

**MODELING AND OUTPUT FEEDBACK CONTROL
OF A FLOATING BALL INSIDE A TUBE**

APPROVED BY SUPERVISING COMMITTEE:

Chunjiang Qian, Ph.D., Chair

Artyom Grigoryan, Ph.D.

Yufang Jin, Ph.D.

Accepted: _____
Dean, Graduate School

Copyright 2012 José L. Gamboa
All Rights Reserved

DEDICATION

This thesis work is dedicated to my family and friends. They are my constant source of encouragement. Also, this thesis is dedicated to Christians on Campus at the University of Texas at San Antonio. They are the greatest group of friends on campus. Furthermore, this work is also dedicated to the Church in San Antonio. They shower me with bountiful amount of prayers. Above all, the honor and glory belongs to God who helped me through this stage in my life.

**MODELING AND OUTPUT FEEDBACK CONTROL
OF A FLOATING BALL INSIDE A TUBE**

by

JOSE LUIS GAMBOA, B.S.

THESIS

Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Electrical and Computer Engineering
August 2012

UMI Number: 1518678

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent on the quality of the copy submitted.

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



UMI 1518678

Copyright 2012 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

ACKNOWLEDGEMENTS

I want to recognize my professor, Dr. Chunjiang Qian, for guiding me through the process of a master thesis work. The knowledge and experience of control theory he possesses opened the way for me to complete this work. Also, I want to recognize my committee members, Dr. Grigoryan for his filtering advice and signal processing expertise and Dr. Jin for her dedicated support to my thesis work and education progress.

I would like to express my appreciation to the staff of the Electrical and Computer Engineering Department for their assistance and advice and also to the many departments at the University of Texas at San Antonio (UTSA) who provided the help when I needed it.

In addition, I want to recognize the Center for Simulation, Visualization, and Real Time Prediction (SiViRT), sponsored by the U.S. National Science Foundation under grant HRD-0932339, who provided support for this work. In particular, the center supported the project of altitude control in a blimp aircraft, which is an application of this thesis work.

Also, I want to show my appreciation to three people in my life. The first one is my mother because of her constant cheering and her words of “You have time for everything.” The second is my family for their faithful support not related to this thesis, but which helped me in ways they do not know. The third is Christians on Campus at UTSA because of their genuine friendship.

August 2012

MODELING AND OUTPUT FEEDBACK CONTROL OF A FLOATING BALL INSIDE A TUBE

José Luis Gamboa, M.S.
The University of Texas at San Antonio, 2012

Supervising Professor: Chunjiang Qian, Ph.D.

This thesis presents modeling, simulation, and real-time output feedback control of a floating ball system that contains a noisy sensor feedback. It concentrates on the control of the vertical position of an object inside a tube pushed upward by an air flow. The procedures of this thesis are to obtain a dynamic model, introduce an effective filtering method, and implement the filtering method on the system in real-time.

The improvement of the floating object's convergence to a desired vertical position is demonstrated by comparing two control techniques with and without the filtering method. First, the results of the convergence of the system output are shown by using a Fuzzy Logic Controller. Then, the new convergence results are presented by using the Fuzzy Logic Controller with the Median Filter. Second, the convergence results of the system output are shown using the tracking controller. Then, again, the new convergence results are shown using the tracking controller with the Median Filter.

Removing the noise of the feedback in a floating ball system leads to less of the unwanted oscillatory response from the system, faster settling time of the floating object to a desired height, and disturbance rejection. This work may further improve an object's floating stability at a desired altitude of a floating object system containing a very noisy sensor feedback.

TABLE OF CONTENTS

Acknowledgements.....	iv
Abstract.....	v
List of Tables	ix
List of Figures.....	x
Chapter 1: Introduction.....	1
1.1: Research Survey	1
1.2: The Floating Ball Plant	2
1.3: Significance of this Research.....	4
1.4: Organization of the Thesis	5
Chapter 2: Properties of the Hardware.....	6
Chapter 3: Fuzzy Logic Controller	16
3.1: Previous Work	16
3.1.1: Results	23
3.1.2: Discussion	27
Chapter 4: Fuzzy Logic Controller with Filter	31
4.1: Fast Fourier Transform.....	31
4.2: Mean Filter	32
4.2.1: Rectangular Mean Filter	32
4.2.2: Triangular Mean Filter.....	36
4.3: Median Filter	40
4.4: Median Filter Implementation in Real-Time	44
4.5: Discussion of FLC and FLC with Filter.....	47

Chapter 5: Dynamic Model by Data Collection	50
5.1: Model Development.....	50
5.1.1: Static Modeling.....	50
5.1.2: Dynamic Modeling	56
5.1.2.1: Data Collection.....	56
5.1.2.2: Control System Model Equation	59
5.1.2.3: Batch Least Square Method	63
5.1.2.4: Model Verification	65
Chapter 6: Model Testing using Fuzzy Logic Controller	67
6.1: System Model.....	67
6.2: Fuzzy Logic Controller	67
6.3: Simulation Results.....	70
6.4: Problems and Discussion	72
Chapter 7: Tracking Controller Development and Simulation.....	73
7.1: System Model.....	73
7.2: Controller Development and State Space Model	74
7.3: Simulation Results.....	80
7.4: Conclusion.....	82
Chapter 8: Tracking Controller Implementation to Real System	83
8.1: Observer without Filter	84
8.2: Observer with Median Filter of Window Size 3	84
8.3: Observer with Median Filter of Window Size 5	85
8.4: Observer with Median Filter of Window Size 7	86

8.5: Discussion	89
Chapter 9: Conclusion and Future Work	90
9.1: Conclusion.....	90
9.2: Future Work	92
Appendix I	94
Appendix II	96
Bibliography	130
Vita	

LIST OF TABLES

Table 2.1	Tube Specifications.....	6
Table 2.2	Floating Object Specifications.....	7
Table 2.3	Fan Specifications.....	7
Table 2.4	Motor Specifications.....	8
Table 2.5	Range Sensor Specifications.....	9
Table 2.6	Data Acquisition Device Specifications.....	11
Table 2.7	DAQ Pin Connections.....	12
Table 2.8	Amplifier Circuit Specifications.....	12
Table 2.9	MATLAB Specifications.....	13
Table 3.1	Comparison between Three-Rule and Five-Rule FLC.....	28
Table 4.1	Comparison between Types of Mean Filter.....	39
Table 4.2	Comparison between Median Filters of Window Size 3 and 5.....	43
Table 4.3	Comparison between Mean and Median Filters.....	44
Table 4.4	Comparison between FLC and FLC with Filter.....	47
Table 5.1	Parameters of the Model Equation.....	65
Table 6.1	Parameters of the Model Equation.....	67
Table 8.1	Comparison between Observer and Observer with Filter.....	87
Table A.1	Data Shifting by Mean Value.....	94

LIST OF FIGURES

Figure 1.1	The Floating Ball Plant	3
Figure 1.2	The Tri-Turbofan Airship	4
Figure 2.1	Plastic Tube.....	6
Figure 2.2	Floating Object.....	7
Figure 2.3	A View of the Fan inside Case	8
Figure 2.4	Motor and Fan Case Added to FBS	8
Figure 2.5	Range Sensor Dimensions	9
Figure 2.6	Range Sensor Front View	10
Figure 2.7	NI USB-6008 Pin Configuration	11
Figure 2.8	NI USB-6008 Top View	12
Figure 2.9	Amplifier Circuit inside Case	13
Figure 2.10	MATLAB Software Environment	14
Figure 2.11	Block Diagram of FBS.....	15
Figure 3.1	Linguistic Rules	17
Figure 3.2	Three-Region Membership Functions	21
Figure 3.3	Three-Region FL and Tube.....	21
Figure 3.4	Five-Region Membership Functions.....	23
Figure 3.5	Three-Rule FLC with Modified Error Equation	24
Figure 3.6	Five-Rule FLC with Modified Error Equation	25
Figure 4.1	Rectangular Mean Filter of Window Size 3	35
Figure 4.2	Rectangular Mean Filter of Window Size 5	35
Figure 4.3	Triangular Mean Filter of Window Size 3.....	38

Figure 4.4	Weighted Mean Filter of Window Size 3	38
Figure 4.5	Median Filter of Window Size 3.....	42
Figure 4.6	Median Filter of Window Size 5.....	43
Figure 4.7	Script of the Three-Regions of FLC	44
Figure 4.8	Script of the Five-Regions of FLC.....	45
Figure 4.9	Three-Rule FLC with Median Filter of Window Size 3	45
Figure 4.10	Five-Rule FLC with Median Filter of Window Size 3	46
Figure 5.1	Experiment Setup.....	51
Figure 5.2	Voltage Fan vs. Height of Ball with Hysteresis.....	52
Figure 5.3	Incorrect Assumption Line	54
Figure 5.4	Input Data.....	57
Figure 5.5	Output Data.....	57
Figure 5.6	Shifted Input Data	58
Figure 5.7	Shifted Output Data	59
Figure 5.8	Output Modeling by Recursive $\hat{y}(k)$ using $y(k)$ Real Output Values	66
Figure 6.1	Script of the Fuzzy Rules.....	68
Figure 6.2	Script of Three-Regions of FLC	69
Figure 6.3	Three-Rule FLC $V_R = 0.4$ Results	70
Figure 6.4	Three-Rule FLC $V_R = 0.7$ Results	71
Figure 6.5	Three-Rule FLC $V_R = 0.8$ Results	71
Figure 7.1	Full-State Feedback Block Diagram.....	74
Figure 7.2	Observer Block Diagram	75
Figure 7.3	Observer Block Diagram with $J = \mathbf{0}$ and Unmeasurable $\mathbf{x}(k)$ State	76

Figure 7.4	Observer $VR = 0.4$ Results	81
Figure 7.5	Observer $VR = 0.5$ Results	81
Figure 7.6	Observer $VR = 0.7$ Results	82
Figure 8.1	Observer and Controller $u(k)$ Only	84
Figure 8.2	Observer Block Diagram with Filter Block	85
Figure 8.3	Observer and Controller $u(k)$ with Filter of Window Size 3	85
Figure 8.4	Observer and Controller $u(k)$ with Filter of Window Size 5	86
Figure 8.5	Observer and Controller $u(k)$ with Filter of Window Size 7	87

CHAPTER 1: INTRODUCTION

1.1 Research Survey

The control of the altitude of a floating object is of importance in this thesis. Settling an object in midair to a specific altitude with more precision is the control problem that is address here. Research work regarding control of the altitude or height of a floating object has not been considered enough [1],[6]. This lack of interest is due to the fact that the physical model of a floating object by an air stream becomes exponentially complicated as more nonlinearities in a floating ball plant are considered [1], [6].

Some researchers, who come across this control problem, approach this issue by different methods. For example, a floating ball plant can be constructed with different hardware components. Also, the control strategies implemented to the floating ball system are many. Furthermore, the floating ball system can offer different applications.

The floating ball plant considered in work [1] is constructed using a gimbaled air jet. In addition, a rotating base for the air jet is added for more complexity. The approach of [1] is to move a ball freely in a three dimensional way by utilizing an air stream and the air stream's angle. [1] mentions that the analysis of the physics regarding the equilibrium point reached by the floating object due to an air stream has been sufficiently investigated. Nevertheless, the trajectory that the floating object takes before reaching this equilibrium point is a control problem not known very well. In [1], this control problem is approached using three control strategies: open loop control, feedback linearization to simplify the nonlinear system, and the same feedback linearization but with a filter.

The work [10] considers a similar plant to that of this thesis, but it incorporates an air compressor to the plant. The compressed air is released in order to lift the ball vertically. Another

difference recognized in the paper is the modeling method. The plant model is approached as a multi-model system rather than just one model. The paper [10] considers more complicated control structure such as multiple models to describe the one system and two-degrees-of-freedom RST digital controllers for each of its models.

The applications of this thesis work are many. For example, the blimp altitude control, air float flow meter control, and the transportation of objects by air [1], [6] are some to name a few.

1.2 The Floating Ball Plant

The floating ball plant in this thesis is a device, where an object inside a vertical tube is lifted up by an air flow created by a fan, not yet regulated by a controller. Within the plant, the floating object is governed by active forces such as the gravity force and the wind force from a fan. In a simple force analysis of the floating ball plant, the lifting force (i.e., the moving force) is the air stream created by the fan. The impeding forces (i.e., the pulling forces) are gravity and the air friction. The floating ball plant is developed by considering these forces and assembling some components as will be described in Chapter 2.

Since the floating ball plant, shown in Figure 1.1, of this thesis utilizes gravity and air lifting forces, it opens a control arena where a control systems engineer can set up control over the floating object. To control the floating ball plant, the Floating Ball System (FBS), with its controlling components, needs to be integrated into the floating ball plant. This FBS has a distance detection sensor as the output and the motor fan voltage as the input. In this control system, the motor voltage is manipulated based on the feedback provided by the sensor output voltage. The motor then actuates rotation of the fan which in turn propels upward the object within the tube.

The FBS is an assembly of simple components such as a hairdryer fan, a plastic tube, and a range sensor which was incorporated all together by the Control, Computation, and Cybernetics (C3) Lab students at the University of Texas at San Antonio (UTSA). The purpose for the development of the FBS was to create a system where an object floats inside a tube at a desired altitude.

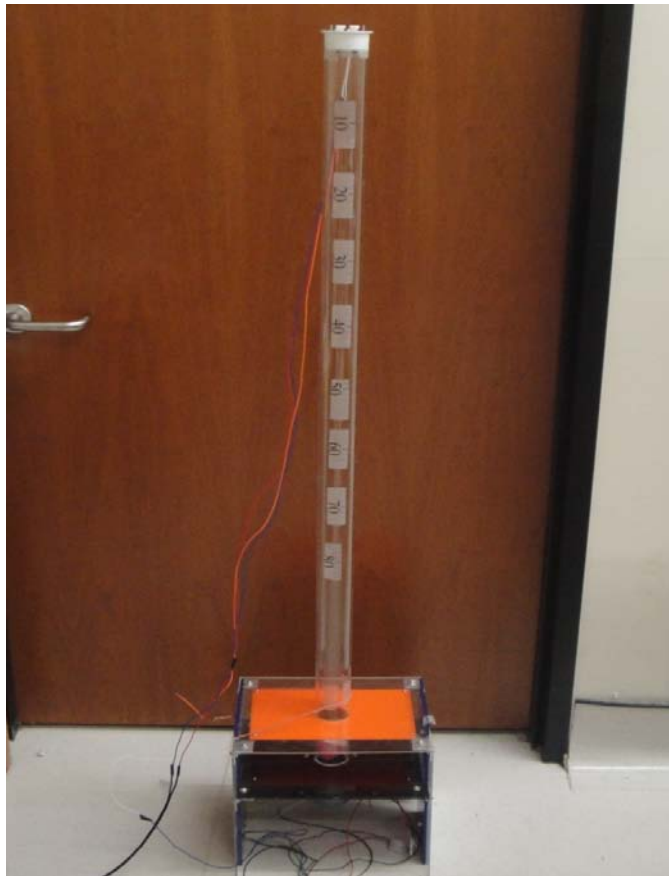


Figure 1.1: The Floating Ball Plant

This floating ball plant, which uses air flow, can be compared to a floating ball plant where the object inside a tube is on top of the water level controlled by a pump. This similar floating ball plant is a much simpler plant in terms of control manipulation than the floating ball plant of this thesis. The system is simpler because the object is already in a stable position due to the buoyant force with no oscillations around a target height position.

1.3 Significance of this Research

This thesis has educational and application significance. At the educational level, the FBS is the final project for the Intelligent Control engineering course offered at UTSA. At the application level, the FBS is a stationary platform for a blimp altitude control system. The blimp system, a tethered UAV application, was already completed through the CREST grant/SiViRT Center. The Tri-Turbofan Blimp in Figure 1.2, used in [13], is utilized in this application as the platform for implementing the three-rule Fuzzy Logic Controller. Additionally, the FBS can play a role as an altitude control platform for many aerial projects such as UAV, airship, and airplane artifacts.



Figure 1.2: The Tri-Turbofan Airship

The FBS, which includes the physical floating ball plant, is used by students to develop their control theory methods. It is in this practical scenario that students can learn and feed their curiosity and expand their theoretical knowledge and their real implementation practice.

The contributions made by this thesis to the electrical engineering field are the procedures, techniques, and optimization used upon a floating ball plant and system. This thesis also helps bridge the gap between theoretical research and engineering practice.

1.4 Organization of the Thesis

In Chapter 2, the FBS hardware and software components are listed. In Chapter 3, the definition and development of the Fuzzy Logic Controller (FLC) is presented. The results of this controller are shown. Then in Chapter 4, the results of the FLC incorporating a filter are also shown. The development of a model for the FBS is considered in Chapter 5. This chapter starts by considering the steady state modeling, but then quickly moves on to the dynamic modeling. In Chapter 6, the developed dynamic model is tested by applying a Three-Rule FLC. The tracking controller using the Output Feedback control is developed in Chapter 7. Then simulations are performed to test the stability of the Observer and the tracking controller. For Chapter 8, this thesis shows the application of the tracking controller and observer to the FBS in real-time. Finally, in Chapter 9, a summary of the entire results with a list of issues and some future works are mentioned.

CHAPTER 2: PROPERTIES OF THE HARDWARE

This section provides specifications of the hardware that are used in the FBS. First, the components of the floating ball plant are presented and then the additional components that comes with the FBS. Finally, the block diagram of the FBS is provided for a view of the entire system.

The Floating Ball Plant Hardware

The Floating ball plant is assembled with a plastic tube, a floating object, and a fan motor.

The Tube

Table 2.1: Tube Specifications

Vendor	Regal Plastics
Material	Acrylic plastic
Outer Diameter	5 cm \approx 2 in
Inner Diameter	4.4 cm \approx 1.75 in
Length	4 ft, 6ft, and 10 ft tubes
Tube Wall Thickness	0.6 cm \approx 0.24 in



Figure 2.1: Plastic Tube

The Floating Object

Table 2.2: Floating Object Specifications

Material	Plastic egg shell
Shape	Oval (The half hemisphere of an egg shell)
Diameter	4.4 cm
Length	2.8 cm

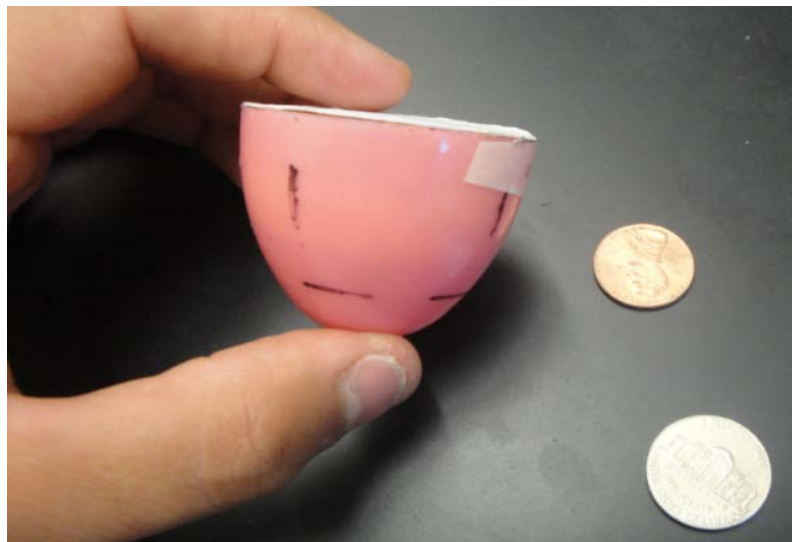


Figure 2.2: Floating Object

The Fan

Table 2.3: Fan Specifications

Fan Type	Generic Hair Dryer
Number of Blades	5

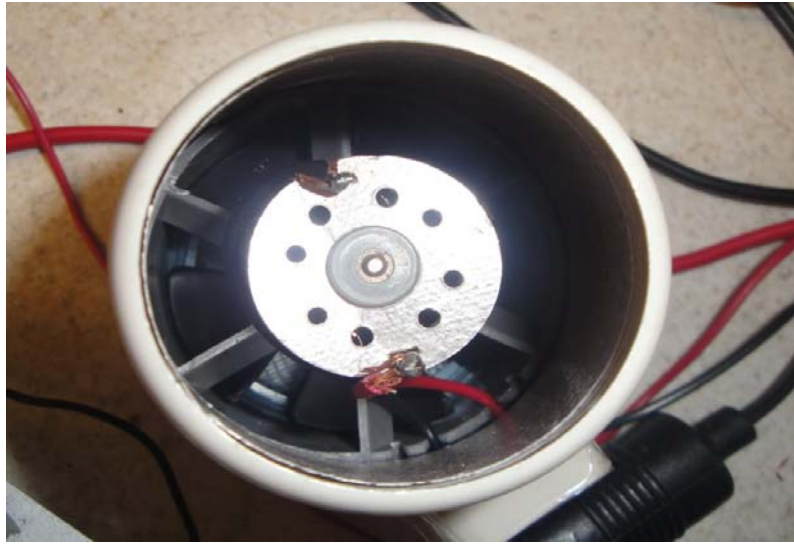


Figure 2.3: A View of the Fan inside Case

The Motor

Table 2.4: Motor Specifications

Max Voltage	$\pm 13V$ DC
Case	Hair Dryer Case

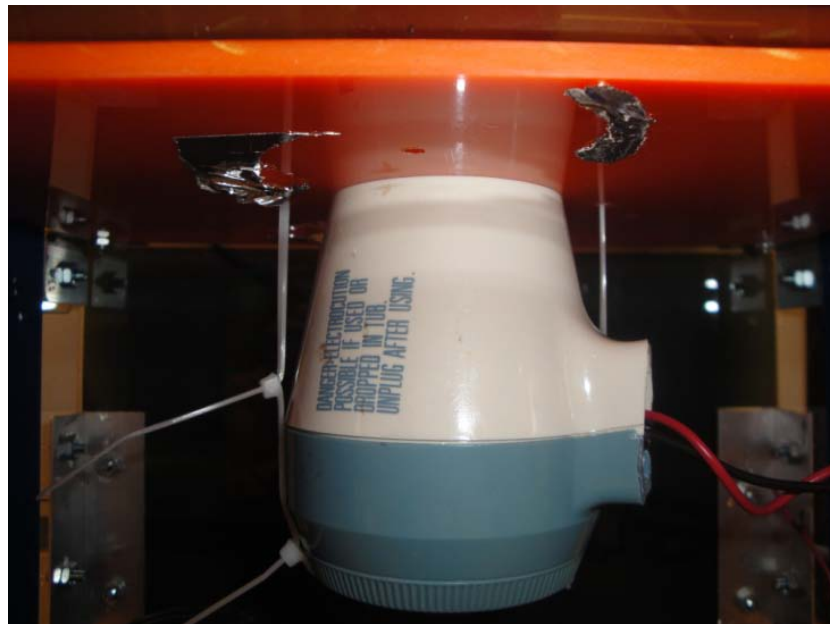


Figure 2.4: Motor and Fan Case Added to FBS

The Floating Ball System (FBS) Hardware

The FBS hardware is the floating ball plant plus the controlling components of the FBS, which includes the range sensor, DAQ device, the amplifier circuit, and MATLAB.

The Range Sensor

Table 2.5: Range Sensor Specifications

Name	SHARP GP2D12
Output Type	Analog
Detection Range	10 cm to 80 cm
Output Voltage Range	0.3V to 3.0V
Supply Voltage	+5V DC
Cost	\$20
Beam Type	Infrared (IR)
Beam Width	Football shaped beam with approximately 16 cm wide in the center [2]
Datasheet	Refer to [15] in Bibliography

- Dimensions (mm):

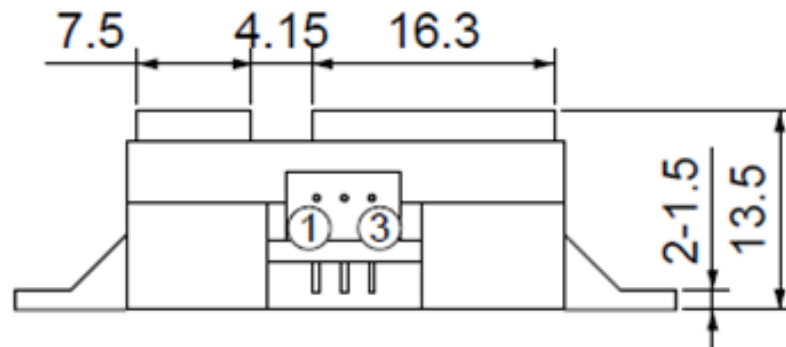


Figure 2.5: Range Sensor Dimensions

- Three Output Leads: The voltage output is labeled as 1, the ground is labeled as 2, and the voltage supply is labeled as 3 shown in Figure 2.5.
- Output Firing Average Time: The first voltage output from the sensor takes an average time of 43.3ms to appear. Then the sensor takes an average time of 38.3 ms for each additional voltage output (27.86 output readings/second, i.e. 27.86 Hz).
- Two Dead Regions (Non-Working Regions): The first dead region is located directly in front of the sensor within a distance of approximately 10cm from sensor. The second dead region is located in front of the sensor at the distance of approximately 80 cm and more.
- Working Region: The working or operating region varies from sensor to sensor but is usually within 10 cm to 80 cm from the view of the sensor.



Figure 2.6: Range Sensor Front View

The Data Acquisition Device

Table 2.6: Data Acquisition Device Specifications

Name	NI USB – 6008
Analog Input (AI) Resolution*	12 Bits differential; 11 bits single-ended
Max AI Sample Rate	10 kilo-Samples/second
User Guide	Refer to [9] in Bibliography

*AI = Analog Input

- Pin Configurations:

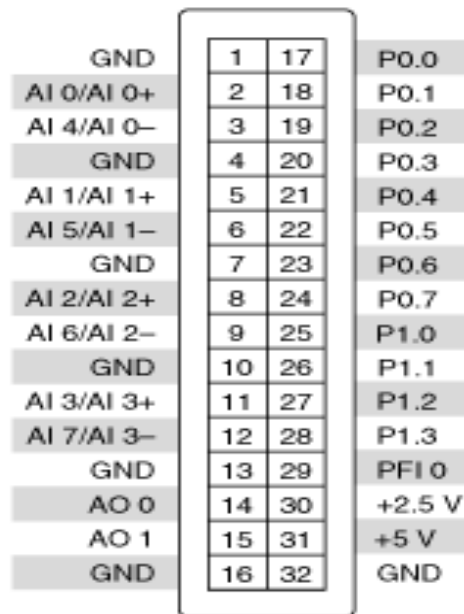


Figure 2.7: NI USB- 6008/6009 Pin Configuration

- Physical Wiring: Figure 2.7 shows the pin designation of the DAQ 6008 device by National Instruments. Table 2.7 shows the pin and hardware connections according to the Block Diagram in Figure 2.11.

Table 2.7: DAQ Pin Connections

From	To
IR Sensor V_o *	PIN 2 (AI 0)
PIN 10 (GND) * or any GND pin	Amplifier Circuit Ground Lead
PIN 14 (AO 1) Control *	Amplifier Circuit Control Positive Voltage Lead
IR Sensor Ground Lead	PIN 10 (GND)
Power Supply 2	PIN 10 (GND)

* V_o = Output Voltage; GND = Ground; AO = Analog Output; IR = Infrared

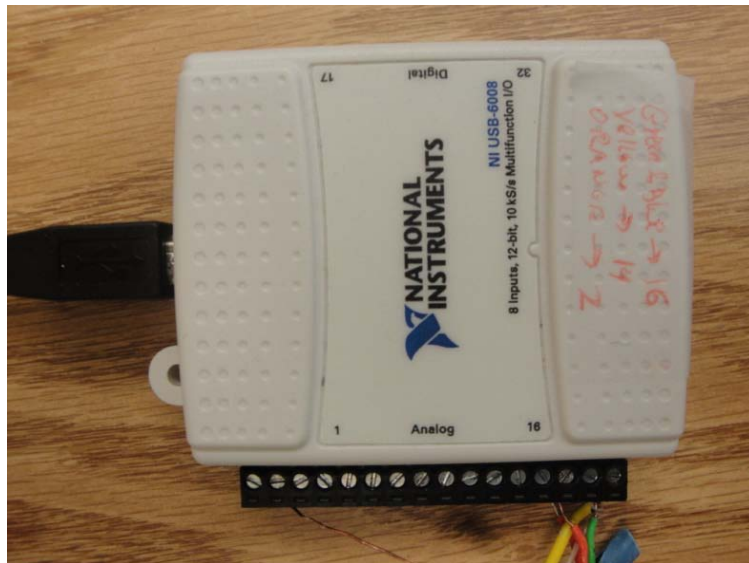


Figure 2.8: NI USB-6008 Top View

The Amplifier Circuit

Table 2.8: Amplifier Circuit Specifications

Supply Voltage	+6V DC
Current	0-3.5 Amps

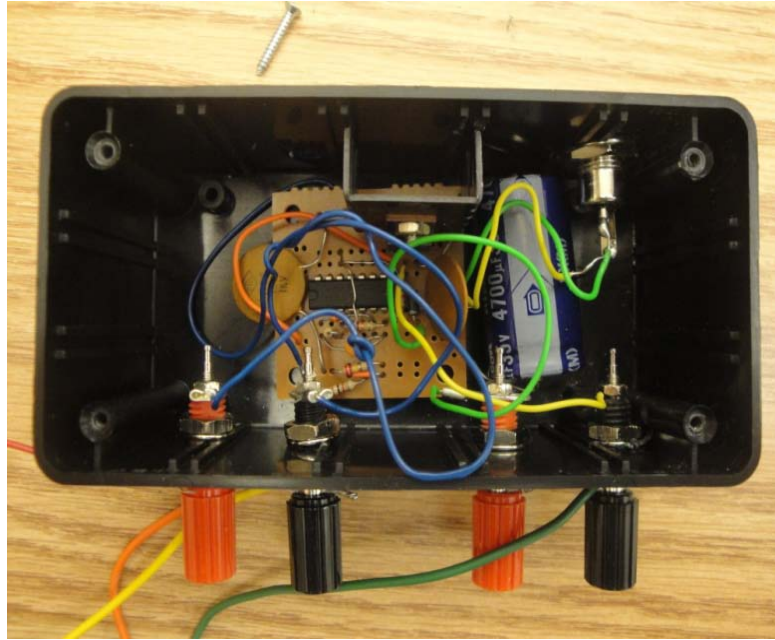


Figure 2.9: Amplifier Circuit inside Case

The MATLAB Software

Table 2.9: MATLAB Specifications

Version	7.9.0 (R2009b)
Bit size	32 – bit (win32)
Software	Refer to [11] in Bibliography

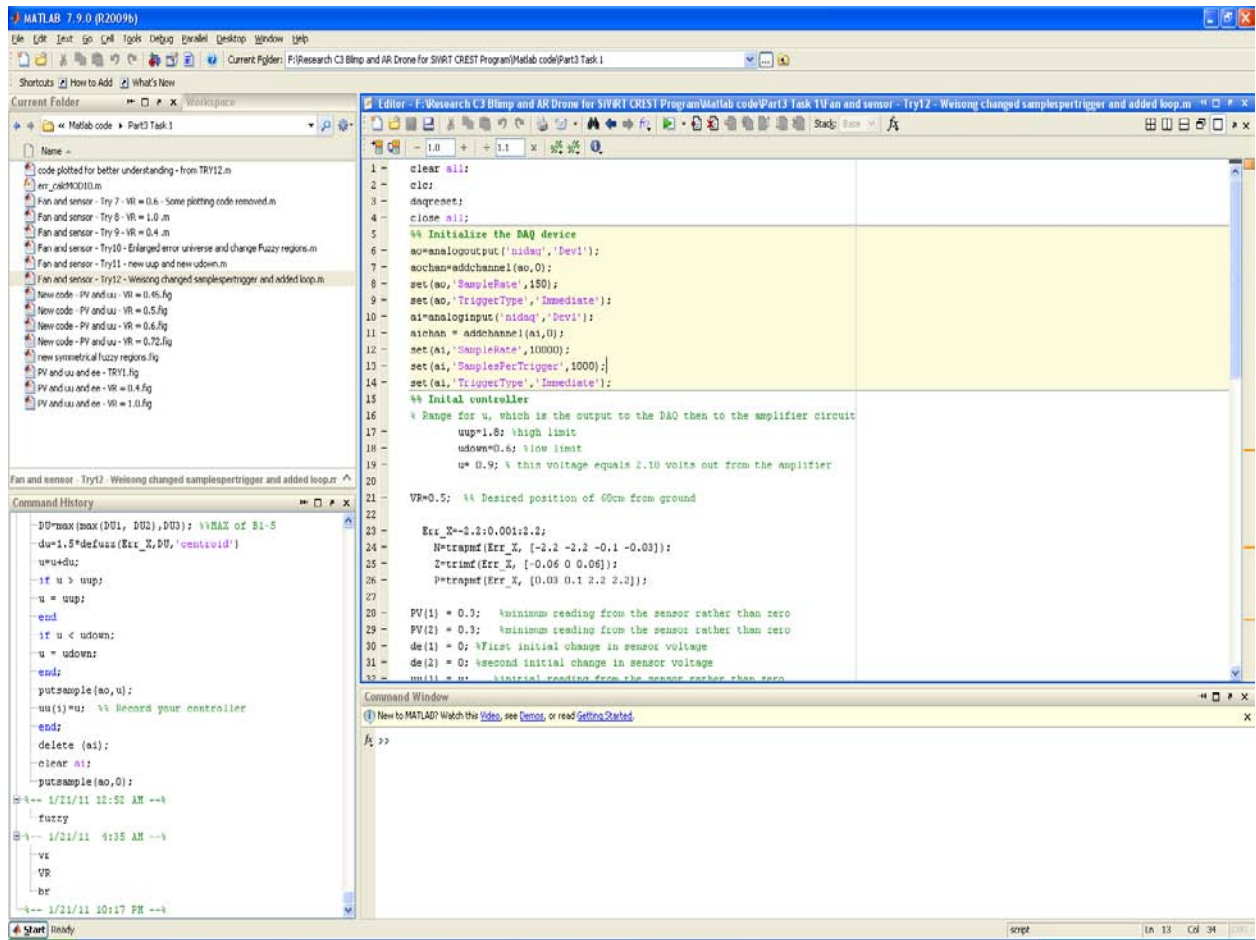


Figure 2.10: MATLAB Software Environment

The Block Diagram of the FBS

The diagram in Figure 2.11 shows all the components wired together.

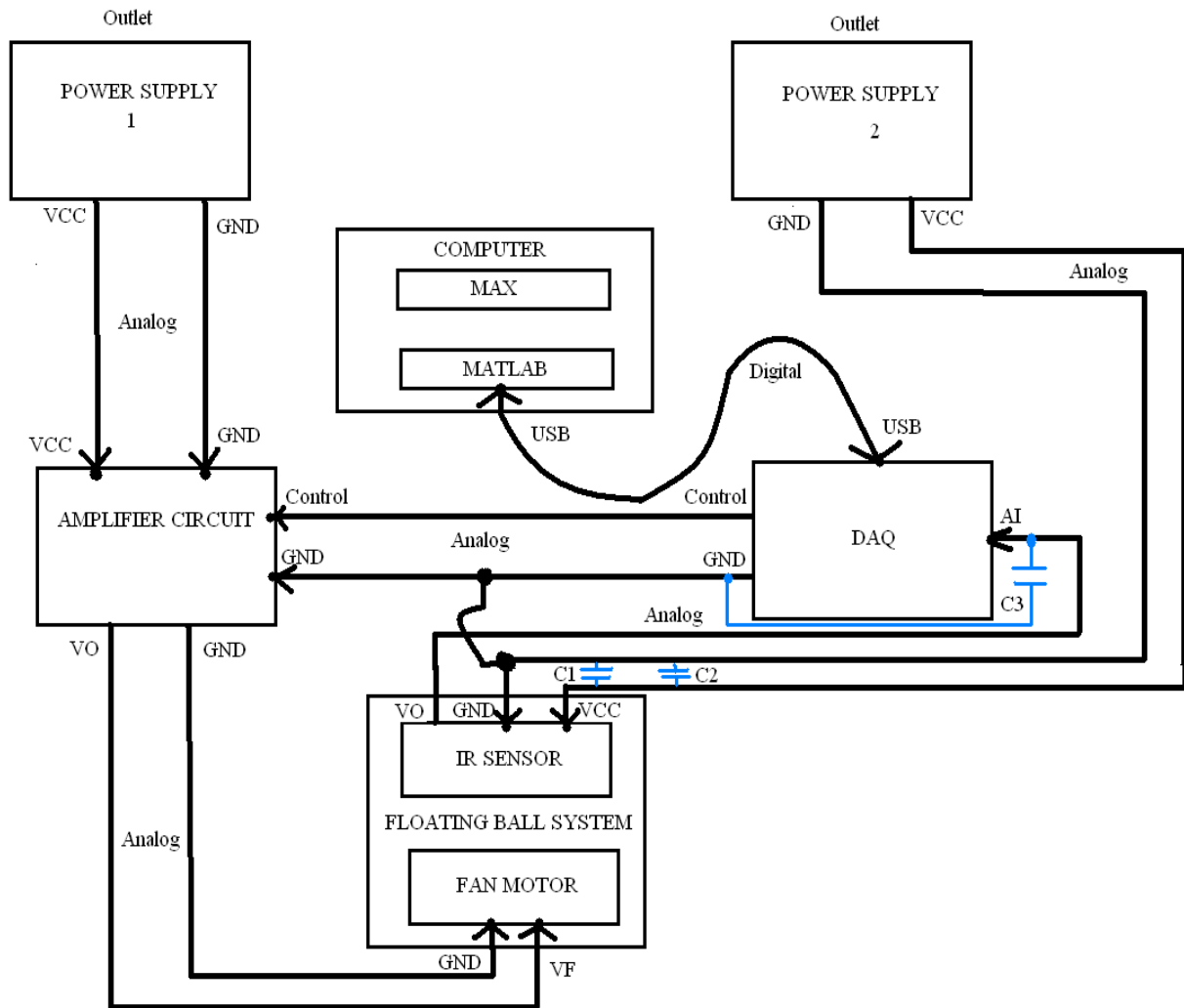


Figure 2.11: Block Diagram of FBS

The signal flow of the FBS system can be followed by referring to Figure 2.11. The signal flow begins with; first, the sensor data is acquired at specific sampling rate from the DAQ into the MATLAB engine. Second, the sensor data value is processed and calculations are made in MATLAB at a speed defined by the processing rate. Third, the processed data, used for real-time control, is sent back to the DAQ. Fourth, the DAQ sends the control value through the Amplifier circuit into the fan motor to lift the object. Finally, fifth, the object's altitude is read by the sensor.

CHAPTER 3: FUZZY LOGIC CONTROLLER

The FLC is one of the two controllers used on the floating ball system (FBS) that will be considered in this thesis. This chapter discusses the implementation of the Fuzzy Logic Controller (FLC) on the FBS. Also, in this chapter, some filter methods are considered and implemented to dampen the oscillatory response of the floating object at a desired height. First, the simulation results of the filters are observed then the results of the real-time filtering implementation. Finally, a comparison table is presented to discuss the filter's smoothing of the noisy sensor signal at various target heights.

3.1 Previous Work

According to the history of the FBS in the UTSA C3 Lab, the FLC was the initial approach to control the FBS. The FLC method was utilized by the researchers of the C3 Lab and students of the course Intelligent Controls offered at UTSA. Every year after the course is taught, the control performance of the FBS is improved due to previous information recorded and results obtained from the researchers and students.

This section shows the results of the Fuzzy Logic (FL) approach to the FBS that was completed before this thesis work. FL is the counterpart of the crisp logic. Since FL includes degree of memberships which are within 0 and 1, it is called "fuzzy". In crisp logic the values are strictly only 0 and 1. The theory of FL allows for the development of a controller method called FLC, which does not use a mathematical model. In addition to the FLC results presented, a description of how the FLC is derived is given.

In general, FL serves as a basic control technique for achieving control of a plant or system that does not have a mathematical model. The FL control depends on the behavior

description as seen by an engineer using the plant. Figure 3.1 shows how the engineer views the floating ball plant and generates rules for the FLC.

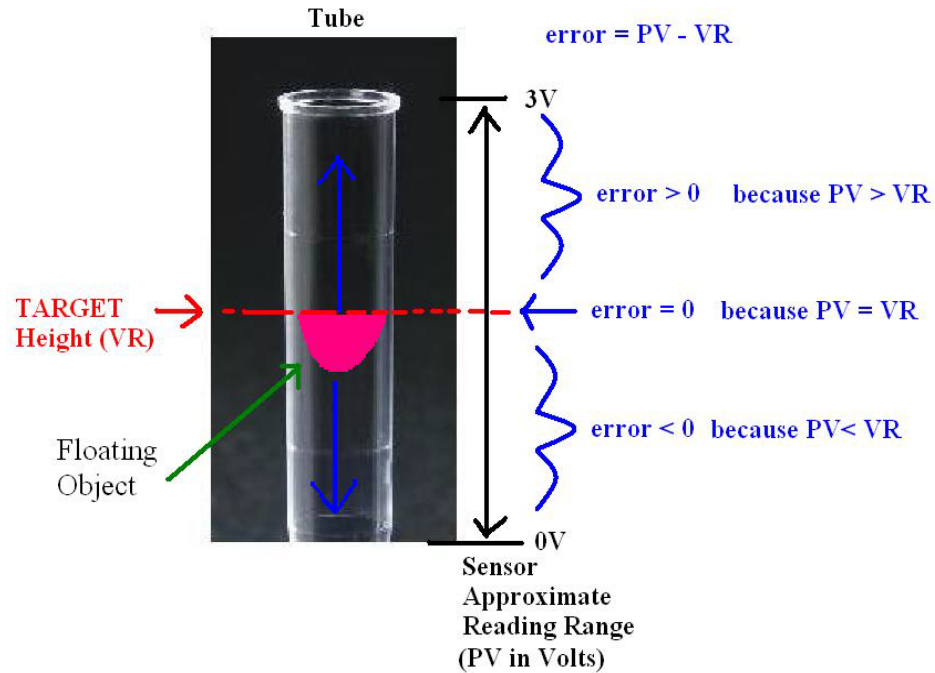


Figure 3.1: Linguistic Rules

The observations made by the engineers in Figure 3.1 results in human rules explained in linguistic form. In addition to these rules used for the development of the FLC, the error equation, the error universe, and the fuzzy regions are needed. Below, the procedures for the three-rule and the five-rule FLCs are mentioned.

As depicted in Figure 3.1, if the object is in the target altitude, then the *error* is zero. However, if the object is above or below the target altitude, then the *error* is nonzero. According to the behavior description just mentioned, the set up of the FLC with three rules is in the following way.

The Three - Rule FLC

The Linguistic Rules

- If error, ee , is negative then *change in controller, du* , is positive.
- If error, ee , is zero then *change in controller, du* , is zero.
- If error, ee , is positive then *change in controller, du* , is negative.

The General Error Equation

The general error equation, also seen in Figure 3.1, is

$$ee(k) = PV(k) - VR \quad (3.1)$$

where $PV(k)$ is the actual voltage and VR (voltage reference) is the set point or desired voltage.

The VR value is a constant value while the $PV(k)$ value is a variable value changing as the object moves up or down. This error formula was modified by a C3 Lab student to improve the performance of the FLC.

The Modified Error Equation

The general error equation becomes

$$ee^*(k) = PV^*(k) - 0.15 \times VR, \quad (3.2)$$

where

$$PV^*(k) = 0.15 \times PV(k) + 2.2 \times speed(k), \quad (3.3)$$

and

$$speed(k) = PV(k) - PV(k-1). \quad (3.4)$$

In Equation (3.2), VR is decreased by a factor of 0.15 (decreased by 15%) and the present voltage $PV(k)$ becomes the pseudo present voltage $PV^*(k)$. In Equation (3.3), $PV^*(k)$ is equal to $PV(k)$ plus a pseudo speed component $speed(k)$, where $PV(k)$ is decreased by a factor 0.15 and $speed(k)$ is increased by a factor of 2.2. Also, in Equation (3.4), $PV(k-1)$ is one previous value from $PV(k)$.

The new error formula $ee^*(k)$ contains the coefficients of 0.15 and 2.2 that were obtained via trial and error experimentations of the FBS. These coefficients were derived in order to deal with two major problems of the FBS: the delay of the system and the nonlinear relationship of the altitude and voltage of the sensor.

The problems of Equation (3.1) are the strict positive and negative $ee(k)$ results in the case when $PV(k)$ is changing fast and is very near to VR . A case of these problems is when the floating object is very close and quickly approaching the target position from the bottom of the tube. In this case, Equation (3.1) results in a negative $ee(k)$ value. The equation does not counteract the quick rising of the floating object by outputting a positive $ee(k)$ value, but rather it continues to provide a negative $ee(k)$ value until the floating object reaches the target position. In response to this case, Equation (3.2) provides a positive error value before the floating object or $PV(k)$ reaches the target from below in order to reverse the response of the controller and hence slow down the ball's velocity when approaching the target. In addition, the development of Equation (3.2) is a response to the slow processing rate of the system.

The Error Universe

The Error Universe is also the error range defined as $Err_X = [-2:0.01:2]$. The Error Universe is derived from the range sensor. According to the specifications of the sensor in Table 2.5 of Chapter 2, the sensor output ranges from approximately 0.3V to 3.0V. This voltage range is mapped in a nonlinear way to the detection distance range of 10cm to 80cm. In order to stay within the working region of the sensor, a safe operating region is capped to a range of [0.4V, 2.4V]. The working operating region of the sensor is described in Chapter 2.

The maximum and minimum values of the Error Universe are obtained using the general equation (3.1). The Error Universe is defined as the error value calculated by subtracting the desired position VR from the actual position $PV(k)$. The maximum Error Universe value is

$$ee(k) = 2.4V - 0.4V = 2.0V,$$

where $PV(k)$ value is the maximum voltage value and VR is the minimum voltage value both read by the sensor.

The minimum Error Universe value is

$$ee(k) = 0.4 - 2.4 = -2.0V,$$

where $PV(k)$ value is the minimum voltage value and VR is the maximum voltage value both read by the sensor.

The Fuzzy Regions

The fuzzy regions of the FLC are called *Negative*, *Zero*, and *Positive* regions. Within the error universe, the *Negative* region is allocated as $N = [-2, -2, -1, 0]$ with a trapezoidal membership function, the *Zero* region as $Z = [-1, 0, 1]$ with a triangular membership function, and the *Positive* region as $P = [0, 1, 2, 2]$ with a trapezoidal membership function. In Figure 3.2, the red line is the *Negative* region, the black line is the *Zero* region, and the blue line is the *Positive* region. The P and N regions cover 50 % of the *Zero* region as depicted in Figure 3.2.

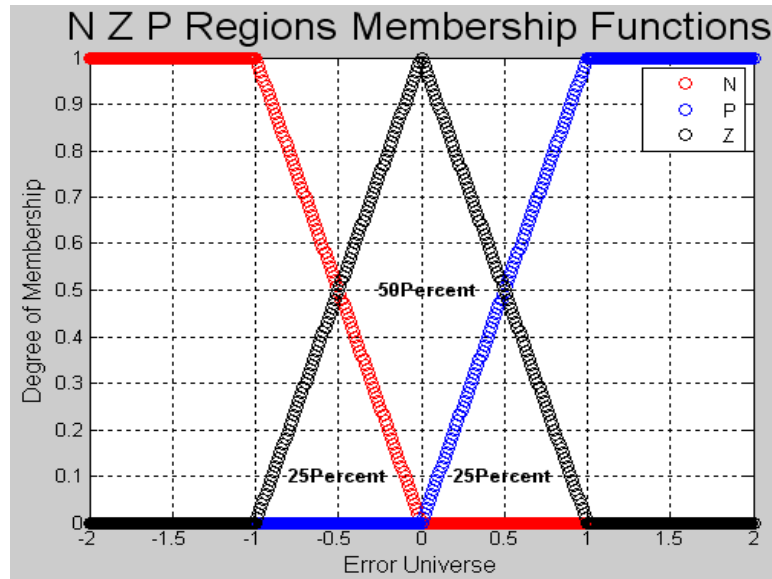


Figure 3.2: Three-Region Membership Functions

In Figure 3.2, the horizontal axis is labeled as the Error Universe. The vertical axis is named the Degree of Membership. The Degree of Membership value, also known as the degree of fulfillment (DOF), is the degree to which a specific error value belongs to a fuzzy region. Figure 3.3 shows how the FBS and the membership functions of the regions are related. The three fuzzy regions correspond to the regions in the tube around the target point.

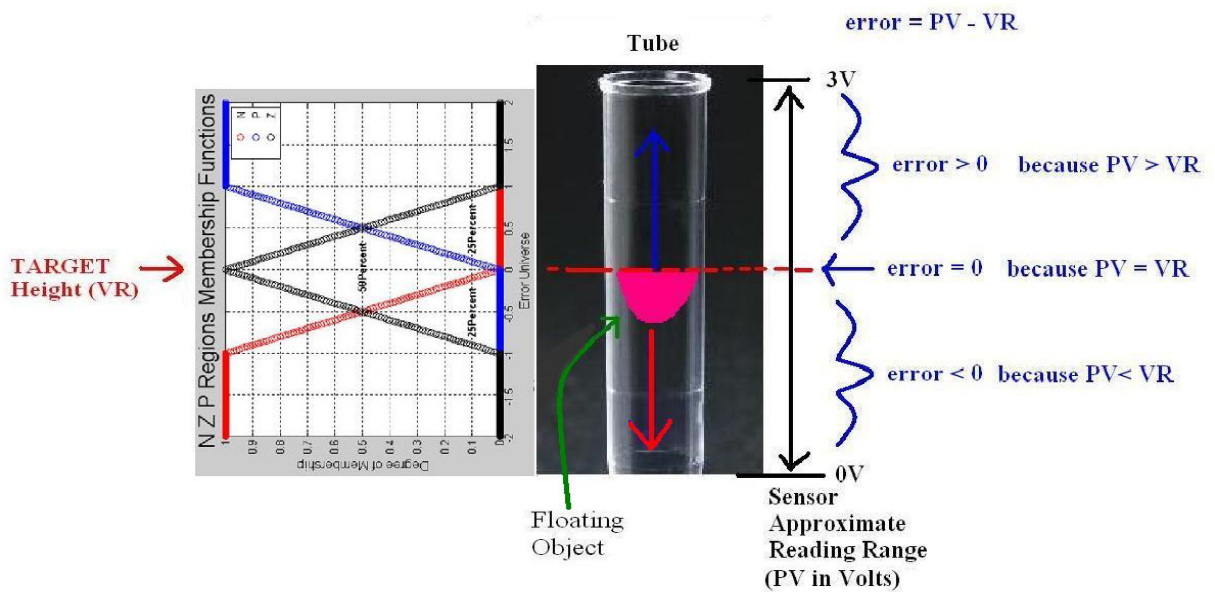


Figure 3.3: Three-Regions FL and Tube

The Five - Rule FLC

Next, the control aspects of the Five-Rule FLC are presented as follows:

The Linguistic Rules

- If *error, ee*, is very negative then *change in controller, du*, is very positive.
- If *error, ee*, is negative then *change in controller, du*, is positive.
- If *error, ee*, is zero then *change in controller, du*, is zero.
- If *error, ee*, is positive then *change in controller, du*, is negative.
- If *error, ee*, is very positive then *change in controller, du*, is very negative.

The Error Universe

The Error Universe is the error range which is represented by $Err_X = [-2:0.1:2]$, where the interval is 0.1. This Error Universe is the same as for the Three-Rules FLC, except for the bigger interval value.

The Fuzzy Regions

The fuzzy regions for the five-rule FL are *Very Negative*, *Negative*, *Zero*, *Positive*, and *Very Positive*. Within the error universe, the *Very Negative* region is allocated as $VN = [-2, -2, -1.5, -0.5]$ with a trapezoidal membership function, the *Negative* region as $N = [-1, -0.5, 0]$ with a triangular membership function, the *Zero* region as $Z = [-0.5, 0, 0.5]$ with a triangle membership function, the *Positive* region as $P = [0, 0.5, 1]$ with a triangular membership function, and the *Very Positive* region as $VP = [0.5, 1.5, 2, 2]$ with a trapezoidal membership function. The *P* and *N* regions cover 50 % of the *Zero* region as shown in Figure 3.4.

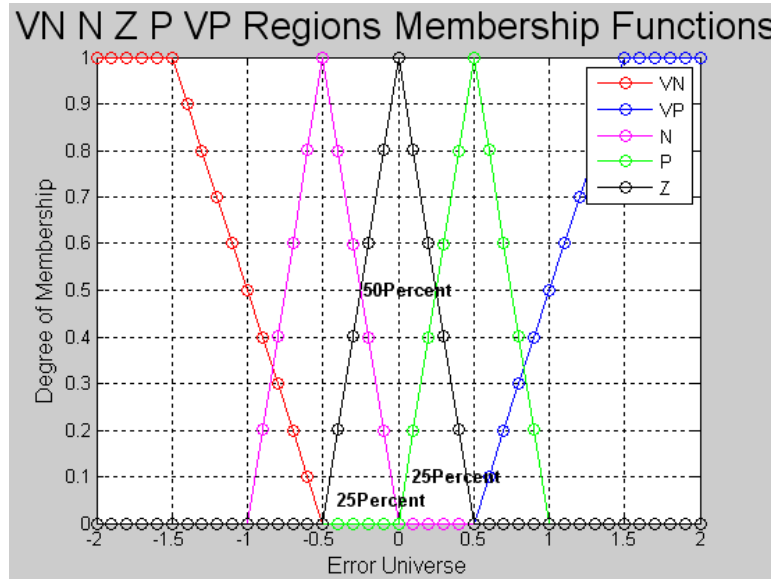
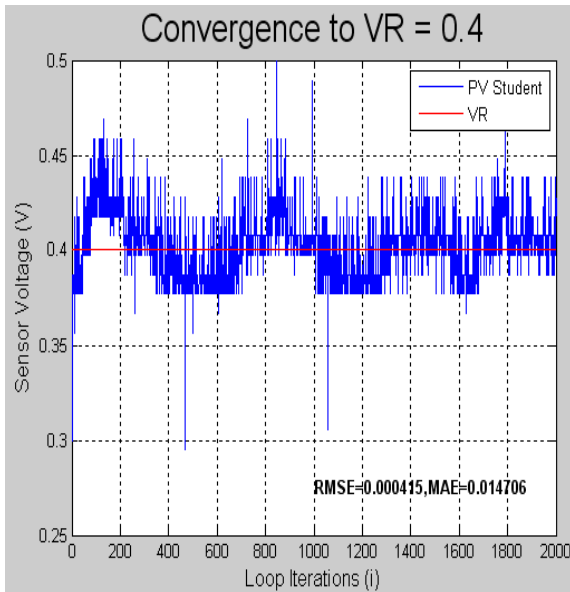


Figure 3.4: Five-Region Membership Functions

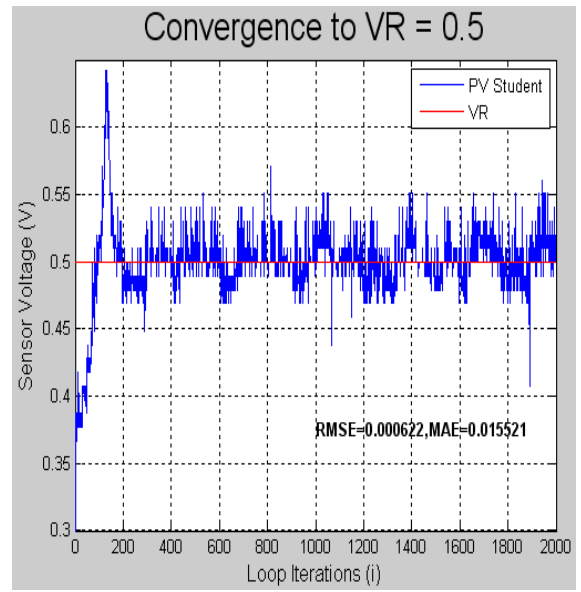
Both the general error equation (3.1) and the modified error equation (3.2) of the Three-Rule FLC also apply for the Five-Rule FLC.

3.1.1 Results

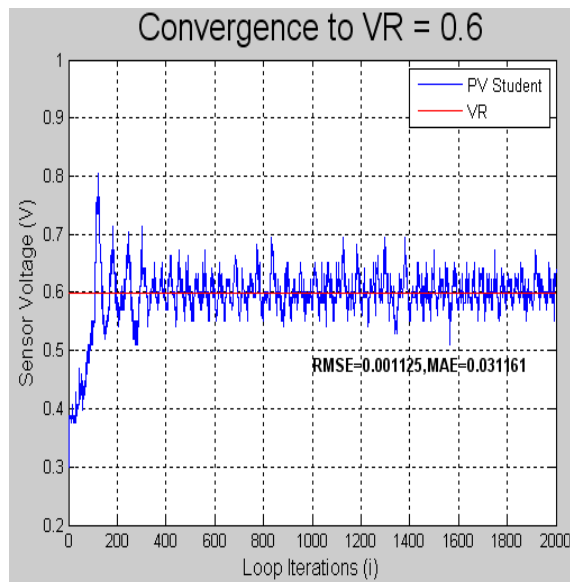
Figures 3.5 and 3.6 show the convergence results of $VR = [0.4V, 0.8V]$ for the Three-Rule and the Five-Rule FLCs respectively. The following results are the real-time convergence response at selected voltage references of the FBS. Given a desired value VR , the system's output $PV Student$ is obtained. The blue signal $PV Student$ is the voltage value corresponding to an altitude of the object. The red signal VR is the desired voltage. Each plot includes the root mean square error (RMSE) value and the mean absolute error (MAE). Both error quantities are statistical measurements that show how far the signal $PV Student$ is from the desired target VR . The closer the RMSE and MAE values are to zero, the better the $PV Student$ approximates the target VR .



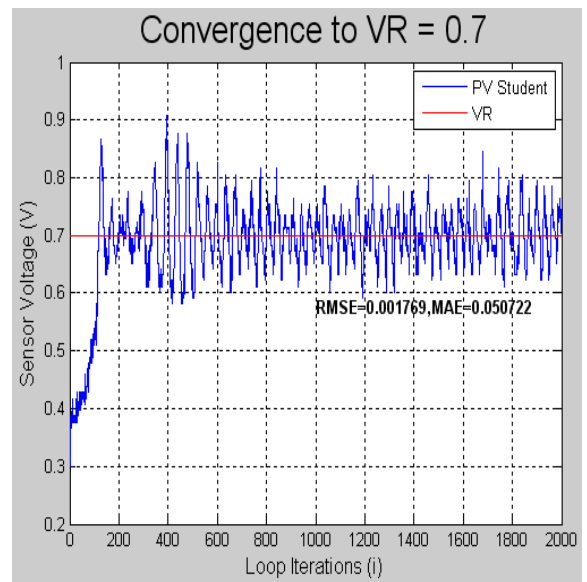
(a)



(b)

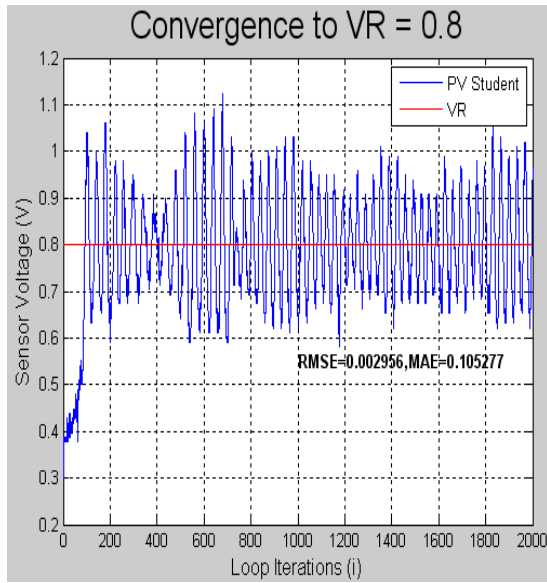


(c)



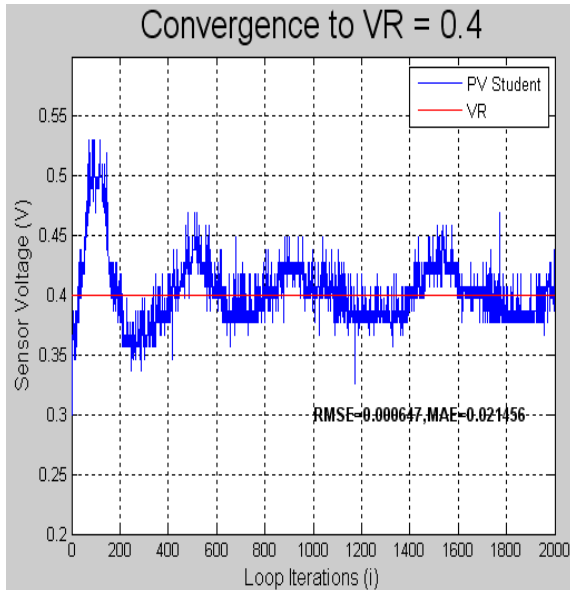
(d)

Figure 3.5: Three-Rule FLC with Modified Error Equation
(a) PV Student Output for $VR = 0.4$; (b) PV Student Output for $VR = 0.5$;
(c) PV Student Output for $VR = 0.6$; (d) PV Student Output for $VR = 0.7$;
(e) PV Student Output for $VR = 0.8$

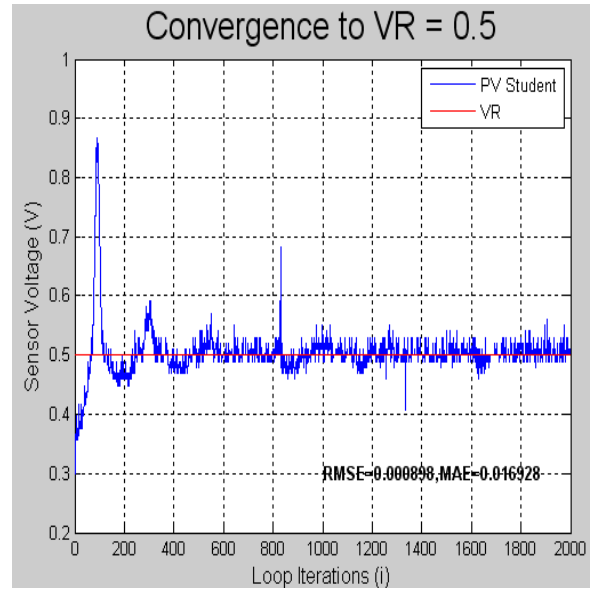


(e)

Figure 3.5 Continued: Three-Rule FLC with Modified Error Equation
 (a) *PV Student* Output for $VR = 0.4$; (b) *PV Student* Output for $VR = 0.5$;
 (c) *PV Student* Output for $VR = 0.6$; (d) *PV Student* Output for $VR = 0.7$;
 (e) *PV Student* Output for $VR = 0.8$

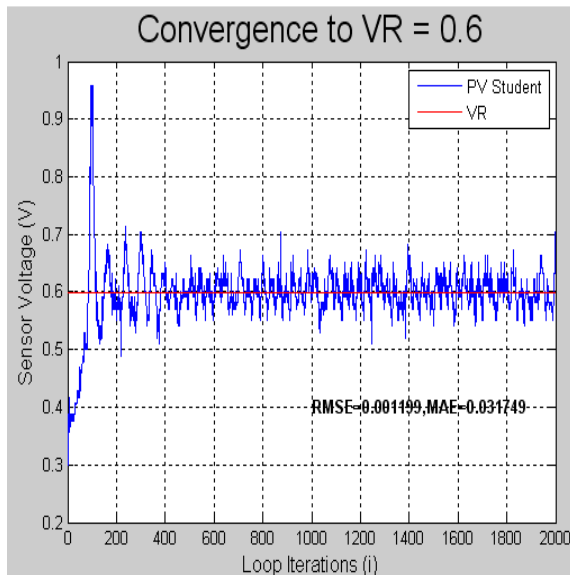


(a)

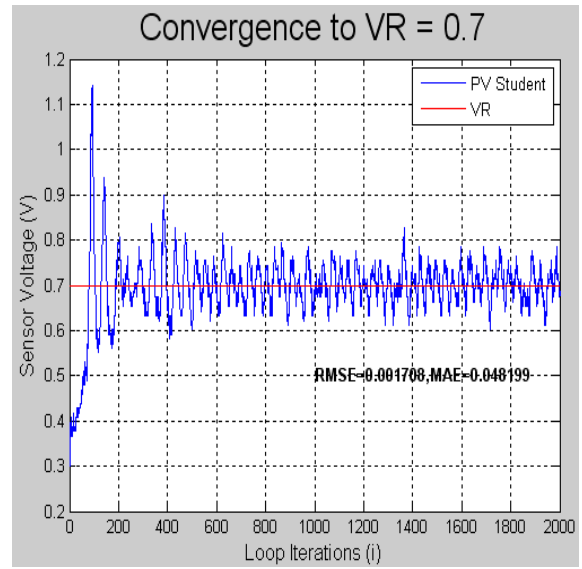


(b)

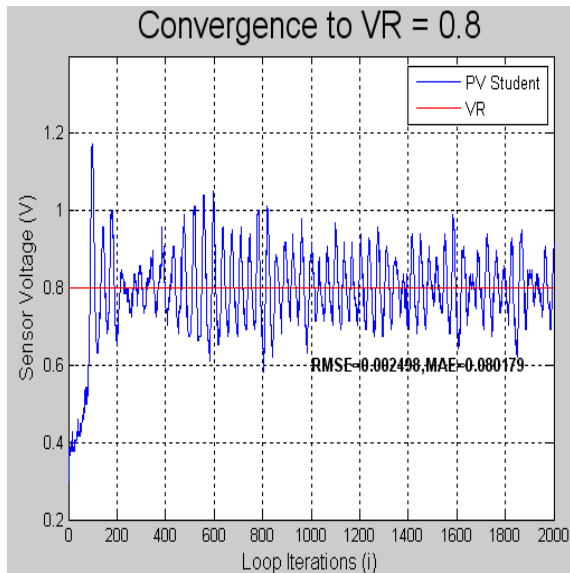
Figure 3.6: Five-Rule FLC with Modified Error Equation
 (a) *PV Student* Output for $VR = 0.4$; (b) *PV Student* Output for $VR = 0.5$;
 (c) *PV Student* Output for $VR = 0.6$; (d) *PV Student* Output for $VR = 0.7$;
 (e) *PV Student* Output for $VR = 0.8$



(c)



(d)



(e)

Figure 3.6 Continued: Five-Rule FLC with Modified Error Equation
 (a) *PV Student* Output for $VR = 0.4$; (b) *PV Student* Output for $VR = 0.5$;
 (c) *PV Student* Output for $VR = 0.6$; (d) *PV Student* Output for $VR = 0.7$;
 (e) *PV Student* Output for $VR = 0.8$

The axes for each plot, as seen in Figures 3.5 and 3.6, are the *Sensor Voltage* for the vertical axis and the *Loop Iterations* for the horizontal axis. The horizontal axis is also the pseudo time axis in seconds. In addition, the *Loop Iterations* are the *number of samples* utilized by the controller at a specific processing speed. The MATLAB commands *tic* and *toc* are used to measure the processing time of the entire *for* loop with 2000 iterations. These processing rate values are presented in the Discussion section below.

Three-Rule FLC Results

The resulting averaged time of the run with 2000 *Loop Iterations* is approximately 127.10 seconds, or 2.12 minutes. This gives us a processing rate (the speed of the calculations, not the sampling rate) of

$$\frac{0.0635 \text{ seconds}}{1 \text{ iteration}} \rightarrow \text{implies} \rightarrow \frac{15.7 \text{ iterations}}{1 \text{ seconds}}$$

or 15.7 Hz. As mentioned before, the number of *Loop Iterations* also corresponds to the number of samples.

Five-Rule FLC Results

The resulting averaged time of the run with 2000 *Loop Iterations* is approximately 123.49 seconds, or 2.06 minutes. This result provides a processing rate (the speed of the calculations, not sampling rate) of

$$\frac{0.0617 \text{ seconds}}{1 \text{ iteration}} \rightarrow \text{implies} \rightarrow \frac{16.20 \text{ iterations}}{1 \text{ second}}$$

or 16.2 Hz.

3.1.2 Discussion:

As seen in Figures 3.5 and 3.6, noise exists in the signal *PV Student*. This noise affects the convergence of the system. The noise causes jumps in the controller action which causes the


system to go strayed as it is converging. Table 3.1 shows the RMSE and MAE results for $VR = [0.4, 1.9]$ at intervals of 0.1 volts.

Table 3.1: Comparison between Three-Rule and Five-Rule FLC

	Fuzzy Logic Controller – 3 Rule		Fuzzy Logic Controller – 5 Rule	
	RMSE	MAE	RMSE	MAE
0.4	0.000415*	0.014706	0.000647	0.021456
0.5	0.000622	0.015521	0.000898	0.016928
0.6	0.001125	0.031161	0.001199	0.031749
0.7	0.001769	0.050722	0.001708	0.048199
0.8	0.002956	0.105277	0.002498	0.080179
0.9	0.004528	0.171674	0.005085	0.192966
1.0	0.004798	0.173744	0.005376	0.204738
1.1	0.005482	0.201173	0.006528	0.252785
1.2	0.006332	0.226285	0.007135	0.269302
1.3	0.006692	0.227232	0.007287	0.264221
1.4	0.006845	0.224342	0.005900	0.196741
1.5	0.007963	0.264484	0.006443	0.205780
1.6	0.008367	0.282379	0.011062	0.369162
1.7	0.009604	0.335697	0.014591	0.523406

Table 3.1 Continued: Comparison between Three-Rule and Five-Rule FLC

1.8	0.010154	0.354730	0.013579	0.492669
1.9	0.010950	0.388547	Brim	Brim
2.0	Brim*	Brim	N/T	N/T
2.1	N/T*	N/T	N/T	N/T
2.2	N/T	N/T	N/T	N/T
2.3	N/T	N/T	N/T	N/T
2.4	N/T	N/T	N/T	N/T
2.5	N/T	N/T	N/T	N/T
2.6	N/T	N/T	N/T	N/T
2.7	N/T	N/T	N/T	N/T
2.8	N/T	N/T	N/T	N/T
2.9	N/T	N/T	N/T	N/T
3.0	N/T	N/T	N/T	N/T

*  = Better than its counterpart FLC

*N/T = Not Tested; Brim = Object reached brim limit of tube

Three-Rule and Five-Rule FLC Discussion

Table 3.1 demonstrates that $VR = 0.4$ has the best convergence because the blue signal is closest to the target value VR , indicated by both the smallest RMSE and MAE values. According to the table, $VR = [0.4, 0.6]$, $VR = [0.9, 1.3]$, and $VR = [1.6, 1.8]$ are the voltage references with good convergence results for the Three-Rule FLC.

Comparing to the Three-Rule FLC, Table 3.1 shows that the Five-Rule FLC performed better at $VR = 0.7, 0.8, 1.4$ and 1.5 since the RMSE and MAE are smaller in value.

Based on the table is concluded that for both FLCs, the $VR = 0.4, 0.5, 0.6$ and 0.7 are the voltage references with good convergences because the MAEs are the lowest. As VR gets bigger, the signal *PV Student* gets worse as it converges to its target VR for both the Three-Rule and the Five-Rule FLCs.

CHAPTER 4: FUZZY LOGIC CONTROLLER WITH FILTER

Essentially, this thesis is trying to solve the main problem of noise from the sensor. Due to the noisy output signal of the sensor, the decision is to use a hardware or software filter. A hardware filter is implemented by placing a bypass capacitor very close to the sensor at the power and ground leads. Because this type of filtering does not achieve much positive results, a software filter is considered.

The requirements of the software filter are

1. Time-domain filtering,
2. Perform real-time filtering, and
3. Achieve smooth filtered signal.

Due to the above requirements, the Mean filter, the Fast Fourier Transform (FFT), and the Median filter are considered as the filtering options. The simulation results of both the Mean and Median filters are analyzed before moving into the real implementation. The simulation involves filtering the data which has already been acquired. Based on the simulation performance of the filter, the decision is made as to which filter should be used in the real implementation. Finally, the Median filter is chosen because of better results in simulation and real-time performance.

All the filters covered in this chapter are tested first in simulation by using previously saved data. Afterwards, the chosen filter is applied to real-time implementation.

4.1 Fast Fourier Transform:

The Fast Fourier Transform (FFT) only meets the first requirement. Because the FFT calculation is usually applied to offline data, that is, data that already has been acquired, there are

not many real-time applications. In addition, the frequency domain is needed to implement the FFT. Therefore, the next filter option is the Mean filter.

4.2 Mean Filter:

The Mean filter meets all of the three requirements for a software filter. For this filter, two major steps are needed in order to proceed to using this filter. First, the window size of the filter is selected and second, the different types of Mean filter such as the Rectangular (Regular) and Triangular (Weighted) Mean filters need to be tested to compare performance.

The Mean (average) filter is the average of data values considered according to the window size. The mean is defined as the summation of the data divided by the number of data values. The values taken into consideration are the past, present, and future values of the data.

One issue that emerges from the window size is delay. For example, if the window size of the Mean filter is 3, then $PV(i-1)$, $PV(i)$, and $PV(i+1)$ are used. This window size introduces a delay of 1 *Loop Iteration* or *sample*. If the window size is 5, then $PV(i-2)$, $PV(i-1)$, $PV(i)$, $PV(i+1)$, and $PV(i+2)$ are needed and a delay of 2 *samples* is introduced. The same pattern of window size to length of delay is applied to a window size of 7 referring to a delay of 3, and so forth. The time delay is dependent on the window size and the processing speed of MATLAB.

4.2.1 Rectangular Mean Filter:

The Rectangular Mean filter is defined as $h_{M3} = [1 \ 1 \ 1]$ for a window size of 3. This window size of 3 has a delay of 1 *sample* because the system waits for one additional value before processing the complete data. The Rectangular Mean filter for a window size of 5 is $h_{M5} = [1 \ 1 \ 1 \ 1 \ 1]$. This window size of 5 has a delay of 2 *Loop Iterations* or *samples* because the system waits for two future values.

This section presents three cases that express the Mean filter's operation on the noisy signal. These cases expose the weakness of the Mean filter which is the inability of the Mean filter to remove bigger noise spikes in the signal. This drawback is due to the lack of noise detection by the Mean filter. The work of developing noise detection rules for the Mean filter is left as additional work that is not covered within the scope of this thesis, but is mentioned in the Future Work, Section 9.2 of Chapter 9. In the following cases, Cases #1, #2, and #3 are developed around their desired target VR and a noise PV is defined as a value outside the desired target VR by at least $1V$.

Case#1: The noise in the 1st sample read

Target is $VR = 0.4V$ and noise is $PV = 0.3V$.

The data are $PV(k-1) = 0.3V$, $PV(k) = 0.4V$, and $PV(k+1) = 0.41V$, where $PV(k-1)$, $PV(k)$, and $PV(k+1)$ are the first, second, and third sample read, respectively. The equation of the Mean filter with a window size of 3 is defined as

$$PV_{filter}(k) = \frac{1 \times PV(k-1) + 1 \times PV(k) + 1 \times PV(k+1)}{1+1+1} \quad (4.1)$$

Equation (4.1) gives

$$PV_{filter}(k) = \frac{1 \times 0.3 + 1 \times 0.4 + 1 \times 0.41}{1+1+1} = 0.37V.$$

Since the Mean filter equation (4.1) is set to the present value k , that is the second value of the data, $PV(k) = 0.4V$ is replaced by $PV_{filter}(k) = 0.37V$. The $PV(k)$ value is replaced even though is not the noisy value.

Case#2: The noise in the 2nd sample read

Target is $VR = 0.5V$ and noise is $PV = 0.4V$.

The data are $PV(k-1) = 0.5V$, $PV(k) = 0.4V$, and $PV(k+1) = 0.49V$.

Equation (4.1) gives

$$PV_{filter}(k) = \frac{1 \times PV(k-1) + 1 \times PV(k) + 1 \times PV(k+1)}{1+1+1} = \frac{1 \times 0.5 + 1 \times 0.4 + 1 \times 0.49}{1+1+1} = 0.4633V.$$

$PV(k) = 0.4V$ is replaced by $PV_{filter}(k) = 0.4633V$. In this case, the noisy value is removed by replacing the 2nd value by the filtered value.

Case#3: The noise in the 3rd sample read

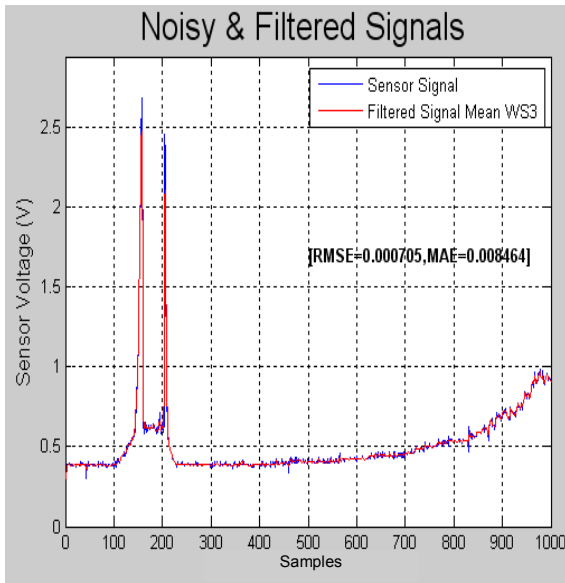
Target is $VR = 0.8V$ and noise is $PV = 0.9V$.

The data are $PV(k-1) = 0.8V$, $PV(k) = 0.81V$, and $PV(k+1) = 0.9V$.

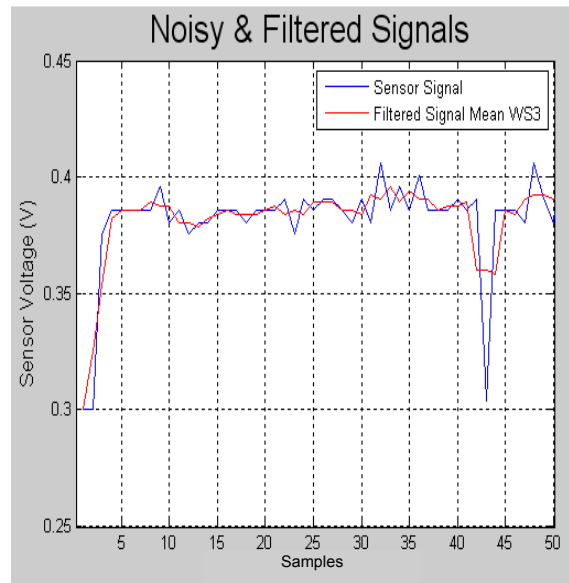
$$PV_{filter}(k) = \frac{1 \times PV(k-1) + 1 \times PV(k) + 1 \times PV(k+1)}{1+1+1} = \frac{1 \times 0.8 + 1 \times 0.81 + 1 \times 0.9}{1+1+1} = 0.8367V$$

$PV(k) = 0.81V$ is replaced by $PV_{filter}(k) = 0.8367V$ even though the $PV(k)$ value is not the noisy value. The three cases presented above show that the Rectangle Mean filter does not have noise detection ability. This inability to detect noise is seen when the 2nd value is replaced even though it is not the noisy value. This effect occurs in Cases #1 and #3.

The simulation of the Rectangular Mean filter is presented in Figure 4.1 and 4.2. The *Sensor Signal* used for these results is data that was acquired previously. First, the plot of the noisy signal with the Mean filter with a window size of 3 is presented and then the plot of the zoomed-in Noisy & Filtered Signal for a better look at the *Filtered Signal*. In Figure 4.1 and 4.2, the left plot is a view of the entire data and the right plot is a view of the low voltage data.

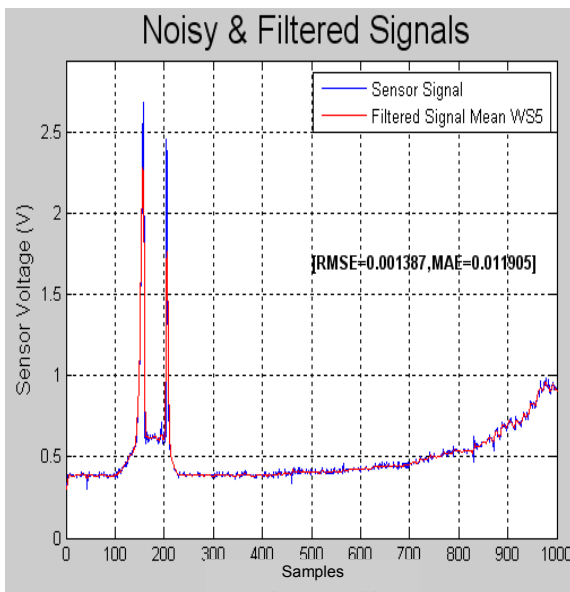


(a)

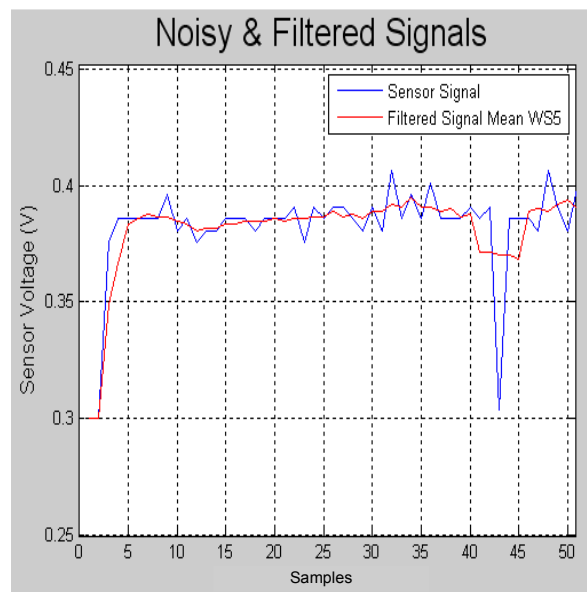


(b)

Figure 4.1: Rectangular Mean Filter of Window Size 3
(a) Sensor signal Output and Filtered Signal;
(b) Sensor signal Output and Filtered Signal (Zoomed-in)



(a)



(b)

Figure 4.2: Rectangular Mean Filter of Window Size 5
(a) Sensor signal Output and Filtered Signal;
(b) Sensor signal Output and Filtered Signal (Zoomed-in)

Mean filter of window size 3 is the best because the high voltage values are not truncated as much. Also, noises are filtered out at the lower voltage levels of [0.3V, 1.0V] as given by the close to zero RMSE = 0.000705 and MAE = 0.008464 values. Although the *Filtered Signal* of the Mean filter of a window size 5 is better at lower voltages, the major problem is that the meaningful signal is truncated at higher voltages. In addition, the RMSE = 0.001387 and MAE = 0.011905 of the filter of window size 5 are bigger than those of the filter of window size 3, meaning there is more filtration happening with the filter of window size 5. Greater truncation effects are seen at higher voltage levels of [1.5V, 3.0V] for larger window sizes.

According to the simulation results of the Rectangular Mean filter in Figure 4.1 and 4.2, filters with higher window sizes have better filtration at the lower voltage levels. Even though this is the case at lower voltage levels, overall these filters negatively affect the system's performance in real-time due to delay.

4.2.2 Triangular Mean Filter:

The Triangular or Weighted Mean filter is another type of Mean filter. In this thesis, the Triangular and the Weighted Mean filters are defined as follows. The Triangular Mean filter does not assign weight values of 1 to the data being considered, but rather different weights are assigned. For example, when using the Rectangular (Regular) Mean filter, the coefficients of value 1 are assigned to the data in Equation (4.1), i.e.,

$$\frac{1 \times PV(i-1) + 1 \times PV(i) + 1 \times PV(i+1)}{1+1+1},$$

but for the Triangular or Weighted Mean filter, weights not equal to 1 can be assigned, i.e.,

$$\frac{w_1 \times PV(i-1) + w_2 \times PV(i) + w_3 \times PV(i+1)}{w_1 + w_2 + w_3}.$$

One case of the Weighted Mean filter is the Triangular Mean filter where two of the weights are equal and the other is greater, thus making a triangle. For example, a Triangular Mean filter is

$$\frac{0.75 \times PV(i-1) + 1 \times PV(i) + 0.75 \times PV(i+1)}{0.75 + 1 + 0.75}$$

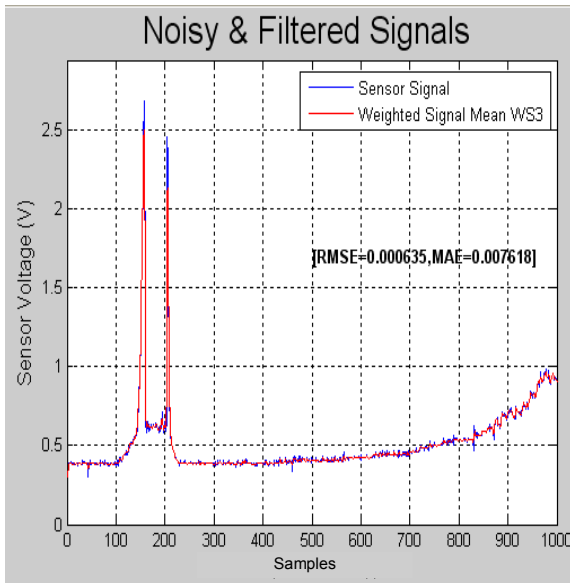
and a general Weighted Mean filter is

$$\frac{4 \times PV(i-1) + 6 \times PV(i) + 0 \times PV(i+1)}{4 + 6 + 0}.$$

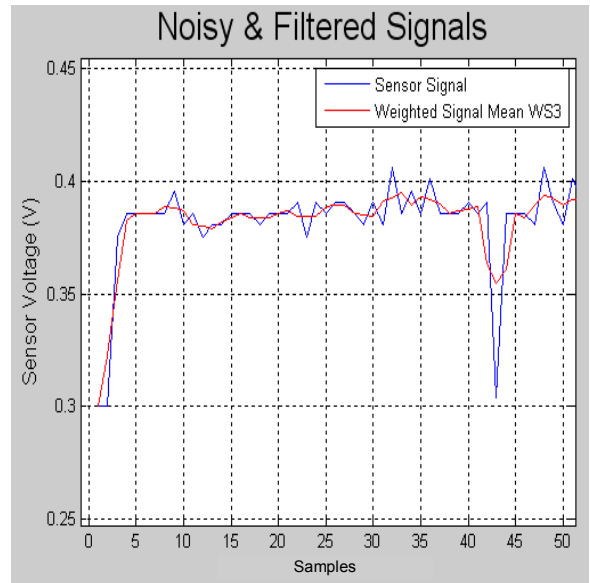
For both the simulations of the Triangular and Weighted Mean filter, these two examples are utilized for filter comparison.

The Triangular and the Weighted Mean filter examples, mentioned previously, are defined as $h_T = [0.75 \ 1 \ 0.75]$ and $h_W = [4 \ 6 \ 0]$ respectively, both with a window size of 3. In this thesis, window sizes of 3 (implying a delay of 1 *sample*) are considered. The reason is because having more than 1 *sample* delay affects the system's update speed.

The results of the Triangular Mean filter and of the Weighted Mean filter are shown in Figure 4.3 and in Figure 4.4 respectively. As mentioned, both filters are not applied yet to real-time data.

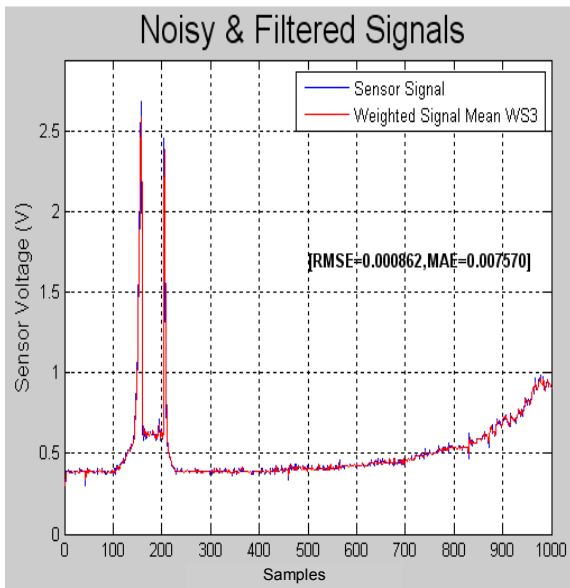


(a)

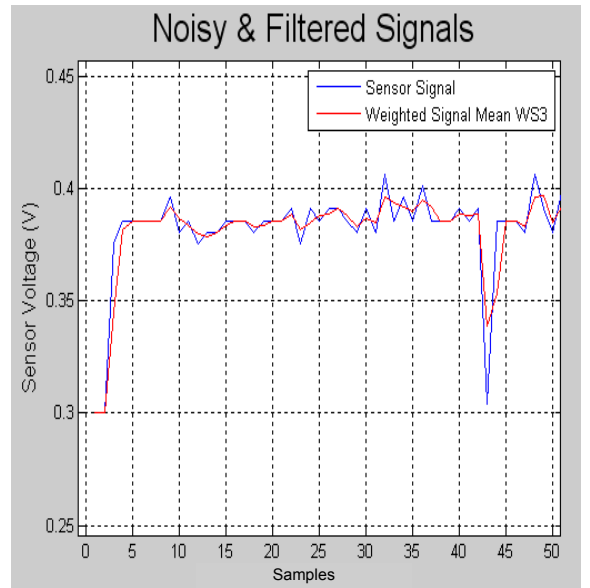


(b)

Figure 4.3: Triangular Mean Filter of Window Size 3
(a) Sensor signal Output and Filtered Signal;
(b) Sensor signal Output and Filtered Signal (Zoomed-in)



(a)



(b)

Figure 4.4: Weighted Mean Filter of Window Size 3
(a) Sensor signal Output and Filtered Signal;
(b) Sensor signal Output and Filtered Signal (Zoomed-in)

By observation of the plots of Figure 4.3 and 4.4, the Triangular Mean filter is better than the Weighted Mean filter because at low voltages the filter removes the noise more effectively.

Table 4.1: Comparison between Types of Mean Filter

Mean Filter WS3	RMSE	MAE	Peak Truncated	Filtration at low voltages	Delay
Rectangular	0.000705	0.008464	Some	Best	1
Triangle	0.000635	0.007618	Not much	Good	1
Weighted	0.000862	0.007570	A little	Not good	1

According to Table 4.1, the best filter is the Rectangular (Regular) Mean filter because it has the 2nd highest RMSE and 1st highest MAE. In addition, the Rectangular Mean filter has “Best” filtration at low voltages and “Some” of the voltage peaks are truncated.

The RMSE and MAE are interpreted differently when analyzing the performance of the filters. For instance, the higher the values of the RMSE and MAE are, the smoother the filtered signal is. The reason is because the filter reduces the noise and thus the filter signal looks more different than the noisy signal. The nonzero statistical measurements indicate that the filtered signal is different than the noisy signal. In the previous use of the RMSE and MAE observed in Chapter 3, the close to zero values of RMSE and MAE means that the *PV* signal is more similar to the target position voltage *VR*.

The Mean filter smoothes out the noisy signal, but higher levels of noise are not completely removed. Although the Mean filter meets all three of the requirements, it introduces the problem of delay. Due to the need of applying this filter in real-time, the effect of delay, where the length of the delay is defined by the window size, is a critical drawback. Since data is obtained online, there is a need to wait for future values to become available in order for the

filter to proceed with the calculation. Another problem is that the Mean filter truncates the data of the signal even if data is not the noise. One solution to this problem is to increase the sampling rate hence making more values available within a time frame.

4.3 Median Filter:

In this section, the results of the Median filter are analyzed. For the Median filter, the window sizes of 3 and 5 are compared. This filter also meets all of the three requirements. Like the Mean filter, this filter also has the problem of delay and the truncation of meaningful data.

The median does not necessarily consider the magnitude of each value of the data; instead it takes into account the number of values. The median of a set of data is defined as the middle number (where a middle number can be found) or the average of the two middle numbers (when no single middle number can be found) in a data where the values are arranged by magnitude. In other words, the median is the middle number of an odd amount of sorted numbers, and is also the average of the two middle numbers in an even amount of sorted numbers.

In the same manner as for the Rectangular Mean Filter, the three cases are presented for the Median Filter. The cases form around their desired target and a noise is a value outside the target by at least 1V. The two-step algorithm of the median calculation is, first, sorts the numbers and, second, finds the median. The sorting of the numbers is the arranging of the numbers in increasing order.

Case#1: The noise in the 1st sample read

Target is $VR = 0.4V$ and noise is $PV = 0.3V$.

The data are $PV(k-1) = 0.3V$, $PV(k) = 0.4V$, and $PV(k+1) = 0.41V$, where $PV(k-1)$, $PV(k)$, and $PV(k+1)$ are the first, second, and third sample read, respectively.

Median Algorithm:

1. Ascending order:

$$PV(k-1) = 0.3V, PV(k) = 0.4V, PV(k+1) = 0.41V$$

2. Find the median

$$PV(k) = 0.4V$$

Since, the Median filter is set to the present value k , $PV(k) = 0.4V$ is replaced by $PV_{filter}(k) = 0.4V$.

Case#2: The noise in the 2nd sample read

Target is $VR = 0.5V$ and noise is $PV = 0.4V$.

The data are $PV(k-1) = 0.5V$, $PV(k) = 0.4V$, and $PV(k+1) = 0.49V$.

Median Algorithm:

1. Ascending order:

$$PV(k) = 0.4V, PV(k+1) = 0.49V, PV(k-1) = 0.5V$$

2. Find the median

$$PV(k+1) = 0.49V$$

$PV(k) = 0.4V$ is replaced by $PV_{filter}(k) = 0.49V = PV(k+1)$.

Case#3: The noise in the 3rd sample read

Target is $VR = 0.8V$ and noise is $PV = 0.9V$.

The data are $PV(k-1) = 0.8V$, $PV(k) = 0.81V$, and $PV(k+1) = 0.9V$.

Median Algorithm:

1. Ascending order:

$$PV(k-1) = 0.8V, PV(k) = 0.81V, PV(k+1) = 0.9V$$

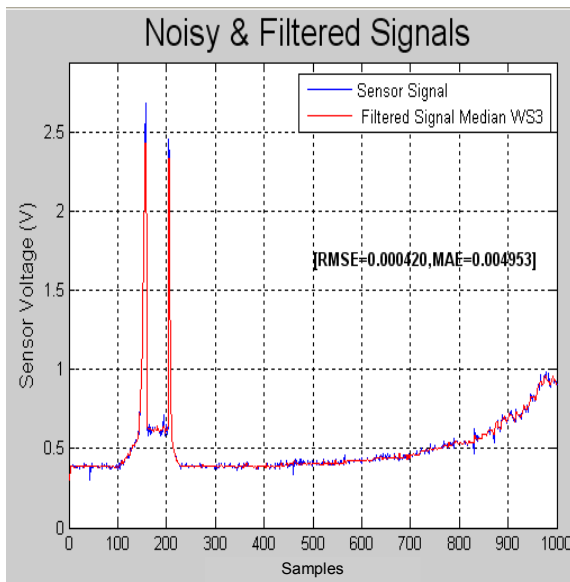
2. Find the median

$$PV(k) = 0.81V$$

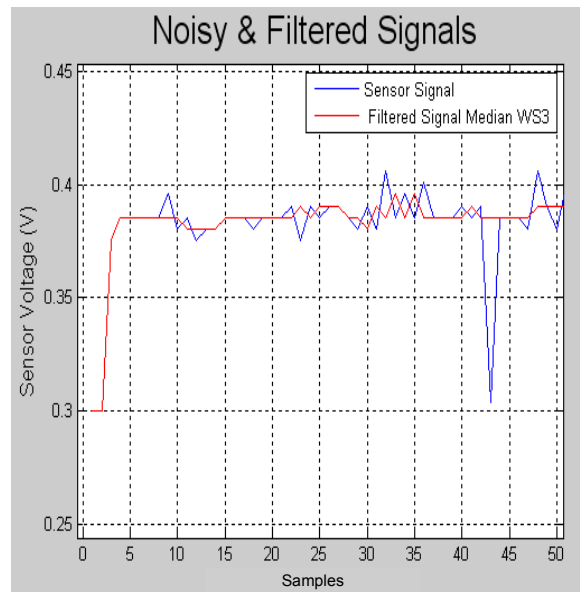
$PV(k) = 0.81V$ is replaced by $PV_{filter}(k) = 0.81V$.

The cases presented above show that the Median filter does not have noise detection ability. This is seen in Cases #1 and #3 where the noise value is not removed.

The simulation aspect of the Median filter is presented in Figure 4.5 and 4.6. Part (a) of both Figure 4.5 and 4.6 shows the plot of the noisy signal with the Median filter of a window size of 3 and 5 respectively. Part (b) of these same figures presents the plot of the Noisy & Filtered Signal in a zoomed-in view for a closer look. In addition, part (a) of the figures presents the entire data and part (b) plots the low voltage levels of the data.

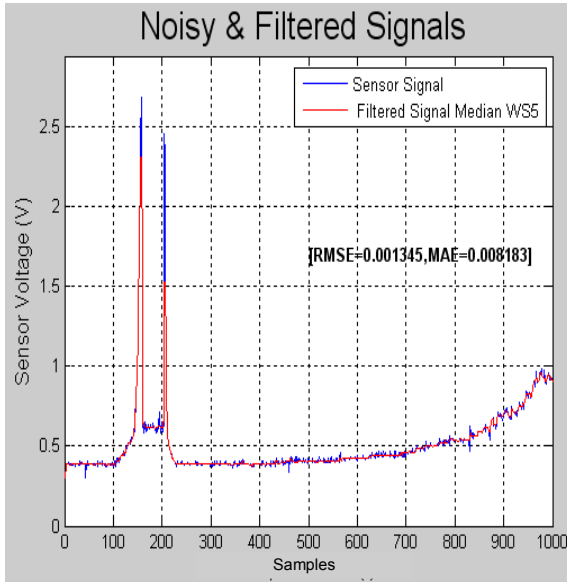


(a)

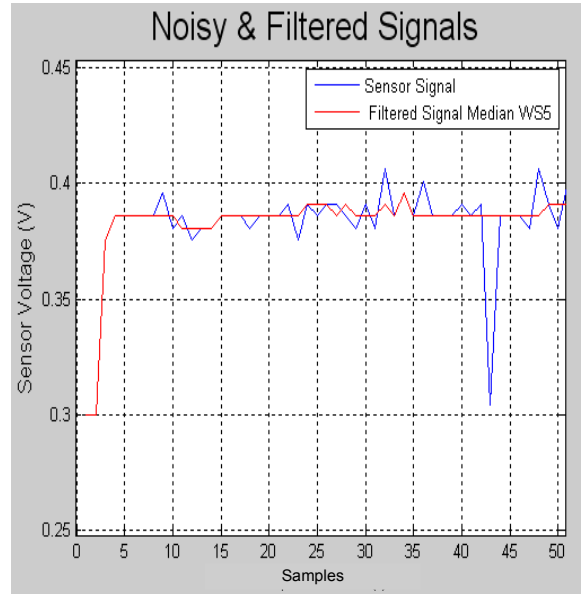


(b)

Figure 4.5: Median Filter of Window Size 3
(a) Sensor Signal Output and Filtered Signal;
(b) Sensor Signal Output and Filtered Signal (Zoomed-in)



(a)



(b)

Figure 4.6: Median Filter of Window Size 5
(a) Sensor Signal Output and Filtered Signal;
(b) Sensor Signal Output and Filtered Signal (Zoomed-in)

Table 4.2: Comparison between Median Filters of Window Size 3 and 5

Median Filter WS	RMSE	MAE	Peak Truncation	Low voltage Filtration	Delay
3	0.000420	0.004953	Some	Good	1
5	0.001345	0.008183	A lot	Best	2

As mentioned before, a delay of 2 Loop Iterations (2 samples) can cause the system's performance to be affected negatively even though the filtering is better. For this reason the Median filter with window size of 3 is chosen. In addition, the Media filter with a window size of 5 causes the peak values (meaningful data) to be truncated significantly as seen in Figure 4.6 (a).

Table 4.3: Comparison between Mean and Median Filters

Filter WS 3	RMSE	MAE	Peak Truncation	Low voltage Filtration	Delay
Rectangular Mean	0.000705	0.008464	A lot	Fair	1
Median	0.000420	0.004953	A little	Best	1

Based on Table 4.3, the Median filter is the top choice for real-time implementation.

4.4 Median Filter Implementation in Real-Time

The real-time application can now be performed since the Median filter has the best performance in simulation. The Median filter is applied to the FLC and better performance is seen compared to previous work. In order to implement the filter to the FLC, the MATLAB script is developed. Appendix II has the entire MATLAB script for the Three-Rule and Five-Rule of the FLC. Figure 4.7 shows a portion of the MATLAB script that defines the fuzzy regions of the Three-Rule FLC.

```
Err_X = (-2:0.01:2);  
N = trapmf(Err_X,[-2 -2 -0.01 0]);  
Z = trimf(Err_X,[-0.01 0 0.01]);  
P = trapmf(Err_X,[0 0.01 2 2]);
```

Figure 4.7: Script of Three-Regions of FLC

The following script in Figure 4.8 defines the fuzzy regions of the Five-Rule FLC.

```

Err_X = (-2:0.01:2);
VN = trapmf(Err_X,[-2 -2 -0.5 -0.01]);
N = trapmf(Err_X,[-0.5 -0.01 0]);
Z = trimf(Err_X,[-0.01 0 0.01]);
P = trapmf(Err_X,[0 0.01 0.5]);
VP = trapmf(Err_X,[0.01 0.5 2 2]);

```

Figure 4.8: Script of Five-Regions of FLC

The performance of the filter is tested at each *VR* value in the same way it was done for the FLC in Section 3.1.1 of Chapter 3. The Three-Rule FLC results are shown in Figure 4.9 and the Five-Rule FLC in Figure 4.10 for selected *VR* values.

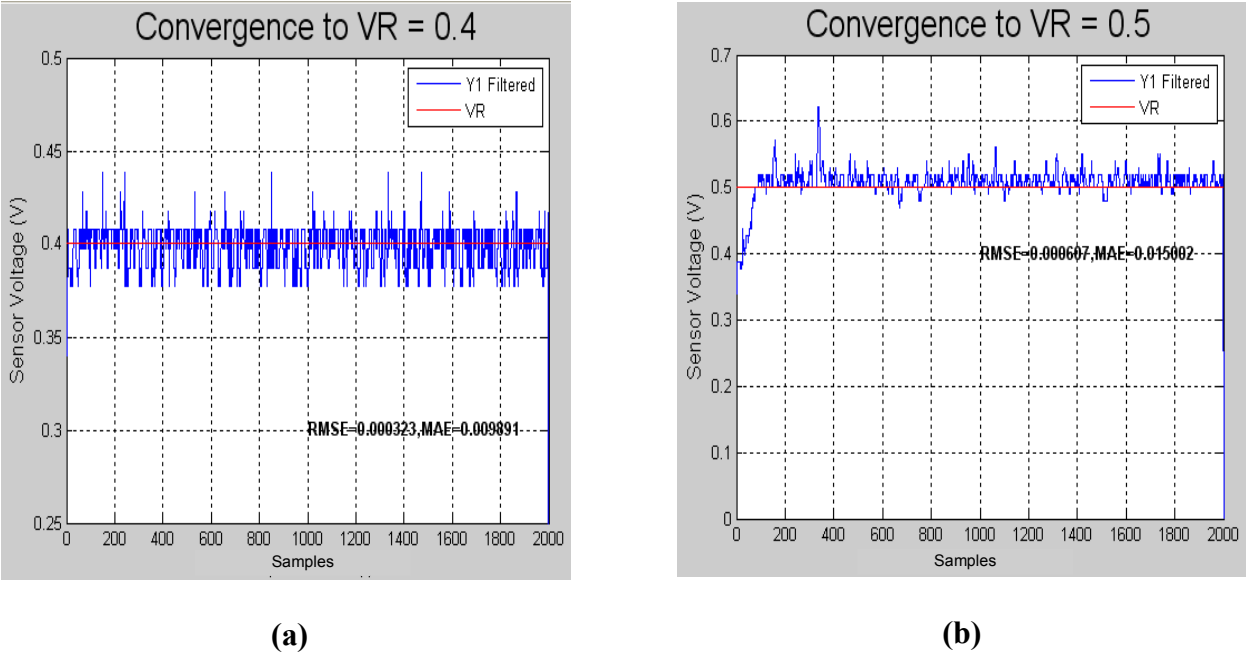
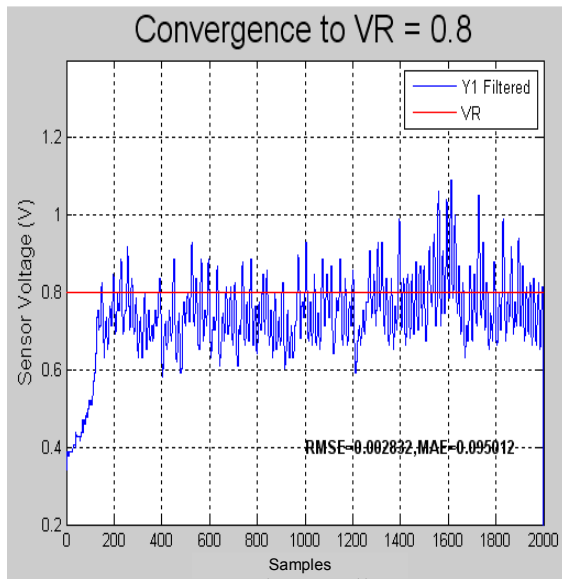
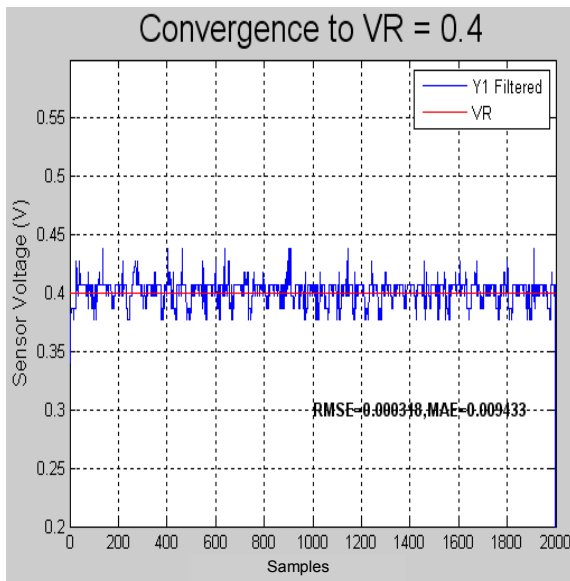


Figure 4.9: Three-Rule FLC with Median Filter of Window Size 3
(a) Y1 Filtered Output for VR = 0.4; (b) Y1 Filtered Output for VR = 0.5;
(c) Y1 Filtered Output for VR = 0.8

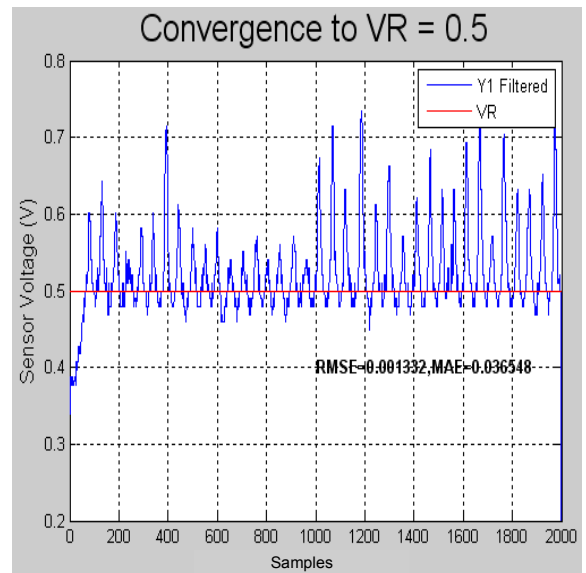


(c)

Figure 4.9 Continued: Three-Rule FLC with Median Filter of Window Size 3
 (a) *Y1 Filtered Output for VR = 0.4*; (b) *Y1 Filtered Output for VR = 0.5*;
 (c) *Y1 Filtered Output for VR = 0.8*

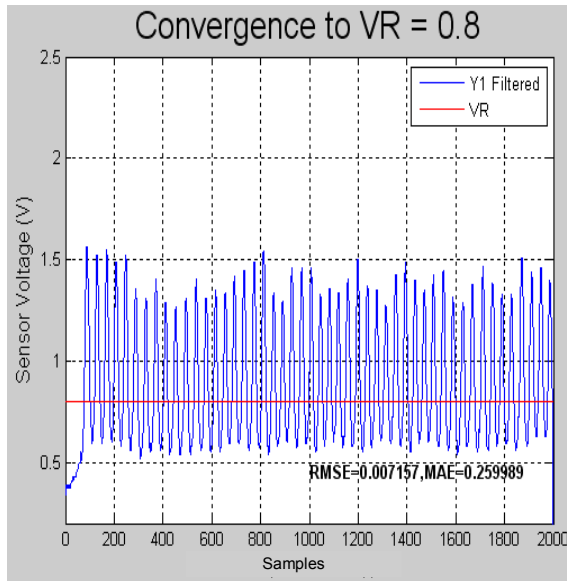


(a)



(b)

Figure 4.10: Five-Rule FLC with Median Filter of Window Size 3
 (a) *Y1 Filtered Output for VR = 0.4*; (b) *Y1 Filtered Output for VR = 0.5*;
 (c) *Y1 Filtered Output for VR = 0.8*



(c)

Figure 4.10 Continued: Five-Rule FLC with Median Filter of Window Size 3
(a) *Y1 Filtered* Output for $VR = 0.4$; (b) *Y1 Filtered* Output for $VR = 0.5$;
(c) *Y1 Filtered* Output for $VR = 0.8$

4.5 Discussion of FLC and FLC with Filter

The MATLAB script that generates the results in Section 4.4 uses the Modified Error Equation (3.2) presented in Section 3.1. Table 4.4 compares the RMSE and MAE results of the FLC in the Previous Work section to that of the FLC with the filter. The Three-Rule and the Five-Rule FLCs with and without the filter are presented.

Table 4.4: Comparison between FLC and FLC with Filter


VR	Student 3 Rule FL		3 Rule FL Code w/ Filter		Student 5 Rule FL		5 Rule FL Code w/ Filter	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
0.4	0.000415	0.014706	0.000323*	0.009891	0.000647	0.021456	0.000318	0.009433
0.5	0.000622	0.015521	0.000607	0.015002	0.000898	0.016928	0.001332	0.036548

Table 4.4 Continued: Comparison between FLC and FLC with Filter

0.6	0.001125	0.031161	0.001294	0.034380	0.001199	0.031749	0.003482	0.122760
0.7	0.001769	0.050722	0.001936	0.05638	0.001708	0.048199	0.005962	0.206979
0.8	0.002956	0.105277	0.002832	0.095012	0.002498	0.080179	0.007157	0.259989
0.9	0.004528	0.171674	0.003465	0.115926	0.005085	0.192966	0.009175	0.339118
1.0	0.004798	0.173744	0.004209	0.142597	0.005376	0.204730	0.009573	0.361224
1.1	0.005482	0.201173	0.004841	0.165382	0.006528	0.252785	0.013909	0.495304
1.2	0.006332	0.226285	0.006322	0.230536	0.007135	0.269302	0.018239	0.642877
1.3	0.006692	0.227232	0.006090	0.198605	0.007287	0.264221	0.016373	0.585464
1.4	0.006845	0.22432	0.007344	0.253678	0.005900	0.196741	0.014980	0.554180
1.5	0.007963	0.264484	0.007204	0.237559	0.006443	0.205780	0.015278	0.573310
1.6	0.008367	0.282379	0.007144	0.213981	0.011062	0.369162	0.014379	0.542519
1.7	0.009604	0.335697	0.011059	0.345101	0.014591	0.523406	Brim	Brim
1.8	0.010154	0.354738	0.008849	0.281623	0.013579	0.492669	N/T	N/T
1.9	0.010950	0.388547	0.009994	0.317909	Brim	Brim	N/T	N/T
2.0	Brim*	Brim	0.011087	0.362689	N/T	N/T	N/T	N/T
2.1	N/T*	N/T	0.010956	0.348081	N/T	N/T	N/T	N/T
2.2	N/T	N/T	0.012191	0.412196	N/T	N/T	N/T	N/T
2.3	N/T	N/T	0.013472	0.461724	N/T	N/T	N/T	N/T
2.4	N/T	N/T	0.014690	0.501879	N/T	N/T	N/T	N/T
2.5	N/T	N/T	0.015406	0.520148	N/T	N/T	N/T	N/T
2.6	N/T	N/T	Brim	Brim	N/T	N/T	N/T	N/T

Table 4.4 Continued: Comparison between FLC and FLC with Filter

2.7	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T
2.8	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T
2.9	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T
3.0	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T

*  = Better than its counterpart Student FL

*N/T = Not Tested; Brim = Object reached brim limit of tube

Table 4.4 shows that the best filtering results occur with the Three-Rule FLC. For $VR = 0.4, 0.5, 0.8-1.3, 1.5-2.5$ the Three-Rule FLC with the filter performs better than the Three-Rule FLC without the filter. The Five-Rule FLC with filter only improves at $VR = 0.4$ when compared to the Five-Rule FLC without filter.

CHAPTER 5: DYNAMIC MODEL BY DATA COLLECTION

Modeling of the FBS is obtained in this chapter. First, an attempt to get a direct relationship between fan voltage (V) and the object's altitude (in) is performed. This process is covered in the Static Modeling section, Section 5.1.1. Second, dynamic modeling is performed using the data collected of the FBS by a previous researcher. The objective of the dynamic modeling is to linearize the nonlinear FBS.

5.1 Model Development

5.1.1 Static Modeling

The static modeling, or also referred to as the steady state modeling, involves the use of a fan voltage as the input and the measurement of the object's height as the output. A voltage is sent to the motor, the fan rotates at a certain revolutions per minute speed, the air flows to the object, and a height measurement in inches is obtained from the floating object. The following assumption is made at the outset, without yet having analyzed the observations or data from the floating ball plant.

Assumption #1: Given a fan voltage as the input, a unique height measurement is obtained as the steady state output. This relationship means that a one-to-one function exists between the voltage and the height.

One-to-One function: A one-to-one function is a function f which takes values from the set u to values of the set PV , where the set u contains all the inputs and the set PV contains all the outputs. If u_1 and u_2 are voltage inputs and $f(u_1) = f(u_2)$, then $u_1 = u_2$. The converse is also true. If $f(u_1) \neq f(u_2)$, then $u_1 \neq u_2$.

In order to test Assumption #1, an experiment is set up. The experimental set-up includes the use of a power supply connected directly to the fan, a measurement tape to measure the

height of the ball, and a stopwatch to measure the time of the final height value also known as the settling time. Using a power supply, the voltage operating range of the fan motor is found to be [11V, 12.6V]. This range of voltages produces the altitude position outputs shown in Figure 5.2. By running this experiment, Assumption#1 is tested.

The experiment tests conducted includes different types of floating objects. The types of floating objects used are half of an eggshell and circular black, yellow, and blue foam objects, where the objects have different weights. In addition, the tests are performed in different lengths of tubes. Figure 5.1 depicts the general experiment set-up used to run the tests.

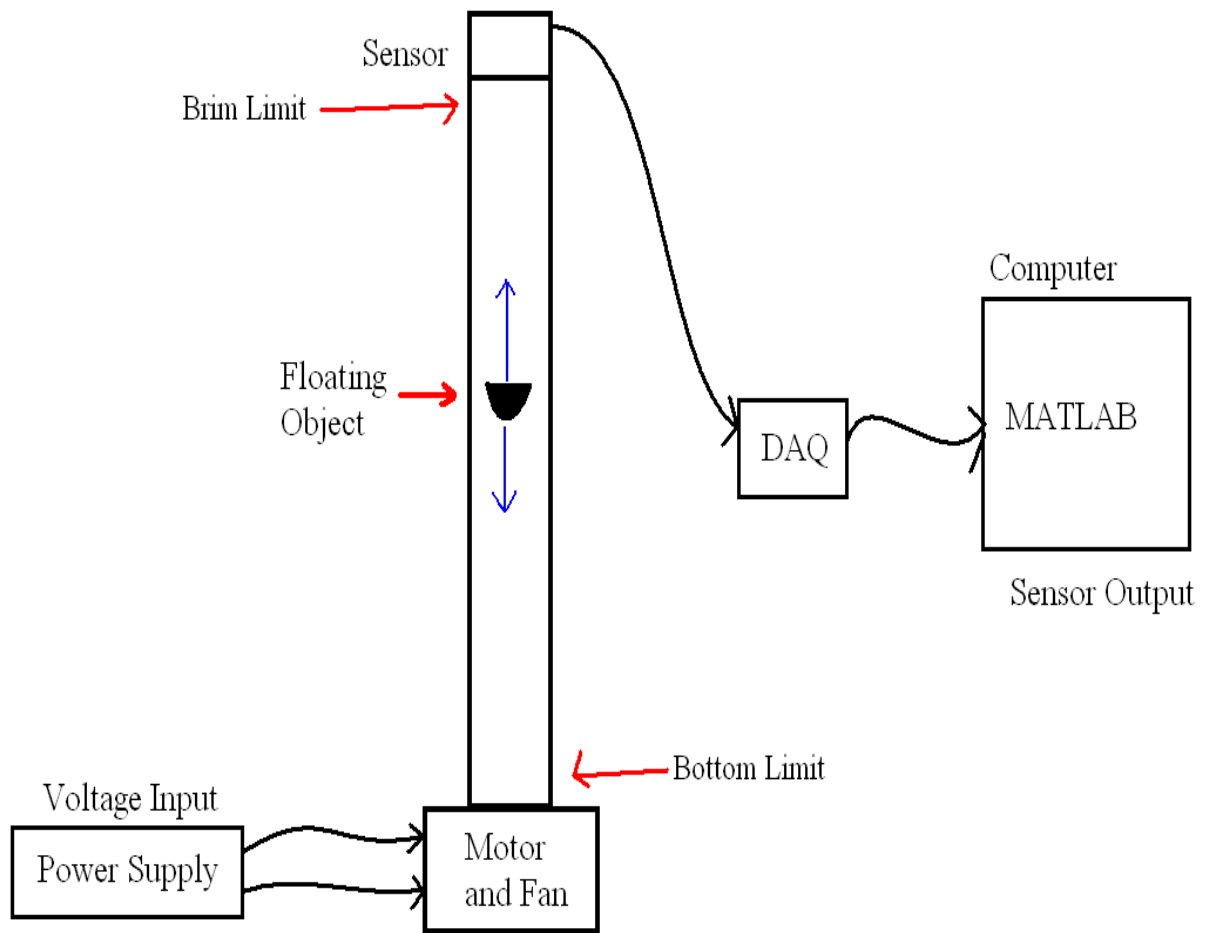


Figure 5.1: Experiment Setup

The results of the experiments are shown in Figure 5.2.

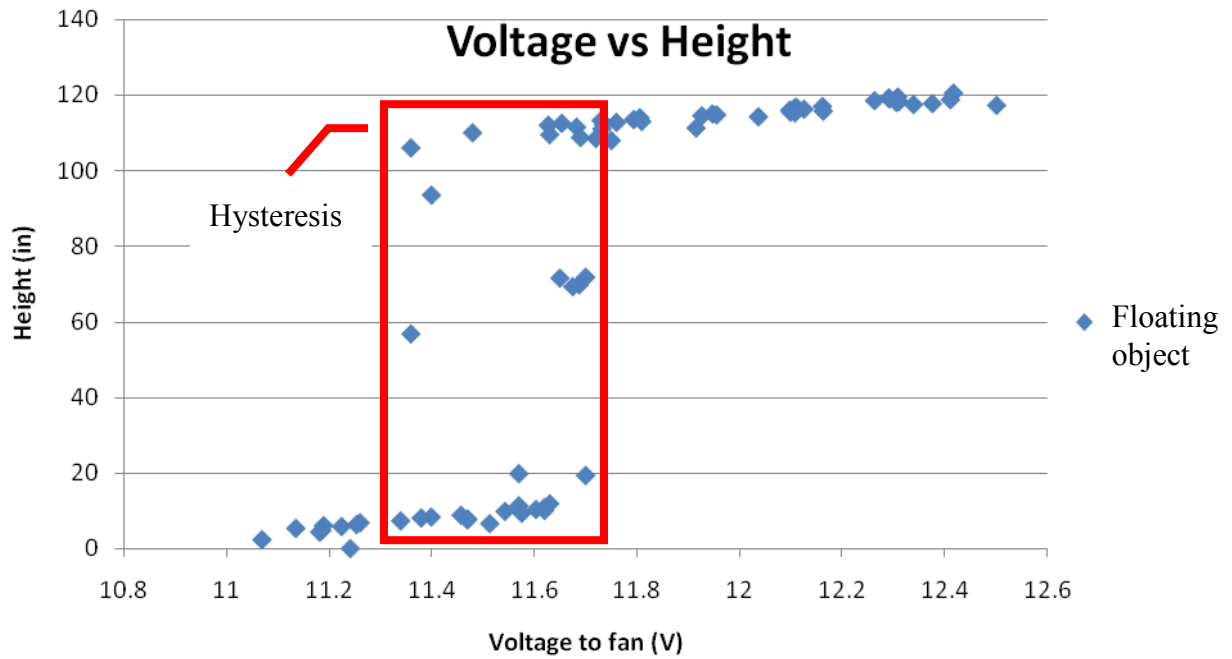


Figure 5.2: Voltage Fan vs. Height of Ball with Hysteresis

Figure 5.2 shows results of the experimental set up using the tallest tube of 10ft and a power supply voltage with a variable range of 0V to 15V. The same overall results, shown in Figure 5.2, occur for smaller tubes of 4 ft. and 6 ft. A blue dot in this same figure shows the average value of the altitude positions of the floating object observed from various tests performed. The average altitude is used due to the oscillatory behavior of the floating object causing different altitude positions when the same voltage is applied to the fan motor. The horizontal axis is the voltage (V) to the fan from the power supply and the vertical axis is the height (in) of the floating object measured with a measuring tape. A “digital” behavior of the floating object is observed in Figure 5.2. Approximately a height of 100 to 120 inches represents a ‘1’ digital output for the voltage range of 11.4V to 12.6V, and approximately a height of 0 to 20 inches represents a ‘0’ digital output for the voltage range of 11V to 11.7V. A hysteresis behavior, labeled with a red box in the plot, is found in the voltage range of 11.4V to 11.7V,

where the ball is either at the brim limit or the bottom limit depending if the floating object is on its way down or its way up.

After the test runs are performed using the set-up in Figure 5.1 and results are obtained shown in Figure 5.2, two ideas are derived.

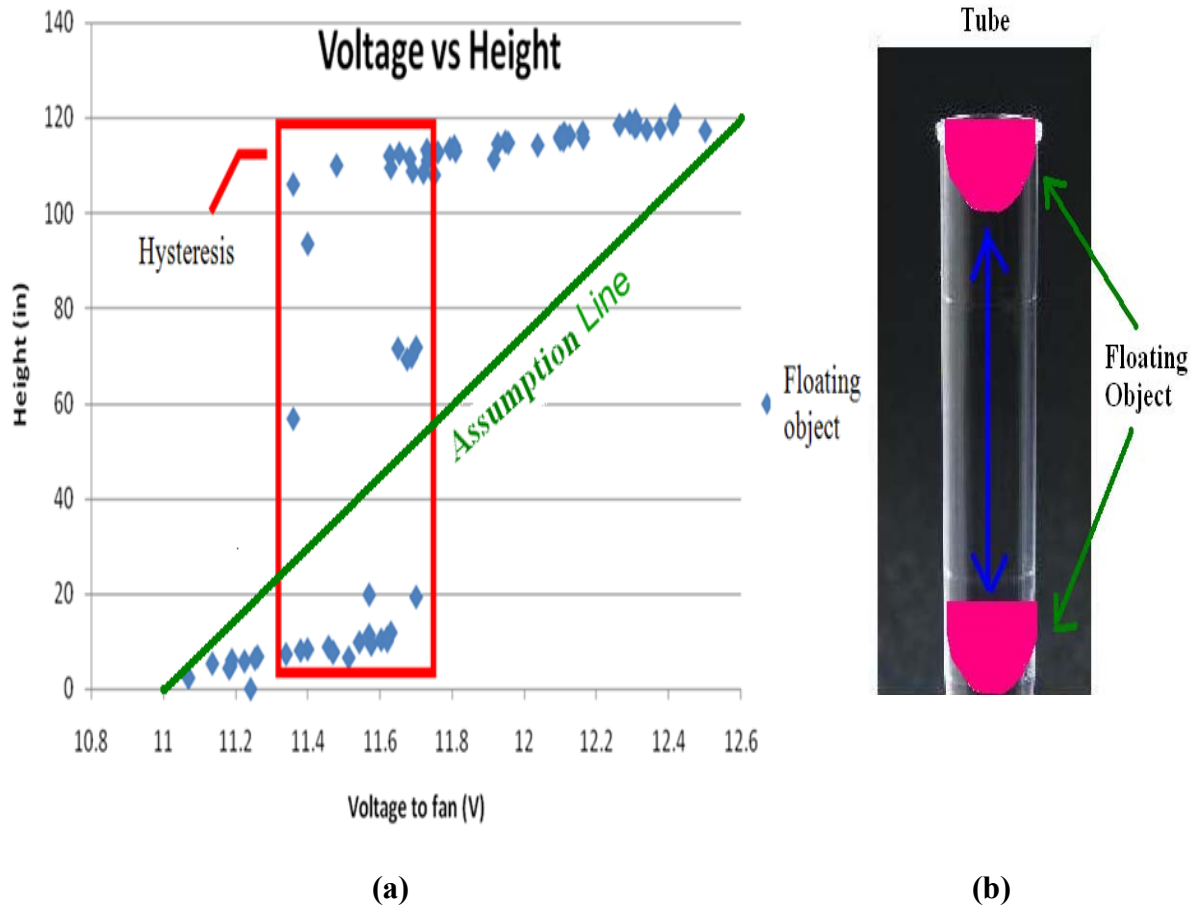
Idea #1: The amount of stream of air coming in from one end of the tube must be the same as the amount of stream of air coming out at the other end of the tube. This indicates that flow of air is needed to lift the object within the tube. This is similar to the case where the flow of water inside a tube or pipe moves objects.

Idea #2: If the air stream from the fan motor, supplied with a constant voltage, is able to lift the floating object fo a relative small altitude (approximately 20 in) then ultimately fo travels all the way to the brim of the tube due to the force of the same air stream.

In [17], the fan, powered by a motor, must develop enough pressure to overcome the resistance and move the air through the crop. In the same way the fan must overcome the resistance of the object's weight and lift the object along the inside of the tube. This characteristic gives Idea#2.

Idea #1 can be understood and realized by having a small plastic tube and blowing air through it. The air needs to go through the tube in order for the air to flow. This means that the tube needs an exit or escape route for the air inside the tube to flow through and thus lift an object. Another example is described to further understand Idea#1. This example involves the use of a human hand. When a hand is formed into the shape of a tunnel and air is blown through it, the air flows in and out of the tunnel. The air flow stops when the other opening of the hand tunnel is closed either with another human hand or with something else. The air blown through the closed tunnel does not flow. The air generated needs an exit route for the air to start flowing.

The two examples mentioned describe how air needs an exit route to flow, and how the air flow is needed to create an air force to lift the object.



**Figure 5.3: Incorrect Assumption Line
(a) Assumption Line; (b) Object Behavior**

Assumption#1 attempts to obtain a one-to-one relationship between the fan voltage and the object's altitude. This assumption is represented by the straight line in Figure 5.3. However, the results of all the tests, shown in Figure 5.3 also, demonstrate that Assumption#1 is incorrect. What the two ideas, Ideas#1 and #2, signify is that the ball tends or concentrates mostly at the bottom and at the brim limits of the tube with little or no altitude position in between. The two ideas apply to all tubes of different height sizes. Therefore, Conclusion 1 is derived.

Conclusion 1: A constant or steady state voltage is not able to settle the floating object to a specific altitude because of Idea#2. In other words, a constant voltage either brings the floating object to an altitude position at the bottom limit or the brim limit but not a position in between.

Conclusion 1 also gives an insight as to what type of voltage input is required to control the altitude of the object inside the tube. For instance, if the controller value remains constant, the result is the same as the behavior seen in Figure 5.3, but if the controller value is an oscillatory (fast and small) voltage then the floating object can converge to a medium height such as 40 in.

All the constant voltages to the fan only result in extreme (digital) floating object behavior with almost no settling at the middle of the tube as shown in Figure 5.3. Nevertheless, since the characteristics of the plant are known, the system is better understood and control can be improved. The positive result from the static modeling is that several characteristics of the floating ball plant are found which can ultimately benefit the dynamic modeling and control over the FBS. For example, in Conclusion 1 an observation is made that a constant voltage from the controller does not bring about a stable convergence at a particular height. Instead, a small varying voltage is needed to maintain the floating object at a middle desired height or altitude.

Another positive result of this type of modeling is that Figure 5.2 reveals a linear characteristic close to the bottom and brim limits of the tube. These regions reveal a more linear behavior than other regions of the tube. In the dynamic data collection, the region close to the bottom limit is considered.

On the other hand, the negative results prove to be very discouraging yet, fortunately, enlightening since it contradicts Assumption #1. Since the Static Modeling (presented in Section 5.1.1) failed, the dynamic modeling is covered in Section 5.1.2.

5.1.2 Dynamic Modeling

The dynamic modeling of the FBS is developed in this section because the steady state modeling approach does not apply to the floating ball plant. The steady state modeling does not work because the floating object does not stay at the same altitude given the same voltage. Consequently, the dynamic modeling utilizes the input variations and observes the corresponding output variations disregarding where the object starts from or what altitude it reaches. In other words, dynamic modeling uses the input-change to output-change correlation.

5.1.2.1 Data Collection

The data collection was obtained by a previous researcher. This data is useful because it shows the input u and the output Y variations within one run. The total amount of *samples* or data obtained is 122 values for each input and output. Each sample of the input u is calculated by the controller, sent to the plant, and becomes the voltage applied to the fan. The output Y is the output voltage of the sensor corresponding to the altitude of the floating object.

The input u and output Y data are collected from one run of the FBS. In Figure 5.4, the input u is shown and the output Y in Figure 5.5.

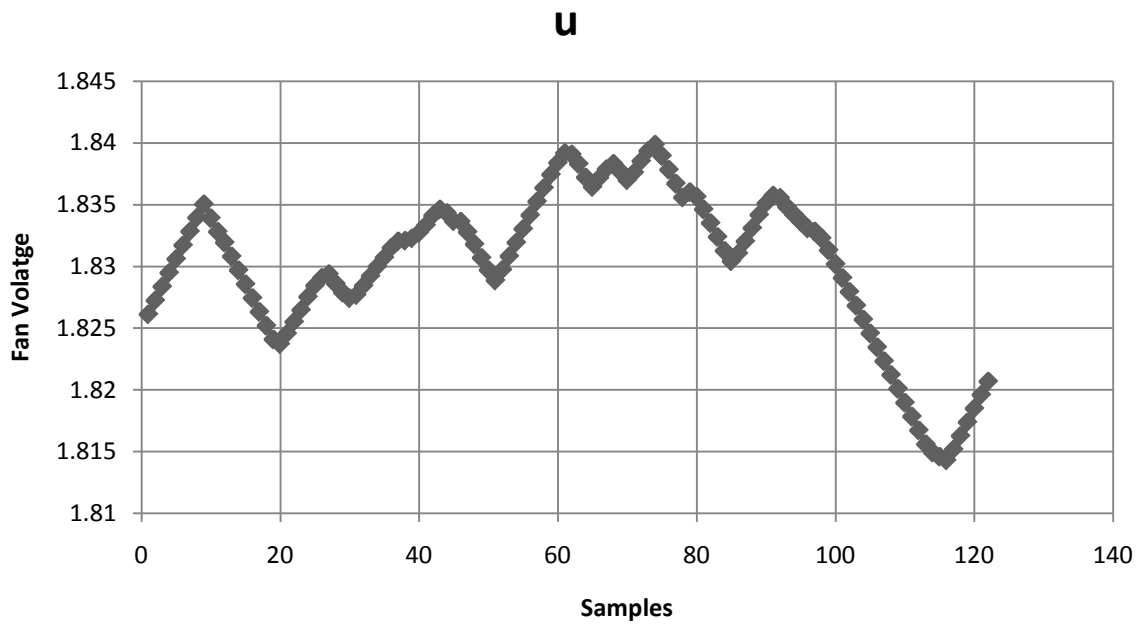


Figure 5.4: Input Data

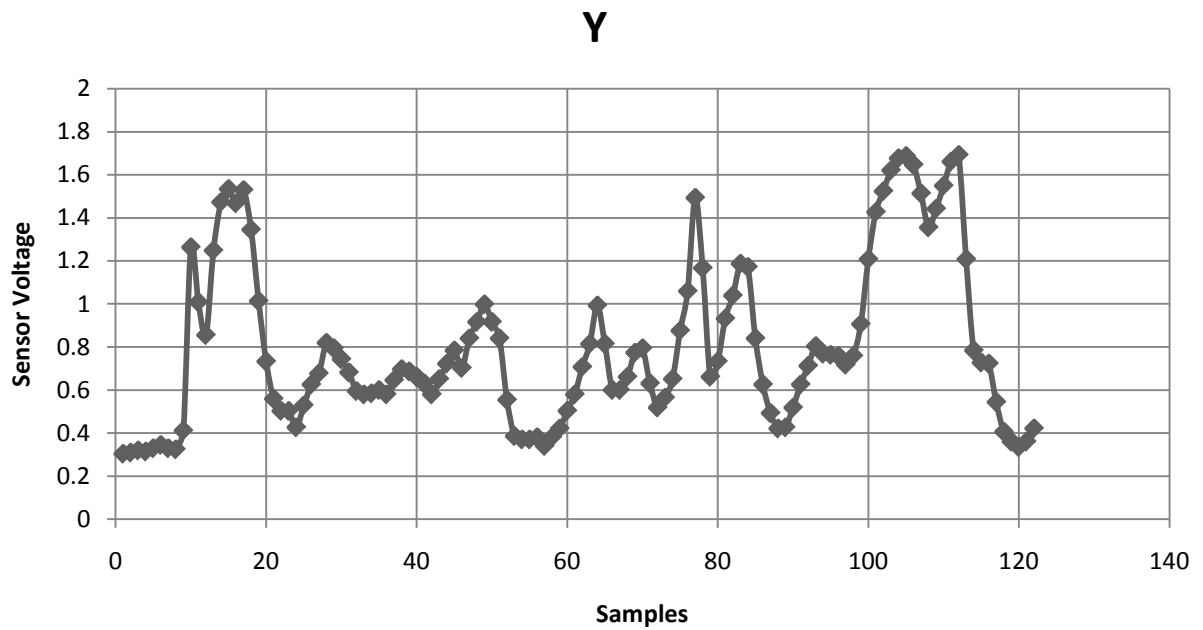


Figure 5.5: Output Data

The input $u \text{ tilde}$ and output $Y \text{ tilde}$ are the same as the u and Y data, but they have been shifted down by the mean value μ_u of the input u and the mean value μ_Y of output Y , respectively.

Equation (5.1) shows

$$\begin{aligned} u \text{ tilde} &= u - \mu_u \\ Y \text{ tilde} &= Y - \mu_Y, \end{aligned} \quad (5.1)$$

where $\mu_u = 1.83\text{V}$ and $\mu_Y = 0.797\text{V}$. All the values of input u are shifted down by the mean value μ_u and become the $u \text{ tilde}$ data shown in Figure 5.6. In the same manner, all the values of output Y are shifted down by the mean value μ_Y and become $Y \text{ tilde}$ shown in Figure 5.7.

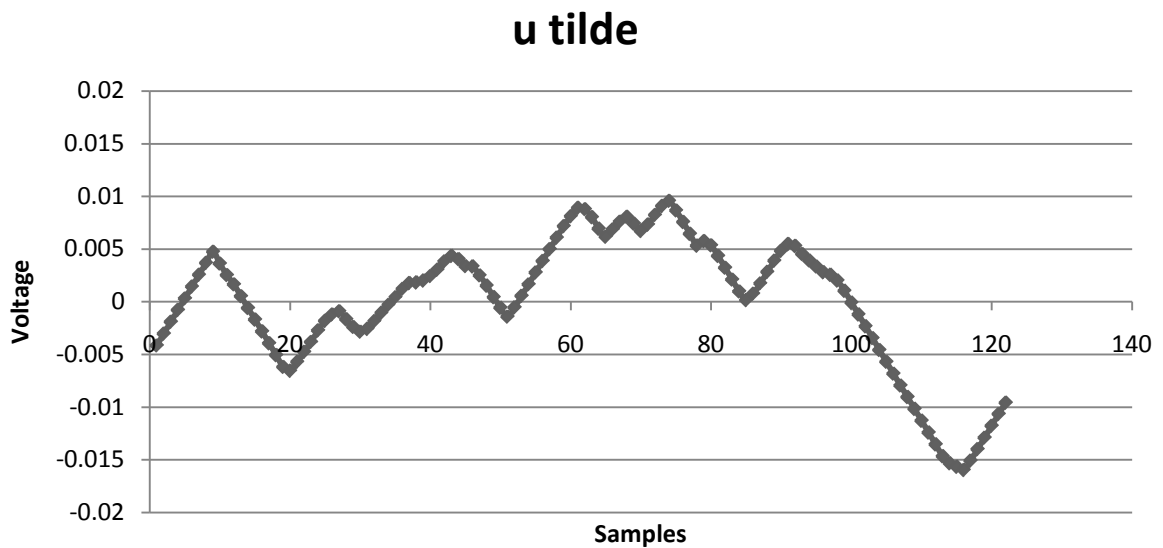


Figure 5.6: Shifted Input Data

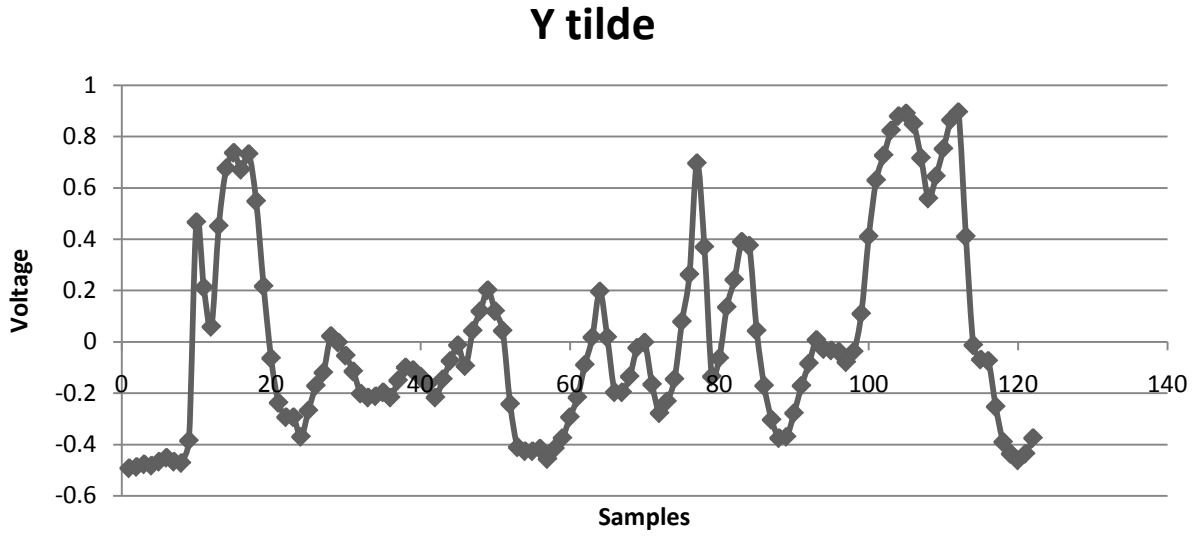


Figure 5.7: Shifted Output Data

The reason for shifting the data is to focus the region of control close to voltage = 0V. In Section 5.1.2.3., data *u tilde* and *Y tilde* are utilized to obtain the coefficients of the Control System Model Equation (5.2).

5.1.2.2 Control System Model Equation

To linearize the input and output relationship of the collected data, different models deriving from the Control System Model Equation (5.2) are considered. Then, the model are verified to see if it tracks or matches the original data. All the model equations presented in this section are tested to see which one gives the best approximation.

The model equation for a control system in [4] is

$$y(k) = \theta_{a1} \times y(k-1) + \theta_{a2} \times y(k-2) + \theta_{a3} \times y(k-3) + \dots + \theta_{aq} \times y(k-q) + \theta_{b0} \times u(k) + \theta_{b1} \times u(k-1) + \theta_{b2} \times u(k-2) + \dots + \theta_{bp} \times u(k-p), \quad (5.2)$$

but the following form

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + a_3 \times y(k-3) + \dots + a_q \times y(k-q) \\ + b_0 \times u(k) + b_1 \times u(k-1) + b_2 \times u(k-2) + \dots + b_p \times u(k-p) \quad (5.3)$$

is used for simplicity.

The following models are considered as representations of the FBS. All the simpler model equations in consideration have the form of the Control System Model Equation (5.2) and some $y(k)$ and $u(k)$ terms are used.

(I)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) \\ + b_0 \times u(k) \quad (5.4)$$

(II)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) \\ + b_0 \times u(k) + b_1 \times u(k-1) \quad (5.5)$$

(III)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) \\ + b_0 \times u(k) + b_2 \times u(k-2) \quad (5.6)$$

(IV)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) \\ + b_0 \times u(k) + b_1 \times u(k-1) + b_2 \times u(k-2) \quad (5.7)$$

(V)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) \\ + b_1 \times u(k-1) \quad (5.8)$$

(VI)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_1 \times u(k-1) + b_2 \times u(k-2) \quad (5.9)$$

(VII)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_2 \times u(k-2) \quad (5.10)$$

(VIII)

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_1 \times u(k-1) + b_2 \times u(k-2) + b_3 \times u(k-3) \quad (5.11)$$

Based on model verification simulations, the models that best approximate the collected data Y tilde given u tilde, are models (II), (III), and (IV). In order to narrow down the number of models, the discrete-time state space model is obtained for each equation model. The state space results for models (II), (III), and (IV), give a nonzero J matrix contrary to the result of $J = \theta$ of Equation (7.8) in Chapter 7. In Chapter 7, in Equation (7.9) the model (VI) has $J = \theta$ for its state space results. Consequently, model (VI) is chosen to represent the FBS over the other models, because the matrix $J = \theta$ results in a simpler steady state model which is less complicated when comparing Equation (7.7) and Equation (7.8). In addition, model (VI) is chosen because of its relatively good approximation results and its simple 2nd order form being adequate to represent the FBS.

To obtain the coefficients of model (VI), the equation model needs to be rewritten in matrix form as shown in Equation (5.12). The matrix form for the Control System Model Equation (5.2) is

$$y(k) = [\theta_{a1}, \theta_{a2}, \theta_{a3}, \dots, \theta_{aq}, \theta_{b0}, \theta_{b1}, \theta_{b2}, \dots, \theta_{bp}] \times \begin{bmatrix} y(k-1) \\ y(k-2) \\ y(k-3) \\ \dots \\ y(k-q) \\ u(k) \\ u(k-1) \\ u(k-2) \\ \dots \\ u(k-p) \end{bmatrix}, \quad (5.12)$$

where θ^T is

$$\theta^T = [\theta_{a1}, \theta_{a2}, \theta_{a3}, \dots, \theta_{aq}, \theta_{b0}, \theta_{b1}, \theta_{b2}, \dots, \theta_{bp}], \quad (5.13)$$

and

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ y(k-3) \\ \dots \\ y(k-q) \\ u(k) \\ u(k-1) \\ u(k-2) \\ \dots \\ u(k-p) \end{bmatrix}. \quad (5.14)$$

Then, the Equation (5.12) reduces to

$$y(k) = \theta^T \times x(k), \quad (5.15)$$

where θ^T is the transpose of θ . As a result, now Model (VI) has the following matrix form from

its original equation where

$$y(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_1 \times u(k-1) + b_2 \times u(k-2), \quad (5.16)$$

has the definitions of

$$\theta^T = [a_1, a_2, b_1, b_2] \quad (5.17)$$

and

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ u(k-1) \\ u(k-2) \end{bmatrix} \quad (5.18)$$

to have the form of

$$y(k) = \theta^T \times x(k). \quad (5.19)$$

5.1.2.3 Batch Least Square Method

The coefficients a_1 , a_2 , b_1 , and b_2 of model (VI), as shown in Equation (5.9), are solved by using the Batch Least Square (BLS) method. The BLS method is useful in approximating unknown variables which maps a given system's input data to its output data. Ideally, this method approximates the coefficients of the Equation (5.2) or Equation (5.3) once the system's input and output data have been collected. The method enables us to get an approximation of each of the coefficients $\theta_{a1}, \theta_{a2}, \dots, \theta_{aq}$ and $\theta_{b0}, \theta_{b0}, \dots, \theta_{bp}$ in Equation (5.2) or the solution of matrix θ in Equation (5.13) with the Equation (5.20) below. One drawback of the BLS method is that it performs offline and is not a real-time estimator because only after the data has been obtained can the BLS model coefficients be obtained [4], [8].

The Batch Least Square (BLS) method uses the equation

$$\hat{\theta}(M) = [\phi^T(M) \times \phi(M)]^{-1} \times \phi^T(M) \times Y(M), \quad (5.20)$$

where M is the size of the ‘batch’ of data considered, and $Y(M)$ and $\Phi(M)$ are given by

$$Y(M) = \begin{bmatrix} y(1) \\ y(2) \\ y(3) \\ \cdot \\ \cdot \\ \cdot \\ y(M) \end{bmatrix} \quad (5.21)$$

and

$$\Phi(M) = \begin{bmatrix} x^T(1) \\ x^T(2) \\ x^T(3) \\ \cdot \\ \cdot \\ \cdot \\ x^T(M) \end{bmatrix} \quad (5.22)$$

respectively, where $\phi^T(M)$ is the transpose of the $\Phi(M)$. The M value size suggests that not all the 122 values of data need to be used.

The model (VI) is simulated by developing the m-file script in the MATLAB software environment. The MATLAB script, written to solve for the unknown parameters using the BLS method, is shown in Appendix II. The script is able to plot not only one size of M but all sizes of M where $M \geq 4, 5, 6, \dots, 118$. This means that all different sizes of batches, that is, $M = 1$ batch,

$M = 2$ batch, $M = 3$ batch, ..., $M = 118$ batch is tested and analyzed within the MATLAB environment.

Based on statistical analysis and visual observation such as averaging, the chosen batch size is $M = 108$ where the unknown parameters are shown in Table 5.1. The batch size of $M = 108$ is the chosen batch size because it provides a better approximation when compared to other batch sizes.

Table 5.1: Parameters of the Model Equation

a_1	1.287852678797189
a_2	-0.326735013498350
b_1	33.141803846436060
b_2	-30.506954781341744

5.1.2.4 Model Verification

The validation the model is presented with the $\hat{y}(k)$ equation given by

$$\begin{aligned} \hat{y}(k) = & a_1 \times y(k-1) + a_2 \times y(k-2) \\ & + b_1 \times u(k-1) + b_2 \times u(k-2), \end{aligned} \quad (5.23)$$

where $y(k)$ is the true output value form the Y tilde data and $\hat{y}(k)$ is the approximation of $y(k)$. Equation (5.23) is a recursive function where the terms of $y(k)$, i.e., $y(k-1)$ and $y(k-2)$, are the real output values of the collected data Y tilde. The terms of $y(k)$ are utilized in order to simulate a real feedback from the FBS.

In order to verify the model's accuracy, the $\hat{y}(k)$ approximations of $y(k)$ are plotted in Figure 5.8 with batch sizes $M = 75$, $M = 97$, and $M = 108$.

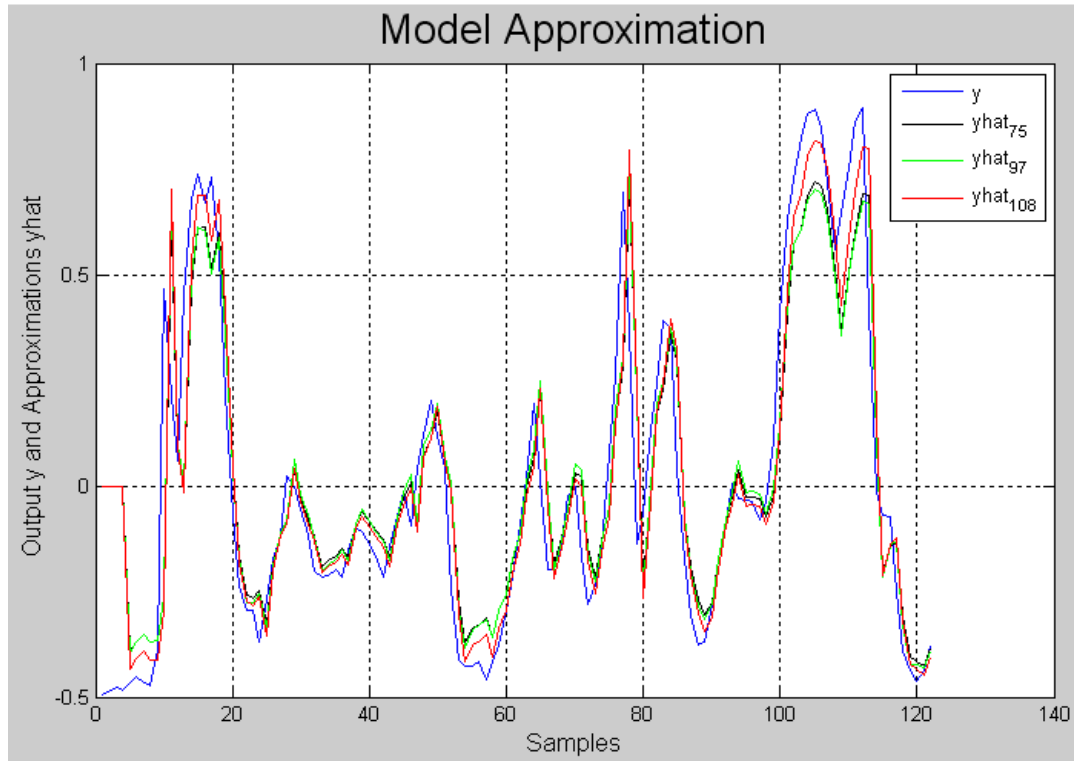


Figure 5.8: Output Modeling by Recursive $\hat{y}(k)$ using $y(k)$ Real Output Values

In Figure 5.8, the real $y(k)$ is the blue signal, the approximation $\hat{y}(k)$ with $M = 75$ is the black signal, $\hat{y}(k)$ with $M = 97$ is the green signal, and $\hat{y}(k)$ with $M = 108$ is the red signal. These $\hat{y}(k)$ approximations are close to the output signal $y(k)$ due to the use of true $y(k)$ terms as recursion values for the approximations $\hat{y}(k)$ in Equation (5.23).

CHAPTER 6: MODEL TESTING USING FUZZY LOGIC CONTROLLER

This chapter covers the testing of the model (VI). The FBS system is represented by the equation model (VI) in Equation (5.9) in Chapter 5, which is in terms of present and past inputs and outputs, and coefficients. The Three-Rule FLC is used to test the stability and convergence results of the model.

6.1 System Model

The equation model (VI) is repeated in Equation (6.1) as

$$\hat{y}(k) = a_1 \times y(k-1) + a_2 \times y(k-2) + b_1 \times u(k-1) + b_2 \times u(k-2), \quad (6.1)$$

where a_1 , a_2 , b_1 , and b_2 parameters are shown again in Table 6.1.

Table 6.1: Parameters of the Model Equation

a_1	1.287852678797189
a_2	-0.326735013498350
b_1	33.141803846436060
b_2	-30.506954781341744

Because the model output $y(k)$ in Equation (5.9) approximates the real output of the FBS, $y(k)$ is changed to $\hat{y}(k)$ in Equation (6.1).

6.2 Fuzzy Logic Controller

A detailed description of the FLC procedures is presented in Chapter 3. Since both the Three-Rule and Five-Rule FLCs, have essentially the same control algorithm, this section uses only the Three-Rule FLC to test the model (VI). To develop the Three-Rule FLC, the linguistic rules, the general error equation, the error universe, and the fuzzy regions are considered.

The Linguistic Rules

The fuzzy logic rules can be express in terms of linguistic rules as follows.

- If the *error, ee*, is Negative then *change in controller, du*, is Positive
- If the *error, ee*, is Zero then *change in controller, du*, is Negative.
- If the *error, ee*, is Positive then *change in controller, du*, is Negative.

The fuzzy rules are represented by the MATLAB commands below.

```
DOF = interp1(Err_X,N,ee);  
DU1 = min(P,DOF);  
DOF = interp1(Err_X,Z,ee);  
DU2 = min(Z,DOF);  
DOF = interp1(Err_X,P,ee);  
DU3 = min(N,DOF);
```

Figure 6.1: Script of the Fuzzy Rules

The General Error Equation

The error equation, used in this Three-Rule FLC, is

$$ee(k) = PV(k) - VR, \quad (6.2)$$

where $ee(k)$ is the error, PV is the actual height in voltage read by the sensor, and VR is the set point (target or desired value). If the error is negative then the floating object is below the target, but if the error is positive then the floating object is above the target. The modified error equation presented in Chapter 3 is not used here because the general error equation is good for simulation, but not in real-time implementation.

The Error Universe

As mentioned in Chapter 3, the error universe is the entire range of the error. The range of the sensor's output is given by the data sheet of the sensor [15]. In order to get the error universe, the sensor needs to detect the object at different locations, especially at maximum and minimum values of the sensor's detecting range. The error range is obtained by subtracting the target output value VR from the sensor's output value $PV(k)$, as shown in Equation (6.2). The target value VR must be within the sensor voltage output range.

The range sensor has a voltage range of [0.3V, 1.9V]. This is a safe voltage range since many IR sensors vary in their voltage range depending on sensor orientation and brightness of the room. The error universe is calculated to be -1.6 to 1.6. The value 1.6V is the result of the maximum sensor output 1.9V subtracted by the minimum target value 0.3V and the value -1.6V is the result of the minimum sensor output 0.3V subtracted by the maximum target value 1.9V. The error universe in MATLAB code is written as $Err_X = [-1.6:0.1:1.6]$ where 0.1 is the interval jump from -1.6 to 1.6.

The Fuzzy Regions

The fuzzy regions for the controller are shown in Figure 6.2 as MATLAB script.

```
N=trapmf(Err_X, [-1.6 -1.6 -0.5 0]);  
Z=trimf(Err_X, [-0.5 0 0.5]);  
P=trapmf(Err_X, [0 0.5 1.6 1.6]);
```

Figure 6.2: Script of Three-Regions of FLC

6.3 Simulation Results

The results of the Three-Rule FLC are presented in this section. The FLC calculates the controller $u(k)$ voltage based on the fuzzy rules and the error value $ee(k)$ and uses the model equation (6.1) to generate the output PV . The PV signal response corresponding to the set points $VR = 0.4, 0.7, 0.8$ are shown in Figure 6.3, 6.4, and 6.5.

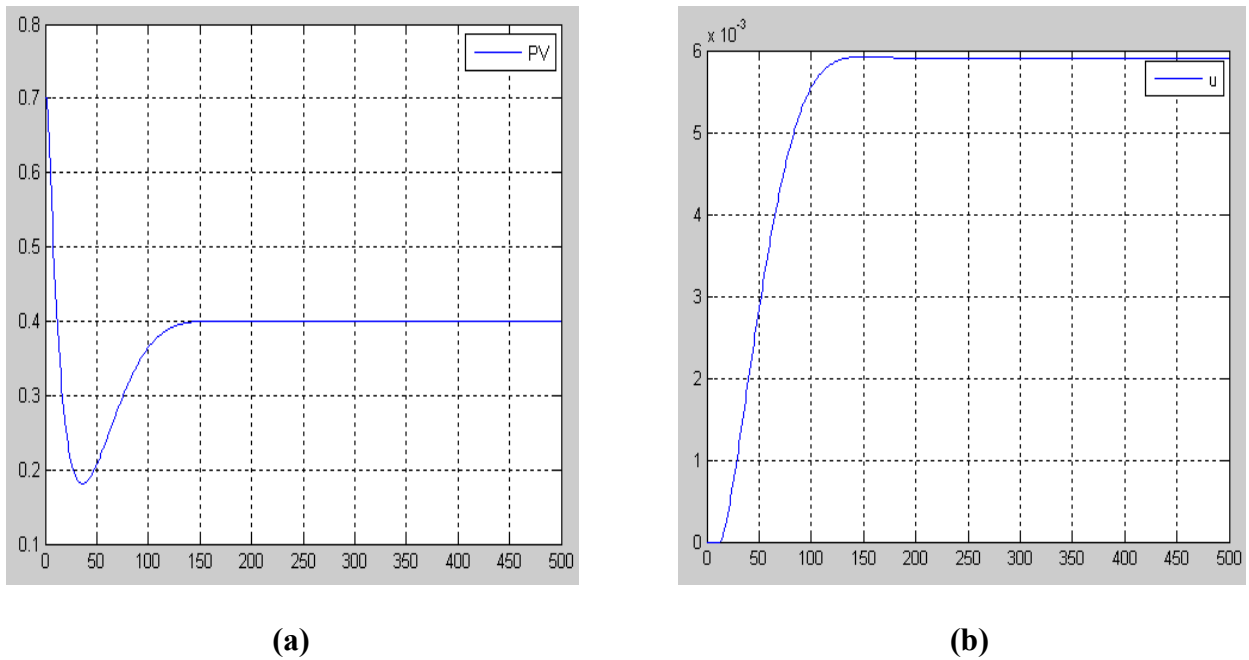
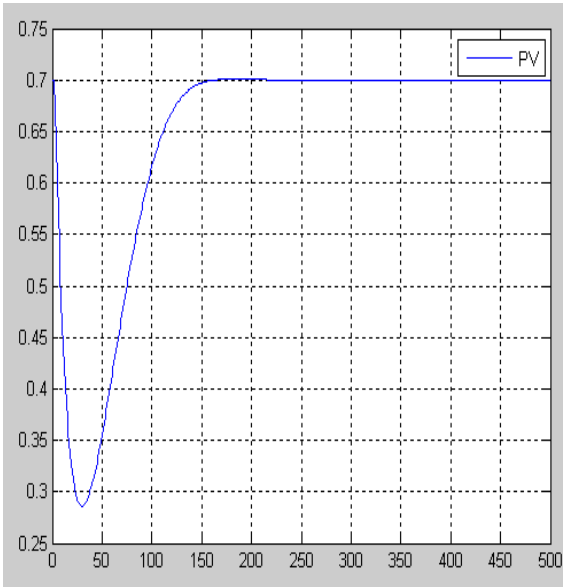
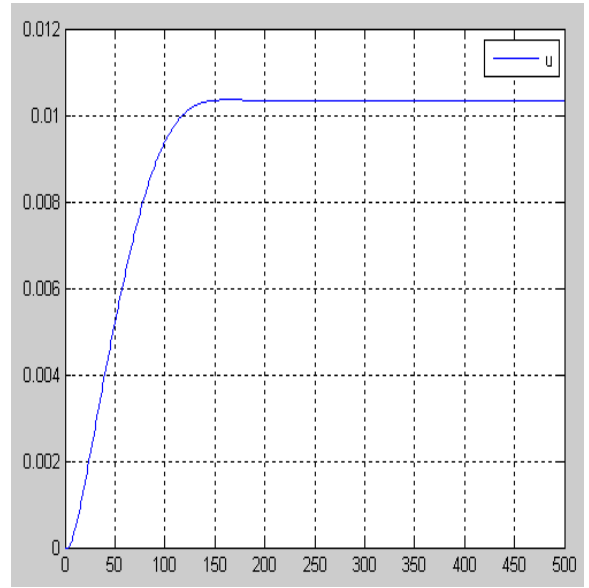


Figure 6.3: Three-Rule FLC $VR = 0.4$ Results
(a) Output PV for $VR = 0.4$; (b) Controller Value u for $VR = 0.4$

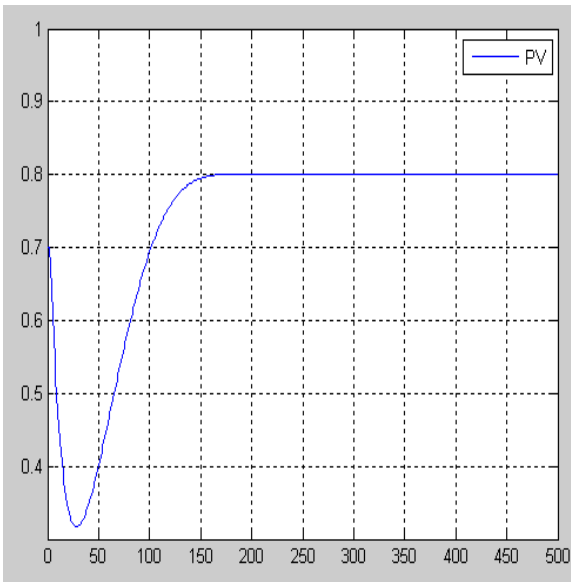


(a)

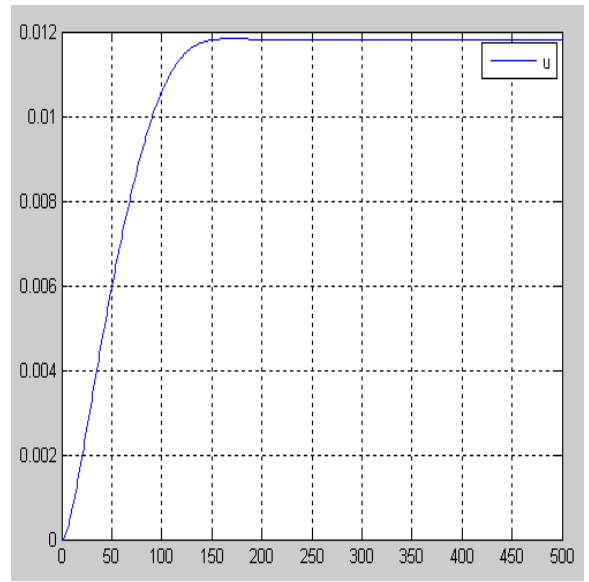


(b)

Figure 6.4: Three-Rule FLC $VR = 0.7$ Results
(a) Output PV for $VR = 0.7$; (b) Controller Value u for $VR = 0.7$



(a)



(b)

Figure 6.5: Three-Rule FLC $VR = 0.8$ Results
(a) Output PV for $VR = 0.8$; (b) Controller Value u for $VR = 0.8$

6.4 Problems and Discussion

One problem is the use of a very small gain in the fuzzy logic simulation. Another problem that emerges is based on the simulation results of the FLC. According to the simulation results, the V_R range is limited to $[0.3V, 0.87V]$ in order to keep $u(k)$ within $[0V, 2V]$ and $y(k)$ (or $PV(k)$) within $[0V, 3V]$. These boundaries of $u(k)$ and $PV(k)$ are derived from the hardware physical limitations.

The real-time implementation is significantly different than the simulation because of the hardware involved such as the fan motor and the sensor. In the real-time implementation and according to Section 5.1.1, the $u(k)$ low voltage range of $[0.0V, 1.6V)$ is not enough to lift the object. This low range of input $u(k)$ results in the output $PV(k)$ equal to approximately $0.35V$. A $PV(k)$ output of $0.35V$ means the object is at the bottom limit of the tube. Once the $u(k)$ value reaches a voltage of approximately $1.6V$, this voltage value causes the floating object to go all the way to the brim limit. However, the object should not reach the brim limit because it enters into one of the dead regions of the sensor. Therefore, a quick control of the object is needed before it reaches the brim limit in order to settle around the target. Once the controller is operating and controlling the object, then a voltage value within the range of $[0.0, 1.6)$ is needed to bring down the ball. To bring the object down the tube a u value of $1.4V$ has the same effect on PV as the u value of $0V$.

CHAPTER 7: TRACKING CONTROLLER DEVELOPMENT AND SIMULATION

In order to develop the tracking controller the following procedures are taken:

- i. The control model system equation is transformed to the discrete-time transfer equation by the Z-Transform;
- ii. The state space model is obtained via the MATLAB command *ssdata*;
- iii. The transfer function poles are calculated from the coefficients of the control system model equation;
- iv. The controller gain \mathbf{K} matrix is solved;
- v. The Observer \mathbf{Lp} matrix is calculated;
- vi. The desired poles of the Controller gain \mathbf{K} and the desired poles of the Observer \mathbf{Lp} are modified in order to have a faster settling time response for the FBS system;
- vii. The state command matrix N_x and the proportionality constant N_u are solved;
- viii. The tracking controller $u(k)$ is fully defined with all its components solved.

7.1 System Model

The steps listed above are now expressed numerically in this section and in the following sections. The model (VI) of the floating ball plant is once again written in Equation (7.1) as

$$\begin{aligned}\hat{y}(k) = & a_1 \times y(k-1) + a_2 \times y(k-2) \\ & + b_1 \times u(k-1) + b_2 \times u(k-2),\end{aligned}\tag{7.1}$$

with the parameters a_1 , a_2 , b_1 , and b_2 in Table 6.1.

The Transfer Function

The transfer function of the model (IV) is obtained by using the Z-transform. Then, Equation (7.1) becomes

$$\frac{Y(z)}{U(z)} = \frac{b_1 \times z^{-1} + b_2 \times z^{-2}}{1 - a_1 \times z^{-1} - a_2 \times z^{-2}}, \quad (7.2)$$

where $y(k)$ transforms to $Y(z)$, $u(k)$ transforms to $U(z)$, $y(k-1)$ transforms to $z^{-1} \times Y(z)$, $y(k-1)$ transforms to $z^{-2} \times Y(z)$, and with similar transformations for $u(k-1)$ and $u(k-2)$. The transfer function now becomes

$$G_d(z) = \frac{Y(z)}{U(z)} = \frac{b_1 \times z + b_2}{z^2 - a_1 \times z - a_2} \quad (7.3)$$

when is multiplied by $\frac{z^2}{z^2}$.

7.2 Controller Development and State Space Model

The FBS has an output (object's position) and an input (fan voltage) and the state variables. The block diagram presented in Figure 7.1 contains the reference r added to a plant with a full-state feedback plus feed-forward. A full-state feedback system is a system where all of its states are measurable [4]. The plant in Figure 7.1 refers to the floating ball plant.

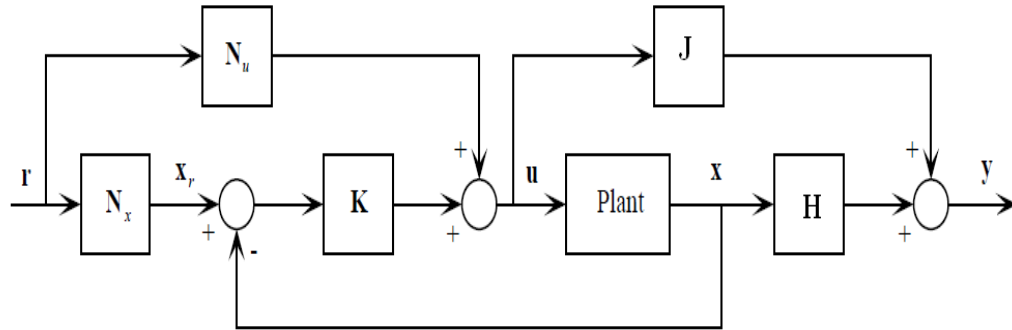


Figure 7.1: Full-State Feedback Block Diagram

The block diagram in Figure 7.1 corresponds to the state space model of

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{\Phi} \times \mathbf{x}(k) + \mathbf{\Gamma} \times \mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{H} \times \mathbf{x}(k) + \mathbf{J} \times \mathbf{u}(k), \end{aligned} \quad (7.4)$$

where $\mathbf{J} = \mathbf{0}$.

The tracking controller $u(k)$ is known as

$$u(k) = -\mathbf{K} \times \mathbf{x}(k) + \mathbf{K} \times \mathbf{N}_x \times r + \mathbf{N}_u \times r, \quad (7.5)$$

where \mathbf{K} is the controller gain, $\mathbf{x}(k)$ is the state variable, \mathbf{N}_x is the state command matrix, r is the reference voltage, \mathbf{N}_u is the proportionality constant, and k is the step of each iteration for the *for* loop seen in the MATLAB script in Appendix II.

For the FBS, the output is the only value measured in the system, not the states. In the FBS, an attempt to measure the speed or acceleration of the floating object by hardware (sensor) in a way obstructs the air flow through the tube. Another way to measure speed and acceleration is by software, that is, by writing a MATLAB code to calculate the derivative. The speed and acceleration calculations from the MATLAB code only goes so far since the speed and acceleration are approximations of the actual speed and acceleration of the object. Therefore, since there are no states measured in the system, the output feedback control with an Observer to estimate the states is necessary. The process that gives the estimated states is called the Observer (Estimator). The Estimator block is shown in Figure 7.2.

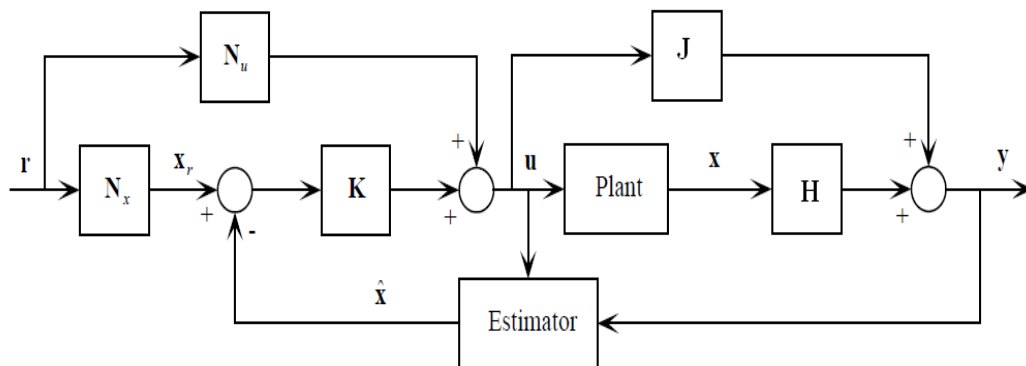


Figure 7.2: Observer Block Diagram

Consequently, the controller $u(k)$ of Equation (7.5) becomes

$$u(k) = -K \times \hat{x}(k) + K \times N_x \times r + N_u \times r, \quad (7.6)$$

where r is the reference input and $\hat{x}(k)$ is the estimated state matrix.

The block diagram in Figure 7.2 corresponds to the state space model of

$$\begin{aligned} \hat{x}(k+1) &= \Phi \times \hat{x}(k) + \Gamma \times u(k) + L_p \times [H \times x(k) - H \times \hat{x}(k)] \\ \hat{y}(k) &= H \times \hat{x}(k) + J \times u(k), \end{aligned} \quad (7.7)$$

where $J = 0$, $\hat{x}(k)$ is the observed state, and $x(k)$ is the unmeasurable state. Due to the unmeasurable state matrix $x(k)$ and $J = 0$, the block diagram of Figure 7.2 becomes the block diagram in Figure 7.3.

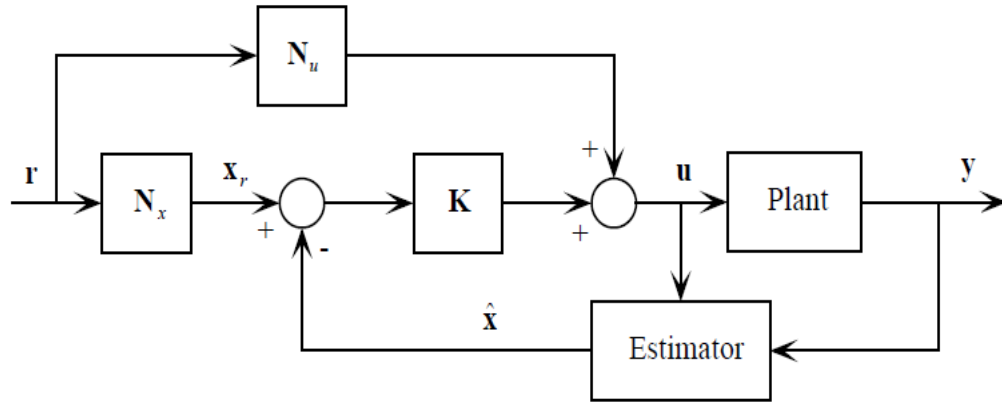


Figure 7.3: Observer Block Diagram with $J = 0$ and Unmeasurable $x(k)$ State

Then, also Equation (7.7) becomes

$$\begin{aligned} \hat{x}(k+1) &= \Phi \times \hat{x}(k) + \Gamma \times u(k) + L_p \times [y(k) - H \times \hat{x}(k)] \\ \hat{y}(k) &= H \times \hat{x}(k) \end{aligned} \quad (7.8)$$

for the block diagram of Figure 7.3.

Then, using the MATLAB command

$$[Phi, Gamma, H, J, Ts] = ssdata(Gd),$$

with Equation (7.3) as $G_d(z)$ and the calculated parameters a_1 , a_2 , b_1 , and b_2 , the state space model of Equation (7.8) is solved giving the following matrices

$$\begin{aligned}\Phi &= \begin{bmatrix} 1.287852678797189 & -0.653470026996700 \\ 0.5 & 0 \end{bmatrix} \\ \Gamma &= \begin{bmatrix} 8 \\ 0 \end{bmatrix} \\ H &= [4.142725480804508 \quad -7.626738695335436] \\ J &= [0]\end{aligned}\tag{7.9}$$

Next, the controllability and the observability are tested for the FBS state space model in Equation (7.8) with the calculated matrices in (7.9). The controllability is verified by meeting the following condition:

Condition: Equation (7.8) is controllable if and only if

$$\text{rank}\left(\left[\Gamma, \Phi \times \Gamma, \dots, \Phi^{n-1} \times \Gamma\right]\right) = n\tag{7.10}$$

where n value corresponds to the $n \times n$ dimensions of Φ .

Since Φ is a 2x2 matrix, $n = 2$ and Equation (7.10) gives

$$\text{rank}\left(\left[\Gamma, \Phi \times \Gamma\right]\right) = \text{rank}\left(\begin{bmatrix} 8 & 10.3028 \\ 0 & 4 \end{bmatrix}\right) = 2 = n.$$

Therefore, the system (7.8) is controllable because the rank is 2, which is equal to n .

The observability of the system is verified by meeting the following condition.

Condition: Equation (7.8) is observable if and only if

$$\text{rank} \begin{pmatrix} H \\ H \times \Phi \\ \cdot \\ \cdot \\ \cdot \\ H \times \Phi^{n-1} \end{pmatrix} = n. \quad (7.11)$$

Then, $n = 2$ because Φ is a 2x2 matrix and Equation (7.11) gives

$$\text{rank} \begin{pmatrix} 4.1427 & -7.6267 \\ 1.5219 & -2.7071 \end{pmatrix} = 2 = n.$$

Similarly, the system (7.8) is observable because the rank = 2 = n .

Next, the poles of the transfer function (7.3) are obtained using the coefficients of the control system model equation in Table 6.1. The denominator equation of the transfer function is

$$z^2 - a_1 \times z - a_2 = 0, \quad (7.12)$$

which factors to $(z-0.347) \times (z-0.94) = 0$ with poles $z_1 = 0.347$, $z_2 = 0.94$.

The controller gain K and Observer Lp are obtained next for the tracking controller $u(k)$ in Equation (7.6) and state space model in Equation (7.8). The following equation is used

$$\det |z \times I_{2 \times 2} - (\Phi - \Gamma \times K)| = (z - 0.347) \times (z - 0.94) \quad (7.13)$$

to solve for controller gain K where z is the variable and I is the identity matrix. Then, the equation

$$\det |z \times I_{2 \times 2} - (\Phi - L_p \times H)| = (z - 0.347) \times (z - 0.94) \quad (7.14)$$

is used to solve for Observer Lp . The determinant formula of a matrix is used for both Equations (7.13) and (7.14). Set up LHS = RHS to solve for controller gain K , where LHS is the left hand side of Equation (7.13) and RHS is Equation (7.12). LHS = RHS becomes

$$\det|z \times I_{2 \times 2} - (\Phi - \Gamma \times K)| = z^2 - a_1 \times z - a_2 \quad (7.15)$$

where Φ and Γ matrices are obtained from (7.9).

From Equation (7.15) the controller K matrix is

$$K = [K_1 \quad K_2] = [2.68 \times 10^{-4} \quad 4.34 \times 10^{-4}]$$

where K is a 1x2 matrix. Similarly, to solve for the Observer Lp matrix set LHS = RHS where LHS is now the left part of Equation (7.14) and RHS remains the same equation. Thus, the set up equation becomes

$$\det|z \times I_{2 \times 2} - (\Phi - L_p \times H)| = z^2 - a_1 \times z - a_2. \quad (7.16)$$

Then, solving for Lp in Equation (7.16) gives

$$L_p = \begin{bmatrix} L_{p1} \\ L_{p2} \end{bmatrix} = \begin{bmatrix} -1.619 \times 10^{-5} \\ -8.679 \times 10^{-5} \end{bmatrix}.$$

The K and Lp solutions are utilized to test the tracking controller $u(k)$ of Equation (7.6). Because the simulation in Figures 7.4, 7.5, and 7.6 results in a slow transient response and settling time from the system, new K and Lp matrices are defined. After testing different K and Lp matrices values, the following new K and Lp matrices are

$$K = [-0.064 \quad 0.12]$$

and

$$L_p = \begin{bmatrix} 10.5418 \\ 5.6097 \end{bmatrix}.$$

These new K and Lp matrices produce a faster transition time and settling time response.

The state command matrix N_x and the proportionality constant N_u are now solved using

the following equation:

$$\mathbf{N} = \begin{bmatrix} \mathbf{N}_x \\ \mathbf{N}_u \end{bmatrix} = \begin{bmatrix} \Phi - \mathbf{I}_{2 \times 2} & \Gamma \\ \mathbf{H} & \mathbf{J} \end{bmatrix}^{-1} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \quad (7.17)$$

The results are

$$\mathbf{N}_x = \begin{bmatrix} 3.04 \\ 1.52 \end{bmatrix}$$

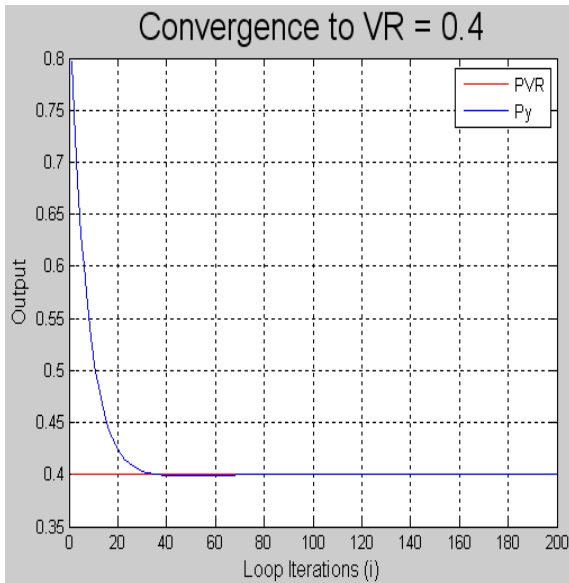
and

$$\mathbf{N}_u = [0.015].$$

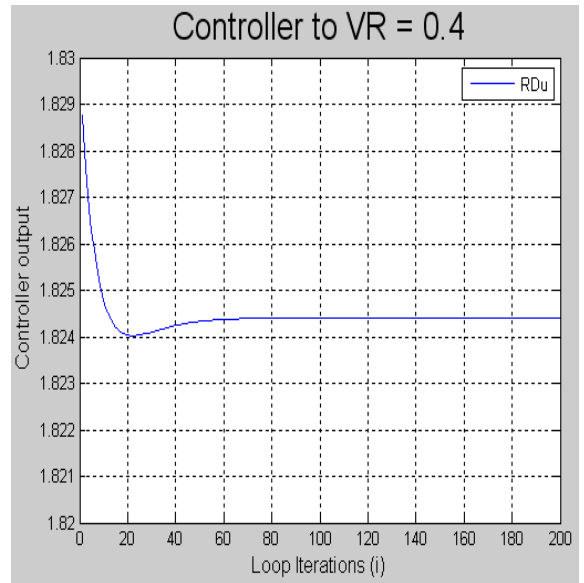
Finally, all the solutions obtained of the components of Equation (7.6) are used to calculate the value $u(k)$.

7.3 Simulation Results

The simulation results for VR or $PVR = 0.4, 0.5,$ and 0.7 are presented in Figure 7.4, 7.5, and 7.6. The MATLAB script for the plots is found in the Appendix II section. The signal PV (or Py) and u (or RDu) stay within the boundaries of the real hardware specifications as seen in all these three figures.

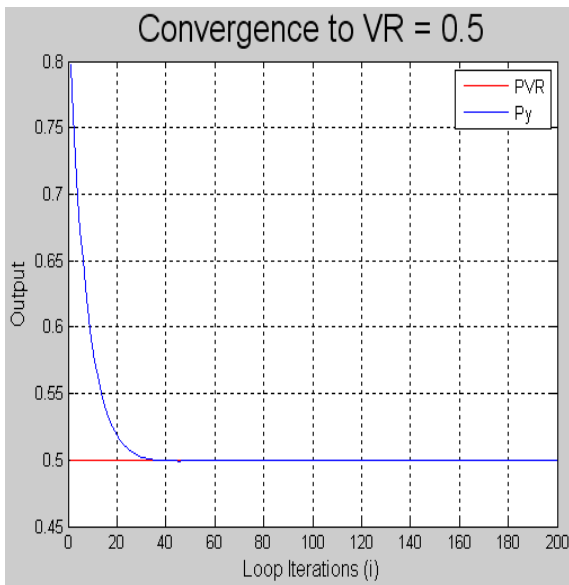


(a)

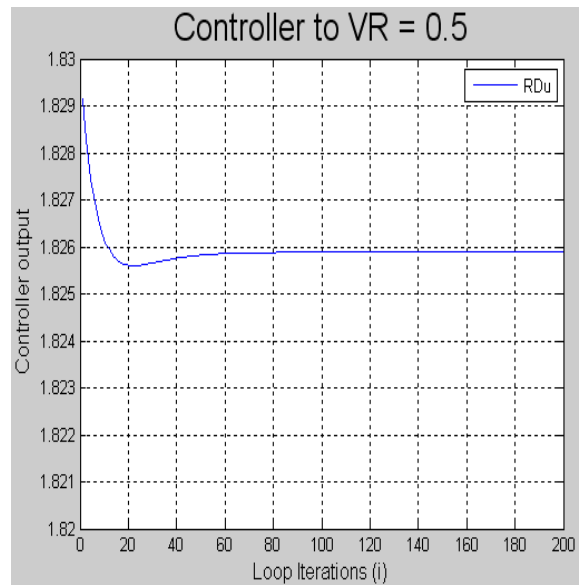


(b)

Figure 7.4: Observer $VR = 0.4$ Results
(a) Output P_y for $VR = 0.4$; (b) Controller Value RD_u for $VR = 0.4$

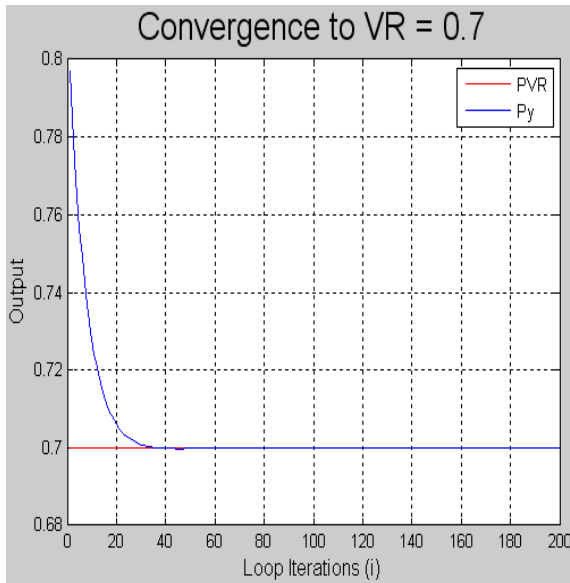


(a)

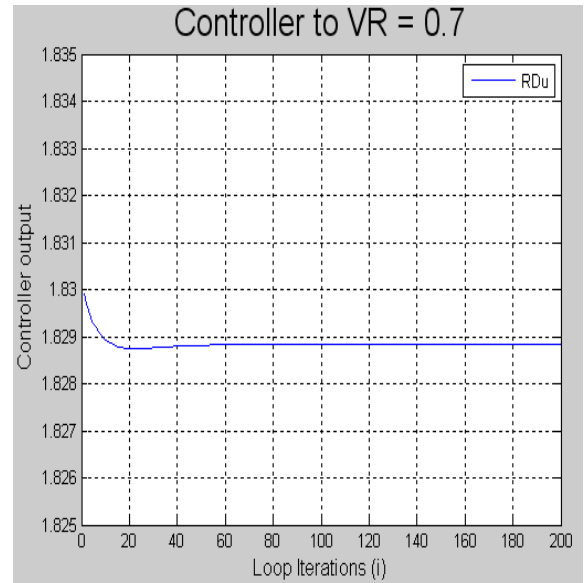


(b)

Figure 7.5: Observer $VR = 0.5$ Results
(a) Output P_y for $VR = 0.5$; (b) Controller Value RD_u for $VR = 0.5$



(a)



(b)

Figure 7.6: Observer $VR = 0.7$ Results
(a) Output P_y for $VR = 0.7$; (b) Controller Value RDu for $VR = 0.7$

7.4 Conclusion

This chapter demonstrates the development of the tracking controller $u(k)$, defined in Equation (7.6), as the second control approach implemented in the FBS. The first one is the FLC seen in Chapter 3. The output feedback control with an observer is utilized and a tracking controller $u(k)$ is designed to deal with a reference input. Then, the discrete-time state space model is obtained from the transfer function of the linearized control model. The state space model is solved for and tested in simulation with the tracking controller $u(k)$. The simulation results shows successful runs at selected target points $VR = 0.4, 0.5, \text{ and } 0.7$.

CHAPTER 8: TRACKING CONTROLLER IMPLEMENTATION TO REAL SYSTEM

Real-time implementation of the tracking controller $u(k)$ and observer is performed in this chapter. Also, this chapter demonstrates that the linear model (VI), developed in Chapter 5, overcomes the nonlinearities of the FBS plant and performs satisfactory in a local region.

The local region is defined as a region of the sensor output voltage close to zero where the linear model more accurately matches the system's behavior. Even though the sensor has a reading range of 0.4V to approximately 3.0V, the linear model is able to provide satisfactory convergence at the region [0.4V, 1.1V].

Real-time implementation involves considering the limit of hardware operation, the transition from theoretical variables to real, physical variables, and adjusting the ideal parameter values to reasonable practical values. When considering the limits of the hardware, the boundary and saturation of the hardware variables are taken into account. To transition from theory to real-time implementation, the sensor output PV needs to replace the $y(k)$ output variable of the state space model of Equation (7.8). The controller gain K and the Observer Lp need to be calibrated to the physical limits of the real system.

The real-time results of the convergence of the object at local target points are shown in the Figures 8.1, 8.3, 8.4, and 8.5. These figures are obtained using the MATLAB script found in Appendix II. The real-time tests are conducted with target points $VR = 0.6$ and $VR = 0.8$. The results show how the floating object, represented by the blue signal $YI Filtered$, achieves convergence to the target altitude position represented by the red signal VR . After the figures, Table 8.1 presents the RMSE and MAE results obtained for each VR value ranging from [0.4V, 1.7V]. Analysis of Table 8.1 allows for making comparisons between different window sizes of the filter, making a decision of the best performance, and obtaining insight about the results.

8.1 Observer without Filter

In this section of the chapter, the Median filter is not used with the observer and with the tracking controller $u(k)$. The noisy sensor signal y from the plant, seen in Figure 7.3 of Chapter 7, is used as the raw input to the observer. The output YI in Figure 8.1 has unwanted spikes and noise from the sensor's output. The sensor's noisy output is due to the current needed to power the sensor's continuous firing of its infrared beam to detect objects [2].

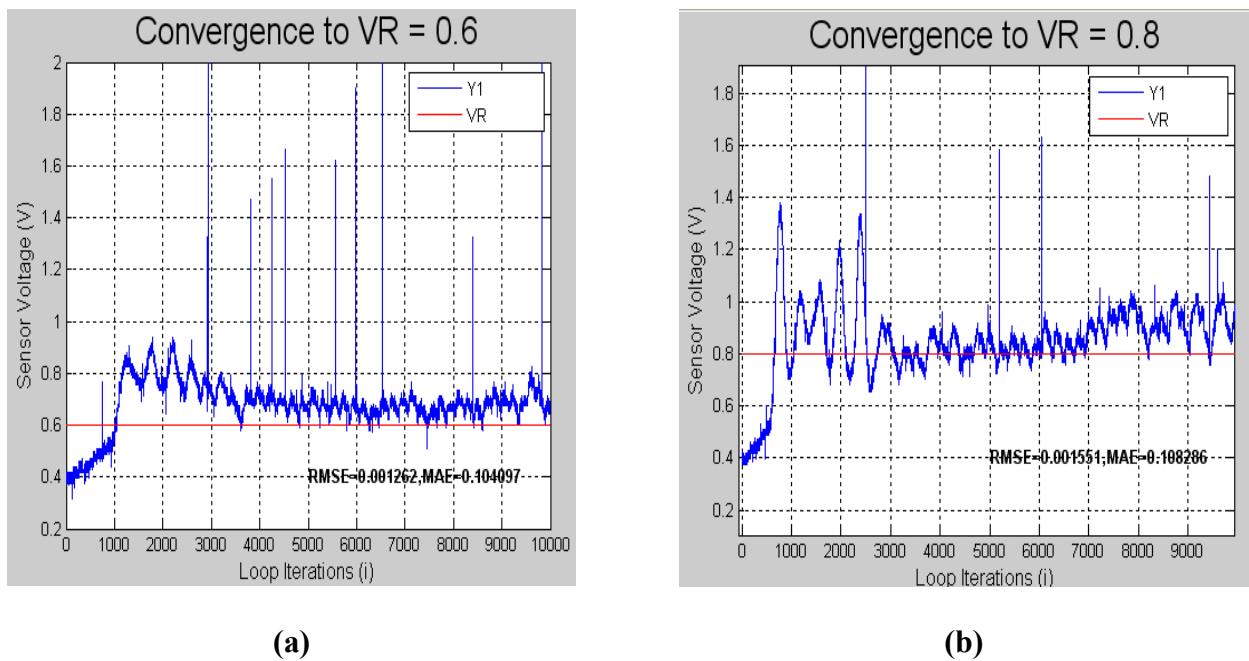


Figure 8.1: Observer and Controller $u(k)$ Only
(a) Output YI for $VR = 0.6$; (b) Output YI for $VR = 0.8$

8.2 Observer with Median Filter of Window Size 3

In this section and in Sections 8.3 and 8.4, the Median filter output $YI_{Filtered}$ (seen in Figure 8.2) is used instead of the noisy sensor value YI . The $YI_{Filtered}$ value becomes the input to the observer and then is utilized by the tracking controller $u(k)$, in Equation (7.6), to generate a control value. Figure 8.3 shows the real-time results of the floating object represented by the $YI_{Filtered}$ signal, converging to $VR = 0.6$ in plot (a) and to $VR = 0.8$ in plot (b).

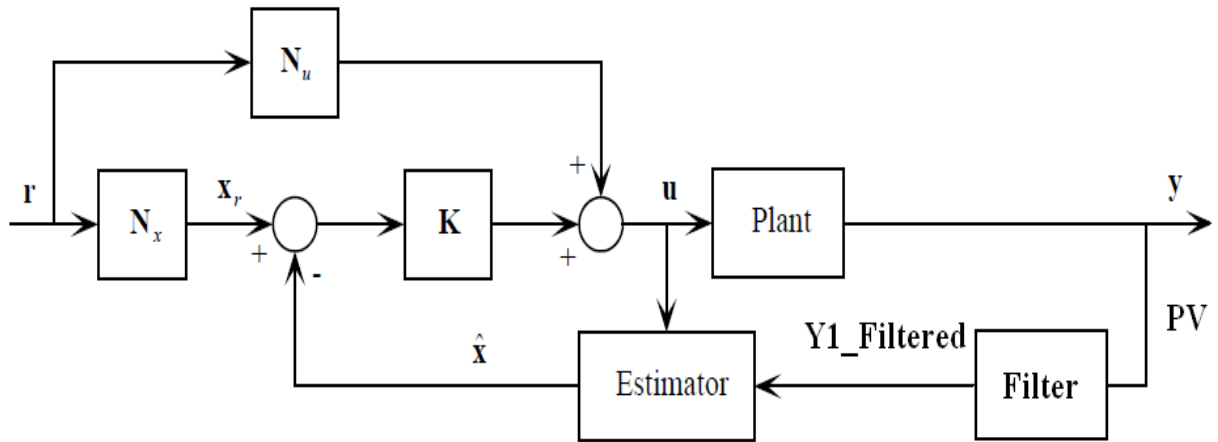
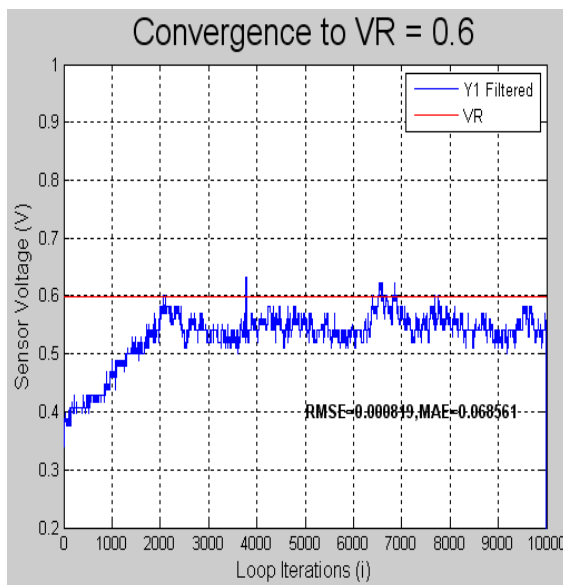
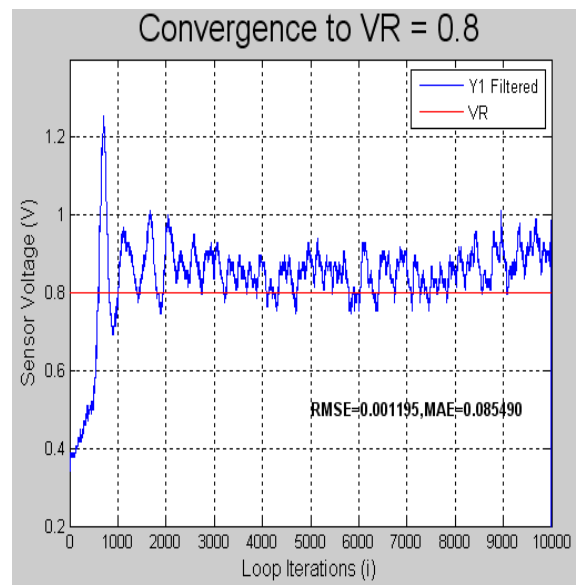


Figure 8.2: Observer Block Diagram with Filter Block



(a)

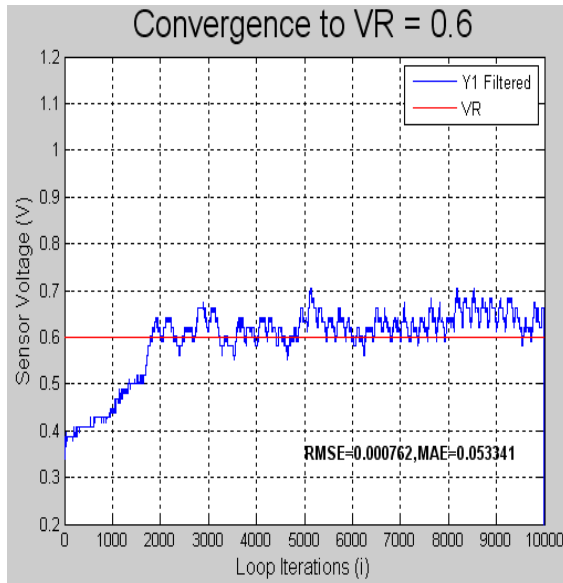


(b)

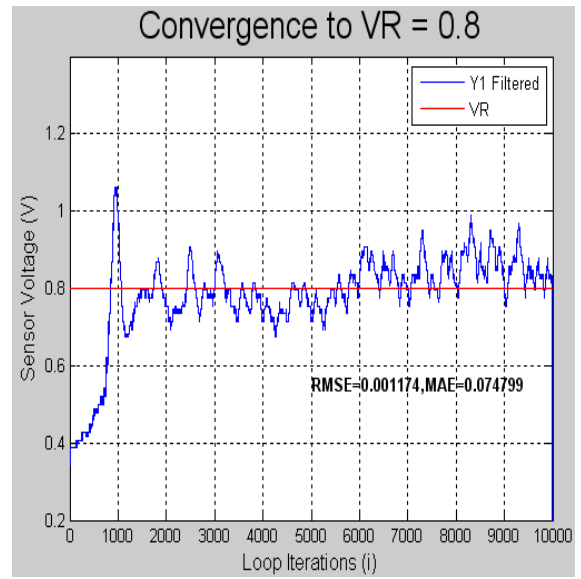
Figure 8.3: Observer and Controller $u(k)$ with Filter of Window Size 3
(a) Output $Y1$ Filtered for $VR = 0.6$; (b) Output $Y1$ Filtered for $VR = 0.8$

8.3 Observer with Median Filter of Window Size 5

Figure 8.4 shows the results of the floating object, represented by the $Y1$ Filtered signal, converging to $VR = 0.6$ and $VR = 0.8$ using the Median filter with window size 5.



(a)

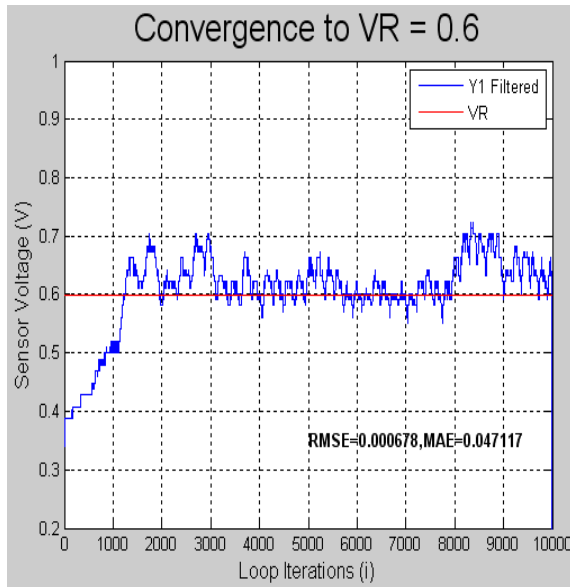


(b)

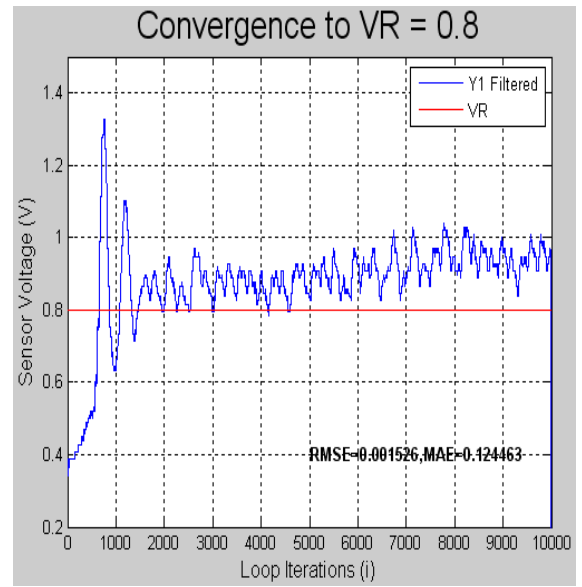
Figure 8.4: Observer and Controller $u(k)$ with Filter of Window Size 5
(a) Output $Y1$ Filtered for $VR = 0.6$; (b) Output $Y1$ Filtered for $VR = 0.8$

8.4 Observer with Median Filter of Window Size 7

Figure 8.5 shows the same floating object represented by the *Y1 Filtered* signal converging to $VR = 0.6$ and 0.8 with a Median filter but of window size 7.



(a)



(b)

Figure 8.5: Observer and Controller $u(k)$ with Filter of Window Size 7
(a) Output $Y1$ Filtered for $VR = 0.6$; (b) Output $Y1$ Filtered for $VR = 0.8$


Comparing Figures 8.3, 8.4, and 8.5, the Observer working with the Median filter of window size 5 performs better than with filter of window size 3 and window size 7. This conclusion where the filter's performance of window size 5 is better is also seen in Table 8.1 by the more RMSE and MAE values highlighted in red. The table also shows the comparison between the Observer with no filter to the Observers with filter of window sizes 3, 5, and 7.

Table 8.1: Comparison between Observer and Observer with Filter

VR	Observer w/ no Filter		Observer w/ WS3 Filter		Observer w/ WS5 Filter		Observer w/ WS7 Filter	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
0.4	0.000306*	0.014494*	0.000168*	0.014807*	0.000220	0.018518	0.000770	0.070939
0.5	0.000467	0.023583	0.000428	0.032638	0.000398	0.023418	0.000605	0.048862
0.6	0.001262	0.10497	0.000819	0.068561	0.000762	0.053341	0.000678	0.047117

Table 8.1 Continued: Comparison between Observer and Observer with Filter

0.7	0.001297	0.102531	0.000878	0.060407	0.000922	0.060850	0.000796	0.050504
0.8	0.001551	0.108286	0.001195	0.085490	0.001174	0.074799	0.001526	0.124463
0.9	0.001360	0.087832	0.001297	0.074141	0.001401	0.087307	0.002362	0.177829
1.0	0.002069	0.170478	0.001927	0.138946	0.001575	0.093823	0.002772	0.199410
1.1	0.002348	0.192327	0.002522	0.158155	0.002194	0.156878	0.002476	0.164370
1.2	0.002363	0.158383	0.002896	0.179571	0.002802	0.197531	0.006437	0.544301
1.3	0.003295	0.198567	0.003030	0.201518	0.006694	0.572136	0.007528	0.654487
1.4	0.006071	0.524654	0.006568	0.572215	0.007497	0.660982	0.006397	0.529314
1.5	0.003311	0.227915	0.007331	0.656777	0.007114	0.630174	0.006735	0.579537
1.6	0.007190	0.642592	0.007405	0.667534	0.006736	0.583052	Brim	Brim
1.7	Brim*	Brim	Brim	Brim	Brim	Brim	N/T	N/T
1.8	N/T*	N/T	N/T	N/T	N/T	N/T	N/T	N/T
1.9	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T
2.0	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T
2.1	N/T	N/T	N/T	N/T	N/T	N/T	N/T	N/T

*  = Better than Observer w/ No Filter

*N/T = Not Tested; Brim = Object reached brim limit of tube.

* Highlighted = 1st overshoot reached the brim limit but object was brought down by hand

*Red = 1st place (lowest-Best); Blue = 2nd place; Purple = 3rd place

8.5 Discussion

Compared to the simulation results in Chapter 7, the real-time application yields less satisfactory results due to the system's nonlinearities, noise, and delay. Nevertheless, the effect of noise can be suppressed using the Median filter.

As shown in Figures 8.3, 8.4, and 8.5, the performances are improved with a filter when compared to Figure 8.1 having no filter. The use of the Median filter with the tracking controller $u(k)$ and the observer further improves the results by removing the noise that can ultimately cause undesired and high levels of oscillations at the target altitude point.

Comparing the two control algorithms, the tracking controller $u(k)$ outperforms the FLC in Chapter 3 at some local target points like $VR = 0.4, 0.5, 0.7,$ and 0.8 when both controllers do not use a filter in real-time. In addition, the tracking controller $u(k)$ performs even better than the FLC mainly at the same local target points when the filter is used by both controllers, based on Table 8.1 and Table 4.4.

CHAPTER 9: CONCLUSION AND FUTURE WORK

9.1 Conclusion

The stability of a floating object in midair inside a tube is important when controlling the altitude of the floating object at a desired altitude point. The use of the Median filter shows successful results in removing the noise which can cause unwanted oscillatory response in the floating object.

Two control techniques are utilized to demonstrate the filter's performance both in simulation and real-time implementation. Based on the statistical results of RSME and MAE, the FLC with the filter provides good results as seen in Table 4.4 and the tracking controller $u(k)$ with the filter provides even better results as seen in Table 8.1.

In summary,

- A second order system is a sufficient model to represent the FBS in a local operating region,
- The dynamic modeling is a better modeling method but the steady state modeling gives insight about the nature of the floating ball plant,
- In real-time application, a constant controller output as at the steady state value does not converge the output of the FBS system because of the characteristics of the floating ball plant seen in the Steady State Modeling,
- The BLS method is sufficient for a local region control but not for the global region of the nonlinear characteristic of the FBS system,
- The Median filter gives better results in real-time application than the Rectangular, Triangular, and Weighted Mean filters,

- Although the Median filter improves the convergence of the system output, the window size of the filter should be kept less than or equal to 5 because of the delay introduced.

Issues Encountered:

1. The major problem encountered in this thesis is that the processing rate is not the same as sampling rate in MATLAB. MATLAB handles the sampling rate apart from the processing rate. The sampling rate is utilized to analyze the data in simulation but not in real-time control. The DAQ's sampling rate is defined by the user within MATLAB.

The processing rate, not the sampling rate, was negatively affecting the system's update speed because the processing rate is too slow. In [11], the MATLAB Help Menu mentions that many data acquisition applications, such as the DAQ, require that data is acquired at a fixed rate, and that the data is processed in some way immediately after it is collected. Even though, the DAQ device has a max sampling rate of 10,000 samples per second, as shown in Table 2.6, the processing rate plays a critical role in making this data useful to update the system. As seen in Chapter 3, the processing rate of the FLC in MATLAB is less than 20Hz. This causes the controller of the FBS to have an overall slow update rate.

Also, the processing rate is dependent upon how fast MATLAB processes *for* and *if* loops, calculates equations, plots in real-time, and generates arrays for variable to be used. Some real causes affecting the processing rate are the real-time plotting and the lack of pre-allocation in the MATLAB code. Once real-time plotting is removed and pre-allocation of the sizes of the variables is added, the processing rate becomes much faster but not enough. Because the FBS needs to be updated with respect to time, the FBS flow signal needs to be understand and analyzed to see where speed is most critical.

2. The dead region of the SHARP sensor adds another problem to the system. The controller gets confused because even though the object was very close to the sensor, the sensor gives an output voltage which corresponds to a far distance value.
3. The nonlinear distance to voltage relationship of the sensor is also a major problem.

9.2 Future Work

The following future work can improve the results of this thesis. This generated future work list is based on the knowledge, experience, and insight learned throughout this thesis. There is space for improvement in the topic of modeling and noisy output feedback. The following tasks are the future work.

1. New linear models that consider the different regions of the sensor nonlinear output need to be developed. For instance, one model for the low voltage region of the sensor voltage output, another model for the central voltage region, and yet another model for the high voltage region. For the model development procedures refer to Section 5.1 of Chapter 5. In addition, the reader should refer to [10] for an example of multiple model development of a similar floating ball plant.
2. The Median filter can be implemented to a FLC that has two input variables ee and de . The variable ee is the error and de is the change in error. The implementation of the Median filter with a FLC of one input variable ee (where $ee = \text{error} = \text{position}$) and one output variable du was achieved in this thesis. Refer to Chapter 3 to obtain insight of the FLC procedures.
3. The Kalman filter can be used with the tracking controller $u(k)$ to improve the convergence of the floating object. The reader can refer to Chapter 7 for the development of the tracking controller $u(k)$.

4. Because of the Recursive Least Square (RLS) method uses of online data, it should be used to model the system instead of the BLS method for a more precise approximation of the coefficients a_1 , a_2 , b_1 , and b_2 . Refer to Section 5.1.2.3 to compare with the procedures of the BLS method.
5. Noise detecting rules for the Rectangular and Triangular Mean Filter need to be formulated and then the results should be compared to the Median Filter method. The reader can refer to Section 4.2.1 for the Rectangular Mean filter and to Section 4.2.2 for the Triangular Mean filter.
6. Noise detecting rules for the Median filter need to be formulated. Then, the Median Filter results with and without the noise detecting rules should be compared. Refer to Chapter 4 for the procedures to develop the Mean and Median filters.

Because this thesis integrates research and education, the next researcher can find this thesis work useful and be able to incorporate this research into his own to expand the engineering field. Since this thesis focuses on a real-life physical system, it can provide assistance with a project of the engineering field and can encourage the assembling of more physical plants.

APPENDIX I

This section presents the data resulting from the shifting of the FBS's raw data by its mean value.

Table A.1: Data Shifting by Mean Value

Shifted Data		
	Input	Output
	u tilde	Y tilde
1	-0.004147589	-0.492999307
2	-0.003027653	-0.487897872
3	-0.001909682	-0.477695003
4	-0.000790717	-0.482796438
5	0.000325192	-0.467492133
6	0.001437804	-0.452187829
7	0.002553712	-0.467492133
8	0.003670665	-0.472593568
9	0.004757348	-0.385869178
10	0.003635726	0.466070422
11	0.002535091	0.210998686
12	0.001656822	0.057955644
13	0.000535201	0.450766118
14	-0.000586421	0.675229246
15	-0.001708043	0.736446463
16	-0.002829664	0.670127811
17	-0.003951286	0.731345028
18	-0.005072908	0.547693378
19	-0.006174998	0.21610012
20	-0.006561977	-0.06447879
21	-0.005715452	-0.237927571
22	-0.004761589	-0.294043353
23	-0.003807725	-0.294043353
24	-0.002737947	-0.370564874
25	-0.001827138	-0.268536179
26	-0.001195361	-0.171608919
27	-0.000889456	-0.120594572
28	-0.001676744	0.022245601
29	-0.002379409	-0.003261573
30	-0.002842386	-0.05427592
31	-0.00258929	-0.115493137
32	-0.001848631	-0.202217527
33	-0.001057829	-0.217521832
34	-0.000282966	-0.212420397
35	0.000439313	-0.197116093
36	0.001230115	-0.217521832
37	0.001763174	-0.15120318
38	0.001814807	-0.100188833
39	0.002008779	-0.110391702
40	0.002443944	-0.135898876
41	0.003053712	-0.166507484
42	0.003844515	-0.217521832
43	0.004347693	-0.146101745
44	0.004053141	-0.074681659
45	0.00338584	-0.013464442
46	0.003352626	-0.095087398
47	0.002508839	0.04265134
48	0.001518802	0.11917286
49	0.000422715	0.200795816
50	-0.000567323	0.11917286
51	-0.00141111	0.04265134
52	-0.000552707	-0.243029005
53	0.000549704	-0.411376351
54	0.001656208	-0.426680656
55	0.002762711	-0.426680656
56	0.003866526	-0.416477786
57	0.004980265	-0.457289264
58	0.006084079	-0.416477786
59	0.007159641	-0.375666308
60	0.008113505	-0.294043353
61	0.008904307	-0.217521832
62	0.008793114	-0.089985963
63	0.00802193	0.017144166
64	0.006931087	0.195694381
65	0.006159903	0.017144166

Table A.1 Continued: Data Shifting by Mean Value

66	0.006882182	-0.197116093
67	0.007604461	-0.197116093
68	0.008039627	-0.135898876
69	0.007412654	-0.023667312
70	0.006709989	-0.003261573
71	0.007319757	-0.166507484
72	0.008248762	-0.278739048
73	0.00908287	-0.232826136
74	0.009586048	-0.146101745
75	0.008668015	0.078361383
76	0.007554527	0.262013033
77	0.006432905	0.695634985
78	0.005311283	0.369143162
79	0.005746449	-0.135898876
80	0.005359469	-0.06447879
81	0.004343872	0.134477165
82	0.003235104	0.241607294
83	0.002113482	0.389548901
84	0.00099186	0.374244597
85	0.000148074	0.04265134
86	0.000779851	-0.171608919
87	0.001752637	-0.304246222
88	0.002828198	-0.375666308
89	0.003897976	-0.370564874
90	0.004826981	-0.278739048
91	0.005458758	-0.171608919
92	0.005279038	-0.084884529
93	0.004542425	0.006941296
94	0.003937816	-0.028768747
95	0.003357257	-0.033870181
96	0.002802637	-0.038971616
97	0.00256222	-0.079783094
98	0.002007599	-0.038971616
99	0.001037057	0.108969991
100	-8.4565E-05	0.40995464
101	-0.001206187	0.629316333
102	-0.002327808	0.726243593
103	-0.00344943	0.823170853
104	-0.004571051	0.879286635

105	-0.005692673	0.889489505
106	-0.006814295	0.848678027
107	-0.007935916	0.716040724
108	-0.009057538	0.557896247
109	-0.01017916	0.644620638
110	-0.011300781	0.751750767
111	-0.012422403	0.863982331
112	-0.013544024	0.894590939
113	-0.014665646	0.40995464
114	-0.015332947	-0.013464442
115	-0.015676082	-0.069580225
116	-0.015970635	-0.074681659
117	-0.01508996	-0.253231875
118	-0.013997929	-0.390970612
119	-0.012888882	-0.436883525
120	-0.011774045	-0.462390699
121	-0.010664998	-0.436883525
122	-0.009589436	-0.375666308

APPENDIX II

This section presents the MATLAB scripts used to generate the plots throughout out this thesis.

CHAPTER 3 CODES:

Code 3.1: For Figure 3.5: Three-Rule FLC with Modified Error Equation Code of Three-Rule Fuzzy Logic Controller with Modified Error Equation

```
clear all;
clc;
close all;
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
duration=0.001;
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'InputType','SingleEnded');
Samples_Per_Trigger = setverify(ai,'SamplesPerTrigger',duration*ai.SampleRate);
set(ai,'TriggerType','Immediate');
%% Initial controller
u= 1.795; %%Controller as the voltage input to the fan
uup=1.8; %%Controller as the maximum voltage
udown=1.6; %%Controller as the minimum voltage
VR=1.8; %% Desired position
%% Define your universe and fuzzy sets here%%
Err_X=(-2:.01:2); %% Universe
N=trapmf(Err_X, [-2 -2 -1 0]); %% Negative Set
Z=trimf(Err_X, [-1 0 1]); %% Zero Set
P=trapmf(Err_X, [0 1 2 2]); %% Positive Set
iterations = 2000;
%Pre-allocation;
PV = zeros(1, iterations);
past_speed = zeros(1, iterations);
eeA = zeros(1, iterations);
endtime3 = zeros(1, iterations);
endtime2 = zeros(1, iterations);
uu = zeros(1, iterations);
%%Initial Values
PV(1) = 0.3;
```

```

PV(2) = 0.3;
uu(1) = u;
uu(2) = u;
%%Error calculation initial values
past_speed(1)=0;
past_speed(2)=0;
%% Processing Speed measured by tic and toc
tic
for i=3: iterations;
    start(ai);
    wait(ai,0.2);
    data=getdata(ai,Samples_Per_Trigger);
    V=data(Samples_Per_Trigger);
    endtime3(i) = toc;
    PV(i)=V; %% Record your position
    %%Error calculation for Real FBS system%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [ee ,past_speed(i)] = err_cal_R1(VR,V,PV(i-1),past_speed(i-1));
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    eeA(i) = ee;
    %%In sensor working region
    DOF=interp1(Err_X,P,ee); %%DOF of Rule 1
    DU1=min(N,DOF); %%B1'
    DOF=interp1(Err_X,Z,ee); %%DOF of Rule 2
    DU2=min(Z,DOF); %%B2'
    DOF=interp1(Err_X,N,ee); %%DOF of Rule 3
    DU3=min(P,DOF); %%B3'
    DU=max(max(DU1, DU2),DU3); %%MAX of B1', B2', B3'
    du=0.1*defuzz(Err_X,DU,'centroid');
    u=u+du;
    %% voltage restrictions
    if u > uup;
        u = uup;
    elseif u < udown;
        u = udown;
    end;
    putsample(ao,u);
    endtime2(i) = toc;
    uu(i)=u; %% Record your controller
end;
endtime1 = toc;
% delete (ai);
% clear ai;

```

Code 3.2: For Figure 3.5 and Figure 3.6

Code for the Error Function

Code name: err_cal_R1.m

```
%% Function with acceleration part
function [Error,Speed_present] = err_cal_R1(V_desired, V_present, V_previous,
Speed_previous)
    Speed_present = V_present - V_previous; % Speed in terms of volts per index of matrix
    Acceleration = Speed_present - Speed_previous;
    QV = V_present+ ((2.2*Speed_present)+(0*Acceleration))/0.15;
    Error = 0.15*(QV - V_desired);
```

Code 3.3: For Figure 3.6: Five-Rule FLC with Modified Error Equation

Code of Five-Rule Fuzzy Logic Controller with Modified Error Equation:

```
clear all;
clc;
close all;
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
duration=0.001;
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'InputType','SingleEnded');
Samples_Per_Trigger = setverify(ai,'SamplesPerTrigger',duration*ai.SampleRate);
set(ai,'TriggerType','Immediate');
%% Initial controller
u= 1.795; %%Controller as the voltage input to the fan
uup=1.8; %%Controller as the maximum voltage
udown=1.6; %%Controller as the minimum voltage
VR=1.6; %% Desired position
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Err_X=(-2:0.1:2); %% Universe
VN=trapmf(Err_X, [-2 -2 -1.5 -0.5]);%% Very Negative Set
N=trimf(Err_X, [-1 -0.5 0]); %% Negative Set
Z=trimf(Err_X, [-0.5 0 0.5]); %% Zero Set
P=trimf(Err_X, [0 0.5 1]); %% Positive Set
VP=trapmf(Err_X, [0.5 1.5 2 2]); %% Very Positive Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iterations = 2000;
%Pre-allocation;
```

```

PV = zeros(1, iterations);
past_speed = zeros(1, iterations);
% eeA = zeros(1, iterations);
endtime3 = zeros(1, iterations);
endtime2 = zeros(1, iterations);
uu = zeros(1, iterations);
%%Initial Values
PV(1) = 0.3;
PV(2) = 0.3;
uu(1) = u;
uu(2) = u;
%%Error calculation initial values
past_speed(1)=0;
past_speed(2)=0;
%% Processing Speed measured by tic and toc
tic
for i=3: iterations;
    start(ai);
    wait(ai,0.2);
    data=getdata(ai,Samples_Per_Trigger);
    V=data(Samples_Per_Trigger);
    endtime3(i) = toc;
    PV(i)=V; %% Record your position
    %%Error calculation for Real FBS system%%
    [ee ,past_speed(i)] = err_cal_R1(VR,V,PV(i-1),past_speed(i-1));
    %%
    DOF=interp1(Err_X,VN,ee); %%DOF of Rule 1
    DU1=min(VP,DOF); %%B1'
    DOF=interp1(Err_X,N,ee); %%DOF of Rule 2
    DU2=min(P,DOF); %%B2'
    DOF=interp1(Err_X,Z,ee); %%DOF of Rule 3
    DU3=min(Z,DOF); %%B3'
    DOF=interp1(Err_X,P,ee); %%DOF of Rule 4
    DU4=min(N,DOF); %%B4'
    DOF=interp1(Err_X,VP,ee); %%DOF of Rule 5
    DU5=min(VN,DOF); %%B5'
    DU = max(max(max(max(DU1, DU2),DU3),DU4),DU5);
    du=0.1*defuzz(Err_X,DU,'centroid');
    %%
    u=u+du;
    %% voltage restrictions
    if u > uup;
        u = uup;
    elseif u < udown;
        u = udown;
    end;
end;

```

```

    putsample(ao,u);
    endtime2(i) = toc;
    uu(i)=u; %% Record your controller
end;
endtime = toc;
Y1(1:iterations) = VR;
Y1_new = PV;
N = iterations;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
MAE=sum(abs(Y1-Y1_new))/N;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f,RMSE,MAE);
h_tx4=text(1000,VR-0.8,S_text,'FontSize',10,'FontWeight','bold');
legend('PV Student','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.6','FontSize',20);
axis([0 2000 0.2 3]);
%Plotting Five Rule Regions
plot(Err_X,VN,'-or')
hold on
plot(Err_X,VP,'-ob')
plot(Err_X,N,'-omagenta')
plot(Err_X,P,'-ogreen')
plot(Err_X,Z,'-oblack')
grid on
title('VN N Z P VP Regions Membership Functions','FontSize',20);
xlabel({'Error Universe'},'FontSize',12);
ylabel({'Degree of Membership'},'FontSize',12);
legend('VN','VP','N','P','Z')
S_text=sprintf('25Percent');
h_tx4=text(-0.4,0.05,S_text,'FontSize',10,'FontWeight','bold');
h_tx4=text(0.1,0.1,S_text,'FontSize',10,'FontWeight','bold');
S_text=sprintf('50Percent');
h_tx4=text(-0.23,0.5,S_text,'FontSize',10,'FontWeight','bold');
=====
*****
=====

```

CHAPTER 4 CODES:

Code 4.1: For Figure 4.1: Rectangular Mean Filter of Window Size 3

Code of Rectangular Mean Filter Simulation with a window size 3

-Load any noisy data as *PV*

```
clear all;
clc
close all
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:1:t_end;
N = length(t);
Y1 = Y;
%% Mean Filtering
% 1. Regular Mean filter (Rectangular)
h=[1 1 1]'; %regular mean filter. *Column array
n_window = length(h); %window size = 3
h=h/sum(h);
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size
Y1_new=Y1;
%Mean Filtering of the noisy signal
for k=L+1:N-L-1 % from 2 to only up to 998:
    Y_inwindow=Y1(k-L:k+L);
    %T1=mean(Y_inwindow,2); % as for h=[1 1 1];
    T1=Y_inwindow*h;
    Y1_new(k)=T1;
end;
rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
mae_1=sum(abs(Y1-Y1_new))/N;
figure(1)
plot(t,Y1)
hold
h_plot3=plot(t,Y1_new,'r');
axis([0,t_end,0,Amplitude+0.25]);
S_text=sprintf('[RMSE=%8.6f,MAE=%8.6f]',rmse_1,mae_1);
```

```

    h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
    legend('Sensor Signal', 'Filtered Signal Mean WS3')
    grid on
    xlabel({'Loop Iterations (i)'},'FontSize',12);
    ylabel({'Sensor Voltage (V)'},'FontSize',12);
    title('Noisy & Filtered Signals ', 'FontSize',20);
    %axis([0 2000 0.2 3.5]);

```

Code 4.2: For Figure 4.2: Rectangular Mean Filter of Window Size 5
Code of Rectangular Mean Filter Simulation with window size 5

-Load any noisy data as *PV*

```

clear all;
clc
close all
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:t_end;
N = length(t);
Y1 =Y;
%% Filter definition
% 1. Regular Mean filter (Rectangular)
h=[1 1 1 1 1]'; %regular mean filter. *Column array
n_window = length(h); %window size = 3
h=h/sum(h);
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size
Y1_new=Y1;
%% Mean Filtering of the noisy signal
for k=L+1:N-L-1 % from 3 to only up to 997:
    Y_inwindow=Y1(k-L:k+L);
    T1=Y_inwindow*h;
    Y1_new(k)=T1;
end;
rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
mae_1=sum(abs(Y1-Y1_new))/N;
figure(1)
plot(t,Y1)

```

```

hold
h_plot3=plot(t,Y1_new,'r');
axis([0,t_end,0,Amplitude+0.25]);
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f',rmse_1,mae_1);
h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
legend('Sensor Signal', 'Filtered Signal Mean WS5')
grid on
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Noisy & Filtered Signals ', 'FontSize',20);

```

Code 4.3: For Figure 4.3: Triangular Mean Filter of Window Size 3

Code of Triangular Mean Simulation with window size 3

-Load any noisy data as *PV*

```

clear all;
clc
close all
h=[0.75 1 0.75]'; %regular mean filter. *Column array
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:1:t_end;
N = length(t);
Y1 =Y;
%% Filter definition
% Weighted Mean filter (3) (Includes Triangle)
%h=[2 8 0]'; %regular mean filter. *Column array
h=h/sum(h);
n_window = length(h); %window size = 3
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size for the For loop
Y1_new=Y1;
%% Mean Filtering of the noisy signal
for k=L+1:N-L-1 % from 3 to only up to 997
Y_inwindow=Y1(k-L:k+L);
T1=Y_inwindow*h;
Y1_new(k)=T1;
end;

```

```

    rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
    mae_1=sum(abs(Y1-Y1_new))/N;
figure(1)
plot(t,Y1)
hold
    h_plot3=plot(t,Y1_new,'r');
    axis([0,t_end,0,Amplitude+0.25]);
    S_text=sprintf('[RMSE=%08.6f,MAE=%08.6f]',rmse_1,mae_1);
    h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
legend('Sensor Signal', 'Weighted Signal Mean WS3')
grid on
xlabel({'Loop Iterations (i)'}, 'FontSize',12);
ylabel({'Sensor Voltage (V)'}, 'FontSize',12);
title('Noisy & Filtered Signals ', 'FontSize',20);

```

Code 4.4: For Figure 4.4: Weighted Mean Filter of Window Size 3 Code of Weighted Mean Simulation with window size 3

-Load any noisy data as *PV*

```

clear all;
clc
close all
h=[4 6 0]'; %regular mean filter. *Column array
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:1:t_end;
N = length(t);
Y1 =Y;
%% Filter definition
% Weighted Mean filter (3) (Includes Triangle)
%h=[2 8 0]'; %regular mean filter. *Column array
h=h/sum(h);
n_window = length(h); %window size = 3
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size for the For loop
Y1_new=Y1;
%% Mean Filtering of the noisy signal
for k=L+1:N-L-1 % from 3 to only up to 997

```

```

    Y_inwindow=Y1(k-L:k+L);
    T1=Y_inwindow*h;
    Y1_new(k)=T1;
end;
rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
mae_1=sum(abs(Y1-Y1_new))/N;
figure(1)
plot(t,Y1)
hold
h_plot3=plot(t,Y1_new,'r');
axis([0,t_end,0,Amplitude+0.25]);
S_text=sprintf('[RMSE=%8.6f,MAE=%8.6f]',rmse_1,mae_1);
h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
legend('Sensor Signal', 'Weighted Signal Mean WS3')
grid on
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Noisy & Filtered Signals ','FontSize',20);

```

Code 4.5: For Figure 4.5: Median Filter of Window Size 3

Code of Median Filter Simulation with window size 3

-Load any noisy data as *PV*

```

clear all;
clc
close all
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:1:t_end;
N = length(t);
Y1 =Y;
%% Median Filter definition
n_window = 3; % Median Window Size
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size
Y1_new=Y1;
%% Median Filtering of the noisy signal
for k=L+1:N-L-1 % from 2 to only up to 501

```



```

    Y_inwindow=Y1(k-L:k+L);
    T1=median(Y_inwindow,2);
    Y1_new(k)=T1;
end;
rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
mae_1=sum(abs(Y1-Y1_new))/N;
%% plotting
figure(1)
plot(t,Y1)
hold
h_plot3=plot(t,Y1_new,'r');
axis([0,t_end,0,Amplitude+0.25]);
S_text=sprintf('[RMSE=%8.6f,MAE=%8.6f]',rmse_1,mae_1);
h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
legend('Sensor Signal', ' Filtered Signal Median WS3')
grid on
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Noisy & Filtered Signals ','FontSize',20);
grid on

```

**Code 4.6: For Figure 4.6: Median Filter of Window Size 5
Code of Median Filter Simulation with a window size 5**

-Load any noisy data as *PV*

```

clear all;
clc
close all
%% Sensor Signal - Not in real time
%% Sensor Signal (Noisy) Definition:
% get sensor data, i.e. PV voltage
%load 'C:\ Documents and Settings\PV.mat'; %Here is an example of loading the noisy data at
%directory C:\ Documents and Settings\PV.mat

Y = PV;
Amplitude = max(PV);
t_end = length(Y); % Assigning index of array to a pseudo-time value, i.e. index i = time t,
%where i=t
t = 1:t_end;
N = length(t);
Y1 =Y;
%% Median Filter definition
n_window = 5; % Median Window Size
L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1, index2,
%index3] for the window size
Y1_new=Y1;

```

```

%% Median Filtering of the noisy signal
for k=L+1:N-L-1 % from 2 to only up to 501
    Y_inwindow=Y1(k-L:k+L);
    T1=median(Y_inwindow,2);
    Y1_new(k)=T1;
end;
rmse_1=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N;
mae_1=sum(abs(Y1-Y1_new))/N;
%% plotting
figure(1)
plot(t,Y1)
hold
h_plot3=plot(t,Y1_new,'r');
axis([0,t_end,0,Amplitude+0.25]);
S_text=sprintf('[RMSE=%8.6f,MAE=%8.6f]',rmse_1,mae_1);
h_tx3=text(500,1.7,S_text,'FontSize',10,'FontWeight','bold');
legend('Sensor Signal', ' Filtered Signal Median WS5')
grid on
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Noisy & Filtered Signals ','FontSize',20);
grid on

```

**Code 4.7: For Figure 4.9: Three-Rule FLC with Median Filter of Window Size 3
Code of Real-Time application for Median Filter Simulation with Three-Rule FLC**

```

clear all;
clc;
close all;
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
duration=0.001;
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'InputType','SingleEnded');
Samples_Per_Trigger = setverify(ai,'SamplesPerTrigger',duration*ai.SampleRate);
set(ai,'TriggerType','Immediate');
%% Initial controller
u= 1.795; %%Controller as the voltage input to the fan
uup=1.8; %%Controller as the maximum voltage
udown=1.6; %%Controller as the minimum voltage

```

```

VR=1.7; %% Desired position
%% Define your universe and fuzzy sets here%%
Err_X=(-2:0.01:2); %% Universe
N=trapmf(Err_X, [-2 -2 -0.01 0]);%% Negative Set
Z=trimf(Err_X, [-0.01 0 0.01]); %% Zero Set
P=trapmf(Err_X, [0 0.01 2 2]); %%Positive Set
iterations = 2000;
%%Pre-allocation;
PV = zeros(1, iterations);
uu = zeros(1, iterations);
past_speed = zeros(1, iterations);
Y1_Filtered = zeros(1, iterations);
endtime3 = zeros(1, iterations);
endtime2 = zeros(1, iterations);
eeA = zeros(1,iterations);
%%Initial Values
uu(1) = u; %u arbitrary initial value for array and plot only; not for actual computation
uu(2) = u; %u arbitrary initial value for array and plot only; not for actual computation
%%Error calculation initial values
past_speed(1)=0; %Past Speed arbitrary initial value for array and plot only; not for actual
%computation
past_speed(2)=0; %Past speed initial value. For actual computation
PV(1) = 0.34; %Noisy signal arbitrary initial value for array and plot only; not for actual
%computation
PV(2) = 0.34; %Noisy signal arbitrary initial value for array and plot only; not for actual
%computation
Y1_Filtered(1)=0.34; %Filtered signal arbitrary initial value. For actual computation
Y1_Filtered(2)=0.34; %Filtered signal arbitrary initial value. For actual computation
tic
for i=3:iterations;
    start(ai);
    wait(ai,0.2);
    data=getdata(ai,Samples_Per_Trigger);
    V=data(Samples_Per_Trigger);
    endtime3(i) = toc;
    PV(i)=V; %% Record your position
    if i>=3; % Median Filter waiting for 1st, 2nd, and 3rd values
        %% Median Filter definition
        n_window = 3; % Median Window Size
        L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1,
        %index2, index3] for the window size
        %%Median Filtering of the noisy signal
        m = i - 1; % from 2 because this is a window size 3 filter
        Y_inwindow=PV(m-L:m+L);
        T1=median(Y_inwindow,2);
        Y1_Filtered(m)=T1;
    end
end

```

```

end
%%Error calculation for Real FBS system
[ee ,past_speed(i)] = err_cal_R1(VR,Y1_Filtered(i-1),Y1_Filtered(i-2),past_speed(i-1));
eeA(i)=ee;
DOF=interp1(Err_X,P,ee); %%DOF of Rule 1
DU1=min(N,DOF); %%B1'
DOF=interp1(Err_X,Z,ee); %%DOF of Rule 2
DU2=min(Z,DOF); %%B2'
DOF=interp1(Err_X,N,ee); %%DOF of Rule 3
DU3=min(P,DOF); %%B3'
DU=max(max(DU1, DU2),DU3); %%MAX of B1', B2', B3'
du=0.1*defuzz(Err_X,DU,'centroid');
u=u+du;
%% voltage restrictions
if u > uup;
    u = uup;
elseif u < udown;
    u = udown;
end;
putsample(ao,u);
endtime2(i) = toc;
uu(i)=u; %% Record your controller
end;
endtime1 = toc;
Y1(1:iterations) = VR;
Y1_new = Y1_Filtered;
N1 = iterations;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
MAE=sum(abs(Y1-Y1_new))/N1;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f,RMSE,MAE);
h_tx4=text(1000,VR-1,S_text,'FontSize',10,'FontWeight','bold');
legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.7','FontSize',20);
axis([0 2000 0.2 3]);

```

**Code 4.8: For Figure 4.10: Five-Rule FLC with Median Filter of Window Size 3
Code of Real-Time application for Median Filter Simulation with Five-Rule FLC**

```

clear all;
clc;
close all;
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
duration=0.001;
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'InputType','SingleEnded');
Samples_Per_Trigger = setverify(ai,'SamplesPerTrigger',duration*ai.SampleRate);
set(ai,'TriggerType','Immediate');
%% Initial controller
u= 1.795; %%Controller as the voltage input to the fan
uup=1.8; %%Controller as the maximum voltage
udown=1.6; %%Controller as the minimum voltage
VR=1.2; %% Desired position
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Err_X=(-2:0.01:2); %% Universe
VN=trapmf(Err_X, [-2 -2 -0.5 -0.01]); %% Very Negative Set
N=trimf(Err_X, [-0.5 -0.01 0]); %% Negative Set
Z=trimf(Err_X, [-0.01 0 0.01]); %% Zero Set
P=trimf(Err_X, [0 0.01 0.5]); %% Positive Set
VP=trapmf(Err_X, [0.01 0.5 2 2]); %% Very Positive Set
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
iterations = 2000;
%Pre-allocation;
PV = zeros(1, iterations);
uu = zeros(1, iterations);
past_speed = zeros(1, iterations);
Y1_Filtered = zeros(1, iterations);
endtime3 = zeros(1, iterations);
endtime2 = zeros(1, iterations);
eeA = zeros(1, iterations);
%%Initial Values
uu(1) = u; %u arbitrary initial value for array and plot only; not for actual computation
uu(2) = u; %u arbitrary initial value for array and plot only; not for actual computation
%%Error calculation initial values

```

```

past_speed(1)=0; %Past Speed arbitrary initial value for array and plot only; not for actual
%computation
past_speed(2)=0; %Past speed initial value. For actual computation
PV(1) = 0.34; %Noisy signal arbitrary initial value for array and plot only; not for actual
%computation
PV(2) = 0.34; %Noisy signal arbitrary initial value for array and plot only; not for actual
%computation
Y1_Filtered(1)=0.34; %Filtered signal arbitrary initial value. For actual computation
Y1_Filtered(2)=0.34; %Filtered signal arbitrary initial value. For actual computation
tic
for i=3:iterations;
    start(ai);
    wait(ai,0.2);
    data=getdata(ai,Samples_Per_Trigger);
    V=data(Samples_Per_Trigger);
    endtime3(i) = toc;
    PV(i)=V; %% Record your position
    if i>=3; % Median Filter waiting for 1st, 2nd, and 3rd values
        %% Median Filter definition
        n_window = 3; % Median Window Size
        L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1,
        %index2, index3] for the window size
        %% Median Filtering of the noisy signal
        m = i - 1; % from 2 because this is a window size 3 filter
        Y_inwindow=PV(m-L:m+L);
        T1=median(Y_inwindow,2);
        Y1_Filtered(m)=T1;
    end
    %%Error calculation for Real FBS system%%%%%%%%%%
    [ee ,past_speed(i)] = err_cal_R1(VR,Y1_Filtered(i-1),Y1_Filtered(i-2),past_speed(i-1));
    %%%%%%%%%%%
    eeA(i) = ee;
    DOF1=interp1(Err_X,VN,ee); %%DOF of Rule 1
    DU1=min(VP,DOF1); %%B1'
    DOF2=interp1(Err_X,N,ee); %%DOF of Rule 2
    DU2=min(P,DOF2); %%B2'
    DOF3=interp1(Err_X,Z,ee); %%DOF of Rule 3
    DU3=min(Z,DOF3); %%B3'
    DOF4=interp1(Err_X,P,ee); %%DOF of Rule 4
    DU4=min(N,DOF4); %%B4'
    DOF5=interp1(Err_X,VP,ee); %%DOF of Rule 5
    DU5=min(VN,DOF5); %%B5'
    DU = max(max(max(max(DU1, DU2),DU3),DU4),DU5);
    du=0.1*defuzz(Err_X,DU,'centroid');
    u=u+du;
    %% voltage restrictions

```

```

if u > uup;
    u = uup;
elseif u < udown;
    u = udown;
end;
putsample(ao,u);
endtime2(i) = toc;
uu(i)=u; %% Record your controller
end;
endtime1 = toc;
Y1(1:iterations) = VR;
Y1_new = Y1_Filtered;
N1 = iterations;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
MAE=sum(abs(Y1-Y1_new))/N1;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f',RMSE,MAE);
h_tx4=text(1000,VR-1,S_text,'FontSize',10,'FontWeight','bold');
legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.7','FontSize',20);
axis([0 2000 0 4]);

```

CHAPTER 5 CODE:

Code 5.1: For Figure 5.8: Output Modeling by Recursive $\hat{y}(k)$ using $y(k)$ Real Output Values

Code of Model Approximation

-Load the shifted u_tilde data

-Load the shifted y_tilde data

```

clear all
clc
close all
% get input data, i.e. fans voltage = controller voltage u
%load 'C:\ Documents and Settings\u_tilde.mat'; %Here is an example of loading the shifted
%input data at directory C:\ Documents and Settings\u_tilde.mat

%get output data, i.e. sensor's values V

```

```

%load 'C:\ Documents and Settings\y_tilde.mat'; %Here is an example of loading the shifted
%output data at directory C:\ Documents and Settings\y_tilde.mat

%% Second New Model of a control system equation
% y(k) = a1*y(k-1)+a2*y(k-2)      % **Change Here if newer model**
%      b2*u(k-1)+b3*u(k-2)      % **Change Here if newer model**
% unknown parameters are a1, a2, b2, and b3    % **Change Here if newer model**
%%
y = y_tilde; %output y_tilde renamed to y
u = u_tilde;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Output matrix of all the y's
Y1 = zeros(1,118); %pre-allocating space
for k = 1:1:118
Y1(k) = y(k+4); % plus 4 in order to align it with the table in the separate sheet
end          % I believe we are not using the Y1(0)=0.314269,Y1(-1)=0.319371,Y1(-
%2)=0.309168,Y1(-3)=0.304066.
Y = Y1'; %transpose of Y1
%Matrix [X1(1),X1(2),X1(3),...,X1(M)]      %% M here starts with 1 in order to collect an all
%inclusive matrix so no need to offset the matrix yet
% X1(1) = [y(1-1)    X1(2) = [y(2-1)  ...  X1(M) = [y(M-1)
%      y(1-2)        y(2-2)        y(M-2)
%      u(1)          u(2)          u(M)
%      u(1-1)]      u(2-1)]      u(M-1)]
X1 = zeros(4,118); %pre-allocating space
for k = 1:1:118
X1(:,k) = [y((k-1)+4);    % **Change Here if newer model**
          y((k-2)+4);    % **Change Here if newer model**
          u((k-1)+4);    % **Change Here if newer model**
          u((k-2)+4)];  % **Change Here if newer model**
end
%Matrix [X(1)';
%      X(2)';
%      X(3)';
%      .
%      .
%      .
%      X(M)']
X_T = X1';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% M>=4, i.e. M=4,5,6,7,8,9,10,11,12,13,.....,118. This is the M maximum
%% value per batch (refer to matrix output Y1 and matrix X1.)
Y_M = zeros(118,118); %pre-allocating space
for M = 1:1:118 %Start M batch from 1
Y_M((1:M),M) = Y(1:M); %I will start the batch size with M=1
end

```



```

%Matrix Phi(M)' = [X1(1),X1(2),X1(3),...,X1(M)]
% M=1 M=2 M=3 M=4 ....
% Phi(1)=[X'(1)] Phi(2)=[X'(1) Phi(3)=[X'(1) Phi(4)=[X'(1) Phi(5)=[X'(1) ... Phi(M)=[X'(1)
% X'(2) X'(2) X'(2) X'(2) X'(2) X'(2)
% X'(3) X'(3) X'(3) X'(3) X'(3)
% X'(4) X'(4) X'(4)
% X'(5) X'(5)
% '
% '
% X'(M)
Phi_M = zeros(118,472);
for M=1:1:118
Phi_M((1:M),(4*(M-3)+9):(4*(M-3)+12))) = X_T((1:M),:); %all the phi for each M=>4. %The
indexing here is modified
end
Phi_M_T = Phi_M'; %transpose of Phi_4
%% Theta_hat formula
% theta_hat = [Phi(M)'*Phi(M)]^-1 *Phi(M)'*Y(M)
Theta_hat = zeros(4,118);
for M = 1:1:118 %Will start at M=1
Theta_hat((1:4),M) = ((Phi_M_T(((4*(M-3)+9):(4*(M-3)+12)),(1:M))*Phi_M((1:M),(4*(M-3)+9):(4*(M-3)+12))))^-1*Phi_M_T(((4*(M-3)+9):(4*(M-3)+12)),(1:M))*Y_M((1:M),M);
%The indexing here is modified
end %we can see that Theta_hat starts at 13th row because the upper rows were reserved for
%M=3,2,1, where each used up four rows
%unknown parameters now known:
a1=zeros(1,118); %pre-allocating space % **Change Here if newer model**
a2=zeros(1,118); %pre-allocating space % **Change Here if newer model**
b2=zeros(1,118); %pre-allocating space % **Change Here if newer model**
b3=zeros(1,118); %pre-allocating space % **Change Here if newer model**
for i = 1:1:118
a1(1,i) = Theta_hat(1,i); % **Change Here if newer model**
a2(1,i) = Theta_hat(2,i); % **Change Here if newer model**
b2(1,i) = Theta_hat(3,i); % **Change Here if newer model**
b3(1,i) = Theta_hat(4,i); % **Change Here if newer model**
end
y_hat=zeros(118,122);
y2_hat=zeros(118,122);
%In the above command we
% y_hat(1) = 0,y_hat(2) = 0,y_hat(3) = 0,y_hat(4) = 0
for k=1:1:118;
for i=1:1:118; %Start with M=1
y_hat(k,i+4) = a1(k)*y((i-1)+4)+a2(k)*y((i-2)+4)+b2(k)*u((i-1)+4)+b3(k)*u((i-2)+4); %y
%is a column matrix. y_hat is a row matrix % **Change Here if newer model**
end
end
end

```

```

% figure(1)
% plot(y);
% hold on;
% plot(y_hat(75,:), 'black');
% plot(y_hat(97,:), 'red');
% plot(y_hat(118,:), 'green');
% legend('y', 'PV_7_5', 'PV_9_7', 'PV_1_1_8');
for k=1:1:118;
    for i=1:1:118; %Start with M=1
        y2_hat(k,4) = y(4);
        y2_hat(k,3) = y(3);
        y2_hat(k,i+4) = a1(k)*y2_hat(k,(i-1)+4)+a2(k)*y2_hat(k,(i-2)+4)+b2(k)*u((i-
1)+4)+b3(k)*u((i-2)+4); % Purely y_hat result. Not mixed with y real values
    end
end
figure(2)
plot(y);
hold on;
plot(y2_hat(75,:), 'black');
plot(y2_hat(97,:), 'green');
plot(y2_hat(108,:), 'red');
grid on
legend('y', 'yhat2_7_5', 'yhat2_9_7', 'yhat2_1_0_8');
xlabel({'Loop Iterations (k)'}, 'FontSize', 12);
ylabel({'Output y and Approximations yhat'}, 'FontSize', 12);
title('Model Approximation', 'FontSize', 20);
%% Results: Get from Theta_hat with the corresponding M value
% M=75
% a1 = 1.12565796098561;
% a2 = -0.284272073029649;
% b2 = 20.8830928179292;
% b3 = -16.8060121992699;
% M=97
% a1 = 1.12901962240891;
% a2 = -0.333642528094603;
% b2 = 0.579010194665750;
% b3 = 3.80311266260797;
% M=108
% a1 = 1.287852678797189
% a2= -0.326735013498350
% b2= 33.141803846436060
% b3 = -30.506954781341744
=====
*****

```

CHAPTER 6 CODE:

Code 6.1: For Figure 6.3: Three-Rule FLC $VR = 0.4$ Results, Figure 6.4: Three-Rule FLC $VR = 0.7$ Results, and Figure 6.5: Three-Rule FLC $VR = 0.8$ Results

-For each plot change the desired voltage from $VR = 0.4$ to $VR = 0.7$ to $VR = 0.8$

Code of Fuzzy Logic simulation

```
clear all;
clc;
close all;
%%
Array_Size = 500;
Gain = 0.00015;
%% Desired position
VR=0.1; %The u stays within [0,2] and PV stays within [0,3] only for VR=[0.3,0.87]
%% Initial controller
u = zeros(1,Array_Size);
u(1)=0.0; %The 0 values
u(2)=0.0; % i am using u(1) and u(2) because we are starting the for loop at i=3 Even though the
%new model starts u with only one previous value
uup=2; %%Controller for above region
udown=0; %%Controller for below region
%% New Data Second model equation parameters: a1,a2,b2,b3 - With Mean Values du and dy
%Corrected
% M=108 batch
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% Define your universe and fuzzy sets here
Err_X=(-1.6:0.1:1.6);
N=trapmf(Err_X, [-1.6 -1.6 -0.5 0]);
Z=trimf(Err_X, [-0.5 0 0.5]);
P=trapmf(Err_X, [0 0.5 1.6 1.6]);
%% Initial value of PV data, i.e. the sensor data. This is required because
%the new model uses two previous values of PV out from the start
PV = zeros(1,Array_Size);
PV(1)=0.7; %the middle between 0.3 and 1.7 of the if loop
PV(2)=0.7;
for i=3:Array_Size % because the new model uses two previous values out from the start
    if PV(i-1)<-0.5 %Outside the sensor region-> >80cm
        u(i) = uup;
    elseif PV(i-1)>=- 0.5 && PV(i-1)<=3.1 %%In sensor working region, put your fuzzy
%controller here
        ee=PV(i-1)-VR;
```

```

DOF = interp1(Err_X,N,ee);
DU1 = min(P,DOF);
DOF = interp1(Err_X,Z,ee);
DU2 = min(Z,DOF);
DOF = interp1(Err_X,P,ee);
DU3 = min(N,DOF);
DU = max(max(DU1,DU2),DU3);
du = Gain*defuzz(Err_X,DU,'centroid');
u(i)=u(i-1)+du;
% u limit
if u(i)>uup
    u(i) = uup;
end
if u(i)<udown
    u(i) = udown;
end
else % PV(i-1)>3.1
    u(i)=udown;
end
PV(i)=a1*PV(i-1)+a2*PV(i-2)+b2*u(i)+b3*u(i-1); %% Simulated System of floating ball
%system
end
figure(1);
plot(PV);
%axis([200 500 (VR-0.0001) (VR+0.0001)])
% figure(2);
% plot(u);

```

CHAPTER 7 CODE:

Code 7.1: For Figure 7.4: Observer $VR = 0.4$ Results, Figure 7.5: Observer $VR = 0.5$ Results, and Figure 7.6: Observer $VR = 0.7$ Results

-For each plot just change from $VR = 0.4$ to $VR = 0.5$ to $VR = 0.7$

Code of Observer simulation

```

clear all
clc
close all
%% From: New Data Second New model equation parameters: a1,a2,b2,b3 from Step 2.1- With
%%Mean Values du and dy Corrected
% y(k) = a1*y(k-1) + a2*y(k-2)
%       + b2*u(k-1) + b3*u(k-2)
% The coefficients obtained via model of a control system equation and

```

```

% through the Batch Least Square method
% M=108
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% Phi, Gamma, H, J, T results only.
% Writing the G(z) function into a transfer function format for ssdata
% command
Ts = 0.0067; % The maximum allowable sampling time for the analog output object of the DAQ
%6008
                % - Maximum update rate 150 Hz, software-timed
Phi = [1.287852678797189, -0.653470026996700
        0.5                , 0];
Gamma = [ 8
          0];
H = [4.142725480804508, -7.626738695335436];
J = 0;
%% N
N = [3.036227048441438
     1.518113524220719
     0.014756949540777];
Nx = N(1:2);
Nu = N(3);
%% controller Gain K
p_c = [0.9,0.9]; %desired poles for controller gain K
K = acker(Phi,Gamma,p_c); %controller gain K
%% Observer Lp
p_o = [0.2,0.2]; % desired poles for observer Lp
Lp = acker(Phi',H',p_o)'; %observer Lp
%%
%Pre-allocation:
size = 200;
r = zeros(1,size);
y = zeros(1,size);
PVR = zeros(1,size);
Py = zeros(1,size);
%Initialization:
u(:,1) = 0; %This is a scalar or matrix 1x1
x(:,1)=[0.0;0.0]; %This matrix is always 2x1
barx(:,1) = [0;0]; %This matrix is always 2x1. Left value is barx1. Right value is barx2
%barx(:,1) = [-0.9652;-0.51436]; %This matrix is always 2x1
RDu(:,1) = 0;
VR = 1;
for k = 1:size;
    r(k) = VR - 0.797065761763504; %dy = 0.797065761763504

```

```

PVR(k) = VR;
u(k) = K*(Nx*r(k)-barx(:,k))+Nu*r(k);
RDu(k) = u(k)+1.83026847123187; %du = 1.83026847123187;
y(:,k) = H*x(:,k)+J*u(:,k); %Output. J=0 here
Py(:,k) = y(:,k) + 0.797065761763504;
x(:,k+1) = Phi*x(:,k)+Gamma*u(k);
barx(:,k+1) = Phi*barx(:,k)+Gamma*u(:,k)+Lp*((H*x(:,k)- 0.797065761763504)-
(H*barx(:,k)- 0.797065761763504));
end
figure(1)
plot(PVR,'r')
hold;
plot(Py);
legend('PVR','Py');
%axis([1 100 -10 10]);
%axis ([100 150 (r(1)- 0.001) (r(1)+0.001)]);
figure(2)
plot(RDu)
legend('RDu')
figure(3)
plot(x(1,:));
hold on
plot(barx(1,:), 'r')
figure(4)
plot(x(2,:));
hold on
plot(barx(2,:), 'r')

```

CHAPTER 8 CODES:

**Code 8.1: For Figure 8.1: Observer and Controller $u(k)$ Only
Real-Time Implementation Code of Observer with No Filter**

```

clear all
clc
close all
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');

```

```

aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'SamplesPerTrigger',10);
set(ai,'InputType','SingleEnded'); %In order to remove the offset value of 1.4
%V=data(10)+1.4 in all other codes. Now it measures according to MAX set to SingleEnded
%input type
set(ai,'TriggerType','Immediate');
%% From: New Data Second New model equation parameters: a1,a2,b2,b3- With %Mean
Values du and dy Corrected
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% PHi, Gamma, H, J, T results only.
% Writing the G(z) function into a transfer function format for ssdata command
Ts = 0.0067; % The maximum allowable sampling time for the analog output object of the DAQ
%6008
Phi = [1.287852678797189, -0.653470026996700; 0.5, 0];
Gamma = [ 8;0];
H = [4.142725480804508, -7.626738695335436];
J = 0;
%% N
N = [3.036227048441438;1.518113524220719;0.014756949540777];
Nx = N(1:2);
Nu = N(3);
%% controller Gain K
p_c = [0.879,0.879];
K = acker(Phi,Gamma,p_c); %controller gain K
%% Observer Lp
p_o = [0.38,0.38];
Lp = acker(Phi',H',p_o); %observer Lp
%%
%Pre-allocation:
size = 10000;
r = zeros(1,size);
PV = zeros(1,size);
PVR = zeros(1,size);
endtime3 = zeros(1, iterations);
endtime2 = zeros(1, iterations);
VR = 0.4;
%Initialization:
barx(:,1) = [-0.9652;-0.51436];
u(:,1) = 0; %This is a scalar or matrix 1x1
RDu(:,1) = 1.67;
tic
for k = 1:size;

```

```

data=getsample(ai);
V=data;
endtime3(k) = toc;
PV(k) = V; %Output from sensor
r(k) = VR - 0.797065761763504-0.1; %dy = 0.797065761763504;
PVR(k) = VR;
u(k) = K*(Nx*r(k)-barx(:,k))+Nu*r(k);
RDU(k) = u(k)+1.83026847123187; %du = 1.83026847123187;
barx(:,k+1) = Phi*barx(:,k)+Gamma*u(:,k)+Lp*((PV(k)- 0.797065761763504)-
(H*barx(:,k)- 0.797065761763504));
putsample(ao,RDU(k)); %Use the offset controller as the actual controller
endtime2(k) = toc;
end
endtime1 = toc;
Y1(1:size) = VR;
Y1_new = PV;
N1 = size;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
%RMSE and MAE calculated
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
MAE=sum(abs(Y1-Y1_new))/N1;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f',RMSE,MAE);
h_tx4=text(5000,VR-0.15,S_text,'FontSize',10,'FontWeight','bold');
legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 0.4','FontSize',20);
axis([0 10000 0 0.8]);

```

**Code 8.2: For Figure 8.3: Observer and Controller $u(k)$ with Filter of Window Size 3
Code of Observer with Median filter of window Size 3:**

```

clear all
clc
close all
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);

```



```

set(ai,'SampleRate',10000);
set(ai,'SamplesPerTrigger',10);
set(ai,'InputType','SingleEnded'); %In order to remove the offset value of 1.4
%V=data(10)+1.4 in all other codes. Now it measures according to MAX set to SingleEnded
%input type
set(ai,'TriggerType','Immediate');
%% From: New Data Second New model equation parameters: a1,a2,b2,b3- With %Mean
Values du and dy Corrected
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% PHi, Gamma, H, J, T results only.
% Writing the G(z) function into a transfer function format for ssdata command
Ts = 0.0067; % The maximum allowable sampling time for the analog output object of the DAQ
%6008
Phi = [1.287852678797189, -0.653470026996700; 0.5, 0];
Gamma = [ 8;0];
H = [4.142725480804508, -7.626738695335436];
J = 0;
%% N
N = [3.036227048441438;1.518113524220719;0.014756949540777];
Nx = N(1:2);
Nu = N(3);
%% controller Gain K
p_c = [0.879,0.879];
K = acker(Phi,Gamma,p_c); %controller gain K
%% Observer Lp
p_o = [0.38,0.38];
Lp = acker(Phi',H',p_o)'; %observer Lp
%%
%Pre-allocation:
size = 10000;
r = zeros(1,size);
PV = zeros(1,size);
Y1_Filtered = zeros(1,size);
PVR = zeros(1,size);
endtime3 = zeros(1, size);
endtime2 = zeros(1, size);
VR = 1.7;
%Initialization:
barx(:,1) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation
barx(:,2) = [-0.9652;-0.51436]; %barx arbitrary initial value. For actual computation
u(:,1) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation
RDu(:,1) = 1.67; %RDu arbitrary initial value. Not for actual computation
PVR(1) = VR; %PVR arbitrary initial value. Not for actual computation

```

```

PV(1) = 0.34; %Noisy signal arbitrary initial value for array only; not for actual computation
Y1_Filtered(1)=0.34; %Filtered signal arbitrary initial value. For actual computation
tic
for k = 2:size;
    data=getsample(ai);
    V=data;
    endtime3(k) = toc;
    PV(k) = V; %Output from sensor
    if k>=3; % Median Filter waiting for 3rd value
        %% Median Filter definition
        n_window = 3; % Median Window Size
        L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1,
%index2, index3] for the window size
        %% Median Filtering of the noisy signal
        m = k - 1; % from 2 because this is a window size 3 filter
        Y_inwindow=PV(m-L:m+L);
        T1=median(Y_inwindow,2);
        Y1_Filtered(m)=T1;
    end
    r(k) = VR - 0.797065761763504; %dy = 0.797065761763504;
    PVR(k) = VR;
    u(k) = K*(Nx*r(k)-barx(:,k))+Nu*r(k);
    RDu(k) = u(k)+1.83026847123187; %du = 1.83026847123187;
    barx(:,k+1) = Phi*barx(:,k)+Gamma*u(:,k)+Lp*((Y1_Filtered(k-1)- 0.797065761763504)-
(H*barx(:,k)- 0.797065761763504));
    putsample(ao,RDu(k)); %Use the offset controller as the actual controller
    endtime2(k) = toc;
end
endtime1 = toc;
Y1(1:size) = VR;
Y1_new = Y1_Filtered;
N1 = size;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
%RMSE and MAE calculated
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
MAE=sum(abs(Y1-Y1_new))/N1;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f',RMSE,MAE);
h_tx4=text(5000,VR-1.1,S_text,'FontSize',10,'FontWeight','bold');
legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.7','FontSize',20);
axis([0 10000 0.2 3.4]);

```

**Code 8.3: For Figure 8.4: Observer and Controller $u(k)$ with Filter of Window Size 5
Code of Observer with Median Filter of window size 5**

```

clear all
clc
close all
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'SamplesPerTrigger',10);
set(ai,'InputType','SingleEnded'); %In order to remove the offset value of 1.4
%V=data(10)+1.4 in all other codes. Now it measures according to MAX set to SingleEnded
%input type
set(ai,'TriggerType','Immediate');
%% From: New Data Second New model equation parameters: a1,a2,b2,b3 from Step 2.1- With
%Mean Values du and dy Corrected
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% PHi, Gamma, H, J, T results only.
% Writing the G(z) function into a transfer function format for ssdata command
Ts = 0.0067; % The maximum allowable sampling time for the analog output object of the DAQ
%6008
Phi = [1.287852678797189, -0.653470026996700; 0.5, 0];
Gamma = [ 8;0];
H = [4.142725480804508, -7.626738695335436];
J = 0;
%% N
N = [3.036227048441438;1.518113524220719;0.014756949540777];
Nx = N(1:2);
Nu = N(3);
%% controller Gain K
p_c = [0.879,0.879];
K = acker(Phi,Gamma,p_c); %controller gain K
%% Observer Lp
p_o = [0.38,0.38];
Lp = acker(Phi,'H',p_o); %observer Lp
%%

```

```

%Pre-allocation:
size = 10000;
r = zeros(1,size);
PV = zeros(1,size);
Y1_Filtered = zeros(1,size);
PVR = zeros(1,size);
endtime3 = zeros(1, size);
endtime2 = zeros(1, size);
VR = 1.7;
%Initialization:
barx(:,1) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation. For
%array and plot
barx(:,2) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation. For
%array and plot
barx(:,3) = [-0.9652;-0.51436]; %barx arbitrary initial value. For actual computation
u(:,1) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation.
%For array and plot
u(:,2) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation.
%For array and plot
RDu(:,1) = 1.67; %RDu arbitrary initial value. Not for actual computation. For array and plot
RDu(:,2) = 1.67; %RDu arbitrary initial value. Not for actual computation. For array and plot
PVR(1) = VR; %PVR arbitrary initial value. Not for actual computation. For array and plot
PVR(2) = VR; %PVR arbitrary initial value. Not for actual computation. For array and plot
PV(1) = 0.34; %Noisy signal arbitrary initial value for array only; Not for actual computation.
%For array and plot
PV(2) = 0.34; %Noisy signal arbitrary initial value for array only; Not for actual computation.
%For array and plot
Y1_Filtered(1)=0.34; %Filtered signal arbitrary initial value. For actual computation
Y1_Filtered(2)=0.34; %Filtered signal arbitrary initial value. For actual computation
tic
for k = 3:size;
    data=getsample(ai);
    V=data;
    endtime3(k) = toc;
    PV(k) = V; %Output from sensor
    if k>=5; % Median Filter waiting for 5th value
        %% Median Filter definition
        n_window = 5; % Median Window Size
        L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1,
%index2, index3] for the window size
        %% Median Filtering of the noisy signal
        m = k - 2; % from 2 because this is a window size 3 filter
        Y_inwindow=PV(m-L:m+L);
        T1=median(Y_inwindow,2);
        Y1_Filtered(m)=T1;
    end
end

```

end

```

r(k) = VR - 0.797065761763504; %dy = 0.797065761763504;
PVR(k) = VR;
u(k) = K*(Nx*r(k)-barx(:,k))+Nu*r(k);
RDu(k) = u(k)+1.83026847123187; %du = 1.83026847123187;
barx(:,k+1) = Phi*barx(:,k)+Gamma*u(:,k)+Lp*((Y1_Filtered(k-2)- 0.797065761763504)-
(H*barx(:,k)- 0.797065761763504));
    putsample(ao,RDu(k)); %Use the offset controller as the actual controller
    endtime2(k) = toc;
end
endtime1 = toc;
Y1(1:size) = VR;
Y1_new = Y1_Filtered;
N1 = size;
plot(Y1_new)
hold
plot(Y1,'r');
grid on
%RMSE and MAE calculated
RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
MAE=sum(abs(Y1-Y1_new))/N1;
S_text=sprintf('RMSE=%8.6f,MAE=%8.6f',RMSE,MAE);
h_tx4=text(5000,VR-1,S_text,'FontSize',10,'FontWeight','bold');
legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.7','FontSize',20);
axis([0 10000 0.2 3.5]);

```

**Code 8.4: For Figure 8.5: Observer and Controller $u(k)$ with Filter of Window Size 7
Code of Observer with Median filter of window size 7**

```

clear all
clc
close all
daqreset;
%% Initialize the DAQ device
ao=analogoutput('nidaq','Dev1');
aochan=addchannel(ao,0);
set(ao,'SampleRate',150);
set(ao,'TriggerType','Immediate');
ai=analoginput('nidaq','Dev1');
aichan = addchannel(ai,0);
set(ai,'SampleRate',10000);
set(ai,'SamplesPerTrigger',10);
set(ai,'InputType','SingleEnded'); %In order to remove the offset value of 1.4

```

```

%V=data(10)+1.4 in all other codes. Now it measures according to MAX set to SingleEnded
%input type
set(ai,'TriggerType','Immediate');
%% From: New Data Second New model equation parameters: a1,a2,b2,b3 from Step 2.1- With
%Mean Values du and dy Corrected
a1 = 1.287852678797189;
a2= -0.326735013498350;
b2= 33.141803846436060;
b3 = -30.506954781341744;
%% PHi, Gamma, H, J, T results only.
% Writing the G(z) function into a transfer function format for ssdata command
Ts = 0.0067; % The maximum allowable sampling time for the analog output object of the DAQ
%6008
Phi = [1.287852678797189, -0.653470026996700; 0.5, 0];
Gamma = [ 8;0];
H = [4.142725480804508, -7.626738695335436];
J = 0;
%% N
N = [3.036227048441438;1.518113524220719;0.014756949540777];
Nx = N(1:2);
Nu = N(3);
%% controller Gain K
p_c = [0.879,0.879];
K = acker(Phi,Gamma,p_c); %controller gain K
%% Observer Lp
p_o = [0.38,0.38];
Lp = acker(Phi',H',p_o)'; %observer Lp
%%
%Pre-allocation:
size = 10000;
r = zeros(1,size);
PV = zeros(1,size);
Y1_Filtered = zeros(1,size);
PVR = zeros(1,size);
endtime3 = zeros(1, size);
endtime2 = zeros(1, size);
VR = 1.6;
%Initialization:
barx(:,1) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation. For
%array and plot
barx(:,2) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation. For
%array and plot
barx(:,3) = [-0.9652;-0.51436]; %barx arbitrary initial value. Not for actual computation. For
%array and plot
barx(:,4) = [-0.9652;-0.51436]; %barx arbitrary initial value. For actual computation

```

```

u(:,1) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation.
%For array and plot
u(:,2) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation.
%For array and plot
u(:,3) = 0; %This is a scalar or matrix 1x1 %u arbitrary initial value. Not for actual computation.
%For array and plot
RDu(:,1) = 1.67; %RDu arbitrary initial value. Not for actual computation. For array and plot
RDu(:,2) = 1.67; %RDu arbitrary initial value. Not for actual computation. For array and plot
RDu(:,3) = 1.67; %RDu arbitrary initial value. Not for actual computation. For array and plot
PVR(1) = VR; %PVR arbitrary initial value. Not for actual computation. For array and plot
PVR(2) = VR; %PVR arbitrary initial value. Not for actual computation. For array and plot
PVR(3) = VR; %PVR arbitrary initial value. Not for actual computation. For array and plot
PV(1) = 0.34; %Noisy signal arbitrary initial value for array only; Not for actual computation.
%For array and plot
PV(2) = 0.34; %Noisy signal arbitrary initial value for array only; Not for actual computation.
%For array and plot
PV(3) = 0.34; %Noisy signal arbitrary initial value for array only; Not for actual computation.
%For array and plot
Y1_Filtered(1)=0.34; %Filtered signal arbitrary initial value. For actual computation
Y1_Filtered(2)=0.34; %Filtered signal arbitrary initial value. For actual computation
Y1_Filtered(3)=0.34; %Filtered signal arbitrary initial value. For actual computation
tic
for k = 4:size;
    data=getsample(ai);
    V=data;
    endtime3(k) = toc;
    PV(k) = V; %Output from sensor
    if k>=7; % Median Filter waiting for 3rd value
        %% Median Filter definition
        n_window = 7; % Median Window Size
        L=(n_window-1)/2; %Offset value in order to have window start at index 1 in [index1,
        %index2, index3] for the window size
        %% Median Filtering of the noisy signal
        m = k - 3; % from 2 because this is a window size 3 filter
        Y_inwindow=PV(m-L:m+L);
        T1=median(Y_inwindow,2);
        Y1_Filtered(m)=T1;
    end
    r(k) = VR - 0.797065761763504; %dy = 0.797065761763504;
    PVR(k) = VR;
    u(k) = K*(Nx*r(k)-barx(:,k))+Nu*r(k);
    RDu(k) = u(k)+1.83026847123187; %du = 1.83026847123187;
    barx(:,k+1) = Phi*barx(:,k)+Gamma*u(:,k)+Lp*((Y1_Filtered(k-3)- 0.797065761763504)-
    (H*barx(:,k)- 0.797065761763504));
    putsample(ao,RDu(k)); %Use the offset controller as the actual controller
    endtime2(k) = toc;

```

```

end
endtime1 = toc;
Y1(1:size) = VR;
    Y1_new = Y1_Filtered;
    N1 = size;
    plot(Y1_new)
    hold
    plot(Y1,'r');
    grid on
    %RMSE and MAE calculated
    RMSE=sqrt((Y1-Y1_new)*(Y1-Y1_new))/N1;
    MAE=sum(abs(Y1-Y1_new))/N1;
    S_text=sprintf('RMSE=%8.6f,MAE=%8.6f,RMSE,MAE);
    h_tx4=text(5000,VR-1,S_text,'FontSize',10,'FontWeight','bold');
    legend('Y1 Filtered','VR');
xlabel({'Loop Iterations (i)'},'FontSize',12);
ylabel({'Sensor Voltage (V)'},'FontSize',12);
title('Convergence to VR = 1.6','FontSize',20);
axis([0 10000 0.2 3.5]);

```

BIBLIOGRAPHY

- [1] A. Becker, R. Sandheinrich, T. Bretl, "Automated Manipulation of Spherical Objects in Three Dimensions Using a Gimbaled Air Jet," IEEE/RSJ International Conference on Intelligent Robots and Systems, October 10-15, 2009, St. Louis, MO, pp. 781 – 786.
- [2] Acroname Robotics. "Sharp IR Rangers Information" 2009. [Online] Acroname Inc. Available: <http://www.acroname.com/robotics/info/articles/sharp/sharp.html> [Accessed 22 February 2011].
- [3] A. Gil, N. Quijano, K. Passino, "The Balls-in-Tubes Experiment," April 5, 2004.
- [4] G. F. Franklin, J. D. Powell, M. Workman., *Digital Control of Dynamic Systems*. 3rd edition, Menlo Park, California: Addison Wesley Longman, 1998.
- [5] J. Franz, S. Cortinas, C. Qian, "FBIT (Floating Ball Instructional Tool)," 2007 ASEE-GSW Annual Conference, March 2007.
- [6] J. Escaño, M. Ortega, F. Rubio, "Position Control of a Pneumatic Levitation System," IEEE, Volume 1, 2005.
- [7] K. Bong, S. Chapin, D. Pregibon, D. Baah, T. Floyd-Smith system, "Compressed-Air Flow Control System," The Royal Society of Chemistry, 2011.
- [8] K. M. Passino, S. Yurkovich., *Fuzzy Control*. Menlo Park, California: Addison-Wesley Longman, 1998.
- [9] National Instruments Corporation, *User Guide and Specifications NI USB – 6008/6009*, 2008.
- [10] L. Ciprian, U. Andreea, P. Dumitru, F. Cristian, "Stabile Algorithms Switching for Multiple Models Control Systems," WSEAS Transactions on Systems, Issue 4, Volume 8, April 2009, ISSN: 1109-2777.
- [11] MATLAB version 7.9.0.529. Natick, Massachusetts: The MathWorks Inc., 2009.
- [12] P. Gluck, "A Delicate Balance: Hovering Balloons in an Air Stream," The Physics Teacher, Volume 4, December 2006.
- [13] R. Jia. Controller Design for a Tri-Turbofan Airship System Based on Particle Swarm Optimization Algorithm & Neural Network Technology. MS Thesis. Department of Electrical and Computer Engineering, The University of Texas at San Antonio, 2008.
- [14] R. Schreiber, "Air Flow Control Using Fuzzy Logic" 1997. Microchip Technology Inc., AN600.

- [15] SHARP Corporation, "Optoelectronic Device," GP2D12 datasheet. Osaka, Japan, 2005.
- [16] T. Teng, J. Shieh, C. Chen, "Genetic Algorithms Applied n Online PID parameters of a Liquid-Level Control System," Transactions of the Institute of Measurement and Control, 2003.
- [17] W. Wilckle, R. Vance Morey, "Selecting and Determining Airflow for Crop Drying, Cooling, and Storage," Regents of the University of Minnesota, College of Agricultural, Food, and Environmental Sciences, 2002.

VITA

José Luis Gamboa was born in Laredo, Peru on February 20th of 1986 and he remained in Peru till the age of nine. In 1995, he journeyed to San Antonio, Texas to be reunited with his family.

He attended the University of Texas at San Antonio (UTSA) from August 2004 to May 2009 and received his Bachelor of Science degree in Electrical Engineering. In August of 2009, he began graduate work at UTSA and will be graduating this August of 2012 receiving a Master of Science degree in Electrical Engineering with a concentration in systems and control.

After graduation, he plans to enroll in a two-year Christian training program. After this training, he plans to pursue a Doctor of Philosophy in Electrical Engineering at UTSA.