INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

**CONTROL AND NAVIGATION SYSTEM FOR
AUTONOMOUS VEHICLES AND ROBOTS USING FUZZY
LOGIC AND KALMAN FILTERING**

by

QI WANG, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirement for the degree of
Master of Engineering

**Department of Mechanical and Aerospace Engineering
Ottawa-Carleton Institute
for Mechanical and Aerospace Engineering**

Carleton University
Ottawa, Ontario
September 22, 1999
©Copyright
1999, Qi Wang

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-48463-7

The undersigned recommend to
the Faculty of Graduate Studies
acceptance of the thesis

**“CONTROL AND NAVIGATION SYSTEM FOR
AUTONOMOUS VEHICLES AND ROBOTS USING FUZZY
LOGIC AND KALMAN FILTERING”**

Submitted by **Qi Wang, B.Eng.**

In partial fulfillment of the requirements for
the degree of Master of Engineering

A handwritten signature in black ink, appearing to read "Robert Bell", written over a horizontal line.

Chair, Department of Mechanical and Aerospace Engineering

A handwritten signature in black ink, appearing to read "J. Jiradell", written over a horizontal line.
Thesis Supervisor

CARLETON UNIVERSITY

ABSTRACT

This thesis presents autonomous vehicle navigation in 3-D or 2-D environment. The vehicle has three different kinds of sensors to navigate in the obstacles populated environment. The obstacles may be static or dynamic. The vehicle's main sensor systems are sonar, global positioning system (GPS) and inertial navigation system (INS). The first sensor is used for obstacle avoidance and object recognition. The second and third sensor is used to determine the position and velocity. The signals from the Global Positioning System (GPS) and Inertial Navigation System (INS) are fused together using Adaptive Fuzzy Logic Kalman filter and the fused signal is fed to the vehicle control system. The control system is based on fuzzy logic controller (FLC). The FLC consists of two geometric modes and three dynamic loops. First group, geometric modes, the controller is making the decision how to follow from the starting point to its final goal and trace the edge of obstacles. Second group, dynamic loops, the controller is changing its velocity, attitude and acceleration dynamically. The results of simulations show that the fully autonomous vehicle can navigate in sparsely as well as densely populated environment. It has been demonstrated that the Fuzzy Adaptive Kalman Filter gives more accurate results than the Extended Kalman Filter does when INS fails or is modeled improperly.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Professor Jerzy Z. Sasiadek, my thesis advisor, for his constant guidance, encouragement and financial support in my work. I greatly benefit from his detailed comments and insights that help me clarify my ideas and present the materials in a suitable way. Without his help, my thesis can not have been completed.

I would like to thank staff of Department of Mechanical and Aerospace Engineering for providing an excellent research environment and various supports in the work.

I am grateful to all the people who help me in the work.

The financial support from Carleton University through graduate scholarships and teaching assistantships are gratefully acknowledged.

I must thank my wife, Li-ping for her continual patience, encouragement and support.

CONTENTS

ACCEPTANCE FORM

ABSTRACT

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1	INTRODUCTION	1
2	FUZZY LOGIC CONTROL	7
1.1	Introduction to Fuzzy Control	7
1.2	Basic Concepts of Fuzzy Sets	10
2.3	Fuzzy Logic	13
2.3.1	Fuzzy Intersection	13
2.3.2	Fuzzy Complement	13
2.3.3	Fuzzy Union	13
2.4	Fuzzy Rule-Base	14
2.5	Fuzzification	14
2.6	Inference Engine	15
2.6.1	Matching Process	15

2.6.2	Inference Step	17
2.7	Defuzzification	17
2.8	Tuning Fuzzy Controller	19
2.8.1	Input Scaling Gains and Output Scaling Gains	19
2.8.2	Tuning Membership Functions	20
3	KALMAN FILTER	22
3.1	Introduction	22
3.2	The Discrete Kalman Filter	23
3.3	Filter Parameters and Tuning	27
3.4	Extended Kalman Filter	29
4	GLOBAL POSITIONING SYSTEM, INERTIAL NAVIGATION SYSTEM AND SONAR SENSOR	34
4.1	Global Positioning System	34
4.1.1	Introduction	34
4.1.2	The Concept of GPS	36
4.1.3	Navigation Satellite Geometry	37
4.1.4	Major Error Sources of GPS	41
4.2	Inertial Navigation System	43
4.2.1	Introduction	43
4.2.2	The System Structure	43
4.3	Sonar-Ring Sensor System	45
5	FUZZY ADAPTIVE KALMAN FILTERING	49
5.1	Introduction	49

5.2	Weighted EKF	52
5.3	INS and GPS	55
5.4	Fuzzy Logic Adaptive System	56
5.4.1	Fuzzy Adaptive Kalman Filtering for Parameter Uncertainties	58
5.4.2	Fuzzy Adaptive Kalman Filtering for Non-white Process Noise	61
5.5	Extended Kalman Filter Model	63
5.5.1	Eight-State GPS/INS Linear Error Model	63
5.5.2	Measurement Model	67
5.6	Simulation Results	69
5.6.1	Simulation Experiment No.1	69
5.6.2	Simulation Experiment No.2	71
6	FUZZY CONTROL SYSTEM OF AUTONOMOUS VEHICLES	82
6.1	Introduction	82
6.2	Assumption of Fuzzy Control System	83
6.3	Path Planning	85
6.4	Path Planning with Dynamic Obstacles	86
6.4.1	Meeting Point and Collision Point	86
6.4.2	Planning and Motion	88
6.4.3	Time Requirement	89
6.5	Path Planning with Static Obstacles	90

6.6	Decision Mode and Trace Mode	91
6.7	Velocity Fuzzy Control Loop	95
6.8	Final Goal Approach Control Loop	97
6.9	Attitude Change Loop	98
6.10	Fuzzy Supervisory Control System	98
6.11	Simulation Experiments	100
6.11.1	Tuning	100
6.11.2	Simulation Results	101
7	CONCLUSIONS	112
	REFERENCES	115
	APPENDIX	120
	Appendix A: Simulation Code	120
A.1	Matlab code for fuzzy adaptive Kalman filtering	120
A.2	C code for fuzzy control system of autonomous vehicles	134

LIST OF TABLES

2.1	Mathematical characterization of triangular membership function	12
4.1	Satellite parameters	38
4.2	Correlated errors affecting pseudorange: Approximate statistical parameters for a stationary observer	42
5.1	Rule table of α for parameter uncertainties	60
5.2	Rule table of scale for parameter uncertainties	61
5.3	Rule table for non-white process noise	63
5.4	Comparison of theoretical and actual error variance	70
5.5	Summary of simulations	71
6.1	Rule table for decision and trace modes	95
6.2	Rule table for velocity control loop	97
6.3	Rule table for stop at final goal	97

LIST OF FIGURES

1.1	Mobile robot control hierarchy	2
1.2	Kalman filter stages	5
2.1	Fuzzy controller	8
2.2	Graphical representation of fuzzy inference	9
2.3	Membership function of five fuzzy sets	10
2.4	Some typical membership functions	11
2.5	Implied fuzzy sets	18
2.6	Fuzzy control gains	19
3.1	Model of random signal process and measurement	23
3.2	Kalman filter loop	27
3.3	An ordinary linearized Kalman filter for an aided inertial navigation system	29
3.4	An extended Kalman filter for an aided inertial navigation system	30
4.1	GPS basic system configuration	35
4.2	Ranging GPS solution	36
4.3	Circular orbital trajectory ($\alpha = 55^\circ$)	39
4.4	The WGS-84 coordinate reference frame (x, y, z) used by GPS and a local level coordinate reference frame (x', y', z')	40

4.5	Range error situation with an ultrasonic sensor	46
4.6	Sonar-ring system	48
5.1	True filter divergence	50
5.2	Apparent filter divergence	51
5.3	Fuzzy adaptive extended Kalman filter	52
5.4	Exponential data weighting	53
5.5	Covariance membership functions for parameter uncertainties	59
5.6	Mean value membership functions for parameter uncertainties	59
5.7	α membership functions for parameter uncertainties	59
5.8	Covariance membership functions for non-white process noise	62
5.9	Mean value membership functions for Non-white process noise	62
5.10	α membership functions for non-white process noise	62
5.11	North channel error model	64
5.12	East channel error model	64
5.13	Altitude channel error model	65
5.14	Actual and theory covariance for 5Q and R	74
5.15	Actual and theory covariance for 5Q and 4R	75
5.16	The position error and innovation sequence of EKF with white noise inputs	76

5.17	Kalman gain and position error for EKF and FLAC-EKF	77
5.18	Innovation sequences of EKF and FLAC-EKF	78
5.19	Position and velocity errors (the smaller mean) of EKF and FLAC-EKF	79
5.20	Position and velocity errors (the larger mean) of EKF and FLAC-EKF	80
5.21	Position and velocity errors (the smaller r) of EKF and FLAC-EKF	81
6.1	Block diagram of fuzzy control system	85
6.2	Collision front	87
6.3	Plan and Motion	88
6.4	Obstacle movement	89
6.5	Physical space and its configuration space	90
6.6	Vehicle's position	92
6.7	Attitude membership functions	94
6.8	Distance membership functions	94
6.9	Change of attitude membership functions	94
6.10	Change-in-velocity membership functions	96
6.11	Velocity membership functions	96
6.12	Change-in-attitude membership functions	98
6.13	Navigation without applying fine tune	103
6.14	Code structure chart	104
6.15	Triangular membership functions	105

6.16	Gaussian membership functions	106
6.17	2 - D navigation	107
6.18	2 - D motion	108
6.19	Avoiding both static and dynamic obstacles	109
6.20	3 - D navigation	110
6.21	3 - D motion	111

CHAPTER 1

INTRODUCTION

In recent years, an increasing amount of robotics research has focused on the problem of planning and executing motion tasks autonomously, without human guidance. An autonomous robot should be able to gather information about the environment and accordingly perform efficient motions for achieving its tasks. The technologies that are associated with autonomous vehicle development have wide ranging applications from automated manufacturing and underground mining to unmanned exploration of the oceans and space. An autonomous vehicle must have the following capabilities [1]:

- Autonomous planning and preparation of actions according to a given task
- Independent execution and supervision of the actions
- Understanding of the environment and interpretation of the results from sensor information
- Independent reaction to unforeseen events
- Passive and active learning capabilities

Figure 1.1 Mobile robot control hierarchy [2]

Navigation is the determination of the position and velocity of a moving vehicle. The three components of position and the three components of velocity make up a six-components of state vector that fully describes the motion of the vehicle. The navigation problem is to find a path from start (S) to goal (G) and traverse it without collision. We can decompose navigation into three sub-tasks: mapping and modeling the environment; path planning and selection; and path traverse and collision avoidance. The relationship between these tasks is shown in Figure 1.1.

Fuzzy logic controllers (FLC) have been used in many applications over the past decade, allowing the application of heuristic information to be used in the control of complex systems. The heuristic information in a FLC consists of a set of linguistic IF – THEN rules, comprising the fuzzy logic rule base. Inputs to the FLC are in the form of linguistic variables, which belong in varying degrees to various fuzzy membership sets. These variables are matched to conditions defined by the individual members of the fuzzy rule base and the response of each rule is weighted according the degree of membership of its inputs. Finally, the weighted variables of the responses of all rules in the rule base are calculated to generate an appropriate output signal.

There have been many successful uses of fuzzy logic control in various knowledge-based systems to control real-time decision-making in the area of command and control for autonomous vehicles [3-10]. In [3], Oriolo and Ulivi described an ' execution monitor' (EM), designed as a fuzzy controller which provides a car-like vehicle moving in a dynamic and partially known environment

with motion autonomy and uses supervised learning for partial automation of its design. In [6], Li incorporated fuzzy logic control with behavior-based control. The 'perception-action' behaviors are formulated by fuzzy sets and fuzzy rules; conflicts and competitions among different behaviors are coordinated by fuzzy reasoning. Saffiotti [8] used fuzzy logic to coordinate small units of control, or behaviors. Goodridge et al. [9] applied a fuzzy behaviors along with a motivational state. But most of studies have only dealt with static obstacles [4] or not included the dynamic characteristics of vehicles [3][5].

Multisensor navigation is the process of estimating the navigation variables of position, velocity, and attitude from a sequence of measurements from more than one navigation sensor. There are essentially two broad categories for navigation and related functions [11]: dead-reckoning sensors and position sensors. Dead-reckoning sensors provide a measure of acceleration or velocity with respect to an Earth-referenced coordinate system, consequently requiring integration with respect to time to provide vehicle position with respect to the Earth. Examples of these types of sensors are inertial navigation system (INS), Doppler radar and air-data sensors. Position sensors provide a position measurement that can be related to Earth-referenced coordinates. Examples of these sensors are radio systems such as the terrestrial-based Loran and the satellite-based Global Positioning System (GPS).

The Kalman filter is used in a number of multisensor systems when it is necessary to fuse dynamic low-level redundant data in real-time. The filter uses the statistical characteristics of a measurement model to recursively determine

estimates for the fused data that are optimal in a statistical sense. Figure 1.2 is the summary of Kalman filter stages.

Figure 1.2 Kalman filter stages

The objective of this thesis is to design and investigate the 3-D guidance and navigation for mobile/flying robots using fuzzy logic and Kalman filtering.

The guidance and navigation has been done in the presence of static and dynamic obstacles. The planner has essentially two geometric modes and three dynamic loops of operation. First group, geometric modes, including decision and trace modes, where the controller is making the decision how to follow from the starting point to its final goal and to trace the edge of the obstacle. The fuzzy logic controller is used to track the trajectory and trace obstacles; Second group, dynamic loops, including velocity fuzzy control, final goal approach control and

attitude change loops, where the controller is changing its velocity, attitude and acceleration.

A method of sensor fusion based on the Adaptive Fuzzy Kalman Filtering has been developed. This method has been applied to fuse position signals from the Global Positioning System (GPS) and Inertial Navigation System (INS) for the autonomous mobile vehicles. The innovation sequence of Extended Kalman Filter (EKF) has been used to monitor the performance of EKF and a fuzzy logic controller is applied to prevent divergence of EKF.

Chapter 2 is an introduction of fuzzy logic control. The description of fuzzy logic systems, four training algorithms for the fuzzy logic systems, concepts of fuzzy set and fuzzy logic, and tuning fuzzy controller are presented.

Chapter 3 describes the Kalman filter theory, algorithms, and filter parameters and tuning.

Chapter 4 is an illustration of Global Positioning System (GPS), Inertial Navigation System (INS), and ultrasonic sensor.

Chapter 5 presents a fuzzy adaptive Kalman filter technique to prevent the divergence of an extended Kalman filter and adapt Kalman filter.

Chapter 6 addresses ways that use fuzzy logic controller to control autonomous vehicles. Two geometric modes and three dynamic loops are applied.

Chapter 7 is the conclusions and discussions.

CHAPTER 2

FUZZY LOGIC CONTROL

2.1 Introduction to Fuzzy Control

Conventional automatic control system design methods involve the construction of mathematical models describing the dynamic system to be controlled and the application of analytical techniques to the model to derive control laws. Such controllers are the products of decades of development and theoretical analysis, and are highly effective.

But in some cases, the mathematical model of the control process may not exist. Even if a relatively accurate model of a dynamic system can be developed, it is often highly nonlinear and too complex to use in controller's development, or may be too "expensive" in terms of computer processing power and memory. A system based on empirical rules may be more effective in this situation.

Fuzzy control provides a formal methodology for representing, manipulating, and implementing a human's heuristic knowledge about how to control a system[12]. Modelling and control techniques based on fuzzy sets attempt to combine numerical and symbolic processing into one framework. On the other hand, fuzzy systems are knowledge-based systems consisting of linguistic If –

Then rule that can be constructed using the expert's knowledge in the given field of interests [13].

The basic configuration of a fuzzy controller is shown in Figure 2.1.

Figure 2.1 Fuzzy Controller

In generally, a fuzzy controller is composed of the following four elements [12]:

1. A fuzzification interfaces which converts controller into information that the inference mechanism can easily use to activate and apply rules. For example, an attitude of 10 degrees may be converted into positive small and zero values;
2. A rule-base which contains a fuzzy logic quantification of the expert's linguistic description of how to achieve good control. Suppose that we control a vehicle with its velocity as inputs and the change of attitude as an output. One of the fuzzy control rules may be expressed in the If-then form:

If the velocity is large **then** the change of attitude is small;

3. An inference engine which emulates the expert's decision making in interpreting and applying knowledge about how to control the plant. It produces fuzzy conclusions;
4. A defuzzification interface which converts the conclusions of the inference mechanism into actual inputs for the process. The most often-used defuzzification methods are centre of gravity and centre average.

A graphical depiction of fuzzy decision making is given as follows:

Figure 2.2. Graphical representation of fuzzy inference

If using "center-average" method (equation 2.3) for defuzzification,

$$u^{crisp} = [0 \times 0.7 + (-25) \times 0.3] / (0.7 + 0.3) = -7.5$$

In this situation, the output of fuzzy controller, change of attitude is -7.5 degrees.

2.2 Basic Concepts of Fuzzy Sets

Let U be a collection of objects, for example, $U = u_i$, and be called the universe of discourse. A fuzzy set F in U is characterized by a membership function $\mu_F : U \rightarrow [0,1]$, with $\mu_F(u_i)$ representing the grade of membership of $u_i \in U$ in the fuzzy set F . A fuzzy set may be viewed as a generalization of the concept of an ordinary set (that is, a crisp set) whose membership function takes any values in the interval $[0,1]$ [14]

Figure 2.3 Membership function of five fuzzy sets

Figure 2.3 shows the membership functions of five fuzzy sets for the attitude of a vehicle. In this example, the universe of discourse is all possible attitudes of a vehicle; that is,

$U = [-\alpha_{\max}, \alpha_{\max}]$, where $\alpha_{\max}$ is the maximum attitude of the vehicle. The fuzzy sets can be named as, "negative large (neglarge)", "negative small (negsmall)", "zero", "positive small (possmall)", and "positive large (poslarge)". At an attitude of -37.5, the fuzzy set "negative small" has membership value 0.5, that is, $\mu_{\text{negative small}}(-37.5) = 0.5$. the fuzzy set "zero" has membership value 0.5, that is, $\mu_{\text{zero}}(-37.5) = 0.5$. and other fuzzy sets have membership value 0, that is, other $\mu_F(-37.5) = 0$.

There are many choices for the shape of the membership function, such as triangular, trapezoidal, Gaussian shapes. These will each provide a different meaning for the linguistic values that they quantify.

Figure 2.4 shows a graphical illustration of a variety of membership functions.

Figure 2.4 Some typical membership functions

Table 2.1 is the mathematical characterization of triangular membership function [12].

Table 2.1 Mathematical characterization of triangular membership function

	Triangular membership functions	
Left	$\mu^L(u) = \begin{cases} 1 & \text{if } u \leq c^L \\ \max\left\{0, 1 + \frac{c^L - u}{0.5w^L}\right\} & \text{otherwise} \end{cases}$	
Center	$\mu^C(u) = \begin{cases} \max\left\{0, 1 + \frac{u - c}{0.5w}\right\} & \text{if } u \leq c^L \\ \max\left\{0, 1 + \frac{c - u}{0.5w}\right\} & \text{otherwise} \end{cases}$	
Right	$\mu^R(u) = \begin{cases} 1 & \text{if } u \leq c^R \\ \max\left\{0, 1 + \frac{u - c^R}{0.5w^R}\right\} & \text{otherwise} \end{cases}$	

where, c^L specifies the "saturation point" and w^L specifies the slope of the non-unity and nonzero part of μ^L . Similarly, for μ^R . For μ^C notice the c is the center of triangle and w is the base-width.

Mathematical characterization of Gaussian membership functions is

$$\mu(u) = \begin{cases} \exp\left(-\frac{1}{2}\left(\frac{u-c}{\sigma}\right)^2\right) & \text{if } c^L \leq u \leq c^R \\ 1 & \text{otherwise} \end{cases} \quad (2.1)$$

where, $c=c^L$, and $\sigma=\sigma^L$ in left, $c=c^R$ and $\sigma=\sigma^R$ in right.

2.3 Fuzzy Logic

2.3.1 Fuzzy Intersection

Let A and B be two fuzzy sets in U. The intersection $A \cap B$ of A and B is a fuzzy set in U with a membership function defined by the following two most commonly used methods:

1. Minimum:

$$\mu_{A \cap B}(u) = \min\{\mu_A(u), \mu_B(u) : u \in U\}$$

2. Algebraic Product:

$$\mu_{A \cap B}(u) = \{\mu_A(u)\mu_B(u) : u \in U\}$$

2.3.2 Fuzzy Complement

The complement ("not") of a fuzzy set A with the membership function $\mu_A(u)$ has a membership function given by $1 - \mu_A(u)$.

2.3.3. Fuzzy Union

The union of $A \cup B$ of A and B is a fuzzy set in U with the membership defined by

1. Maximum:

$$\mu_{A \cup B}(u) = \max \{ \mu_A(u), \mu_B(u) : u \in U \}$$

2. Algebraic Sum:

$$\mu_{A \cup B}(u) = \{ \mu_A(u) + \mu_B(u) - \mu_A(u)\mu_B(u) : u \in U \}$$

2.4 Fuzzy Rule-Base

A fuzzy rule base consists of a collection of fuzzy IF-THEN rules to make a "fuzzy implication". The rules of a linguistic fuzzy model have the general form:

If premise **Then** consequent

The multi-input single-output (MISO) form of a linguistic rule is

If x_1 is A_1 and ... and x_m is A_m **then** y is B

Where A_1, B is fuzzy sets, and $x_i \in U$ and $y \in V$ are linguistic variables.

The fuzzy rule base is the heart of the fuzzy logic system in the sense that all other three components (fuzzy interference, fuzzification, and defuzzification) are used to interpret these rules and make them usable for specific problems.

2.5 Fuzzification

Fuzzy sets are used to quantify the information in the rule base, and the interference engine operates on fuzzy sets to produce fuzzy sets. The fuzzification performs a mapping from a crisp point $x = (x_1, \dots, x_n)^T \in U$ into a fuzzy set A' in U . Quite often "singleton fuzzification" is used, where A' is a fuzzy singleton with support x , that is,

$$\mu_{A'}(x') = \begin{cases} 1 & x' = u \\ 0 & \text{otherwise} \end{cases}$$

Fuzzification process as the act of obtaining a value of an input variable and finding the numeric values of the membership functions that are defined for that variable.

For example, in Figure 2.3, if $\alpha=75$, the fuzzification process amounts to finding the values of the input membership functions for this. In this case,

$$\mu_{poss\ small}(\alpha)=1, \text{ with all others zero.}$$

Some think of the membership function values as an "encoding" of the fuzzy controller numeric input values. The encoded information is then used in the fuzzy inference process that starts with "matching".

2.6 Inference Engine

In a fuzzy inference engine, fuzzy logic principles are used to combine the fuzzy IF-THEN rules in the fuzzy rule base into a mapping from fuzzy sets in $U=U_1 \times U_2 \dots \times U_n$ to fuzzy sets in V .

2.6.1 Matching Process

In order to perform inference process we must first quantify each of the rules with fuzzy logic.

For a linguistic rule

If x_1 is A_1 and ... and x_m is A_m **then** y is B

Hence, the main item to focus on is how to quantify the logical "and" operation that combines the meaning of linguistic terms.

If the singleton fuzzification is used, for the given u_i inputs,

$$\begin{aligned}\mu_{A_1}^{\cdot}(u_1) &= \mu_{A_1}(u_1) \\ \mu_{A_2}^{\cdot}(u_2) &= \mu_{A_2}(u_2) \\ &\vdots \\ \mu_{A_n}^{\cdot}(u_n) &= \mu_{A_n}(u_n)\end{aligned}$$

We denote the certainty of the premise by $\mu_{premise}$. Generally, we use following two ways to define it:

- Minimum:

$$\mu_{premise} = \min\{\mu_{A_1}^{\cdot}(u_1), \mu_{A_2}^{\cdot}(u_2) \cdots \mu_{A_n}^{\cdot}(u_n)\}$$

- Product:

$$\mu_{premise} = \mu_{A_1}^{\cdot}(u_1) \mu_{A_2}^{\cdot}(u_2) \cdots \mu_{A_n}^{\cdot}(u_n)$$

We say that a rule is "on at time t " if its premise membership function $\mu_{premise} > 0$. Hence, the inference mechanism seeks to determine which rules are on to find out which rules are relevant to the current situation.

2.6.2 Inference Step

Here, we need to determine the conclusions when the rules are on.

The rule indicates that if the promise is true then the action indicated by its consequent should be taken. The membership function for the conclusion reached by rule (i) is given:

$$\mu_{(i)}(u) = \min \{ \mu_{premise(i)}, \mu_{consequence(i)} \}$$

It is an implication that we can be no more certain about our consequent than our premise. While the input to the inference process is the set of rules that are on, its output is the set of implied fuzzy sets that represent the conclusions reached by all the rules that are on.

2.7 Defuzzification

The defuzzification performs a mapping from fuzzy sets in V to a crisp point $y \in V$.

Generally, there are two kinds of defuzzification methods.

1. "Center of gravity " (COG)

Let b_i denote the center of the membership function of the consequent of rule(i)(where it reaches its peak), and let $\int \mu_{(i)}$ denote the area under the membership function $\mu(i)$. The COG method computer u^{crisp} to be

$$u^{crisp} = \frac{\sum_i b_i \int \mu_{(i)}}{\sum_i \int \mu_{(i)}} \quad (2.2)$$

2. " Center-average "

$$u^{crisp} = \frac{\sum_i b_i \mu_{premise_{i,1}}}{\sum_i \mu_{premise_{i,1}}} \quad (2.3)$$

We call it the "center-average" method since equation (2.3) is a weighted average of the center values of the output membership function centers. Basically, the center-average method replaces the areas of the implied fuzzy sets that are used in COG with the values of $\mu_{premise_{i,1}}$.

Figure 2.5 Implied fuzzy sets

2.8 Tuning Fuzzy Controller

2.8.1 Input Scaling Gains and Output Scaling Gains

Figure 2.6 Fuzzy control gains

When the scaling gains at inputs u_i and outputs y_i of fuzzy controller are changed, the performance of the resulting fuzzy control system has a significant impact.

Generally, the scaling gains have following effects [12]:

- If the scaling gains = 1, there is no effect on the membership functions.
- If the input gains < 1 , the membership function are uniformly "spread out" by factor of $1/g_i$. It changes the meaning of the linguistics so that the membership functions represent larger numbers.
- If the output gains < 1 , there is the effect of contracting the output membership functions and hence making the meaning of their associated linguistics quantify smaller numbers.
- If the input gains > 1 , the membership functions are uniformly "contracted", the membership functions now represent smaller numbers.

- If the output gains >1 , there are the effect of spreading out the output membership functions and hence making the meaning of their associated linguistics quantify larger numbers.

2.8.2 Tuning Membership Functions

Sometimes, the rule-base and membership functions cannot achieve the desired performance by tuning only the scaling gains. Often, what is needed is a more careful consideration of how to specify additional rules or better membership functions.

There are many parameters of membership functions to tune, such as, membership function shapes, positioning, and number or type of rules. If we add more membership functions and rules, these can provide for a "finer" control, which can sometimes reduce chattering or oscillations.

One of very useful methods of real implementations for fuzzy control systems is output membership function tuning. In this method we will tune the positioning of the output membership functions by characterizing their centers by a function. Suppose that we use c^i , $i = -n, \dots, -2, -1, 0, 1, 2, \dots, n$ to denote the centers of the output membership functions for a fuzzy controller, where the indices i for the c^i are the linguistic-numeric values used for the output membership function.

Generally, we use

$$c^i = hi,$$

Suppose, that we choose instead

$$c^i = h \text{sign}(i) i^2$$

where, $sign(i)$ returns the sign of the member i and $sign(0) = 1$. The effect of this is to make the "gain" of fuzzy controller smaller when the signals are small and larger as the signals grow larger.

Another way is that the width and center of input and output membership functions can be divided unevenly. The fuzzy controller will weight more on the dense area and be more sensitive in this area. For instance, if the centers of nine membership functions are $\{-1, -0.6, -0.2, -0.1, 0, 0.1, 0.2, 0.7, 1\}$, the fuzzy controller will pay more attention to the area from -0.2 to 0.2. If, the width of seven membership functions are $\{0.6, 0.5, 0.2, 0.1, 0.2, 0.5, 0.7\}$, for the same reason, the fuzzy controller will take care of more in the center area.

CHAPTER 3

KALMAN FILTER

3.1 Introduction

In 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem. Since that time, due in large part to advances in digital computing, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. Generally, the linear least-squares filter problem is this: Given the spectral characteristics of additive combination of signal and noise, what linear operation on this input combination will yield the best separation of the signal from the noise? In this case, "best" means minimum mean-square error in a time series.

The filter is very powerful in several aspects: it can estimate past, present, and even future states, and do so even when the precise nature of the modeled system is unknown. Given some initial estimates, it allows the parameters of a model to be predicted and adjusted with each new measurement, providing an

estimate of error at each update. It can incorporate the effects of noise (from both measurement and modeling), and its computational structure.

3.2 The Discrete Kalman Filter

Figure 3.1 Model of random signal process and measurement

The random process to be estimated can be modeled in the form

$$\mathbf{x}_{k+1} = \Phi_k \mathbf{x}_k + \mathbf{w}_k \quad (3.1)$$

The observation (measurement) of the process is assumed to occur at discrete points in time in accordance with the linear relation

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \quad (3.2)$$

where

$\mathbf{x}_k = (n \times 1)$ state of the process at time t_k , i.e., $\mathbf{x}_k = \mathbf{x}(t_k)$

$\Phi_k = (n \times n)$ matrix that relates $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ in the absence of a forcing function (If

$\mathbf{x}_k$ is a sample of continuous process, Φ_k is the usual state transition matrix)

$\mathbf{w}_k = (n \times 1)$ vector --- assumed to be a white sequence with known covariance structure

$\mathbf{z}_k = (m \times n)$ vector measurement at time t_k

$\mathbf{H}_k$ = linear connection matrix between output $\mathbf{y}_k$ and state $\mathbf{x}_k$

$\mathbf{v}_k = (m \times n)$ measurement error --- assumed to be a white sequence with known covariance structure and having zero crosscorrelation with the $\mathbf{w}_k$ sequence

The covariance matrices for the $\mathbf{w}_k$ and $\mathbf{v}_k$ vectors are given by

$$E[\mathbf{w}_k \mathbf{w}_i^T] = \begin{cases} \mathbf{Q}_k, & i = k \\ \mathbf{0}, & i \neq k \end{cases} \quad (3.4)$$

$$E[\mathbf{v}_k \mathbf{v}_i^T] = \begin{cases} \mathbf{R}_k, & i = k \\ \mathbf{0}, & i \neq k \end{cases} \quad (3.5)$$

$$E[\mathbf{w}_k \mathbf{v}_i^T] = 0, \quad \text{for all } k \text{ and } i \quad (3.6)$$

Suppose, the plant to be controlled is described by

$$\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{w}(t) \quad (3.7)$$

If the dynamic system has fixed parameters (i.e., $\mathbf{F}$ is a constant), the Φ may be written as an exponential series.

$$\Phi_k = e^{\mathbf{F}\Delta t} = \mathbf{I} + (\mathbf{F}\Delta t) + (\mathbf{F}\Delta t)^2/2! + \dots \quad (3.8)$$

We assume at this point that we have an initial estimate of the process at some point in time t_k and that this estimate is based on all our knowledge about the processor prior t_k . This prior estimate will be denoted as $\hat{\mathbf{x}}_k^-$ where the "hat" denotes estimate, and the "super minus" is a reminder that this is our best estimate

prior to assimilating the measurement at t_k . We also assume that we know the error covariance matrix associated with $\hat{\mathbf{x}}_k^-$. We define the estimation error to be

$$\mathbf{e}_k^- = \mathbf{x}_k - \hat{\mathbf{x}}_k^-$$

and the associated error covariance matrix is

$$\mathbf{P}_k^- = E[\mathbf{e}_k^- \mathbf{e}_k^{-T}] = E[(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)(\mathbf{x}_k - \hat{\mathbf{x}}_k^-)^T] \quad (3.9)$$

With the assumption of a prior estimate $\hat{\mathbf{x}}_k^-$, we now seek to use the measurement $\mathbf{z}_k$ to improve the prior estimate. We choose a linear blending of the noisy measurement and prior estimate in accordance with the equation

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_k^-) \quad (3.10)$$

where:

$\hat{\mathbf{x}}_k$ = updated estimate

$\mathbf{K}_k$ = blending factor (yet to be determined)

We use minimum mean - square error as the performance criterion.

$$\mathbf{P}_k = E[\mathbf{e}_k \mathbf{e}_k^T] = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T \quad (3.11)$$

We want to find the particular $\mathbf{K}_k$ that minimizes the individual terms along the major diagonal of $\mathbf{P}_k$, because these terms represent the estimation error variances for the elements of the state vector being estimated. By using differential calculus approach, we proceed to differentiate the trace of $\mathbf{P}$ with

respect to $\mathbf{K}$, and set the derivative equal to zero and solve for the optimal gain.

The result

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \quad (3.12)$$

The particular $\mathbf{K}_k$ namely the one that minimizes the mean - square estimation error, is called the Kalman gain.

The covariance matrix associated with optimal estimate is computed.

$$\mathbf{P}_k = \mathbf{P}_k^- - \mathbf{K}_k (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k) \mathbf{K}_k^T \quad (3.13)$$

or

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \quad (3.14)$$

The updated estimated $\hat{\mathbf{x}}_k$ is projected ahead via the transition matrix. Because it has zero mean and is not correlated with any of the previous $\mathbf{w}_k$. Thus we have

$$\hat{\mathbf{x}}_{k+1}^- = \Phi_k \hat{\mathbf{x}}_k \quad (3.15)$$

The expression for update estimated $\mathbf{P}_{k+1}$

$$\begin{aligned} \mathbf{P}_{k+1}^- &= E[\mathbf{e}_{k+1}^- \mathbf{e}_{k+1}^{-T}] = E[(\Phi_k \mathbf{e}_k + \mathbf{w}_k)(\Phi_k \mathbf{e}_k + \mathbf{w}_k)^T] \\ &= \Phi_k \mathbf{P}_k \Phi_k^T + \mathbf{Q}_k \end{aligned} \quad (3.16)$$

Figure 3.2 Kalman filter computing sequence

3.3 Filter Parameters and Tuning

In the actual implementation of the filter, each of the measurement error covariance matrix $\mathbf{R}$ and the process noise covariance matrix $\mathbf{Q}$ (given by (3.5) and (3.4) respectively) might be measured prior to operation of the filter. In the case of the measurement error covariance $\mathbf{R}$ in particular this makes sense--because we need to be able to measure the process (while operating the filter) we should generally be able to take some off-line sample measurements in order to determine the variance of the measurement error.

In the case of $\mathbf{Q}$, often times the choice is less deterministic. For example, this noise source is often used to represent the uncertainty in the process model (3.1).

Sometimes a very poor model can be used simply by "injecting" enough uncertainty via the selection of $\mathbf{Q}$. Certainly in this case one would hope that the measurements of the process would be reliable.

In either case, whether or not we have a rational basis for choosing the parameters, often times superior filter performance (statistically speaking) can be obtained by "tuning" the filter parameters $\mathbf{Q}$ and $\mathbf{R}$. The tuning is usually performed off-line, frequently with the help of another (distinct) Kalman filter. Changing $\mathbf{Q}$ or $\mathbf{R}$ in a filter structure does affect the transient duration and the steady state operation. Increasing $\mathbf{Q}$ would indicate either stronger noises driving the dynamics or increased uncertainty in the adequacy of the model itself to depict the true dynamics accurately. As a result, the filter gains will generally increase, thereby weighting the measurements more heavily: this is reasonable since increased $\mathbf{Q}$ dictates that we should put less confidence in the output of the filter's own dynamics model. By similar reasoning, increased $\mathbf{R}$ would indicate that the measurements are subjected to a stronger corruptive noise, and so should be weighted less by the filter. In fact, this will decrease the gain values.

The difference $\mathbf{z} - \mathbf{H}_k \hat{\mathbf{x}}_k^-$ in (3.10) is called the measurement innovation, or the residual. The residual reflects the discrepancy between the predicted measurement $\mathbf{H}_k \hat{\mathbf{x}}_k^-$ and the actual measurement $\mathbf{z}$. A residual of zero means that the two are in complete agreement.

3.4 Extended Kalman Filter

Many dynamic systems and sensors are not absolutely linear, but they not far from it.

There are two basic ways of linearizing the problem. One is to linearize about some nominal trajectory in state space that does not depend on the measurement data. The resulting filter is usually referred to as simply a linearized Kalman filter (Figure 3.3). The other method is to linearize about a trajectory that is continually updated with the state estimates resulting from the measurements. When this is done, the filter is called an extended Kalman filter (EKF) (Figure 3.4). The linearized filter is always estimating incremental quantities, and then the total quantity is reconstructed by adding the incremental estimate to the nominal part.

Figure 3.3 An ordinary linearized Kalman filter for an aided Inertial navigation system

Figure 3.4 An extended Kalman filter for an aided
inertial navigation system

The EKF approach is to apply the standard Kalman filter (for linear systems) to nonlinear systems with additive white noise by continually updating a linearization around the previous state estimate, starting with an initial guess. In other words, we only consider a linear Taylor approximation of the system function at the previous state estimate and that of the observation function at the corresponding predicted position. This approach gives a simple and efficient algorithm to handle a nonlinear model. However, convergence to a reasonable estimate may not be obtained if the initial guess is poor or if the disturbances are so large that the linearization is inadequate to describe the system.

Generally, the extended Kalman filter is used where the mission time is long, such as a ship at sea for many weeks or months. Otherwise, the reference

trajectory would diverge from real trajectory beyond acceptable limits. On the other hand, if the mission time is short, the actual trajectory will match the preprogrammed one closely. In this case an ordinary linearized Kalman filter might well be preferred.

The equations of EKF are given as following:

Nonlinear dynamic model:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, k) + \mathbf{w}_k \quad (3.17)$$

where

$$\mathbf{w}_k \sim N(0, \mathbf{Q});$$

Nonlinear measurement model:

$$\mathbf{z}_k = h(\mathbf{x}_k, k) + \mathbf{v}_k \quad (3.18)$$

where:

$$\mathbf{v}_k \sim N(0, \mathbf{R});$$

Nonlinear implementation equations for approximate time and measurement updates are given as following:

Computing the predicted state estimate:

$$\hat{\mathbf{x}}_{k+1}^- = f(\hat{\mathbf{x}}_k, k) \quad (3.19)$$

Computing the predicted measurement:

$$\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_k^-, k) \quad (3.20)$$

Linear approximation equations:

$$\Phi_k \approx \left. \frac{\partial f(x, k)}{\partial x} \right|_{x=\hat{\mathbf{x}}_k^-} \quad (3.21)$$

Conditioning the predicted estimate on the measurement:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \hat{\mathbf{z}}_k) \quad (3.22)$$

$$\mathbf{H}_k \approx \left. \frac{\partial h(x, k)}{\partial x} \right|_{x=\hat{\mathbf{x}}_k^-} \quad (3.23)$$

Computing the *a priori* covariance matrix:

$$\mathbf{P}_{k+1}^- = \Phi_k \mathbf{P}_k \Phi_k^T + \mathbf{Q}_k \quad (3.24)$$

Computing the Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}_k^T [\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R}_k]^{-1} \quad (3.25)$$

Computing the *a posteriori* covariance matrix:

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^- \quad (3.26)$$

These equations represent an approximate, computationally feasible linear measurement update for the estimate and error covariance. The residual is

computed using the nonlinear measurement function $h(.,k)$ evaluated at a priori estimate $\hat{\mathbf{x}}_k^-$. The error covariance is found using the Jacobian matrix $\mathbf{H}(\hat{\mathbf{x}}_k^-)$.

Since the time and measurement updates depend on Jacobians evaluated at the current estimate, the error covariance and Kalman gain $\mathbf{K}_k$ cannot be computed off-line for the EKF. They must be computed in real-time, as the data become available.

CHAPTER 4

GLOBAL POSITIONING SYSTEM, INERTIAL NAVIGATION SYSTEM AND SONAR SENSOR

4.1 Global Positioning System

4.1.1 Introduction

Since the 1960s, the use of satellites was established as an important means of navigation on Earth. Global Positioning System (GPS) is a passive, survivable, continuous, space-based system that provides any suitably equipped user with highly accurate three-dimensional position, velocity, and time information anywhere on or near Earth. Position and time determination is accomplished by the reception of GPS signals to obtain ranging information as well as messages transmitted by the satellites.

GPS has the basic system configuration illustrated in Figure 4.1. The monitoring and satellite control sites are dispersed around the world.

Figure 4.1 GPS basic system configuration

The fully operational GPS satellite constellation comprises 24 satellites, four each in six 55°-inclined orbit planes spaced 60° apart in longitude. The control segment monitors the GPS satellites, checking both their operational health and exact position in space. The master control station transmits corrections for the satellite's ephemeris constants and clock offsets back to the satellites themselves. The satellites can then incorporate these updates in the signals they send to GPS receivers.

4.1.2 The Concept of GPS

The concept of GPS is illustrated in Figure 4.2.

Figure 4.2 Ranging GPS solution [11]

The measurements are not range and range rate, but quantities described as pseudorange and pseudorange range rate. Initially an autonomous user is not expected to precisely synchronize to the satellite system time. In the GPS positioning problem, we need to solve for at least three degrees of freedom for position and one degree of freedom for time.

Neglecting for the moment the clock and other measurement errors, the range to satellite i is given as

$$R_i = \sqrt{(X_i - x)^2 + (Y_i - y)^2 + (Z_i - z)^2} \quad (4.1)$$

where:

$[X_i, Y_i, Z_i]^T$ = Cartesian position coordinates of satellite i

$[x, y, z]^T$ = Cartesian position coordinates of observer

The actual measurement is in form

$$\psi_i = R_i + c\Delta t \quad (4.2)$$

where:

ψ_i = noiseless pseudorange

Δt = receiver offset from the satellite system time

c = speed of light

The observer position $[x, y, z]^T$ is "slaved" to the coordinate frame of reference used by the satellite system. For GPS, this reference is a geodetic datum called WGS-84 (Figure 4.4) (for World Geodetic System of 1984), which is Earth - Centered and Earth - Fixed. The datum also defines the ellipsoid that crudely approximates the surface of the earth. Although the satellite positions are reported in WGS - 84 coordinates, it is sometimes useful to deal with a locally level frame of reference, where the $x' - y'$ plane is tangential to the surface of the Earth ellipsoid.

4.1.3 Navigation Satellite Geometry

The exact description of a satellite orbital trajectory can be very complex. A perfectly elliptical orbit can be fully described by six parameters: size and flatness of the ellipse, orientation of the elliptical plane, orientation of ellipse in

its plane, and position of the satellite in the elliptical orbit. These are known as *Keplarian* parameters.

For simplicity of our simulation, these orbits are assumed to be perfectly circular with a radius R (26560 km) and inclined 55 degrees to the equatorial plane. As seen in Figure 4.3, the parameter Ω , called the right ascension, is the longitude where the orbital plane intersects the equatorial plane as the orbit crosses over from the southern to the Northern Hemisphere. Although Ω is very nearly constant in inertial space, it changes with respect to an Earth-fixed frame of reference because of the rotation of the Earth. A second parameter θ represents the location of the satellite as the angular phase in the circular orbit using the right ascension as the reference point. For the assumption of a circular orbit, θ changes at a constant rate, which gives rise to a period of approximately 43082 sec (slightly less than one half a day).

Table 4.1   Satellite Parameters

Satellite ID	Ω_0 (°)	θ_0 (°) ^a
1	326	68
2	26	340
3	146	198
4	86	271
5	206	90

^a θ_0 is the value of θ at the reference time t_0

Ω_0 is the value of Ω at the reference time t_0

Figure 4.3 Circular orbital trajectory ($\alpha = 55^\circ$) [16]

If the Earth- Centered Earth-fixed coordinates are given by Figure 4.3, the satellite position in the orbital model can be calculated by

$$x = R[\cos \theta \cos \Omega - \sin \theta \sin \Omega \cos 55^\circ] \quad (4.3)$$

$$y = R[\cos \theta \sin \Omega + \sin \theta \cos \Omega \cos 55^\circ] \quad (4.4)$$

$$z = R \sin \theta \sin 55^\circ \quad (4.5)$$

where

$$\theta = \theta_0 + (t - t_0) \cdot \frac{360}{43082} \text{ deg}$$

$$\Omega = \Omega_0 - (t - t_0) \cdot \frac{360}{86164} \text{ deg}$$

$$R = 26560000m$$

Figure.4.4 The WGS-84 coordinate reference frame (x, y, z) used by GPS and a locally level coordinate reference frame (x', y', z') [16]

One way to use the locally level coordinate frame and yet avoid the complications of the transformation is to adopt as the observer's local reference.

the "zero-zero" location where we simply have $x'=y$, $y'=z$, and $z'=x-r$. (r is the Earth radius -6380 km).

The satellite positions in the locally level coordinate frame chosen above, can be written as functions of time as follows:

$$x'(t) = R[\cos \theta(t) \sin \Omega(t) + \sin \theta(t) \cos \Omega(t) \cos 55^\circ] \quad (3.6)$$

$$y'(t) = R \sin \theta(t) \sin 55^\circ \quad (3.8)$$

$$z'(t) = R[\cos \theta(t) \cos \Omega(t) - \sin \theta(t) \sin \Omega(t) \cos 55^\circ] - r \quad (3.9)$$

4.1.4 Major Error Sources of GPS

The major sources of common errors are following:

1. Ionospheric Refraction. The ionosphere is a shell of electrically charged atoms and molecules that surrounds the Earth, stretching from a height of about 50 km to more than 1000 km. In the ionosphere, the signal interacts with free electrons and ions, causing an increase in its phase velocity with a corresponding decrease in its group velocity.
2. Tropospheric Refraction. The troposphere causes the propagation velocity of the signal to be slowed, compressing the signal wavelength.
3. Selective Availability. Selective availability (SA) is a deliberate error introduced by the US Department of Defense for security reasons to degrade the position solutions of non-military receivers.
4. Ephemeris errors. Normally, the difference between the actual satellite location and the location computed from the broadcast ephemeris is small, less

than 1 to 3 meters, but this error can be increased significantly with selective availability. Ephemeris error are very slowly varying biases but can spatially decorrelate over large distances.

5. Other Sources. Multipath and receiver noise at the reference station and the user.

Table 4.2 Correlated Errors Affecting Pseudorange: Approximate Statistical Parameters for a Stationary Observer [16]

Error Component	Standard Deviation ^a	Time Constant	Determining Factors
Satellite Position	3, 3 m	1/2 h	GPS Control Segment
Iono refraction with correction	1, 5 m	1/2 h	Solar flare activity: local time of day
Tropo refraction with correction	2, 2 m	1 h	Local atmospheric condition
Multipath	1, 5 m	10 min	Low elevations(<15°): Interfering signal paths
Selective availability	0, 30 m	3 min	GPS Control Segment

^a The two numbers given are for the PPS (authorized) and SPS (unauthorized) operational modes, respectively.

4.2 Inertial Navigation System

4.2.1 Introduction

Inertial navigation is a technique for determining a vehicle's position and velocity by measuring its acceleration and processing the acceleration information in a computer.

The techniques of inertial navigation evolved from fire-control technology, the marine gyrocompass, and conventional aircraft instrumentation. The earliest practical applications — and the heaviest expenditure of funds—were for ballistic-missile-guidance systems and for ship's inertial navigation systems. In the late 1950s, increased procurement of military aircraft led to the development of aircraft inertial navigators. Many of the disadvantages of inertial systems can be overcome through aiding with other sensors such as GPS, radar, or star-trackers.

In 1996, inertial navigation systems were widely used in military vehicles. Many ships, submarines, guided missiles, space vehicles, and virtually all-modern military aircraft are equipped with inertial navigation systems. Large commercial airliners routinely make use of inertial systems for navigation and steering.

4.2.2 The System Structure

Inertial navigation systems typically use an internal set of inertial sensors, consisting of accelerometers and gyroscopes. An accelerometer is a device that measures the force required accelerating a proof mass; thus, it measures the

acceleration of the vehicle containing the accelerometer. The purpose of the gyroscopes ("gyros") in an inertial navigation system is to space-stabilize the accelerometers. The combined accelerometer and gyro systems are either gimbaled or strapdown. With a gimbaled system, the sensor system is housed in a mechanical device, which is free to rotate in three dimensions. The accelerometer system can then be maintained in a stable reference frame by using the gyros to measure the angular rotation about the axes, this permit the accelerometer outputs to be integrated into velocity and position. With a strapdown system the inertial sensor assembly is fixed on the vehicle and cannot move. The gyroscopes track the rotation of the vehicle, and a stable reference frame is then determined computationally in software using the gyro measurements as input to the algorithm. In fact, the attitude integration algorithms permit the construction of an "analytic" platform.

In an inertial navigation systems, a platform (either gimbaled or analytic) measures acceleration in a coordinate frame that has a prescribed orientation relative to the Earth. Usually, the stabilized coordinate frame is locally leveled (two horizontal axes, one vertical). The computer, which may be the vehicle's central computer or a navigation computer, calculates the vehicle's position and velocity from the outputs of the two horizontal accelerometers. The computer also calculates gyroscope torque signals that maintain the platform in the desired orientation relative to the Earth.

4.3 Sonar-Ring Sensor System

There were many papers related to a mobile robot using sonar-ring to navigate and avoid obstacles [17][18]. Ando and Yuta [17] developed wall-following algorithms to make an autonomous robot be able to follow a wall in various sharps. Twelve directional sonar-rings were used. Ajejandro and Boumdine [18] used seven ultrasonic sensors to perceive the environment.

(a)

(b)

(c)

(d)

Figure. 4.5 Range error situation with an ultrasonic sensor

Only when the measurement direction is almost perpendicular to the target plane, the ultrasonic range sensor can get the range data between the sensor and the target plane [17](Figure 4.3 (a)). If the transducer is rotated so that the axis of the beam is at an angle to the wall (beam angle in Figure 4.3(b)), the measured

range will be the distance along the edge of the beam to the wall, not the distance along the axis of the beam. Thus, the apparent range is shorter than the axial range, resulting in range uncertainty, that is a function of transducer rotation and beam angle. If a small object is placed in front of the wall, the measured range is correct, but the lateral position of the object is only known to be within certain range (Figure.4.3 (c)). If the rotation of the transducer is greater than the beam angle, the beam will be reflected away from the transducer, the wall will be invisible (Figure.4.3 (d)).

To measure the distance between a vehicle and obstacles, a vehicle is equipped with sixteen directional sonar-rings that measure the range data of its direction. Fifteen sensors consist of three group's — front, right and left. Each group has five ultrasonic sensors, and spans an angle of about 75° . Another sensor senses the back of the vehicle (Figure 4.6). When the robot is moving, the measurements of each group of sensors are checked, and the smallest distance of each group is retained as the distance to the obstacle. Several sensors sense the obstacles in the same time. Each sensor has a certain angular range. The ranges of sensor overlap, but they could be combined to form a larger angular range, so it includes the whole obstacle. The angular range between the first and last sensor that detected obstacle determines the outer edges of the obstacle. If the distance to obstacles is smaller than the safety distance, the angle range is impassable. The sensor data are mapped into the fuzzy set, such as "large", "medium" and "small". A fuzzy controller can make the decision based on a set of fuzzy rules and membership functions. The use of a fuzzy set to determine nearness helps to

overcome uncertainty in sensor readings by avoiding the sharp cut-offs that are present in standard rule based systems [19].

Figure 4.6 Sonar-ring system

CHAPTER 5

FUZZY ADAPTIVE KALMAN FILTERING

5.1 Introduction

When navigating and guiding an autonomous vehicle, the position and velocity of the vehicle must be determined. The Global Positioning System (GPS) is a satellite-based navigation system that provides a user with the proper equipment access to useful and accurate positioning information anywhere on the globe [16]. However, several errors are associated with the GPS measurement. It has superior long-term error performance, but poor short-term accuracy. For many vehicle navigation systems, GPS is insufficient as a stand-alone position system. The integration of GPS and Inertial Navigation System (INS) is ideal for vehicle navigation systems. In general, the short-term accuracy of INS is good; the long-term accuracy is poor. The disadvantages of GPS/INS are ideally cancelled. If the signal of GPS is interrupted, the INS enables the navigation system to coast along until GPS signal is reestablished [16]. The requirements for accuracy, availability and robustness are therefore achieved.

Kalman filtering is a form of optimal estimation characterized by recursive evaluation, and an internal model of the dynamics of the system being estimated. The dynamic weighting of incoming evidence with ongoing expectation produces estimates of the state of the observed system [20]. An extended Kalman filter (EKF) can be used to fuse measurements from GPS and INS. In this EKF, the corrected INS data are used as a reference trajectory, and GPS data are applied to update and estimate the error states of this trajectory. The Kalman filter requires that all the plant dynamics and noise processes are exactly known and the noise processes are zero mean white noise. If the theoretical behavior of a filter and its actual behavior do not agree, divergence problems will occur. There are two kinds of divergence: Apparent divergence and True divergence [21][22]. In the apparent divergence, the actual estimate error covariance S_k remains bounded, but it approaches a larger bound than does predicted error covariance P_k . In true divergence, P_k remains bounded, but the actual estimation covariance S_k eventually becomes infinite.

Figure 5.1 True filter divergence

Figure 5.2 Apparent filter divergence

The divergence due to modeling errors is critical in Kalman filter application. If, the Kalman filter is fed information that the process behaved one way, whereas, in fact, it behaves another way, the filter will try to continually fit a wrong process. When the k increases, P_k and Kalman gain K_k go to zero. In this situation, measurement does not provide enough information to estimate all the state variables of the system. in other words, the computed estimation error matrix becomes unrealistically small, and the filter disregards the measurement. then the problem is particularly severe. Thus, in order to solve the divergence due to modeling errors, we can estimate unmodeled states, but it add complexity to the filter and one can never be sure that all of the suspected unstable states are indeed model states[21]. Other possibilities [24] are: (1) limiting of effective filter memory and overweighing most recent data, (2) finite memory filtering. (3) to add fictitious process noise in the model to compensate for the unmodeled nonlinear dynamics. It makes sure that the Kalman filter is driven by white noise, and prevents the filter from disregarding new measurement. In this paper, a fuzzy

logic adaptive controller (FLAC) is used to prevent the Kalman filter from divergence. The fuzzy logic adaptive controller (FLAC) will continually adjust the noise strengths in the filter's internal model, and tune the filter as well as possible. The FLAC performance is evaluated by the simulation of the fuzzy adaptive extended Kalman filtering scheme of Figure 5.3.

Figure 5.3 Fuzzy adaptive extended Kalman filter

5.2 Weighted EKF

Because the processes of both GPS and INS are nonlinear, a linearization is necessary. An extended Kalman filter is used to fuse the measurements from

the GPS and INS. In order to prevent divergence by keeping the filter from discounting measurements for large k , the exponential data weighting [22] is used.

The models and implementation equations for the weighted extended Kalman filter are as following:

Let us set the model covariance matrices equal to

$$\mathbf{R}_k = \mathbf{R}\alpha^{-2(k-1)} \quad (5.1)$$

$$\mathbf{Q}_k = \mathbf{Q}\alpha^{-2(k-1)} \quad (5.2)$$

where, $\alpha \geq 1$, and constant matrices $\mathbf{Q}$ and $\mathbf{R}$. For $\alpha > 1$, as time k increases, the $\mathbf{R}$ and $\mathbf{Q}$ decrease, so that we are giving more credibility to the recent data by decreasing the noise covariance exponentially with k . If $\alpha = 1$, it is a regular EKF.

Figure 5.4 Exponential data weighting

By defining the weighted covariance

$$\mathbf{P}_k^{\alpha^-} = \mathbf{P}_k^- \alpha^{2k} \quad (5.3)$$

The Kalman gain can be computed:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_k^- \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R} \alpha^{-2(k+1)})^{-1} \\ &= \mathbf{P}_k^{\alpha^-} \mathbf{H}_k^T \left(\mathbf{H}_k \mathbf{P}_k^{\alpha^-} \mathbf{H}_k^T + \frac{\mathbf{R}}{\alpha^2} \right)^{-1} \end{aligned} \quad (5.4)$$

The predicted state estimate is:

$$\hat{\mathbf{x}}_{k+1}^- = f(\hat{\mathbf{x}}_k^-, k) \quad (5.5)$$

The predicted measurement is:

$$\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_k^-, k) \quad (5.6)$$

The linear approximation equations can be presented in form:

$$\Phi_k \approx \left. \frac{\partial f(x, k)}{\partial x} \right|_{x=\hat{x}_k^-} \quad (5.7)$$

The predicted estimate on the measurement can be computed:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - \hat{\mathbf{z}}_k) \quad (5.8)$$

$$\mathbf{H}_k \approx \left. \frac{\partial h(x, k)}{\partial x} \right|_{x=\hat{x}_k^-} \quad (5.9)$$

Computing the *a priori* covariance matrix:

$$\mathbf{P}_{k+1}^- = \Phi_k \mathbf{P}_k \Phi_k^T + \mathbf{Q} \alpha^{-2(k+1)} \quad (5.10)$$

Re-writing (5.9) gives:

$$\mathbf{P}_{k+1}^{\alpha-} = \alpha^2 \Phi_k \mathbf{P}_k^{\alpha} \Phi_k^T + \mathbf{Q} \quad (5.11)$$

Computing the *a posteriori* covariance matrix gives:

$$\mathbf{P}_k^{\alpha} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_k^{\alpha-} \quad (5.12)$$

The initial condition is:

$$\mathbf{P}_{ij}^{\alpha-} = \mathbf{P}_{ij}$$

In equation(5.8). the term $\mathbf{z}_k - \hat{\mathbf{z}}_k$ is called residuals or innovations. It reflects the degree to which the model fits the data.

5.3 INS and GPS

The inertial navigation system (INS) consists of a sensor package, which includes accelerometers and gyros to measure accelerations and angular rates. By using these signals as input, the attitude angle and three-dimensional vectors of velocity and position are computed [23]. The errors in the measurements of force made by the accelerometers and the errors in the measurement of angular change in orientation with respect to inertial space made by gyroscopes are two fundamental error sources, which affect the error behavior of an inertial system.

The inertial system error response, related to position, velocity, and orientation is divergent with time due to noise input [11]. There are biases associated with the accelerometers and gyros. In order to correct the errors of INS, the GPS measurements are used to estimate the inertial system errors, subtract them from the INS outputs, and then obtain the corrected INS outputs. There is number of errors in GPS, such as ephemeris errors, propagation errors, selective availability, multi-path, and receiver noise, etc. By using differential GPS (DGPS), most of the errors can be corrected, but the multi-path and receiver noise cannot be eliminated.

5.4 Fuzzy Logic Adaptive System

It is assumed that both, the process noise $\mathbf{w}_k$ and the measurement noise $\mathbf{v}_k$ are zero-mean white sequences with known covariance $\mathbf{Q}$ and $\mathbf{R}$ in the Kalman filter. If the Kalman filter is based on a complete and perfectly tuned model, the residuals or innovations should be a zero-mean white noise process. If the residuals are not white noise, there is something wrong with the design and the filter is not performing optimally [22]. The Kalman filters will diverge or coverage to a large bound. In practice, it is difficult to know the exact values for $\mathbf{Q}$ and $\mathbf{R}$. In order to reduce computation, we have to ignore some errors, but sometimes those unmodeled errors will become significant. These are the instrument bias errors of INS, sometimes the $\mathbf{w}_k$ may be not zero mean. In those cases, the residuals can be used to adapt the filter. In fact, the residuals are the differences between actual measurements and best measurement predictions based

on the filter's internal model. A well-tuned filter is that where the 95% of the autocorrelation function of innovation series should fall within the $\pm 2\sigma$ boundary [25]. If the filter diverges, the residuals will not be zero mean and become larger.

There are very few papers on application of fuzzy logic to adapt the Kalman filter. In [26], fuzzy logic is used to the on-line detection and correction of divergence in a single state Kalman filter. There were three inputs and two outputs to fuzzy logic controller (FLC), and 24 rules were used. In our works, the adaptive fuzzy logic controller has been introduced and designed. The purpose of the fuzzy logic adaptive controller (FLAC) is to detect the bias of measurements and prevent divergence of the extended Kalman filter. It has been applied in three axes — East (x), North (y), and Altitude (z). The covariance of the residuals and the mean of residuals are used as the inputs to FLAC for all three fuzzy inference engines (x, y, and z). As an input to FLAC, the covariance of the residuals and mean values of residuals are used to decide the degree of divergence. The value of covariances relates to $\mathbf{R}$. If the residual has zero mean, the equation for covariance of the residual is:

$$\tilde{\mathbf{P}}_z = \mathbf{H}_k \mathbf{P}_k^- \mathbf{H}_k^T + \mathbf{R} \quad (5.13)$$

The fuzzy adaptive Kalman filtering has been used for guidance and navigation of mobile robots, especially for 3-D environment. The navigation of flying robots requires fast, and accurate on-line control algorithms. The “regular” Extended Kalman Filter requires high number of states for accurate navigation and positioning and is unable to monitor the parameters changing. The FLAC requires smaller number of states for the same accuracy and therefore it would

need less computational effort. Alternatively, the same number of states (as in “regular” filter) would allow for more accurate navigation.

5.4.1 Fuzzy Adaptive Kalman Filtering for Parameter Uncertainties

Sometimes, uncertain or time varying parameters are considered to exist in the **Q** and **R** matrices. The fuzzy adaptive Kalman filtering is used to detect and then adapt the filter on-line. There are two groups of fuzzy controllers. In those two fuzzy controllers, the covariance of the residuals and the mean of residuals are used as the inputs to both controllers for all three fuzzy inference engines. The exponential weighting α for first group controller and the scales for second group controller of three axes are the outputs.

The first group, which output is α is used to detect the filter divergence and adapt the EKF. Generally, when the covariance is becoming large, and mean value is moving away from zero, the Kalman filter is becoming unstable. In this case, a large α will be applied. A large α means that process noises are added. It can ensure that in the model all states are sufficiently excited by the process noise. When the covariance is extremely large, there are some problems with the GPS measurements, so the filter cannot depend on these measurements anymore, and a smaller α will be used. By selecting appropriate α , the fuzzy logic controller will adapt the Kalman filter optimally and try to keep the innovation sequence acting as zero-mean white noise.

Figure 5.5 Covariance membership functions for parameter uncertainties

Figure 5.6 Mean value membership functions for parameter uncertainties

Figure 5.7 α Membership functions for parameter uncertainties

The fuzzy logic controller uses 9 rules, such as:

*If the covariance of residuals is large **and** the mean value is zero **Then** α is zero.*

*If the covariance of residuals is zero **and** the mean value is large **Then** α is small.*

The second group, which output is scale is used to detect the change of measurement noise covariance $\mathbf{R}$. From equation (5.13), the $\mathbf{R}$ is related to the covariance of residual, the larger the covariance of residual, the more the measurement noise. When the fuzzy logic controller finds that the covariance of residual is larger than that expected, it applies a large scale to adjust the α .

$$\alpha_{\text{adjust}} = (\alpha - 1) \times \text{scale} + 1 \quad (5.14)$$

A sample rule is:

*If the covariance of residuals is small **and** the mean values is small **then** the scale is small.*

Table 5.1 Rule table of α for parameter uncertainties

α		Mean Value		
		Z	S	L
P	Z	Z	S	S
	S	S	L	S
	L	Z	NS	NS

S --- Small; L --- Large;

Z --- Zero; NS --- Negative Small

Table 5.2 Rule table of scale for parameter uncertainties

Scale		Mean Value		
		Z	S	L
P	Z	Z	Z	Z
	S	S	S	S
	L	L	S	Z

5.4.2 Fuzzy Adaptive Kalman Filtering for Non-white Process Noise

It is assumed that the process noise $\mathbf{w}_k$ is white noise for Kalman filtering. But sometime the process noise could be correlated with itself, non-white. In this case, we can add a shaping filtering that manufactures colored noise $\mathbf{w}_k$ with a given spectral density from white noise, but it will increase the state variables. In some real-time situation, the computing time have a restriction for increasing the state variables. We can use a fuzzy adaptive Kalman filtering to adaptive the process noise rather than add more state variables. There are 9 rules and therefore, little computational time is needed. The membership functions for this fuzzy control are showed in figure 5.8, 5.9, and 5.10.

The FLAC uses 9 rules, such as:

*If the covariance of residuals is large **and** the mean values are zero **Then** α is large.*

*If the covariance of residuals is zero **and** the mean values are large **Then** α is zero.*

Figure 5.8 Covariance membership functions for non-white process noise

Figure 5.9 Mean value membership functions for non-white process noise

Figure 5.10 α Membership functions for non-white process noise

Table. 5.3 Rule table for non-white process noise

α		Mean Value		
		Z	S	L
P_z	Z	S	Z	Z
	S	Z	L	M
	L	L	M	Z

S --- Small;   M --- Medium;   L --- Large;   Z --- Zero;

5.5 Extended Kalman Filter Model

The state variables used in simulation are:

$$\mathbf{x}_k = [x_k, \dot{x}_k, y_k, \dot{y}_k, z_k, \dot{z}_k, c\Delta t, c\dot{\Delta t}] \quad (5.15)$$

where, x points East, y points North, and z is the altitude. Other two states are GPS range bias state and range drift state.

5.5.1 Eight-state GPS/INS Linear Error Model

$$\mathbf{x}_{k+1} = \Phi \mathbf{x}_k + \mathbf{w}_k \quad (5.16)$$

The error models are represented in Figure 5.11, 5.12, and 5.13[16].

Figure 5.11 North channel error model

North channel:

$$\frac{d^2 \Delta y'}{dt^2} = a_y + g \phi_x$$

$$-\frac{d\phi_x}{dt} = \frac{1}{R} \frac{d\Delta y'}{dt} + \omega_y \phi_z + \varepsilon_x$$

Azimuth error :

$$\frac{d\phi_z}{dt} = \varepsilon_z$$

Figure 5.12 East channel error model

The driving noise $\mathbf{w}_k$, assumed time invariant is

$$\mathbf{w}_k = [0, \ t\sigma_x, \ 0, \ t\sigma_y, \ 0, \ t\sigma_z, \ cS_f t + \frac{cS_g t^2}{2}, \ cS_g t]^T \quad (5.18)$$

where, σ_y , σ_x , and σ_z are standard deviations associated with X, Y, and Z. t is the sample time.

$$S_f = 0.4(10^{-18})\text{sec.}$$

$$S_g = 1.58(10^{-18})\text{sec}^{-1}$$

S_f and S_g are standard deviations associated with clock offset and velocity respectively.

The process noise covariance matrix for a constant velocity model is

$$\mathbf{Q} = \begin{bmatrix} \frac{\sigma_x^2 t^3}{3} & \frac{\sigma_x^2 t^2}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{\sigma_x^2 t^2}{2} & \sigma_x^2 t & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\sigma_y^2 t^3}{3} & \frac{\sigma_y^2 t^2}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\sigma_y^2 t^2}{2} & \sigma_y^2 t & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\sigma_z^2 t^3}{3} & \frac{\sigma_z^2 t^2}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{\sigma_z^2 t^2}{2} & \sigma_z^2 t & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & (S_f t + \frac{S_g t^3}{3})c^2 & \frac{S_g t^2 c^2}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{S_g t^3 c^2}{2} & S_g t c^2 \end{bmatrix} \quad (5.19)$$

5.5.2 Measurement Model

Four pseudorange measurements (one per satellite) are sufficient for a standalone GPS solution. Each pseudorange measurement is of the general form

Total measured pseudorange = True range + Range bias

+ White measurement noise

To use pseudorange in the Kalman filter procedure, it must be linearized. The observation processed by the Kalman filter is the difference between the receiver-measured pseudorange and the estimated pseudorange based on the corrected INS computed position and a corrected GPS receiver clock time. Only the incremental perturbations in the INS and clock estimate errors then appear on the right side of the measurement equation along with a white measurement noise. There will be a linearized pseudorange measurement for each satellite, so the final form of the measurement model for the Kalman filter is

$\mathbf{z}$ = Actual receiver measurement

– Predicted measurement based on corrected INS position
and corrected receiver time

The resulting $\mathbf{H}$ matrix and observation $\mathbf{z}$ are

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \quad (5.20)$$

For the nominal point of linearization $\mathbf{x}_0 = [x_0, y_0, z_0]^T$

$$h_i^j = \left[\frac{\partial \psi_i}{\partial x} \right] = \frac{-(X_i - x_0)}{\sqrt{(X_i - x_0)^2 + (Y_i - y_0)^2 + (Z_i - z_0)^2}}$$

$$h_y^i = \left[\frac{\partial \psi_i}{\partial y} \right] = \frac{-(Y_i - y_0)}{\sqrt{(X_i - x_0)^2 + (Y_i - y_0)^2 + (Z_i - z_0)^2}}$$

$$h_z^i = \left[\frac{\partial \psi_i}{\partial z} \right] = \frac{-(Z_i - z_0)}{\sqrt{(X_i - x_0)^2 + (Y_i - y_0)^2 + (Z_i - z_0)^2}} \quad (5.21)$$

for $i=1, \dots, 4$.

$$H = \begin{bmatrix} h_x^{(1)} & 0 & h_y^{(1)} & 0 & h_z^{(1)} & 0 & 1 & 0 \\ h_x^{(2)} & 0 & h_y^{(2)} & 0 & h_z^{(2)} & 0 & 1 & 0 \\ h_x^{(3)} & 0 & h_y^{(3)} & 0 & h_z^{(3)} & 0 & 1 & 0 \\ h_x^{(4)} & 0 & h_y^{(4)} & 0 & h_z^{(4)} & 0 & 1 & 0 \end{bmatrix} \quad (5.22)$$

The **R** matrix reflects the additive white measurement noise components in which in this model would be a 4×4 matrix

$$\mathbf{R} = \mathbf{E}[\mathbf{v} \mathbf{v}^T]$$

$$\mathbf{R} = \begin{bmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & r & 0 \\ 0 & 0 & 0 & r \end{bmatrix} \quad (5.23)$$

The initial conditions of $\mathbf{x}_0$ and $\mathbf{P}_0$ theoretically will not affect the final Kalman filter estimation results. But they do effect how fast the Kalman filter will reach its stable value.

The initial values are

$$\hat{\mathbf{x}}_0^- = [0, 0, 0, 0, 0, 0, 0, 0]^T \quad (5.24)$$

$$\mathbf{P}_0^- \approx \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} \quad (5.25)$$

5.6 Simulation Results

MATLAB codes has been used to simulate and test the proposed method.

For all simulations, the sample period is 1 second.

5.6.1 Simulation Experiment No.1

The first part of simulation uses the fuzzy adaptive Kalman filtering for parameter uncertainties.

The designed standard deviation of GPS measurement $\mathbf{R}$ is 5 [m]. The designed standard deviations of $\mathbf{Q}$ for INS are 0.0012 meter, 0.0012 meter, and 0.0012 meter for the East (x), North (y), and Altitude (z) respectively.

The simulations (Table 5.4 and Figure 5.14, 5.15) show that after the filter is stable, the actual error covariance of fuzzy logic adaptive EKF almost agrees with the theory error covariance. In the Table 5.4, the designed parameters are $\mathbf{Q}$ and $\mathbf{R}$. The $5\mathbf{Q}$, $2\mathbf{R}$ etc. mean that the real time parameters are 5 and 2 time as large as the designed $\mathbf{Q}$ and $\mathbf{R}$. In figure 5.14, and 5.15, The dashed lines are the covariance of theory EKF, and the solid lines are the covariance of fuzzy adaptive EKF.

Table 5.4 Comparison of theoretical and actual error variance

$\mathbf{Q}$	$\mathbf{R}$	Theory			Actual		
		$\mathbf{P}_x$	$\mathbf{P}_y$	$\mathbf{P}_z$	$\mathbf{P}_x$	$\mathbf{P}_y$	$\mathbf{P}_z$
5Q	R	3.1711	2.5540	0.8344	3.3912	2.7227	0.8072
5Q	2R	5.3293	4.2877	1.3979	5.3121	4.1030	1.1796
3Q	2R	4.6896	3.7694	1.2268	4.8469	4.0864	1.2989
5Q	4R	8.9612	7.2002	2.3417	8.3122	7.7340	2.5005

5.6.2 Simulation Experiment No.2

In the second part of simulation, we simulate the inputs as non-white process noises. Table 5.5 shows the summary of simulation parameters.

Table 5.5 Summary of Simulations

Fig. No.	INS noise variances	Means of INS w	GPS noise variances
Figure 5.16	$\sigma_x=0.00006$ $\sigma_y=0.00006$ $\sigma_z=0.000006$	$x=0$ $y=0$ $z=0$	$r=(4m)^2$
Figure 5.17 Figure 5.18 Figure 5.19	$\sigma_x=0.00006$ $\sigma_y=0.00006$ $\sigma_z=0.000006$	$x=0.00012$ $y=0.00006$ $z=0.000012$	$r=(4m)^2$
Figure 5.20	$\sigma_x=0.00006$ $\sigma_y=0.00006$ $\sigma_z=0.000006$	$x=0.00036$ $y=0.00024$ $z=0.000048$	$r=(4m)^2$
Figure 5.21	$\sigma_x=0.00006$ $\sigma_y=0.00006$ $\sigma_z=0.000006$	$x=0.00012$ $y=0.00006$ $z=0.000012$	$r=(2m)^2$

In the Figure 5.16, there were not INS biases, so the mean for the East (x). North (y), and Altitude (z) were zero and inputs to EKF were zero-mean white noise. The EKF estimated the errors of INS and corrected the errors successfully. The mean of the innovation sequence of EKF was almost zero.

It was assumed that the measurements of INS have some biases for other simulations. The dashed lines were EKF, and the solid lines were fuzzy adaptive

EKF. Figure 5.17 showed the Kalman gains and position error for EKF and fuzzy adaptive EKF. The Kalman gain went to zero with k for EKF. The Kalman gain for fuzzy adaptive EKF were changing dynamically. Because of the almost zero Kalman gain of EKF, it was unable to correct the divergence by measurement (equation 5.8). By changing the α of Fuzzy adaptive EKF and then applying a large Kalman gain when fuzzy adaptive EKF detected the divergence of EKF, the measurements from GPS could correct estimated errors and prevent EKF from divergence .

From figure 5.18, the innovation sequences of EKF had a large drift, and was stable at a high mean value. The fuzzy adaptive EKF clearly improved the performance of EKF, and the mean value of residuals was much smaller than that of EKF. Since the innovation sequence or residuals are the differences between actual measurements and best measurement predictions based on the filter's internal model, consistent mismatch indicates erroneous model formulation.

Other figures present the corrected position (first column) and velocity (second column) errors. The corrected error is the current INS error minus estimated INS error. Although the INS measurements had some problem. (erroneous model, unmodeled states, or INS fail, etc.), the fuzzy adaptive EKF significantly reduced the corrected position and velocity errors. The simulations (Figure 5.19, Figure 5.20) also showed that the corrected errors of EKF were proportional to the mean values of INS measurements. In other word, the more errors are not modeled, the worse the EKF performs. The simulation in Figure 5.21 showed that good GPS measurements could improve the performance of

fuzzy adaptive EKF. It is because the larger Kalman gain is used, the more GPS measurements are counted. There was no improvement for EKF as a result of almost zero on Kalman gain.

Figure 5.14 Actual and theory covariance for 5Q and R

Figure 5.15 Actual and theory covariance for 5Q and 4R

Figure 5.16 The position error and innovation sequence of EKF with white noise inputs

Figure 5.17 Kalman gain and position error for EKF and FLAC-EKF

Figure 5.18 Innovation sequences of EKF and FLAC-EKF

Figure 5.19 Position and velocity errors (the smaller means) of EKF and FLAC-EKF

Figure 5.20 Position and velocity errors (the larger means) of EKF and FLAC-EKF

Figure 5.21 Position and velocity errors (the smaller r) of EKF and FLAC-EKF

CHAPTER 6

FUZZY CONTROL SYSTEM OF AUTONOMOUS VEHICLES

6.1 Introduction

When a vehicle move in a known environment, we can plan the path in advance, so that the vehicle can follow the planned path and does not collide with any of obstacles. In here, a vehicle can be a free-flying robot, spacecraft, aircraft, helicopter, mobile robot or any autonomous guided vehicle. In general, obstacles may not be fully known when we plan the path. In this case, the vehicle has to handle an event such as unexpectedly encountering some moving or static obstacles on its way and an original plan may no longer be executable in the new situation. The vehicle has to change its original path, and revise its motion to achieve the collision-free path [27]. In the navigation of vehicles, the environment uncertainty and complexity is a key issue. The capability of planning in dynamic domain is important for autonomous vehicles. The vehicle should perform assigned task by itself without the need to specify every action that is to be taken

by the vehicle. It is very difficult to build a precise and complete real-world mathematical model of the vehicle for preplanning a collision-free path.

For fuzzy logic control, instead of a complete model of the controlled process, the human knowledge -- how to control a vehicle can be used. For example, when a driver finds his car too close to an obstacle, in order to avoid collision with the obstacle, he will try to go away from the obstacle. If the obstacle is in the right of his car, he will turn left. In this situation, the linguistic If-Then rule, which he may use, is "If the distance to obstacle is small and the attitude is neglarge then change of the attitude is poslarge." Although this kind of linguistic information is not precise, it provides important information about how to control a vehicle.

6.2 Assumption of Fuzzy Control System

The fuzzy controller discussed is based on the following assumptions:

1. A vehicles uses GPS/INS and sensory systems, which measure the distance, position, velocity, and attitude, so that the vehicle can accurately detect its own position and can observe the obstacles by its sensory system.
2. The vehicle does not have any knowledge of obstacles in advance.
3. The start point and goal point for the vehicle is known, so one can make a rough path planning.

4. Obstacles are assumed to be described polygons. They are enlarged by constant distance R . The vehicles are then reduced to their reference points[28].

$$R_{\text{v}} = \sqrt{(W/2)^2 + (L/2)^2}$$

$$R_{\text{e}} = \sqrt{(L/2)^2 + (H/2)^2}$$

$$R_{\text{vz}} = \sqrt{(W/2)^2 + (H/2)^2} \quad (6.1)$$

where , W is the width of a vehicle, L is length of a vehicle, and H is height of a vehicle.

There are two geometric modes and three dynamic characteristic loops in the fuzzy control system.

Two geometric modes:

- Decision mode
- Trace mode

Three dynamic characteristic loops:

- Velocity fuzzy control loop
- Final goal approach control loop
- Attitude change loop

The block diagram of the fuzzy control system is shown as follow:

Figure 6.1 Block diagram of fuzzy control system

6.3 Path Planning

Path planners are generally divided into global and local ones, the former work is off-line, and the latter is on-line [29]. The off-line path planning generates a rough path by using known information. Motion planning problems in a static domains are usually solved in the following two steps. First, a graph is defined to represent a geometric structure of the environment. Next, graph search is performed to find a connected component between the node containing the start point and the node containing the destination point [27].

In this fuzzy control system, we first initial the elementary motion command, such as "move from A to B then C and final D", and then the on-line control, different fuzzy modes or loops will be applied. The on-line control is the

more important one. When the sensor system of a vehicle finds new information about the environment, the on-line control will make an appropriate decision. The vehicle needs to design an appropriate plan to execute a given task as well as handle unexpected events that arrive either from the outside world or from the vehicle itself.

6.4 Path Planning with Dynamic Obstacles

6.4.1 Meeting Point and Collision Point

We assume that the robot can move faster than any of the obstacles.

- Meeting point

If moving point V and R meet at a location X at the same time t , the location X is called a meeting point. The time t is called the meeting time.

- Collision fronts

A set of collision points around the start point, with respect to a certain speed, can be decomposed into subsets which we call collision fronts.

Figure 6.2 shows a movement, say (L, d_L, v_L) . Let v_a and v_b be the two endpoints of L , and let P_a and P_b be accessible points corresponding v_a and v_b , with respect to R 's initial location O , start time t_0 and speed v . The set of accessible points corresponding to all points in L forms a segment. This segment takes the form of either a straight-line segment or a conic section. We will call this segment a collision front of L (with respect to O , t_0 , and v). For an environment that

Figure 6.2 Collision front

contains more than one movement, there can be more than one collision front, each of which corresponds to same movement. In this case, however, it is possible that only a part of a collision front is accessible. Notice that if two moving obstacles do not collide, then the corresponding collision fronts do not intersect. The characterisation of the collision front is as follows. Suppose that R departs O at time t_0 and keep moving at a constant speed v along a ray, say l , emanating from O . The point robot R does not meet any of the obstacle in motion if and only if l does not intersect any of the collision front. The collision front of a stationary edge L is L itself or consists of subsets of L .

6.4.2 Planning and Motion

Figure 6.3 Plan and motion

In Figure 6.3, point S and G are the start and destination points, respectively. Triangle ABC is an obstacle moving towards the right, and rectangle $DEFH$ is an obstacle moving towards the left. Point Q , V , and T are the accessible points of C from S , of F from Q , and of H from V , respectively. The dashed objects show the locations of the corresponding obstacles at the indicated times. The sequence of line segments (SQ , QV , VT , TG) constitutes the final motion. The robot forecasts the movement of obstacles and uses the concept of visibility to plan the path.

6.4.3 Time Requirement

Figure 6.4 Obstacle movement

When we plan a path in dynamic environment, the time required to generate a collision-free motion is most important (Figure 6.4).

We consider an obstacle $ABCD$, BD lying parallel to the y -axis. Let (x_0, y_0) be the location of D at the start time, l be the length of BD , v be the speed of the obstacle, v_{max} be the speed of the robot, and θ be the angle formed by the direction of the movement and x -axis. Suppose that a point P at (x_0, Y) in BD is accessible from origin $O(x_R, y_R)$. We set the start time t_0 .

Coordinates x and y must satisfy

$$t(M) = \frac{\sqrt{(x - x_R)^2 + (y - y_R)^2}}{v_{max}} = \frac{\sqrt{(x - x_0)^2 + (y - Y)^2}}{v_L} \quad (6.2)$$

$$y = (x - x_0) \tan \theta + Y \quad (6.3)$$

$$y_0 \leq Y \leq y_0 + l \quad (6.4)$$

The points (x,y) that satisfy (6.2) and (6.3) form the collision front. These equations define a quadratic relationship, the collision front lies either on a parabola, hyperbola, or ellipse. The curve is a hyperbola when $v_{\max} > v_L \cos \theta$, parabola when $v_{\max} = v_L \cos \theta$, and ellipse when $v_{\max} < v_L \cos \theta$, respectively. Generally, the two endpoints of a collision front can be computed by setting Y to y_0 and $y_0 + l$ in equation (6.3) and (6.4).

6.5 Path Planning with Static Obstacles

The configuration space is a transformation from physical space in which the robot is of finite-size (e.g., a polygon) into another space in which the robot is treated as a point. Intuitively, the configuration space is obtained by shrinking the robot to a point, while growing the obstacles by the size of the robot.

Figure. 6.5 Physical space and its configuration space

Figure 6.5 illustrates the concept of configuration space. P is a fixed object in physical space, and R is the robot. The object delineated by thick line is $CO_R(P)$ with respect to the reference point on R . The robot will follow the edges of grown obstacle.

6.6 Decision Mode and Trace Mode

In those modes, only the geometric information will be considered. When a driver drives a car, he will check his distance to other vehicles, and depend on the distance and his car's attitude deciding how to avoid other vehicles. In our fuzzy control system, two inputs and one output to fuzzy controller are used. Two inputs are attitude of a vehicle and distance to planned path or to obstacles. The output is the change of attitude. When the attitude and the change of attitude steer a vehicle far away from the preplanned path or edges of obstacle, it is defined as positive change. Otherwise, it is negative change. When a vehicle starts its motion, the decision mode is applied. At every step, the fuzzy logic controller checks the data that come from the sonar-ring and GPS/INS. First, it makes sure where is the subgoal or final goal, which depend on the data from GPS/INS and preplanned paths. Then, the data that come from sonar-ring are checked. The data that come from front of sonar-ring are considered to have priority. If, there are no obstacles within the safety range in front of vehicles or obstacles do not influence the movement of a vehicle, the decision mode will be continued. It is important for a vehicle to follow the planned path in decision mode. Otherwise, the trace mode will be applied. The vehicles will try to avoid obstacles and follow the edges of

obstacles. In trace mode, it is important to act quickly so as to avoid imminent collision and keep its safety during its moving around the obstacles.

Figure 6.6 Vehicle's Position

The position for the vehicle is:

$$x(k+1) = \Delta S \cos(\theta + \beta - \gamma) + x(k) \quad (6.5)$$

$$y(k+1) = \Delta S \sin(\theta + \beta - \gamma) + y(k) \quad (6.6)$$

Δs is the displacement of the vehicle in unit time.

Some rules for those two modes are as follows:

*If the distance (D) to path (obstacles) is large **and** the attitude (θ) is poslarge **then** the change of attitude (β) is neglarge.*

*If the distance (D) to path (obstacles) is large **and** the attitude (θ) is possmall **then** the change of attitude (β) is neglarge.*

This rule shows the situation that when the distance between a vehicle and edges of obstacles or a path is large, and the vehicle is moving away the path or obstacles, the vehicle will turn a large steer angle to follow the path or obstacles.

*If the distance to path (obstacles) is large **and** the attitude is zero **then** the change of attitude is neglarge.*

This rule shows the situation that when the distance between a vehicle and obstacles or a path is large, and the vehicle is almost paralleling the path or edges of obstacles (a linguistic quantification of zero does not imply that the attitude = 0 exactly), the vehicle will turn a large steer angle to follow the path or obstacles.

*If the distance to path (obstacles) is large **and** the attitude is negsmall **then** the change of attitude is negsmall.*

*If the distance to path (obstacles) is large **and** the attitude is neglarge **then** the change of attitude is zero.*

*If the distance to path (obstacles) is small **and** the attitude is poslarge **then** the change of attitude is neglarge.*

*If the distance to path (obstacles) is small **and** the attitude is possmall **then** the change of attitude is neglarge.*

*If the distance to path (obstacles) is small **and** the attitude is zero **then** the change of attitude is negsmall.*

We use the "negsmall" as an abbreviation for "negative small in size" and so on for the other variable. Some membership functions are showed as figure 6.7, 6.8, 6.9, 6.10, and 6.11.

Figure 6.7 Attitude membership functions

Figure 6.8 Distance membership functions

Figure 6.9 Change of attitude membership functions

The rule-base array that we use for the decision and trace modes is shown in Table 6.1. The rule-base is a 3×5 array, as we have 3 fuzzy sets for distance and 5 fuzzy sets for attitude on the input universes of discourse.

Table 6.1 Rule table for decision and trace modes

Change of Attitude		Attitude				
		NL	NS	Z	PS	PL
Distance to path or obstacles	Z	PL	PS	Z	NS	NL
	S	PS	Z	NS	NL	NL
	L	Z	NS	NL	NL	NL

PL --- poslarge: PS --- possmall: NL --- neglarge: NS --- negsmall:

Z --- zero: S --- small: L --- large:

6.7 Velocity Fuzzy Control Loop

In this loop and following two other control loops, the vehicle's dynamic characteristics will be taken into account. In the decision mode, the shortest time is the most important, so that a vehicle will try to reach the subgoals or final goal as soon as possible. The vehicle will accelerate until it reaches its highest allowable speed and keeps moving at this speed. If the sensor system detects obstacles, the vehicle will make a decision whether to change its speed or not. If the obstacle do

not influence its movement, it will keep its high speed, and the decision mode will be continued. Otherwise, it will switch to the trace mode. In trace mode, both the safety and shortest time will be considered, but safety is the more important one. This is because when a vehicle moves with great speed in the trace mode, it may collide with obstacles. It will slow down and keep a slow speed around obstacles.

The rule table (Table 6.2) shows that when a vehicle is far away from obstacles, it will move at high speed, and when it is near to obstacles, the vehicle will reduce its speed and move slowly.

Figure 6.10 Change-in-velocity membership functions

Figure 6.11 Velocity membership functions

Table 6.2 Rule table for velocity control loop

Change-in-Velocity		Velocity		
		Z	S	L
Distance to Obstacles	Z	PS	Z	NS
	S	PS	Z	NS
	L	PL	PS	Z

6.8 Final Goal Approach Control Loop

When the vehicle senses the final goal, and the distance is small, the loop will slow down the speed of the vehicle, and then stop at final goal. Two inputs using in final goal approach control loop are distance to final goal and the velocity of the vehicle. The output is the change-in-velocity.

Table 6.3 Rule table for stop at final goal

Change-in-Velocity		Velocity		
		Z	S	L
Distance to Obstacles	Z	Z	NS	NL
	S	PS	Z	NL
	L	PS	Z	NS

6.9 Attitude Change Loop

When a vehicle is moving at high speed, only a small change of attitude will be allowed. It is dangerous to corner at speed. Otherwise, the change of attitude could be large. In this mode, the maximum change-in-attitude is $120^\circ/\text{s}$, the minimum change-in-attitude is $10^\circ/\text{s}$.

The rules for this loop are as follows:

*If the velocity is large **then** the change-in-attitude is small.*

*If the velocity is small **then** the change-in-attitude is medium.*

*If the velocity is zero **then** the change-in-attitude is large.*

Figure 6.12 Change-in-attitude membership functions

6.10 Fuzzy Supervisory Control System

In order to reduce the oscillating behaviour and avoid the collision with obstacles in trace mode (Figure 6.13), a fuzzy supervisory control system has been used. It expands or compresses the universes of discourse by simply changing the scaling gains. The input and output universes of discourse of the fuzzy controller are normalized on the range $[-1,1]$. The scaling gains are used to map the actual

inputs of the fuzzy control system to the normalized universe of discourse $[-1,1]$. When the universe of discourse is expanded, a "coarse control" is achieved, and when it is compressed, a "fine control" is achieved. When the distance is large, we use the "coarse control". The vehicle can make large change of attitude or acceleration, so that it can follow the planned path and accelerate quickly. If the distance is very small, the "fine control" is applied. In the "fine control", the scaling gains for change of attitude and acceleration are changed dynamically. The scaling gains proportion the distance to preplanned path or edges of obstacles. The more close a vehicle to preplanned paths or edges of obstacles, the smaller the scaling gains of attitude and acceleration.

The rules we can use are:

If the distance (D) to preplanned paths or edges of obstacles is larger than 1.5 m, then the scaling gains for change of attitude and acceleration are $g_{attitude\_nomal}$ and $g_{acceleration\_nomal}$.

If the distance (D) to preplanned paths or edges of obstacles is equal to or smaller than 1.5 m, then the scaling gains for change of attitude and acceleration are c_1D and c_2D .

where c_1 , and c_2 are the coefficient to adjust the total gains.

By this method, when the vehicle is close to the path or edges of obstacles, only a very small change of attitude and acceleration will be allowed. It will reduce the oscillating behaviour and avoid the collision with obstacles in trace mode.

6.11 Simulation Results

The proposed method has been simulated using the C codes. (Figure 6.14).

6.11.1 Tuning

If the performance of the fuzzy controller was not satisfactory, fine-tune or redesign the fuzzy controller by trial and error and repeat the procedure until the performance was satisfactory.

In our simulation, different numbers and types of membership functions were used in simulation. We found that using more membership functions (with the same gains), for example, 11 triangular membership functions had little influence on the performance of fuzzy control system. In general, if we use more membership functions, the real path of vehicle was more close to the planned path. However, there is a saturation point, beyond which the increased number of membership functions does not help. When 5 triangular membership functions (Figure 6.15) and 5 gaussian membership functions (Figure 6.16) were used, we got almost the same results as 7 triangular membership functions. The real path of 5-gaussian-membership function was the smoothest. Using less membership functions reduce the computational time. In the experiments, 5 Gaussian membership functions were used in simulation.

We must be cautioned that the changes of gains do not stretch the simulation model beyond its range of validity.

6.11.2 Simulation Experiments

As a first step, two dimensions and only the static obstacles were used to simulate the process (Figure 6.17, Figure 6.18). There were five static obstacles. This constitutes a densely populated area for the purpose of this experiment. The vehicle used distance to obstacles, and attitude as inputs and change of attitude as an output to avoid obstacles. The sensor range was 20m. Both geometric and dynamic features were used. At first, the decision mode was applied. The vehicle accelerated until it reached its highest allowable speed. When it sensed an obstacle, but the obstacle did not influence the movement of the vehicle, the decision mode still activated and it kept its high speed and followed the planned path. After it sensed obstacles and these obstacles were within the safety distance, it transferred to the trace mode, decelerated and moved around obstacles keeping an almost constant speed and distance away from obstacles. After it switched back to the decision mode, it speeded up again. Finally, the final goal was sensed, it slowed down and then stopped at final goal.

The second step was the simulation of avoiding both static and dynamic obstacles (Figure 6.19). We assumed the velocity of the moving obstacle was constant and was lower than the velocity of the vehicle. In order to avoid the obstacle successfully, the vehicle did not reduce its speed. When the vehicle encountered the dynamic obstacle, it forecasted the movement of the dynamic obstacle, planed the path and then transferred to the trace mode.

The third step was to develop 3-D trajectory planner (Figure 6.20, Figure 6.21). In 3-D trajectory planner, The XY, XZ and YZ planes were computed individually and then those three vectors were added in X, Y and Z-axis.

The equations for calculating X, Y, and Z are as follows:

$$x(k+1) = \frac{\Delta s(x_{vx} + x_{vz})}{\sqrt{(x_{vx} + x_{vz})^2 + (y_{vx} + y_{vz})^2 + (z_{vx} + z_{vz})^2}} + x(k) \quad (6.7)$$

$$y(k+1) = \frac{\Delta s(y_{vx} + y_{vz})}{\sqrt{(x_{vx} + x_{vz})^2 + (y_{vx} + y_{vz})^2 + (z_{vx} + z_{vz})^2}} + y(k) \quad (6.8)$$

$$z(k+1) = \frac{\Delta s(z_{vx} + z_{vz})}{\sqrt{(x_{vx} + x_{vz})^2 + (y_{vx} + y_{vz})^2 + (z_{vx} + z_{vz})^2}} + z(k) \quad (6.9)$$

where: Δs is the displacement of the vehicle in unit time.

In the 3-D simulation, 5 obstacles were used. The vehicle successfully avoided obstacles and reached the final goal.

The problem with the 3D-trajectory planner is its computational requirements. The parallel processing is seen as a means of easing this computational bottleneck. In this system, the three dimensions can be computed in parallel [30].

Figure 6.13 Navigation without applying fine tune

Figure 6.14 Code structure chart

Figure 6.15 Triangular membership functions

Figure 6.16 Gaussian membership functions

Figure 6.17 2-D Navigation

Figure 6.18 2-D Motion

Fig.6.19 Avoiding both static and dynamic obstacles

Figure 6.20 3-D Navigation

Figure 6.21 3-D Motion

CHAPTER 7

CONCLUSIONS

7.1 Conclusions

Fuzzy control provides a systematic procedure of transforming a knowledge base into a nonlinear mapping, where the knowledge base consists of a collection of fuzzy IF - THEN rules. In this thesis:

- 1) A fuzzy adaptive extended Kalman filter has been developed to detect and prevent the EKF from divergence. By monitoring the innovation sequences (residuals), the fuzzy logic adaptive controller can evaluate the performance of an EKF. If the filter does not perform well, which the residuals are not zero-mean white noises, it would apply an appropriate weighting factor α to ensure that in the model all states are sufficiently excited by the process noise or a heavier weighting of real-world measurements. It improves the accuracy of an EKF. The simulation results show that the fuzzy logic adaptive controller significantly reduces the corrected position and velocity errors when the EKF results diverge and can adjust Kalman filter properly during the parameter uncertainties. In the fuzzy logic adaptive controller, little computational time is needed.

It can be used to navigate and guide autonomous vehicles or robots and achieved a relatively accurate performance in the situation that sensors fail. Also, the fuzzy adaptive controller can use lower order state-model without compromising accuracy significantly. Another words, for any given accuracy, the fuzzy adaptive Kalman filter may be able to use a lower order state model. The fuzzy adaptive controller makes the necessary trade-off between accuracy and computational burden due to the increased dimension of the error state vector and associated matrices.

- 2) We use a fuzzy logic control system to guide and navigate autonomous vehicle. The fuzzy control system includes two geometric and three dynamic character loops. It can achieve a) Following the planned path or edges of obstacles: b). Accelerating and decelerating the velocity: c). Obstacle avoidance. The simulations for fuzzy logic control system of autonomous vehicles show that the fuzzy control is an efficient and stable method. This system can be used in 3D cases for free-flying robots as well as for 2D autonomous guided vehicle. In this thesis, both static and dynamic obstacles were used. The robot was able to navigate in densely populated areas avoiding “traps”.

7.2 Future research

In future research, we are going to develop an adaptive fuzzy control system to control and navigate the autonomous vehicles. especially in dynamic environment. The possibility to use the adaptive fuzzy logic to modify existing

fuzzy controller when none of fuzzy control systems completely fits the current situation will be considered. A fuzzy control system for avoiding unrestricted moving obstacles or controlling multi-vehicle movement would be developed. We can construct a more accurately fuzzy logic control system by combining numerical and linguistic information using adaptive fuzzy systems.

REFERENCES

- [1] Lee C.S. George. " Sensor - Based Robots: Algorithms and Architectures". Springer - Verlag. Berlin Heidelberg 1991.
- [2] Mckerrow, P. J.. "Introduction to Robotics". Addison-Wesley Publishing Company, Inc. 1991.
- [3] Oriolo G., and Ulivi G.. " Real-Time Map Building and Navigation for Autonomous Robots in Unknown Environments". IEEE Transactions on Systems, Man, and Cybernetics. 1998, pp. 316 - 331.
- [4] Hodge N. E. and Trabia M. B.. "Navigation Fuzzy Logic Control of an Autonomous Vehicle". the IASTED International Conference CONTROL'97, 1997, pp. 258 -262.
- [5] Gariner Ph. And Fraichard Th.. " A Fuzzy Motion Controller for a Car-like Vehicle", International Conference on Intelligent Robots and Systems, 1996, pp. 1171 -1178.

- [6] Li W. " ' Perception - Action' Behavior Control of a Mobil Robot in Uncertain Environments using Fuzzy Logic", International Conference on Intelligent Robots and Systems, 1994, pp. 439 -446.

- [7] Yen J. and . Pfluger N, " Path Planning and Execution Using Fuzzy Logic", AIAA Conference on Guidance, Navigation, and Control, 1991, pp1691 - 1698.

- [8] Saffiotti A.. "Fuzzy Logic in Autonomous Robotics: behavior coordination". IEEE International Conference on Fuzzy Systems, 1997, pp 573 - 578.

- [9] Goodridge, S. G., Luo, R. C., and Kay, M. G.," Multilayered fuzzy behavior fusion for real - time control of systems with many sensors". IEEE International Conference on Multisensor Fusion and Integration for Intelligent System (MFI), 1994, pp. 272 - 279.

- [10] Hartmut, S., Jorg, H., and Liliane, P.. " A Fuzzy System for Indoor Mobile Robot Navigation". IEEE International Conference on Fuzzy Systems, 1995, pp. 83 - 88.

- [11] Kayton M. and Fried W.R.. " Avionics Navigation Systems". John Wiley & Sons, Inc. 1997.

- [12] Passino, K. M., and Yurkovich, S., "Fuzzy control", Addison-Wesley Longman, 1998.

- [13] Babuska, R., and Verbruggen, H.B., " An overview of fuzzy modeling for control", Control Engineering Practice, A Journal of IFAC, Volume 4 Number 11, November 1996., pp. 1593-1606.

- [14] Wang L. X., " Adaptive Fuzzy Systems and Control". Prentice-Hall, Inc., 1994.

- [15] Grewal, M. S. and Andrews, A. P., " Kalman Filter, Theory and Practice", Prentice-Hall, Inc., 1993.

- [16] Brown, R. G. and Hwang, P.Y.C. " Introduction to Random Signals and Applied Kalman Filtering ". John Wiley & Sons, Inc., 1992

- [17] Ando, Y. and Yuta, S., "Following a Wall by Autonomous Mobile Robot with a Sonar-Ring", IEEE International Conference on Robotics and Automation, 1995, pp. 2599 -2606.

- [18] Alejandro, R. and Boumedine M., " Real-Time Navigation in Unknown Environments Using Fuzzy Logic and Ultrasonic Sensing", IEEE International Symposium on Intelligent Control, 1996, pp.26-29.

- [19] Yen, J., Pfluger, N., Lea, B., Murphy, M., and Jani, Y., " Employing Fuzzy Logic For Navigation and Control in an Autonomous Mobile System ", American Control Conference, 1993, pp. 1850 – 1854.

- [20] Abidi, Mongi A. and Gonzalez, Rafael C. " Data Fusion in Robotics and Machine Intelligence ", Academic Press, Inc., 1992.

- [21] Gelb, A., " Applied Optimal Estimation ", The MIT Press, 1974.

- [22] Lewis, F. L., " Optimal Estimation with an Introduction to Stochastic Control Theory ", John Wiley & Sons, Inc., 1986.

- [23] Jochen, P., Meyer-Hilberg, W., and Thomas, Jacob. "High Accuracy Navigation and Landing System using GPS/IMU System Integration". 1994 IEEE Position Location and Navigation Symposium. pp.298 – 305.

- [24] Maybeck, P. S., " Stochastic Models, Estimation, and Control ". Academic Press Inc., 1982.

- [25] Cooper, S. and Durrant-Whyte H., " A Kalman Filter Model for GPS Navigation of Land Vehicles ", 1994 IEEE Int. Conf. on Intelligent Robot and Systems, pp. 157 – 163.
- [26] Abdelnour, G., Chand, S., Chiu, S., and Kido T., " On-line Detection & Correction of Kalman Filter Divergence by Fuzzy Logic ", 1993 American Control Conf., pp 1835 – 1839.
- [27] Fujimura, K., "Motion planning in dynamic environments". Springer-Verlag, 1991.
- [28] Zhang , J., and Raczowsky, J., "Robust subgoal planning and motion execution for robots in fuzzy environments", 1996 IEEE Int. Conf. on Intelligent Robots and Systems. pp. 447-453.
- [29] Sasiadek, J., and Duleba I., " 3D local trajectory planner". 1997 AIAA Guidance, Navigation and Control Conf., pp.517-526.
- [30] Vaidehi, V., and Krishnan, C.N., "Mapping the Kalman tracking algorithm onto the transputer network", 1998 IEEE Transactions on Aerospace and Electronic System. pp. 681-686.

Appendix A: Simulation Code

The control and navigation system of autonomous vehicles, developed in Chapter 5, and 6 respectively, were tested by simulation. The code was written in Borland C and Student Matlab 5. Section A.1 is the simulation of fuzzy adaptive Kalman filtering for Chapter 5. Section A.2 is the simulation of fuzzy control system for Chapter 6.

A.1 Matlab code for fuzzy adaptive Kalman filtering

The first part of this simulation is initialization of Kalman filter and other parameters. The movements of satellites, and vehicles are simulated. We use actual trajectory and then corrupt it by white noise to simulate the measurements of GPS and INS. There are two parts of simulations, one is fuzzy adaptive Kalman filtering, and the other is the extended Kalman filtering. The final results are compared by graphics outputs.


```

%-----
%                               INITIALIZATION AND MODELING                               **
%                               **
%-----

% x(k+1)=A*x(k)+w(k);
% z(k)=H(k)*x(k)+v(k);
% prior estimate x_
% x1=East position error; x2=East velocity;
% x3=North position error; x4=North velocity;
% x5=altitude error; x6=altitude rate;
% x7=range bias state; x8=range drift state;
innvar1=0; innmp=[0;0;0;0]; Pz=0;
inn2=[0;0;0;0]; c1=0; c3=0; c2=0; cc=0;
x_=[0; 0; 0; 0; 0; 0; 0; 0];
xins=[0;0;0;0;0;0;0;0];
y=[0;0;0;0;0;0;0;0];

r=6380000;

% prior error covariance P_
P_=[100 0 0 0 0 0 0 0;
    0 10 0 0 0 0 0 0;
    0 0 100 0 0 0 0 0;
    0 0 0 10 0 0 0 0;
    0 0 0 0 100 0 0 0;
    0 0 0 0 0 10 0 0;
    0 0 0 0 0 0 100 0;
    0 0 0 0 0 0 0 10];

A=[1 1 0 0 0 0 0 0;
   0 1 0 0 0 0 0 0;
   0 0 1 1 0 0 0 0;
   0 0 0 1 0 0 0 0;
   0 0 0 0 1 1 0 0;
   0 0 0 0 -2*9.8/r 1 0 0;
   0 0 0 0 0 1 1 0;
   0 0 0 0 0 0 0 1];

% R=E[v(k)*inv(v(k))]
rr=25;
R=[rr 0 0 0;
   0 rr 0 0;
   0 0 rr 0;
   0 0 0 rr];

aa=.1;
ox=.0006*aa; oy=.0006*aa; oz=.00006*aa; of=0.4*10^(-18)*aa;
og=1.58^(-18)*aa; c=3*10^8;

a=1/3;
b=1/2;

Pz=0; PPz=0; innvar1=0; Pzdot=0; jj=0; ii=0; alpha=1; iij=0;

```

```

zgps=[0;0;0;0]; mean1=0; mean2=0; mean3=0;mean4=0; mean=[0;0;0;0];alpha11=1;alpha22=1;
alpha33=1;
meandot1=0; meandot2=0; meandot3=0; mean11=0; mean22=0; mean33=0;mea1=0;mea2=0;mea3=0;
innv1(1)=0; innv1(2)=0; innv1(3)=0; innv(1)=0; innv(2)=0; innv(3)=0;
% Q=E[w(k)*inv(w(k))]
Q=[a*ox*ox b*ox*ox 0 0 0 0 0;
    b*ox*ox ox*ox 0 0 0 0 0;
    0 0 a*oy*oy b*oy*oy 0 0 0;
    0 0 b*oy*oy oy*oy 0 0 0;
    0 0 0 0 a*oz*oz b*oz*oz 0 0;
    0 0 0 0 b*oz*oz oz*oz 0 0;
    0 0 0 0 0 of*of+a*og*og*c^2 b*og*og*c^2;
    0 0 0 0 0 b*og*og*c^2 og*og];

```

```

N=3000;t=[0;0;0;0;0;0;0;0];

```

```

y=[0.0,0.0,0.0,0.0,0];

```

```

% .....
%          PARAMETERS DECLARATION AND INITIAL RANDOM NOISE          **
%          **
% .....

```

```

omega01 = 326*pi/180;
omega02 = 26*pi/180;
omega03 = 146*pi/180;
omega04 = 86*pi/180;

```

```

theta01 = 68*pi/180;
theta02 = 340*pi/180;
theta03 = 198*pi/180;
theta04 = 271*pi/180 ;    %end of parameters declaration
c=1;
for k=1:N

```

```

    rnd1(k)=randn;    %initial random noise
    rnd2(k)=randn;
    rnd3(k)=randn;
    rnd4(k)=randn;
    rnd5(k)=randn;
    rnd6(k)=randn;
    rnd7(k)=randn;
    rnd8(k)=randn;
    rnd9(k)=rand;
    rnd10(k)=rand;
    rnd11(k)=rand;
    rnd12(k)=rand;

```

```

%input for w
if(k<3000)
    % w=[ox*(rnd1(k));ox*(rnd1(k));oy*(rnd2(k));oy*(rnd2(k));
    %oz*(rnd3(k));oz*(rnd3(k));(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];

    w=c*[ox*(rnd1(k)+2/c);ox*(rnd1(k)+2/c);oy*(rnd2(k)+0.5/c);oy*(rnd2(k)+.5/c);
        oz*(rnd3(k)+1/c);oz*(rnd3(k)+1/c);(of*c+og*c)*(rnd4(k)+.5/c);c*og*(rnd4(k)+.6/c)];
elseif k<11500
    %    w=[ox*(rnd1(k)+2/c);ox*(rnd1(k)+2/c);oy*(rnd2(k)-1/c);oy*(rnd2(k)-1/c);

```

```

elseif (k<20000)
%      oz*(rnd3(k)+1/c);oz*(rnd3(k)+1/c);(of*c+og*c)*(rnd4(k)-.5/c);c*og*(rnd4(k)-.6/c)];
w=c*[0*ox*(rnd1(k)+4/c);ox*(rnd1(k)+4/c);0*oy*(rnd2(k)+1/c);oy*(rnd2(k)+1/c);
      0*oz*(rnd3(k)+1/c);oz*(rnd3(k)+4/c);(of*c+og*c)*(rnd4(k)+.5/c);c*og*(rnd4(k)+.6/c)];
%w=[0;ox*(rnd1(k)+5000);0;oy*(rnd2(k)+5000);
    % 0;0;(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];

else
w=[ox*(rnd1(k));ox*(rnd1(k));oy*(rnd2(k));oy*(rnd2(k));
  oz*(rnd3(k));oz*(rnd3(k));(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];
end

if k<1000      %vehicle movement
    y=[10*k*.01*(k-1);10*k*.01;10*k*.01*(k-1);10*k*.01;5*(k-1);5;0;0];
else
    y=[50*(k-1);50;50*(k-1);10;50*(k-1);5;0;0];
end
xins=A*xins+w; %INS error
xins1=xins+x_; %corrected INS errors

xtrue1(k)=xins(1,1);
xtrue2(k)=xins(3,1);
xtrue3(k)=xins(5,1);
xpx(k)=xins1(1,1);
xpy(k)=xins1(3,1);
xpz(k)=xins1(5,1);
xv(k)=xins1(2,1);
yv(k)=xins1(4,1);
zv(k)=xins1(6,1);
if abs(xv(k))>0.5
    xv(k)=0.5;
end
if abs(yv(k))>0.5
    yv(k)=0.5;
end
if abs(zv(k))>0.5
    zv(k)=0.5;
end
x1(k)=x_(1,1); x2(k)=x_(3,1); x3(k)=x_(5,1);

%*****
%      CACULATE THE COORDINATES OF SATELLITES      **
%*****
%
OmegaRate = 2*pi/86164;
ThetaRate = 2*pi/43082; %rates equations
t=k-1;
omega1 = omega01 - OmegaRate*t;
omega2 = omega02 - OmegaRate*t;
omega3 = omega03 - OmegaRate*t;
omega4 = omega04 - OmegaRate*t;
theta1 = theta01 + ThetaRate*t;
theta2 = theta02 + ThetaRate*t;
theta3 = theta03 + ThetaRate*t;
theta4 = theta04 + ThetaRate*t; % computer omega and theta
Rs=26560000;

```

```

r=6380000;
X11=Rs*(cos(theta1)*cos(omega1)-sin(theta1)*sin(omega1)*cos(55*pi/180));
Y11=Rs*(cos(theta1)*sin(omega1)+sin(theta1)*cos(omega1)*cos(55*pi/180));
Z11=Rs*sin(theta1)*sin(55*pi/180);
X21=Rs*(cos(theta2)*cos(omega2)-sin(theta2)*sin(omega2)*cos(55*pi/180));
Y11=Rs*(cos(theta1)*sin(omega1)+sin(theta1)*cos(omega1)*cos(55*pi/180));
Z11=Rs*sin(theta1)*sin(55*pi/180);
X21=Rs*(cos(theta2)*cos(omega2)-sin(theta2)*sin(omega2)*cos(55*pi/180));
Y21=Rs*(cos(theta2)*sin(omega2)+sin(theta2)*cos(omega2)*cos(55*pi/180));
Z21=Rs*sin(theta2)*sin(55*pi/180);
X31=Rs*(cos(theta3)*cos(omega3)-sin(theta3)*sin(omega3)*cos(55*pi/180));
Y31=Rs*(cos(theta3)*sin(omega3)+sin(theta3)*cos(omega3)*cos(55*pi/180));
Z31=Rs*sin(theta3)*sin(55*pi/180);
X41=Rs*(cos(theta4)*cos(omega4)-sin(theta4)*sin(omega4)*cos(55*pi/180));
Y41=Rs*(cos(theta4)*sin(omega4)+sin(theta4)*cos(omega4)*cos(55*pi/180));
Z41=Rs*sin(theta4)*sin(55*pi/180);
Xv=xpx(k)+y(1,1)+3*rand1(k);
Yv=xpy(k)+y(3,1)+3*rand2(k);
Zv=xpz(k)-r+y(5,1)+3*rand3(k);

```

```

%-----
%          MEASUREMENT MATRIX H          **
%                                           **
%-----

```

```

PR1=sqrt((X11-Xv)^2+(Y11-Yv)^2+(Z11-Zv)^2);
PR2=sqrt((X21-Xv)^2+(Y21-Yv)^2+(Z21-Zv)^2);
PR3=sqrt((X31-Xv)^2+(Y31-Yv)^2+(Z31-Zv)^2);
PR4=sqrt((X41-Xv)^2+(Y41-Yv)^2+(Z41-Zv)^2);
pr1=sqrt((X11-Xv+xpx(k))^2+(Y11-Yv+xpy(k))^2+(Z11-Zv+xpz(k))^2);
pr2=sqrt((X21-Xv+xpx(k))^2+(Y21-Yv+xpy(k))^2+(Z21-Zv+xpz(k))^2);
pr3=sqrt((X31-Xv+xpx(k))^2+(Y31-Yv+xpy(k))^2+(Z31-Zv+xpz(k))^2);
pr4=sqrt((X41-Xv+xpx(k))^2+(Y41-Yv+xpy(k))^2+(Z41-Zv+xpz(k))^2);

```

```

h1x=-(X11-Xv)/PR1;
h1y=-(Y11-Yv)/PR1;
h1z=-(Z11-Zv)/PR1;
h2x=-(X21-Xv)/PR2;
h2y=-(Y21-Yv)/PR2;
h2z=-(Z21-Zv)/PR2;
h3x=-(X31-Xv)/PR3;
h3y=-(Y31-Yv)/PR3;
h3z=-(Z31-Zv)/PR3;
h4x=-(X41-Xv)/PR4;
h4y=-(Y41-Yv)/PR4;
h4z=-(Z41-Zv)/PR4;

```

```

zins1=PR1;
zins2=PR2;
zins3=PR3;
zins4=PR4;

```

```

Zins=[zins1;zins2;zins3;zins4];

```

```

H=[h1x 0 h1y 0 h1z 0 1 0;
   h2x 0 h2y 0 h2z 0 1 0;
   h3x 0 h3y 0 h3z 0 1 0;

```

```
h4x 0 h4y 0 h4z 0 1 0];
```

```
zgps1=pr1;
zgps2=pr2;
zgps3=pr3;
zgps4=pr4;
```

```
%-----
%                               **
%                               **
%-----
```

MEASUREMENTS OF GPS

```
if k<3000
    Zgps=[zgps1;zgps2;zgps3;zgps4]+4*[rnd5(k);rnd6(k);rnd7(k);rnd8(k)];
%elseif k<1200
    %Zgps=[zgps1;zgps2+5;zgps3;zgps4]+4*[rnd5(k)+.5;rnd6(k)/2+.4;rnd7(k)+.2;rnd8(k)];
elseif k<2000
    Zgps=[zgps1;zgps2;zgps3;zgps4]+10*[rnd5(k)+5;rnd6(k);rnd7(k);rnd8(k)];

    Zgps=[zgps1;zgps2;zgps3;zgps4]+12*[rnd9(k);rnd10(k);rnd11(k);rnd12(k)];
else
    Zgps=[zgps1;zgps2;zgps3;zgps4]+2*[rnd5(k);rnd6(k);rnd7(k);rnd8(k)];
end
```

```
%-----
%                               **
%                               **
%-----
```

INPUTS TO FUZZY CONTROLLER

```
inn=(Zgps-Zins);
in1(k)=inn(1,1);
in2(k)=inn(2,1);
in3(k)=inn(3,1);
innv4(k)=inn(4,1);
innvar1=(inn(1,1)^2+inn(2,1)^2+inn(3,1)^2+inn(4,1)^2)/4;
if k>20
    kk=k-20;
end
if k<12
    alpha1=1;alpha2=1;
    alphas1(k)=1;
    alphas2(k)=1;
    alphas3(k)=1;
else
    if ii~=2
        mea1=ii*mea1/(ii+1)+inn(1,1)/(ii+1); %caculate the average
        mea2=ii*mea2/(ii+1)+inn(2,1)/(ii+1);
        mea3=ii*mea3/(ii+1)+inn(3,1)/(ii+1);
        innv1(1)=ii*innv1(1)/(ii+1)+inn(1,1)^2/(ii+1);
        innv1(2)=ii*innv1(2)/(ii+1)+inn(2,1)^2/(ii+1);
        innv1(3)=ii*innv1(3)/(ii+1)+inn(3,1)^2/(ii+1);
        ii=ii+1;
    else
        mean1=mea1;
        mean2=mea2;
        mean3=mea3;
        innv(1)=innv1(1);
        innv(2)=innv1(2);
```

```

innv(3)=innv1(3);
mean=[mean1;mean2;mean3;0];
mea1=inn(1,1); %keep the old mean
mea2=inn(2,1);
mea3=inn(3,1);
ii=0;
end

for ji=1:3 %input for fuzzy controllers of X, Y, and Z
    if ji==1
        mean=abs(mean1);
    elseif ji==2
        mean=abs(mean2);
    else
        mean=abs(mean3);
    end
    Pz=innv(ji);
    g1=1/(rr*16); % gain for covariance
    g0=0.2; % gain for alpha
    if ji==1
        Pz1=mean1-mea1;
        meandot1=cc*meandot1/(cc+1)+Pz1/(cc+1);
    elseif ji==2
        Pz2=mean2-mea2;
        meandot2=cc*meandot2/(cc+1)+Pz2/(cc+1);
        g0=0.15;
    else
        Pz3=mean3-mea3;
        meandot3=cc*meandot3/(cc+1)+Pz3/(cc+1);
        g0=0.15;
    end
end

% .....
%                                     FUZZY CONTROL                                **
%                                     .....                                **
% .....

% Initialize parameters for the fuzzy controller
nume=3; % Number of input membership functions for the Pz universe of discourse
numie=3; % Number of input membership functions for mean universe of discourse
g2=0.05; %gain for mean

we=0.5*(1/g1);
% we is half the width of the triangular input membership function bases

wie=0.5*(1/g2);
% Similar to we but for the mean universe of discourse
base=.67*g0;
% Base width of output membership fuctions of the fuzzy controller

ce=[0 0.05 1]*(1/g1);
% Centers of input membership functions for the covariance
cie=[0 0.95 1]*(1/g2);
% Centers of input membership functions for the mean

% rule-base

```

```

rules=[0.1 0 1;
       0 1 0.5;
       0 0.5 0]*g0;

% Fuzzy controller calculations:
ie_count=0;e_count=0; % These are used to count the number of
                    % non-zero mf certainties of e and int e

if Pz<=ce(1) % Takes care of saturation of the left-most membership function
    mfe=[1 0 0]; % i.e., the only one on is the left-most one
    e_count=e_count+1;e_int=1; % One mf on, it is the left-most one.

    elseif Pz>=ce(numie) % Takes care of saturation of the right-most mf
    mfe=[0 0 1];
        e_count=e_count+1;e_int=numie; % One mf on, it is the right-most one
    else

    for i=1:numie
        if Pz<=ce(i)
            mfe(i)=max([0 1+(Pz-ce(i))/we]);

            % In this case the input is to the left of the center ce(i) and we compute
            % the value of the mf centered at ce(i) for this input e
            if mfe(i)~=0

                % If the certainty is not equal to zero then say that have one mf on by incrementing our count

                e_count=e_count+1;
                e_int=i; % This term holds the index last entry with a non-zero term
            end
        else

            mfe(i)=max([0,1+(ce(i)-Pz)/we]);

            if mfe(i)~=0

                e_count=e_count+1;
                e_int=i; % This term holds the index of the last entry with a non-zero term
            end
        end
    end
end
end
if mean<=cie(1) % Takes care of saturation of left-most mf
    mfi=[1 0 0];
    ie_count=ie_count+1;
    ie_int=1;
elseif mean>=cie(numie)
    % Takes care of saturation of the right-most mf
    mfi=[0 0 1];
    ie_count=ie_count+1;
    ie_int=numie;
else
    for i=1:numie
        if mean<=cie(i)
            mfi(i)=max([0,1+(mean-cie(i))/wie]);
            if mfi(i)~=0

```

```

        ie_count=ie_count+1;
        ie_int=i; % This term holds last entry with a non-zero term
    end
else
    mfe(i)=max([0,1+(cie(i)-mean)/wie]);
    if mfe(i)~=0
        ie_count=ie_count+1;
        ie_int=i; % This term holds last entry with a non-zero term
    end
end
end
end
end

% The next loops calculate the crisp
num=0;
den=0;
for r=(e_int-e_count+1):e_int % Scan over e indices of mfs that are on
    for l=(ie_int-ie_count+1):ie_int % Scan over int e indices of mfs that are on
        prem=min([mfe(r) mfe(l)]);
        % Value of premise membership function
        num=num+rules(r,l)*base*(prem-(prem)^2/2);
        den=den+base*(prem-(prem)^2/2);
    end
end
alpha(ji)=num/(den)+1; %get alpha
end

alpha11=alpha(1);
alpha22=alpha(2);
alpha33=alpha(3);
alphan1(k)=alpha(1);
alphan2(k)=alpha(2);
alphan3(k)=alpha(3);

end
alpha1=[alpha11 0 0 0;0 alpha22 0 0;0 alpha33 0;0 0 0 1];
alpha2=[alpha11 0 0 0 0 0 0;0 alpha11 0 0 0 0 0;
        0 0 alpha22 0 0 0 0;0 0 0 alpha22 0 0 0;
        0 0 0 0 alpha33 0 0;0 0 0 0 0 alpha33 0 0;
        0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 1];

% .....
%                                WEIGHTED KALMAN FILTERING                                **
%                                                                                          **
% .....

K=P_*H'*inv(H*P_*H'+R*inv(alpha1*alpha1'));
kkf(k)=K(1,1);
x=x_+K*inn;
I=eye(size(K*H));
P=(I-K*H)*P_;
x_=A*x;
P_=alpha2*alpha2'*(A*P*A'+Q);
p1(k)=P_(1,1);
T(k)=k-1;
end

```



```

%*****
%                               EXTENDED KALMAN FILTERING                               **
%                               **
%*****
%caculate alpha=1
P_=[100 0 0 0 0 0 0;
    0 10 0 0 0 0 0;
    0 0 100 0 0 0 0;
    0 0 0 10 0 0 0;
    0 0 0 0 100 0 0;
    0 0 0 0 0 10 0;
    0 0 0 0 0 0 10;
    0 0 0 0 0 0 0 10];

innvar1=0; innmp=[0;0;0;0]; Pz=0;
inn2=[0;0;0;0]; c1=0; c3=0; c2=0;
x_=[0; 0; 0; 0; 0; 0; 0; 0];
xins=[0;0;0;0;0;0;0;0];
y=[0;0;0;0;0;0;0;0];
Pz=0; PPz=0; innvar1=0; Pzdot=0; jj=0; ii=0; alpha=1;
zgps=[0;0;0;0]; mean1=0; mean2=0; mean3=0; mean4=0; mean=0; alpha11=1; alpha22=1; alpha33=1;
meandot1=0; meandot2=0; meandot3=0; mean11=0; mean22=0; mean33=0; mea1=0; mea2=0; mea3=0;
innv1(1)=0; innv1(2)=0; innv1(3)=0; innv(1)=0; innv(2)=0; innv(3)=0;
t=[0;0;0;0;0;0;0;0];

zgps=[0;0;0;0];
xt=[0;0;0;0;0;0;0;0];
y=[0;0;0;0;0;0;0;0];

omega01 = 326*pi/180;
omega02 = 26*pi/180;
omega03 = 146*pi/180;
omega04 = 86*pi/180;

theta01 = 68*pi/180;
theta02 = 340*pi/180;
theta03 = 198*pi/180;
theta04 = 271*pi/180;

%k=1;
for k=1:N

    if(k<10000)
        %w=[ox*(rnd1(k));ox*(rnd1(k));oy*(rnd2(k));oy*(rnd2(k));
        %oz*(rnd3(k));oz*(rnd3(k));(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];

        w=c*[ox*(rnd1(k)+2/c);ox*(rnd1(k)+2/c);oy*(rnd2(k)+0.5/c);oy*(rnd2(k)+0.5/c);
        oz*(rnd3(k)+1/c);oz*(rnd3(k)+1/c);(of*c+og*c)*(rnd4(k)+.5/c);c*og*(rnd4(k)+.6/c)];
        %w=[0;ox*(rnd1(k)+5000);0;oy*(rnd2(k)+5000);
        %    0;0;(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];
    elseif k<11500
        w=c*[0*ox*(rnd1(k)+4/c);ox*(rnd1(k)+4/c);0*oy*(rnd2(k)+1/c);oy*(rnd2(k)+1/c);
        0*oz*(rnd3(k)+1/c);oz*(rnd3(k)+4/c);(of*c+og*c)*(rnd4(k)+.5/c);c*og*(rnd4(k)+.6/c)];
        %w=c*[0;0;0;oy*(rnd2(k)+500/c);
        %    0;0;(of*c+og*c)*(rnd4(k)+.5/c);c*og*(rnd4(k)+.6/c)];
    end
end

```

```

else
w=[ox*(rnd1(k));ox*(rnd1(k));oy*(rnd2(k));oy*(rnd2(k));
oz*(rnd3(k));oz*(rnd3(k));(of*c+og*c)*(rnd4(k));c*og*(rnd4(k))];
end

if k<1000
y=[10*k*.01*(k-1);10*k*.01;10*k*.01*(k-1);10*k*.01;5*(k-1);5;0;0];
else
y=[5*(k-1);5;5*(k-1);1;5*(k-1);5;0;0];
end
xins=A*xins+w;
xins1=xins+x_;
xtrue11(k)=xins(1,1);
xtrue21(k)=xins(3,1);
xtrue31(k)=xins(5,1);
%xins=[0;0;0;0;0;0;0;0];
xpx1(k)=xins1(1,1);
xpy1(k)=xins1(3,1);
xpz1(k)=xins1(5,1);
xv1(k)=xins1(2,1);
yv1(k)=xins1(4,1);
zv1(k)=xins1(6,1);
if abs(xv1(k))>0.5
xv1(k)=0.5;
end
if abs(yv1(k))>0.5
yv1(k)=0.5;
end
if abs(zv1(k))>0.5
zv1(k)=0.5;
end

x11(k)=x_(1,1); x21(k)=x_(3,1); x31(k)=x_(5,1);
%caculate the coordiate of satellites
OmegaRate = 2*pi/86164;
ThetaRate = 2*pi/43082;
t=k-1;
omega1 = omega01 - OmegaRate*t;
omega2 = omega02 - OmegaRate*t;
omega3 = omega03 - OmegaRate*t;
omega4 = omega04 - OmegaRate*t;
theta1 = theta01 + ThetaRate*t;
theta2 = theta02 + ThetaRate*t;
theta3 = theta03 + ThetaRate*t;
theta4 = theta04 + ThetaRate*t ;
Rs=26560000;
r=6380000;
%caculate the coordinate of satilletes
X11=Rs*(cos(theta1)*cos(omega1)-sin(theta1)*sin(omega1)*cos(55*pi/180));
Y11=Rs*(cos(theta1)*sin(omega1)+sin(theta1)*cos(omega1)*cos(55*pi/180));
Z11=Rs*sin(theta1)*sin(55*pi/180);
X21=Rs*(cos(theta2)*cos(omega2)-sin(theta2)*sin(omega2)*cos(55*pi/180));
Y11=Rs*(cos(theta1)*sin(omega1)+sin(theta1)*cos(omega1)*cos(55*pi/180));
Z11=Rs*sin(theta1)*sin(55*pi/180);
X21=Rs*(cos(theta2)*cos(omega2)-sin(theta2)*sin(omega2)*cos(55*pi/180));
Y21=Rs*(cos(theta2)*sin(omega2)+sin(theta2)*cos(omega2)*cos(55*pi/180));

```

```

Z21=Rs*sin(theta2)*sin(55*pi/180);
X31=Rs*(cos(theta3)*cos(omega3)-sin(theta3)*sin(omega3)*cos(55*pi/180));
Y31=Rs*(cos(theta3)*sin(omega3)+sin(theta3)*cos(omega3)*cos(55*pi/180));
Z31=Rs*sin(theta3)*sin(55*pi/180);
X41=Rs*(cos(theta4)*cos(omega4)-sin(theta4)*sin(omega4)*cos(55*pi/180));
Y41=Rs*(cos(theta4)*sin(omega4)+sin(theta4)*cos(omega4)*cos(55*pi/180));
Z41=Rs*sin(theta4)*sin(55*pi/180);
Xv=xpx(k)+y(1,1)+3*rnd1(k);
Yv=xpy(k)+y(3,1)+3*rnd2(k);
Zv=xpz(k)+y(5,1)+3*rnd3(k);
PR1=sqrt((X11-Xv)^2+(Y11-Yv)^2+(Z11-Zv)^2);
PR2=sqrt((X21-Xv)^2+(Y21-Yv)^2+(Z21-Zv)^2);
PR3=sqrt((X31-Xv)^2+(Y31-Yv)^2+(Z31-Zv)^2);
PR4=sqrt((X41-Xv)^2+(Y41-Yv)^2+(Z41-Zv)^2);
pr1=sqrt((X11-Xv+xpx1(k))^2+(Y11-Yv+xpy1(k))^2+(Z11-Zv+xpz1(k))^2);
pr2=sqrt((X21-Xv+xpx1(k))^2+(Y21-Yv+xpy1(k))^2+(Z21-Zv+xpz1(k))^2);
pr3=sqrt((X31-Xv+xpx1(k))^2+(Y31-Yv+xpy1(k))^2+(Z31-Zv+xpz1(k))^2);
pr4=sqrt((X41-Xv+xpx1(k))^2+(Y41-Yv+xpy1(k))^2+(Z41-Zv+xpz1(k))^2);

```

```

h1x=-(X11-Xv)/PR1;
h1y=-(Y11-Yv)/PR1;
h1z=-(Z11-Zv)/PR1;
h2x=-(X21-Xv)/PR2;
h2y=-(Y21-Yv)/PR2;
h2z=-(Z21-Zv)/PR2;
h3x=-(X31-Xv)/PR3;
h3y=-(Y31-Yv)/PR3;
h3z=-(Z31-Zv)/PR3;
h4x=-(X41-Xv)/PR4;
h4y=-(Y41-Yv)/PR4;
h4z=-(Z41-Zv)/PR4;

```

```

zins1=PR1;
zins2=PR2;
zins3=PR3;
zins4=PR4;

```

```
Zins=[zins1;zins2;zins3;zins4];
```

```

H=[h1x 0 h1y 0 h1z 0 1 0;
   h2x 0 h2y 0 h2z 0 1 0;
   h3x 0 h3y 0 h3z 0 1 0;
   h4x 0 h4y 0 h4z 0 1 0];

```

```

zgps1=pr1;
zgps2=pr2;
zgps3=pr3;
zgps4=pr4;

```

```
if k<3000
```

```
    Zgps=[zgps1;zgps2;zgps3;zgps4]+4*[rnd5(k);rnd6(k);rnd7(k);rnd8(k)];
```

```
% elseif k<1400
```

```
    % Zgps=[zgps1;zgps2+5;zgps3;zgps4]+4*[rnd5(k)-.5;rnd6(k)/2-.4;rnd7(k)-.2;rnd8(k)];
```

```
elseif k<2000
```

```
    Zgps=[zgps1;zgps2;zgps3;zgps4]+10*[rnd5(k)+5;rnd6(k);rnd7(k);rnd8(k)];
```

```

    %Zgps=[zgps1;zgps2;zgps3;zgps4]+12*[rnd9(k);rnd10(k);rnd11(k);rnd12(k)];
else
    Zgps=[zgps1;zgps2;zgps3;zgps4]+2*[rnd5(k);rnd6(k);rnd7(k);rnd8(k)];
end
inn=(Zgps-Zins);
in11(k)=inn(1,1);
in21(k)=inn(2,1);
in31(k)=inn(3,1);
in41(k)=inn(4,1);
innvar1=(inn(1,1)^2+inn(2,1)^2+inn(3,1)^2+inn(4,1)^2)/4;

alpha=1;
if k>1549
    P11(k)=P(1,1);
    P22(k)=P_(3,3);
    P33(k)=P_(5,5);
end
%R=alpha*R;
%kalman filter
K=P_*H'*inv(H*P_*H'+R/(alpha^2));

kkn(k)=K(1,1);

x=x_+K*inn;
I=eye(size(K*H));
P=(I-K*H)*P_;
x_=A*x;
P_=alpha^2*(A*P*A'+Q);

p11(k)=P_(1,1);
T(k)=k-1;
end

%*****
%                               GRAPHIC OUTPUTS                               **
%*****
K
subplot(4,2,1)
plot(T,kkf,T,kkn,'--');
xlabel(' Time(s)')
ylabel('Kalman gain K')
title('Kalman gain K')
grid on

subplot(4,2,2)
plot(T,in11)
xlabel(' Time(s)')
ylabel('Innovation(m)')

subplot(4,2,3)
plot(T,xpx,T,xpx1,'--');
xlabel('Time(s)')
ylabel('X Position Errors(m)')

```

```

%axis([0 N -5 5])
grid on
subplot(4,2,4)
plot(T,xv,T,xv1,'--')
xlabel('Time(s)')
ylabel('X Velocity Errors(m/s)')
%axis([0 N -0.2 0.2])
grid on
subplot(4,2,5)
plot(T,xpy,T,xpy1,'--');
xlabel('Time(s)')
ylabel('Y Position Errors(m)')
%axis([0 N -5 5])
grid on
subplot(4,2,6)
plot(T,yv,T,yv1,'--');
xlabel('Time(s)')
ylabel('Y Velocity Errors(m/s)')
%axis([0 N -0.2 0.2])
grid on

subplot(4,2,7)
plot(T,xpz,T,xpz1,'--');
xlabel('Time(s)')
ylabel('Z Position Errors(m)')
%axis([0 N -5 5])
grid on
subplot(4,2,8)
plot(T,zv,T,zv1,'--');
xlabel('Time(s)')
ylabel('Z Velocity Errors(m/s)')
%axis([0 N -0.2 0.2])

grid on

clear

```

A.2 C code for fuzzy control system of autonomous vehicles

At every step, the distances to each edge of obstacles and planed paths were checked, and we applied the fuzzy controls to the edge, which has minimum distance in trace mode or the planed path in decision mode. In different situations, different fuzzy controls would be on. The results were written to a file, and then Matlab was used to draw the figures.

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define pi 3.14159

float member_functionG(int,int,float,float,float);
float member_function(int,int,float,float,float);
float fuzzy_control(float,float,int,int);
float ucrisp(int,float[],float[],float,float,int,int,int);
float ucrisp1(int,float[],float,float,int);
float angle(float,float,float,float,float,float,float,float,float,float);
float cacul_bata(float,float,float,float,float,float,float,float,float,float,float);
float follow_edge(float,float,float,float,float[],float[],float,float,float[],
float[],float,float,int,int);
int transfer(float,float,float,float,float,float,float[],float[],float,int);
int transfer1(float,float,float,float,float,float,float,float);
int decide_intersection_point(float[],float[],float[],float[],int,int);
float coord(float bata1,float vo,float v,float xoo,float yoo,
float xm,float ym);

/*-----*/
/*                                MAIN                                */
/*-----*/
void main(void)
{
float M,D,xs,xg,xm,x1,xa,ys,yg,ym,y1,ya,g0,g1,g2,K,t,w,l,dist_obstacle,alpha1,
steer_angle,alpha,bata,delta_bata,saft_distance,alpha_o,xs1,ys1,xg1,yg1,xa1,ya1,
x11,x12,x13,x14,y11,y12,y13,y14,d1,d2,xang,yang,ta,v,delta_v,gb,gb1,gbv,bata1,
g_dv,g_v,gd,v0,step,delta_bata,alpha_m,alpha_m1,mx1,mx2,my1,my2,Dang1,xoo,yoo,vo,xang1,yang1,
Dang[30],xdo[30],ydo[30],x[30],y[30],xd[30],yd[30],k[30],disto[30],xo[30],yo[30];
int nr,ce,j,i,kt,ka,c,number_obstacle,min_index,min_indexa,ia,de,dt,tr.move_obstacle,trc;

FILE *fp;
if((fp=fopen("fuzzy","wb"))==NULL)
{
printf("cannot open directory file\n");
exit (1); }
/*initialization*/
```

```

M=0; xm=0; ym=0; x11=0; y11=0; x12=470; y12=900;
x13=901; y13=601; x14=900; y14=350; v=0; step=0.1; delta_v=0; delta_bata=0;
t=0; w=40; l=50; mx2=1000; my2=1000;
number_obstacle=5;
saft_distance=sqrt(pow(0.5*w,2)+pow(0.5*l,2));
xo[0]=100.001; yo[0]=500; xo[1]=100; yo[1]=450; xo[2]=425; yo[2]=450.001;
xo[3]=425.001; yo[3]=500.001;
xo[4]=350; yo[4]=600.001; xo[5]=350.001; yo[5]=350; xo[6]=400; yo[6]=350.001;
xo[7]=400.001; yo[7]=600;
xo[8]=210.1; yo[8]=278.1; xo[9]=210; yo[9]=230.8; xo[10]=550; yo[10]=230;
xo[11]=550.1; yo[11]=278;
xo[12]=100.1; yo[12]=500.1; xo[13]=100; yo[13]=300; xo[14]=100.2; yo[14]=300.1;
xo[15]=200.1; yo[15]=500.2;
xo[16]=500.1; xo[17]=500; xo[18]=550; xo[19]=550.1;
yo[16]=630; yo[17]=278; yo[18]=278.1; yo[19]=630.1;

```

```
/*enlarge the obstacles*/
```

```
for(i=1;i<=number_obstacle;i++)
```

```

{
    c=i*4-1; /*obstacles are quadrilateral*/
    for(j=c-3;j<=c;j++)
    {
        if(j==(c-3))
        {
            x[j]=xo[j]-saft_distance;
            y[j]=yo[j]+saft_distance;
        }
        else if(j==(c-2))
        {
            x[j]=xo[j]-saft_distance;
            y[j]=yo[j]-saft_distance;
        }
        else if(j==(c-1))
        {
            x[j]=xo[j]+saft_distance;
            y[j]=yo[j]-saft_distance;
        }
        else
        {
            x[j]=xo[j]+saft_distance;
            y[j]=yo[j]+saft_distance;
        }
    }
}
trc=0; bata=0; bata1=-3; vo=0.1;

```

```
/*output to a file*/
```

```

fprintf(fp,"Xo1=[%f %f %f %f %f %f];\nYo1=[%f %f %f %f %f %f];\n",
xo[0],xo[1],xo[2],xo[3],xo[0],yo[0],yo[1],yo[2],yo[3],yo[0]);
fprintf(fp,"Xo2=[%f %f %f %f %f %f];\nYo2=[%f %f %f %f %f %f];\n",
xo[4],xo[5],xo[6],xo[7],xo[4],yo[4],yo[5],yo[6],yo[7],yo[4]);
fprintf(fp,"Xo3=[%f %f %f %f %f %f];\nYo3=[%f %f %f %f %f %f];\n",
xo[8],xo[9],xo[10],xo[11],xo[8],yo[8],yo[9],yo[10],yo[11],yo[8]);
fprintf(fp,"Xo4=[%f %f %f %f %f %f];\nYo4=[%f %f %f %f %f %f];\n",
xo[12],xo[13],xo[14],xo[15],xo[12],yo[12],yo[13],yo[14],yo[15],yo[12]);
fprintf(fp,"Xo5=[%f %f %f %f %f %f];\nYo5=[%f %f %f %f %f %f];\n"

```

```

xo[16],xo[17],xo[18],xo[19],xo[16],yo[16],yo[17],yo[18],yo[19],yo[16]);
fprintf(fp,"x1=[%f %f %f];\ny1=[%f %f %f];\n",x11,x12,x13,y11,y12,y13);
fprintf(fp,"L=[%f %f %f %f];\n",xm,ym,v,t);
//fprintf(fp,"L=[%f %f %f %f];\n",bata,delta_v,t,delta_bata);
//preplanned path
xs1=x11; ys1=y11;   xg1=x12;   yg1=y12;
while(58<=sqrt(pow(fabs(xm-x13),2)+pow(fabs(ym-y13),2)))
{
    v0=v;
    d1=sqrt(pow(fabs(xm-x12),2)+pow(fabs(ym-y12),2));
    //d2=sqrt(pow(fabs(xm-x13),2)+pow(fabs(ym-y13),2));
    if(d1<100)
    {
        xs1=x12; ys1=y12;   xg1=x13;   yg1=y13;
    }
    xa=xg1-xs1;   ya=yg1-ys1;
    alpha1=atan2(ya,xa); /*angle of physical goal line*/
    K=ya/xa;   /*slope of physical goal line*/
    x1=((ym-ys1)*K+pow(K,2)*xs1+xm)/(pow(K,2)+1); /*intersection point to physical goal line*/
    y1=((xm-xs1)*K+pow(K,2)*ym+ys1)/(pow(K,2)+1);
    D=sqrt(pow(fabs(xm-x1),2)+pow(fabs(ym-y1),2)); /*distance to physical goal line*/
/* x[17]=x[17]+0.35*vo*step*cos(bata1);
y[17]=y[17]+0.35*vo*step*sin(bata1);
x[20]=x[20]+0.35*vo*step*cos(bata1);
y[20]=y[20]+0.35*vo*step*sin(bata1);
x[18]=x[18]+0.35*vo*step*cos(bata1);
y[18]=y[18]+0.35*vo*step*sin(bata1);
x[19]=x[19]+0.35*vo*step*cos(bata1);
y[19]=y[19]+0.35*vo*step*sin(bata1); */
    if(ym<=920)
    {
        min_index=0;   min_indexa=0;
        for(i=1;i<=number_obstacle,i++)
        {
            c=i*4-1; /*obstacles are quadrilateral*/
            /*check all obstacles and find the nearest edge of obstacle*/
            for(j=c-3;j<=c;j++)
            {
                if(j==c)
                    k[j]=(y[j]-y[j-3])/(x[j]-x[j-3]);/*get the slope of edges of
                                                            obstacles*/
                else
                    k[j]=(y[j+1]-y[j])/(x[j+1]-x[j]);
                xd[j]=((ym-y[j])*k[j]+pow(k[j],2)*x[j]+xm)/(pow(k[j],2)+1);
                yd[j]=((xm-x[j])*k[j]+pow(k[j],2)*ym+y[j])/(pow(k[j],2)+1);
                xdo[j]=((ym-yo[j])*k[j]+pow(k[j],2)*xo[j]+xm)/(pow(k[j],2)+1);
                ydo[j]=((xm-xo[j])*k[j]+pow(k[j],2)*ym+yo[j])/(pow(k[j],2)+1);
                disto[j]=sqrt(pow(fabs(xm-xd[j]),2)+pow(fabs(ym-yd[j]),2));/*distance
                                                            to obstacles*/
                de=decide_intersection_point(xo,yo,xdo,ydo,j,c);
                if(de==1)/*the line of distance to obstacle intersect with real
                    obstacle*/
                    disto[j]=disto[j];
                else
                    disto[j]=100;
            }
            /*the nearest edge of obstacle */

```



```

        if(disto[j]<disto[min_index])
            min_index=j;
        else
            min_index=min_index;
    }
    for(ia=c-3;ia<=c;ia++)
    {
        Dang[ia]=sqrt(pow(fabs(xm-xo[ia]),2)+pow(fabs(ym-yo[ia]),2));
        if(Dang[ia]<Dang[min_indexa])
            min_indexa=ia;
        else
            min_indexa=min_indexa;
    }
    Dang1=Dang[min_indexa]-saft_distance;
    ka=min_indexa;
    dist_obstacle=disto[min_index];
    kt=min_index; /*get the edge of obstacles which is acting*/
    if(kt==3||kt==7||kt==11||kt==15)
    {
        ya1=y[kt-3]-y[kt];
        xa1=x[kt-3]-x[kt]; /*right turn*/
    }
    else
    {
        ya1=y[kt+1]-y[kt];
        xa1=x[kt+1]-x[kt];
    }
    alpha_o=atan2(ya1,xa1); /*angle of obstacle edge*/
    dt=transfer(ys1,yg1,xs1,xg1,ym,xm,xd,yd,alpha_o,kt);/*decide whether transfer to
                                                                    decision mode*/
    if(dt==0&&Dang1>6)
    {
        dist_obstacle=100;
    }
    if(dist_obstacle>=saft_distance&&Dang1>6) /*decision mode*/
    {
        alpha=alpha1; /*angle of physical goal line*/
        x1=x1; y1=y1; yg=yg1; ys=ys1; /*follow phasical goal*/
        xg=xg1; xs=xs1; D=D; ta=0;
        dist_obstacle=100;
    }
    else
    {
        if(Dang1>dist_obstacle)
        {
            alpha=alpha_o; /*follow the edges of obstacles*/
            if(kt==11||kt==3||kt==15||kt==7)
            {
                yg=y[kt-3]; xg=x[kt-3]; xs=x[kt]; ys=y[kt];
            }
            else
            {
                xg=x[kt+1]; yg=y[kt+1]; xs=x[kt]; ys=y[kt];
            }
            x1=xd[kt] y1=yd[kt];
        }
    }

```

```

        D=dist_obstacle; ta=0; dist_obstacle=disto[min_index];
    }
else
{
    D=Dang1;
    xang=xm-xo[ka];
    yang=ym-yo[ka];
    ta=1;
    dist_obstacle=Dang1;
}
}
else
{
    if(trc!=1)
    {
        xoo=x[17]; yoo=y[17];
        k[16]=(y[17]-y[16])/(x[17]-x[16]);
        xd[16]=((ym-y[16])*k[16]+pow(k[16],2)*x[16]+xm)/(pow(k[16],2)+1);
        yd[16]=((xm-x[16])*k[16]+pow(k[16],2)*ym+y[16])/(pow(k[16],2)+1);
        dist_obstacle=sqrt(pow(fabs(xm-xd[16]),2)+
        pow(fabs(ym-yd[16]),2));/*distance to obstacles*/

        de=decide_intersection_point(x,y,xd,yd,16,16);
        if(de==1)/*the line of distance to obstacle intersect with real
        obstacle*/
            dist_obstacle=dist_obstacle;
        else
            dist_obstacle=100;

        mx1=coord(bata1,vo,v,xoo,yoo,xm,ym); /* printf("mx1=%f",mx1); */
        my1=(mx1-xoo)*tan(bata1)+yoo;
        xang1=mx1-xm;
        yang1=my1-ym; /* printf("my1=%f",my1); */

        alpha_m=atan2(yang1,xang1); /* printf("d=%f",dist_obstacle); */
        xoo=x[18]+sqrt((x[17]-mx1)*(x[17]-mx1)+
        (y[17]-my1)*(y[17]-my1))*cos(bata1);
        yoo=y[18]+sqrt((x[17]-mx1)*(x[17]-mx1)+
        (y[17]-my1)*(y[17]-my1))*sin(bata1);
        mx2=coord(bata1,vo,v,xoo,yoo,mx1,my1);
        my2=(mx2-xoo)*tan(bata1)+yoo;
        xang1=mx2-mx1;
        yang1=my2-my1;
        alpha_m1=atan2(yang1,xang1);
        tr=transfer1(ys1,yg1,xs1,xg1,ym,xm,mx2,my2);
    }

    else
    {
        tr=transfer1(ys1,yg1,xs1,xg1,ym,xm,mx2,my2); }
        if(dist_obstacle>=99||tr==0||xm>=mx2)//decision mode
        {
            alpha=alpha1; /*angle of physical goal line*/
            x1=x1; y1=y1; yg=yg1; ys=ys1; /*follow phasical goal*/
            xg=xg1; xs=xs1; D=D; ta=0;

```

```

        dist_obstacle=100;    trc=0;
        move_obstacle=0;
    }

    else //trace mode
    {
        trc=1;    dist_obstacle=40;
        move_obstacle=2;
    }
}
if(move_obstacle!=1&&move_obstacle!=2)
{
    if(D>1.5)
    {
        g0=0.01; g1=0.8;
        g2=1;
    }
    else
    {
        g0=.01; g1=.8; g2=1.2*D;
    }
    /*input to fuzzy control:D and steer_angle*/
    D=g0*D; /* printf("D=----%f",D/g0); */
    if(ta==0)
        steer_angle=g1*angle(yg.ys.xs.x1.xm.xg.ym.y1.alpha.bata);
    else
        steer_angle=atan2(yang.xang)+pi/2;
    ce=0; g_v=.02; nr=2;
    v=g_v*v0;
    if(dist_obstacle>=3.5)
        g_dv=10;
    else
        g_dv=.001*D;
    gd=.01;

    delta_v=g_dv*fuzzy_control(gd*dist_obstacle.v.nr.ce); /*caculate
    accelerate*/
    v=v0+delta_v*step;
    /*update velocity*/
    ce=3; gb=.1; gbv=.05; gb1=.01;
    delta_bata=gb*fuzzy_control(gbv*delta_v.gb1*v.nr.ce); /*relation between
    angular and velocity*/
    ce=1;
    delta_bata=g2*fuzzy_control(D.steer_angle.nr.ce); /*output
    change_in_attitude*/
    if(delta_bata>delta_bata)
        delta_bata=delta_bata;
    bata=cacul_bata(yg.ys.xs.xg.x1.xm.y1.ym.bata,delta_bata);/* caculata
    attitude */
}
else if(move_obstacle==2)
{
    if(xm<mx1)
        bata=alpha_m;
    else
        bata=alpha_m1;
}

```

```

    }

    M=v0*step+0.5*delta_v*step*step;      /*caculate displacement*/
    t=t+step;
    xm=0.35*M*cos(bata)+xm; /*robot center position*/
    ym=0.35*M*sin(bata)+ym;

    fprintf(fp,"%f %f %f %f;\n",xm,ym,v,t);
    // fprintf(fp,"%f %f %f %f;\n",bata,delta_v,t,delta_bata);
}
/*distance to final goal*/
while(5<=sqrt(pow(fabs(xm-x13),2)+pow(fabs(ym-y13),2)))
{
    v0=v;
    D=sqrt(pow(fabs(xm-x13),2)+pow(fabs(ym-y13),2));
    gd=.01;
    D=gd*D;
    v=g_v*v0;
    ce=2;
    delta_v=g_dv*fuzzy_control(D,v,nr,ce); /*caculate accelerate*/
    v=v0+delta_v*step; /*update velocity*/
    M=v0*step+0.5*delta_v*step*step;      /*caculate displacement*/
    t=t+step;
    x1=((ym-ys1)*K+pow(K,2)*xs1+xm)/(pow(K,2)+1); /*intersection point to
    physical goal line*/
    y1=((xm-xs1)*K+pow(K,2)*ym+ys1)/(pow(K,2)+1);
    D=sqrt(pow(fabs(xm-x1),2)+pow(fabs(ym-y1),2));
    g0=0.01;
    D=g0*D;
    steer_angle=g1*angle(yg,ys,xs,x1,xm,xg,ym,y1,alpha,bata);
    delta_bata=g2*fuzzy_control(D,steer_angle,nr,ce); /*output
    change_in_attitude*/
    bata=cacul_bata(yg,ys,xs,xg,x1,xm,y1,ym,bata,delta_bata);/* caculata
    attitude */
    xm=0.3*M*cos(bata)+xm; /*robot center position*/
    ym=0.3*M*sin(bata)+ym;
    t=t+step;
    printf("dv=%f",delta_v);
    fprintf(fp,"%f %f %f %f;\n",xm,ym,v,t);
    //fprintf(fp,"%f %f %f %f;\n",bata,delta_v,t,delta_bata);
}
xm=xg1;
ym=yg1;
v=0; delta_v=0;
xa=x12-x11; ya=y12-y11; bata=atan2(ya,xa);
fprintf(fp,"%f %f %f %f;",xm,ym,v,t);
//fprintf(fp,"%f %f %f %f;",bata,delta_v,t,delta_bata);
fclose(fp);      scanf("i=%d",&j);
}

```

```

/*****
/*          MAIN FUNCTION OF FUZZY CONTROLLER          */
/*          */
*****/

float fuzzy_control(float D,float steer_angle,int nr,int ce)
{
float xd,mf1[11],mf2[11],w,cl,i0,u;
int i,N;
if(ce==0||ce==2)
{
N=4; i0=2;
for(i=i0;i<=N;i++)
{
xd=steer_angle; w=.5; cl=w*N/2; /*xd--v and D,w--the base-width of membership*/
mf1[i]=member_functionG(N,i,xd,w,cl); /*functions,cl--saturation point,N--number*/
xd=D; w=.5; cl=w*N/2; /*of linguistic values*/
if(i<nr)
mf2[i]=0;
else
mf2[i]=member_functionG(N,i,xd,w,cl);
}
}
else if(ce==1)
{
N=4; i0=0;

for(i=i0;i<=N;i++)
{
xd=steer_angle; w=.5; cl=w*N/2; /*xd--e and de/dt,w--the base-width of membership*/
mf1[i]=member_functionG(N,i,xd,w,cl); /*functions,cl--saturation point,N--number*/
xd=D; w=.5; cl=w*N/2; /*of linguistic values*/
if(i<nr)
mf2[i]=0;
else
mf2[i]=member_functionG(N,i,xd,w,cl);
}
}
else
{
N=4; i0=0;

for(i=nr;i<=N;i++)
{
xd=steer_angle; w=.5; cl=w*N/2; /*xd--velocity ,w--the base-width of membership*/
mf1[i]=member_functionG(N,i,xd,w,cl); /*functions,cl--saturation point,N--number*/
u=ucrisp1(N,mf1,w,cl,nr);
}
}
w=.5; cl=w*N/2;
if(ce==3)
return u;
else
return ucrisp(N,mf1,mf2,w,cl,nr,ce,i0);
}

```

```

/*****
/*                                MEMBERSHIP FUNCTIONS, TRIANGULAR                                */
/*                                */
*****/

```

```
float member_function(int N,int i,float xd,float w,float cl)
```

```

{
float mf[22];
mf[i]=0;
if(i<N/2)
{
if(xd<0)
{
if(xd<=-cl)
mf[0]=1;
else
{
if(((i-N/2-1)*w<xd)&&(xd<(i-N/2)*w))
mf[i]=1-((i-N/2)*w+xd)/w;
else if(((i-N/2)*w<xd)&&(xd<(i-N/2+1)*w))
mf[i]=1+((i-N/2)*w-xd)/w;
else
mf[i]=0;
}
}
else
mf[i]=0;
}
else if(i==N/2)
{
if((-w<xd)&&(xd<w))
{
if(xd<=0)
mf[N/2]=1+xd/w;
else
mf[N/2]=1-xd/w;
}
else
mf[N/2]=0;
}
else
{
if(xd>0)
{
if(xd>=cl)
mf[N]=1;
else
{
if(((i-N/2+1)*w>xd)&&(xd>(i-N/2)*w))
mf[i]=1-(xd-(i-N/2)*w)/w;
else if(((i-N/2)*w>xd)&&(xd>(i-N/2-1)*w))
mf[i]=1+(xd-(i-N/2)*w)/w;
else
mf[i]=0;
}
}
}
}

```

```

    else
        mf[i]=0;
    }
    return mf[i];
}

```

```

/*****
/*          GAUSSIAN MEMBERSHIP FUNCTION          */
/*          */
*****/
float member_functionG(int N,int i,float xd,float w,float cl)
{
    float mf[22],o;
    o=.5*w;
    if(fabs(xd-w*(i-N/2))>=w)
        mf[i]=0;
    else if (fabs(xd)>=cl)
        mf[i]=1;
    else
        mf[i]=exp(-0.5*pow(((xd-w*(i-N/2))/o),2));
    return mf[i];
}

```

```

/*****
/*          FUZZIFICATION AND THE CACULATION OF OUTPUT*/
/*          */
*****/
float ucrisp(int N,float mf1[],float mf2[],float w,float cl,int nr,int ce,int i0)
{
    float prem[11][11],rule[11][11],ucrisp.num,den;
    int i,j;
    for(i=i0;i<=N;i++) /*column*/
    {
        for(j=nr;j<=N;j++) /*row*/
        {
            if(mf1[i]>mf2[j]) /*using the minimum of the two membership values*/
                prem[i][j]=mf2[j];
            else
                prem[i][j]=mf1[i];
            /* prem[i][j]=mf1[i]*mf2[j]; */ /*using the product*/
        }
    }
    /*initialization of the rule-bases*/
    if(ce==0) /*for accelerate*/
    {
        if(i==2&&j==4)
            rule[i][j]=4;
        else if(i==2)
            rule[i][j]=3;
        else if(i==3&&j==4)
            rule[i][j]=3;
        else if(i==3)
            rule[i][j]=2;
        else if(i==4&&j==4)

```

```

    rule[i][j]=2;
    else
    rule[i][j]=1;
    }
    else if(ce==1)    /*for change_in_attitude*/
    {
    if(i+j<=2)
    rule[i][j]=4;
    else if(i+j==3)
    rule[i][j]=3;
    else if(i+j==4)
    rule[i][j]=2;
    else if(i+j==5)
    rule[i][j]=1;
    else
    rule[i][j]=0;
    }
    else if(ce==2)    /*go to final goal*/
    {
    if(i==4&&j==4)
    rule[i][j]=1;
    else if(i==4)
    rule[i][j]=0;
    else if(j==4&&i==3)
    rule[i][j]=2;
    else if(j==3&&i==3)
    rule[i][j]=1;
    else if(j==2&&i==3)
    rule[i][j]=0;
    else if(j==2&&i==2)
    rule[i][j]=2;
    else
    rule[i][j]=3;
    }
    }
}
/*computing crisp(COG)*/
num=0; den=0;
for(i=0;i<=N;i++)
{
    for(j=nr;j<=N;j++)
    {
        /*num=num+w*2*(prem[i][j]-pow(prem[i][j],2)/2)*w*(rule[i][j]-N/2);
        den=den+w*2*(prem[i][j]-pow(prem[i][j],2)/2); */
        num=num+prem[i][j]*w*(rule[i][j]-N/2);
        den=den+prem[i][j];
        if(den==0)
        den=0.0000001;
    }
}

ucrisp=num/den;
if(ucrisp>=cl)
ucrisp=cl;
else if(ucrisp<=-cl)
ucrisp=-cl;

```



```

else
ucrisp=ucrisp;
return ucrisp;
}

```

```

/*****
/*                      FUZZIFICATION AND THE CACULATION OF OUTPUT                      */
/*                      */
*****/

```

```

float ucrisp1(int N,float mf1[],float w,float cl,int nr)
{
float prem[11],rule[11],ucrisp.num,den;
int j;
    num=0; den=0;
    for(j=nr;j<=N;j++)
    {
        prem[j]=mf1[j];
        /*initialization of the rule-bases*/
        rule[4]=2;
        rule[3]=3;
        rule[2]=4;
        /*computing crisp(COG)*/
        /* num=num+w*2*(prem[i][j]-pow(prem[i][j],2)/2)*w*(rule[i][j]-N/2);
        den=den+w*2*(prem[i][j]-pow(prem[i][j],2)/2); */
        num=num+prem[j]*w*(rule[j]-N/2)+.05;
        den=den+prem[j];
    }

ucrisp=num/den;
if(ucrisp>=cl)
ucrisp=cl;
else if(ucrisp<=-cl)
ucrisp=-cl;
else
ucrisp=ucrisp;
return ucrisp;
}

```

```

/*****
/*                      CACULATE THE STEERING ANGLE                      */
/*                      */
*****/

```

```

float angle(float yg,float ys,float xs,float x1,float xm,float xg,float ym,float y1,
float alpha,float bata)
{
float angle;
if(fabs(yg-ys)>fabs(xs-xg)) /* go Y direction*/
{
    if(yg>ys)
    {

```

```

if(x1>xm)
angle=bata-alpha;
else
angle=alpha-bata;
}
else
{
if(x1>xm)
angle=alpha-bata;
else
angle=bata-alpha;
}
}
else /*X direction*/
{
if(xg>xs)
{
if(ym>y1)
angle=bata-alpha;
else
angle=alpha-bata;
}
else
{
if(yg<ys)
{
if(ym>y1)
{
if(bata<0)
angle=alpha-bata;
else
angle=pi*2-bata+alpha;
}
else
{
if(bata<0)
angle=bata-alpha;
else
angle=-(pi*2-bata+alpha);
}
}
}
else
{
if(ym>y1)
{
if(bata<0)
angle=-(2*pi+bata-alpha);
else
angle=alpha-bata;
}
else
{
if(bata>0)
angle=bata-alpha;
else
angle=2*pi+bata-alpha;
}
}
}
}

```

```

    }
  }
}
return angle;
}

```

```

/*****
                                RANGE OF ANGLE
*****/

```

```

float cacul_bata(float yg, float ys, float xs, float xg, float x1, float xm, float y1, float ym, float bata, float dalta_bata)
{
  if(fabs(yg-ys)>fabs(xs-xg))
  {
    if(yg>ys)
    {
      if(x1>xm)
        bata=bata+dalta_bata;
      else
        bata=bata-dalta_bata;
      if((bata>pi-0.4)||bata<-pi/2)
        bata=pi-0.4;
      else if(bata<=0.4)
        bata=0.4;
      else
        bata=bata;
    }
    else
    {
      if(x1>xm)
        bata=bata-dalta_bata;
      else
        bata=bata+dalta_bata;
      if(bata<-pi+0.4||bata>pi/2)
        bata=-pi+0.4;
      else if(bata>=0.4)
        bata=-0.4;
      else
        bata=bata;
    }
  }
  else
  {
    if(xg>xs)
    {
      if(ym>y1)
        bata=bata+dalta_bata;
      else
        bata=bata-dalta_bata;
      if(bata>=pi/2-0.4)
        bata=pi/2-0.4;
      else if(bata<=-pi/2+0.4)
        bata=-pi/2+0.4;
    }
  }
}

```

```

    else
        bata=bata;
}
else
{
    if(ym>y1)
        bata=bata-dalta_bata;
    else
        bata=bata+dalta_bata;
    if(bata<=-pi)
        bata=2*pi+bata;
        else if(bata>pi)
            bata=bata-2*pi;

    else
        bata=bata;
    if(bata<0&&bata>=-pi/2-0.4)
        bata=-pi/2-0.4;
    else if(bata>0&&bata<=pi/2-0.4)
        bata=pi/2-0.4;
    else
        bata=bata;
}
}
return bata;
}

```

```

/*.....*/
/*                                DECIDE THE REAL OBSTACLE                                */
/*.....*/
int decide_intersection_point(float xo[],float yo[],float xdo[],float ydo[],int j,int c)
{
    float A[30],B[30];

    if(j==c)
    {
        A[j]=sqrt(pow(fabs(xo[j]-xo[j-3]),2)+pow(fabs(yo[j]-yo[j-3]),2));
        B[j]=sqrt(pow(fabs(xo[j]-xdo[j]),2)+pow(fabs(yo[j]-ydo[j]),2))
            +sqrt(pow(fabs(xo[j-3]-xdo[j]),2)+pow(fabs(yo[j-3]-ydo[j]),2));
    }
    else
    {
        A[j]=sqrt(pow(fabs(xo[j]-xo[j+1]),2)+pow(fabs(yo[j]-yo[j+1]),2));
        B[j]=sqrt(pow(fabs(xo[j]-xdo[j]),2)+pow(fabs(yo[j]-ydo[j]),2))
            +sqrt(pow(fabs(xo[j+1]-xdo[j]),2)+pow(fabs(yo[j+1]-ydo[j]),2));
    }
    if(A[j]==B[j]) /*the point of intersection inside the obstacle*/
        return 1;
    else
        return 0;
}

```

```

/*****
/*          TRANSFER FROM DECISION MODE TO TRACE MODE          */
/*          *****/
/*****

```

```

int transfer(float ys1,float yg1,float xs1,float xg1,float ym,
            float xm,float xd[],float yd[],float alpha_o,int kt)
{
    if(fabs(ys1-yg1)>fabs(xs1-xg1)) /*go Y direction*/
    {
        if(((ys1>yg1)&&(ym<yd[kt])&&(alpha_o>-0.4))
            ||((yg1>ys1)&&(ym>yd[kt])&&((alpha_o>pi-0.4)||((alpha_o>-pi&&alpha_o<-pi/2))))
            return 0;
        else
            return 1;
    }
    else /*X direction*/
    {
        if(((xs1>xg1)&&(xm<xd[kt])&&(alpha_o>-pi/2-0.4))
            ||((xg1>xs1)&&(xm>xd[kt])&&(alpha_o>pi/2-0.4)))
            return 0;
        else
            return 1;
    }
}

```

```

int transfer1(float ys1,float yg1,float xs1,float xg1,float ym,
            float xm,float mx2,float my2)
{
    if(fabs(ys1-yg1)>fabs(xs1-xg1)) /*go Y direction*/
    {
        if(((ys1>yg1)&&(ym<my2))
            ||((yg1>ys1)&&(ym>my2)))
            return 0;
        else
            return 1;
    }
    else /*X direction*/
    {
        if(((xs1>xg1)&&(xm<mx2))
            ||((xg1>xs1)&&(xm>mx2)))
            return 0;
        else
            return 1;
    }
}

```

```

/*****
/*          COORDINATE          */
/*          */
*****/

float coord(float bata1, float vo, float v, float xoo, float yoo,
float xm, float ym)
{
float a, b, c;
a=vo*vo+vo*vo*tan(bata1)*tan(bata1)-v*v-v*v*tan(bata1)*tan(bata1);
b=-2*(xm*vo*vo+xoo*vo*vo*tan(bata1)*tan(bata1)-vo*vo*(yoo-ym)*tan(bata1)
-xoo*v*v*tan(bata1)*tan(bata1)-xoo*v*v);
c=xm*xm*vo*vo+vo*vo*xoo*xoo*tan(bata1)*tan(bata1)-2*vo*vo*(yoo-ym)*xoo*tan(bata1)
+(yoo-ym)*(yoo-ym)*vo*vo-xoo*xoo*v*v-v*v*xoo*xoo*tan(bata1)*tan(bata1);

return (-b+sqrt(b*b-4*a*c))/(2*a);
}

```