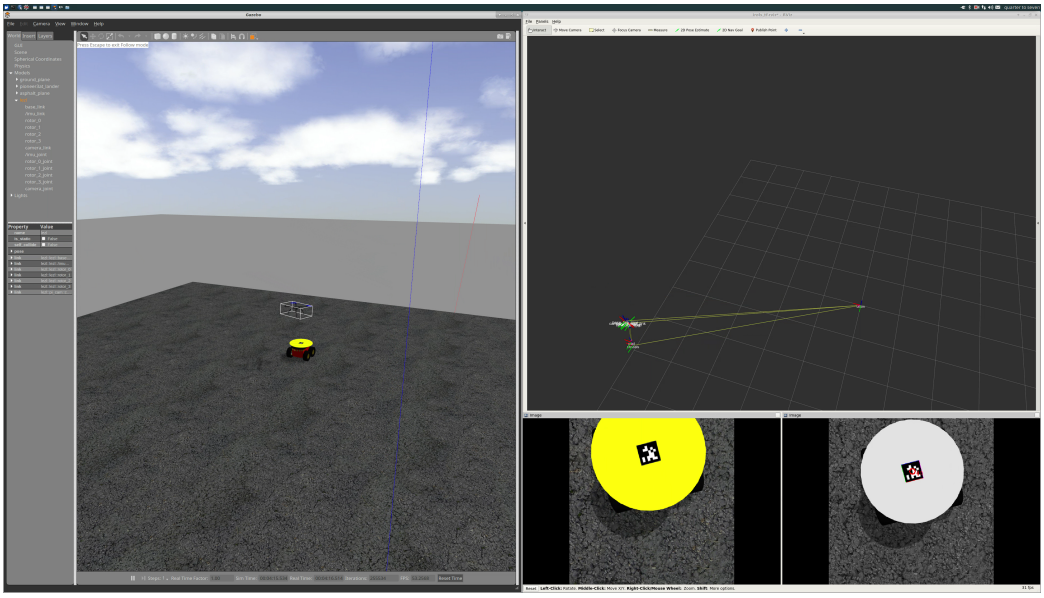


Figure B.5: The simulation state just before the landing sequence has completed. The estimates at this range are very reliable.