



UNIVERSITY OF TASMANIA

Department of Electrical Engineering and Computer Science

Hobart Campus

Final Year Thesis

**FUZZY ADAPTIVE SLIDING MODE
TRACKING CONTROL FOR
BALANCING BEAM SYSTEM**

Written by: Nguyen Vu Thanh

Supervisor: Dr Zhihong Man

This thesis is submitted in partial fulfillment of the requirements for the degree of
Bachelor of Engineering (Honours) at University of Tasmania, Australia

October 2000

Abstract

In this project, a **fuzzy adaptive sliding mode controller (FASMC)** for a non-linear, two-degree of freedom tracking system with uncertainties and disturbance was designed, simulated in Matlab and then implemented in the real system.

Firstly, a sliding variable was defined to describe the desired error dynamics of the controlled system. In order to eliminate the effects of parameter uncertainties and disturbances, adaptive rules were applied to online learn and adjust the system parameters in the Liapunov sense. The combination of **adaptive rules** and **sliding mode control** scheme drives the system dynamics into sliding mode and assures its stability based on **Liapunov stability theory**.

Due to the discontinuous nature of sliding control, there is unavoidable switching around the sliding mode surface. It may create severe chattering, which then results in accelerated wear of control motors, poor system reliability, or even instability and uncontrollability. To reduce the **chattering effect**, a fuzzy tuning controller is utilized around the switching surface.

Simulation and practical results showed that the controller not only assures fast tracking, stability and robustness of an adaptive sliding mode controller but also effectively diminishes the chattering effect.

Acknowledgements

First of all, I would like to extend a special note of thanks to my supervisor, Dr Zhihong Man, for his whole-heart help, enthusiasm and persistent encouragement during the project. With his professional knowledge in control engineering and fuzzy logic, he not only provided strategic direction but also much valuable advice in theoretical design and practical implementation of the project. As a generous and benevolent lecturer, he encouraged us in such a way that we felt guilty if we did not do our best. It is a privilege for me to work under his instruction, which is not just about technical knowledge but more notably, about the way to live.

I wish to thank Mr. John McCulloch, of the departmental support staff, for his punctual and precious assistance in overcoming subtle and critical issues in hardware systems and software debugging. I also want to express sincere thanks to my project partner, “veteran” Kim Cheong Sim, for his hard work, friendship and understanding. Without his cooperation, encouragement and tremendous contribution, accomplishing this project would have been impossible.

Last, but certainly not least, I would like to thank my family and friends for their love and enthusiastic support, especially my brother, Nguyen Vu Thanh, for his talented advice and incredible help in software development.

Nguyen Vu Thanh

Hobart, Friday, September 15, 2000

Contents

Abstract	2
Acknowledgements	3
Contents	4
1 Introduction	9
2 Balancing beam and controller hardware systems	12
2.1 An overview of the physical system & hardware interaction (dataflow)	12
2.2 Balancing beam description and mathematical model	15
2.3 Sensory hardware	17
2.3.1 Optical encoder and PCL-833 card.....	17
2.3.2 Potentiometers	17
2.3.2.1 Description	17
2.3.2.2 Crossover tracking.....	18
2.3.3 Tachometer	20
2.4 Control driving hardware.....	21
2.4.1 Description.....	21
2.4.2 Dead-zone effect	21
2.5 Amplifier circuits.....	23
3 The involved theories	24
3.1 Liapunov stability criterion	24
3.1.1 Introduction.....	24
3.1.2 Stability in the sense of Liapunov.....	25
3.1.3 Asymptotic stability in the large & uniform stability	26
3.1.4 Second method of Liapunov	28
3.1.5 Discussion of Liapunov stability	30
3.2 Adaptive control	32

3.2.1	Introduction.....	32
3.2.2	Definition of adaptive control system.....	33
3.2.3	Adaptive control structure and algorithm	35
3.2.4	Discussion of adaptive control.....	36
3.2.4.1	Advantages of adaptive control.....	36
3.2.4.2	Disadvantages of adaptive control	37
3.2.5	Why should adaptive control be used for this project?.....	38
3.3	Sliding mode control	39
3.3.1	Introduction.....	39
3.3.2	The principle of sliding mode control.....	40
3.3.3	Discussion of sliding mode control	41
3.3.3.1	Advantages of sliding mode control.....	41
3.3.3.2	Disadvantages of sliding mode control – Chattering effect .	42
3.3.4	Why should sliding mode control be used for this project?	43
3.4	Fuzzy logic control	45
3.4.1	Introduction.....	45
3.4.2	Fuzzy set theory	46
3.4.2.1	Definitions of fuzzy set	46
3.4.2.2	Basic operations on fuzzy set.....	47
3.4.3	Fuzzy interference and fuzzy reasoning	49
3.4.3.1	Fuzzy if-then rules.....	49
3.4.3.2	Fuzzy interference	49
3.4.3.3	The compositional rule of inference.....	50
3.4.3.4	Fuzzy reasoning.....	51
3.4.4	General structure of a fuzzy logic controller	53
3.4.5	Logic processing in a fuzzy controller.....	54
3.4.5.1	Fuzzification.....	55
3.4.5.2	Fuzzy reasoning (rule evaluation)	56
3.4.5.3	Defuzzification	59

3.4.6	Design method of a fuzzy logic controller.....	61
3.4.7	Discussion of the fuzzy logic controller	63
3.4.7.1	Advantages of the fuzzy logic controller	63
3.4.7.2	Disadvantages of the fuzzy logic controller	64
4	Design controller for the balancing beam system	65
4.1	Introduction	65
4.2	Adaptive sliding mode controller	66
4.2.1	Control signal and stability proof	66
4.3	Fuzzy controller to reduce chattering effects.....	71
4.3.1	The boundary layer control approach in brief	71
4.3.2	Fuzzy logic approach and the FLC design.....	72
4.3.2.1	Fuzzy controller 3-3-7	76
4.3.2.2	Fuzzy controller 5-5-7	78
4.4	Summary.....	80
4.5	The effects of controller parameters	83
5	Testing by Matlab simulation	87
5.1	Introduction	87
5.1.1	Simulation settings.....	88
5.1.2	Simulation reference signals	90
5.1.3	Simulation outputs	90
5.2	Simulation results under ideal conditions.....	91
5.2.1	Triangular reference signal	91
5.2.2	Sine-wave reference signal	92
5.3	Simulation results with noise, disturbances and delay	93
5.3.1	Triangular reference signal	93
5.3.2	Sine-wave reference signal	94
5.4	Analysis of simulation results.....	96
6	Practical implementation	98
6.1	Software development	98

6.2	Practical results.....	101
6.2.1	Introduction.....	101
6.2.2	Square wave reference signal	102
6.2.3	Sine wave reference signal	103
6.2.4	Joystick reference signal.....	105
6.3	Analysis of practical results.....	107
6.4	Some additional remarks	110
6.4.1	Limitations of the physical plant	110
6.4.2	How to choose optimum controller parameters	113
7	Conclusion and future improvements	114
	References & Bibliography	117
	Appendix A: Matlab programs for simulation & off-line data plotting.....	121
A.1	Simulation program	121
A.2	Programs for off-line data plotting	126
A.2.1	For both horizontal and vertical axes.....	126
A.2.2	For horizontal axis	128
A.2.3	For vertical axis	129
	Appendix B: The controller's software.....	130
B.1	Fuzzy controller.....	130
B.1.1	The header file of fuzzy class	130
B.1.2	The implemented code of fuzzy class in C++	133
B.2	Full FASMC software	150
B.3	Device driver for PC-30 interface card.....	172
B.4	Device driver for PCL833 optical encoder.....	175
B.4.1	The optical encoder device driver (833Drive.c)	175
B.4.2	Optical encoder functions for project (PCL833.c).....	183
	Appendix C: Interfacing card and encoder.....	186
C.1	PC-30 interface card	186

C.1.1	A/D subsystem.....	186
C.1.2	D/A subsystem.....	187
C.2	PCL-833 encoder.....	188
Appendix D: Signal amplification stage.....		190
D.1	Summing amplifier circuit.....	191
D.2	Buffer circuit.....	192
D.3	Power amplifier circuit	193

1 Introduction

In classical control theory, the system dynamic behaviours are often described by a set of differential equations (state equations or transfer functions). The designed controller will be based on that model with the assumption that all system states and parameters should be known in advance or measurable. However, this method is often difficult, if not impossible, to apply for real systems with a high degree of **non-linearities, uncertainty dynamics, unknown parameter bounds and external disturbances**. If the system is modelled by just a simplified set of equations, which does not adequately reflect the real system characteristics, the controller may not drive the system into its desired dynamics or assure its stability. That is the reason for the developing of modern control theories such as the Liapunov stability criteria, sliding mode and adaptive control theories, neural networks and fuzzy logic control.

In this project, the controlled system is a 2-degree-of-freedom balancing beam with high non-linearities, uncertainty dynamics and parameter variations. It is a **tracking system** since the system output is required to follow the input signal (pre-set signal or joystick). Over the years, the following control schemes have been designed and implemented:

1. Self-tuning controller
2. Linear Quadratic Controller (LQC)
3. Model Reference Adaptive Controller (MRAC)
4. Sliding Mode Controller (SMC)
5. Adaptive Sliding Mode Controller (ASMC)

In early designs, only the horizontal axis motion is controlled but the achieved result is limited. Succeeding control schemes improved the performance of the horizontal axis and were extended to control vertical axis motion with restricted outcome due to its higher non-linearities and uncertainties. The latest controller (ASMC) showed quite reasonable tracking performance, robustness and stability for both axes.

In the adaptive sliding mode controller (ASMC), system parameters are adaptively learned and adjusted, and then the control action forces the system trajectories to intersect a manifold of the sliding surface. The system trajectories (dynamics) are then constrained to the sliding surface for all subsequent time via the use of high speed switching controls. The advantage of this control scheme is the fast tracking response and robustness against external disturbances and variations in system parameters. However, the main drawback is severe chattering associated with the switching around the sliding surface because it can excite undesirable high frequency dynamics.

The ASMC scheme controlled the **chattering effect** by applying a **Boundary Layer** around the sliding surface and the result obtained is superior to previous controllers. However, there is a trade-off between chattering and robustness since better reduction of chattering leads to the degradation in robustness and disturbance rejections. Moreover, under **high noise and disturbance conditions** (e.g. with joystick reference input), the chattering is not effectively reduced and the tracking performance is also degraded. For that reason, a new controller using **fuzzy logic** is proposed in this thesis to form a **Fuzzy Adaptive Sliding Mode Controller (FASMC)** for the balancing beam system.

Simulation and practical results showed that the new controller has effectively diminished the chattering effects without sacrificing the tracking performance, stability and robustness of an adaptive sliding mode controller.

The thesis is organized as follows: In section 2, the balancing beam system, its mathematical model, and controller hardware data interaction will be described. Section 3 will briefly introduce some theories and suitable discussions related to the designed controller (e.g. Liapunov stability criteria, sliding mode, adaptive and fuzzy control). The system model and theories in those sections will be applied to theoretically design the controller in section 4. The fuzzy controller is designed **based on real data obtained from last year's controller (1999)**, with the help of the Matlab Fuzzy Toolbox, and will be fully described in this section.

After that, section 5 describes the testing of the proposed controller by Matlab simulation. Upon the success of the theoretical controller, the controller software for the real balancing beam will be developed on C++ environment and practical results will be shown in section 6, and compared with previous ASMC controller results. Some remarks about the designed controller and practical problems are also presented in this section. Section 7 will provide the overall conclusion and suggestions for further improvements.

2 Balancing beam and controller hardware systems

The main purpose of this thesis is to propose a new control scheme and design for a tracking balancing beam system. This chapter will provide an overview of the completely controlled system and briefly demonstrate its general controller hardware data interaction. Then, the physical controlled plant will be described, its mathematical model will be developed and other related controller hardware will be fully illustrated.

2.1 An overview of the physical system & hardware interaction (dataflow)

The complete physical system is shown in Figure 2.1.1, which consists of four main parts (excluding the controller software in the computer):

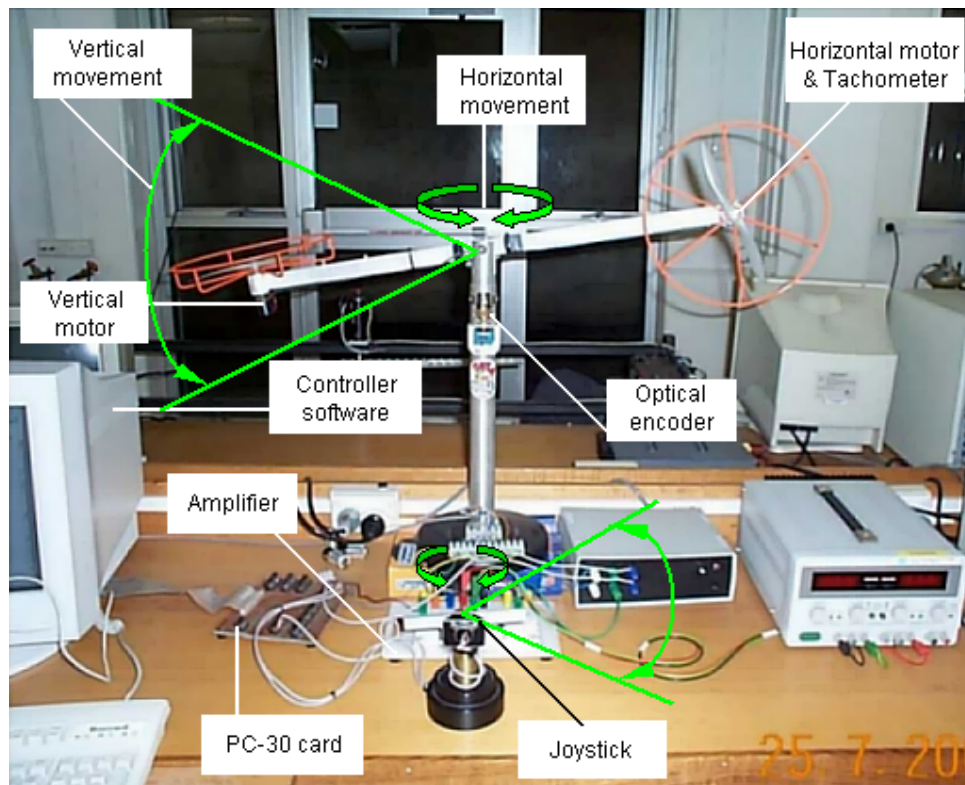


Figure 2.1.1: The completely physical system

2. Balancing beam and controller hardware systems

1. The controlled system (balancing beam)
2. The sensory hardware
3. The control driving hardware (motors to move the beam to desired positions)
4. Amplifier stage with negative feedback from tachometer

The interaction (data flow) between the controlled plant and the controller hardware system is shown in Figure 2.1.2. In fact, there are two separate controllers for horizontal axis and vertical axis, however the control algorithm and hardware for both axes are identical, so from now on only one controller consider will be considered.

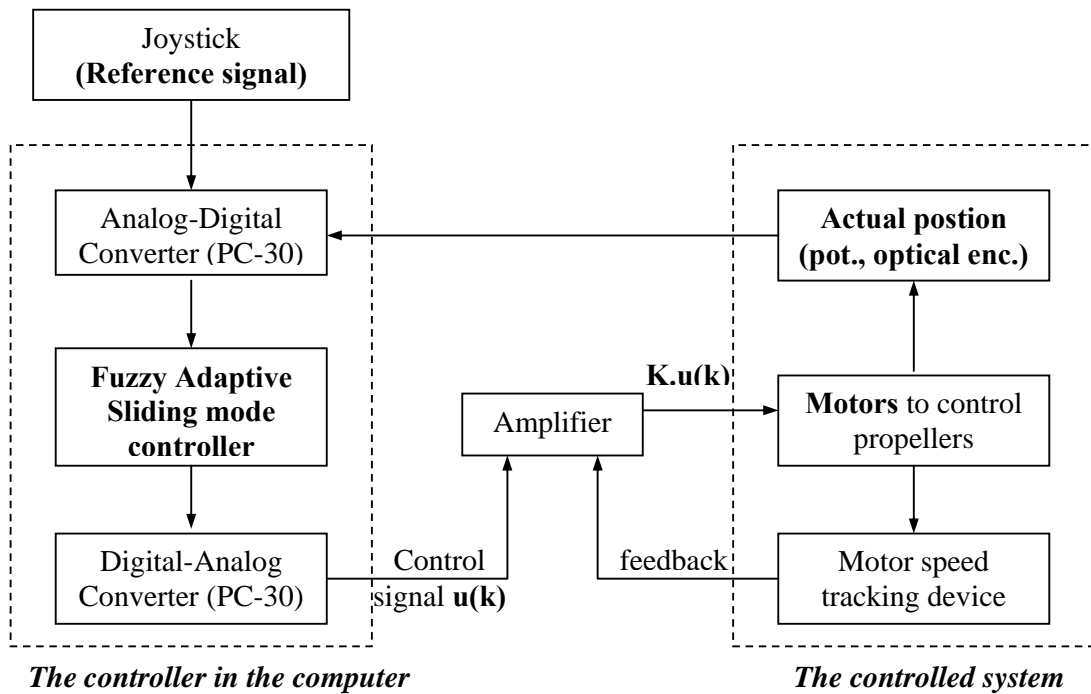


Figure 2.1.2: The data flow between system hardware

Through the ADC PC-30 card (Analog-Digital Conversion), the controller (software in the computer) receives the reference position either defined by joystick (read through potentiometers) or pre-set signal in controller software, and the actual position of the beam axis (determined by potentiometers for vertical axis or optical encoder for

horizontal axis). The only aim of ADC card is to convert the analog signals from joystick and encoder into digital data.

Based on the desired and current positions of the beam, the historical data and the control algorithm, the controller updates its controller parameters to produce a suitable control signal. This control signal is sent to the DAC (Digital-Analog Conversion) PC-30 card to convert digital data into an analog control signal. Since the output current of the signal from the computer and required input current of driving motors are quite different, an amplifier is utilized. This amplifier stage does not increase the control signal voltage but only the current level (up to values enough to drive the motor), and provides isolation and noise reduction between motors and the computer.

Through the amplifier, the output voltage corresponding to control signal is applied to the motor, which in turn drives the fan and produces suitable tangential force to move the beam to the reference position. A motor speed-tracking device then gives negative voltage feedback to the amplifier to prevent run-away acceleration of the motor.

The new actual position is then read by the controller and compared with the reference position to produce a new control signal. This cycle happens in every sampling period and provides real-time control for the balancing beam system.

2.2 Balancing beam description and mathematical model

The balancing beam can independently rotate about two axes: horizontal and vertical (elevation) as shown in Figure 2.1.1. Each motion is controlled by the speed variation and torque of a separate propeller, which is driven by a motor located at each end of the beam.

Since the motions of the two axes are independent, the balancing beam can be considered as consisting of two sub-systems. Each system corresponds to a single axis of rotation, which is described in the block diagram below:

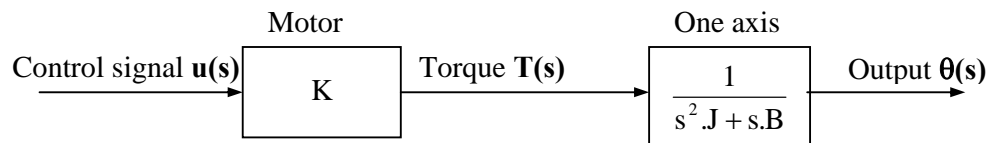


Figure 2.2.1: The block diagram of a sub-system (one axis)

where:

- $T(s)$: Output torque of a motor (corresponding to the control signal)
- K : Gain of the motor (constant)
- J : Moment inertial of the beam (varied and uncertain)
- B : Viscous-friction coefficient of the beam (varied and uncertain)

In order to place the optical encoder for sensing the azimuth of the beam axis, a gear with a 1:1 ratio is installed. This gear does not affect the system dynamics model, however it introduces some frictions in the system.

In the frequency domain, the reference model can be stated as:

$$\theta(s) = \frac{K}{s^2 \cdot J + s \cdot B} \cdot U(s)$$

$$\Rightarrow K \cdot U(s) = s^2 \cdot \theta(s) \cdot J + s \cdot \theta(s) \cdot B \quad (2.2.1)$$

Using the inverse Laplace transform with the assumption that all initial conditions are zero, the equation in the time domain is obtained:

$$K \cdot u(t) = J \cdot \ddot{\theta}(t) + B \cdot \dot{\theta}(t)$$

$$\Rightarrow \ddot{\theta}(t) = -\frac{B}{J} \dot{\theta}(t) + \frac{K}{J} u(t)$$

$$\Rightarrow \ddot{\theta}(t) = -f(\dot{\theta}, \theta) + b \cdot u(t) \quad (2.2.2)$$

where:

- $f(\dot{\theta}, \theta) = \frac{B}{J} \dot{\theta}(t) < f_o$
- $b = \frac{K}{J}, 0 < b < b_o$
- f_o, b_o : unknown upper bounds of system parameters to be adaptively controlled.

This developed model is rather simple since it does not take into account the non-linearity characteristics of the system. Moreover, the vertical and horizontal axes are different in gravity, friction and disturbance sensation and these differences should be considered in describing their dynamics behaviours. However, the exploitation of adaptive rules can provide strong robustness and the effects of those non-linearities,

uncertainties and inaccuracies can be eliminated. Therefore, the obtained model will be used in designing and implementation of the system controller.

2.3 Sensory hardware

2.3.1 Optical encoder and PCL-833 card

The optical encoder is installed on the horizontal axis of the beam to determine the position of that axis. It has its own card (PCL-833), which is a 3-axis quadrature encoder and counter add-on card for IBM PC/AT family, to convert data and send to the controller through PC-30. Since three quadrature encoders are 24 bit up/down counters and there exists a 16-Mhz oscillator time base with wide range multiplier, the data resolution is quite high. Moreover, the optical encoder has high noise resistance and so provides rather accurate positions.

The PCL-833 card has nine different input sources, which are handled by an on-board interrupt controller. Each input can accept either a single-ended or a differential signal, and has a decoding circuit for incremental quadrature encoding. Quadrature inputs can work with or without index, corresponding to linear or rotary encoder feedback, therefore can keep track of the crossover effect (see Potentiometer section).

2.3.2 Potentiometers

2.3.2.1 Description

There are three potentiometers installed in the system to sensor the locations of the joystick and the beam. Particularly, they are located at:

- Horizontal axis of the joystick
- Vertical axis of the joystick
- Vertical axis of the balancing beam

For each position of the measured axis, the potentiometer will produce a corresponding output voltage, lying in the range of 0 to 5 volts at full deflection. This analog signal will be converted into digital data by the ADC PC-30 card (at 12-bit resolution), and then the current position of the measured axis is updated to the controller.

2.3.2.2 Crossover tracking

From the experiment data, it is known that the characteristics of the potentiometers are almost linear, except at regions near the crossover from 0 to 5 volts and vice versa.

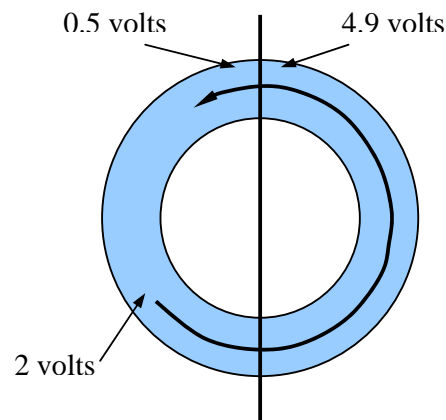


Figure 2.3.1: Crossover due to inertia

The system can become uncontrollable, as follows:

1. Suppose the joystick reference signal was suddenly moved from 2 volts to 4.9 volts, the controller would produce maximum allowable voltage to move the beam to the desired position as fast as possible. However, once the beam reaches the 4.9 volts position, the moment inertia (due to the mass of beam in motion) creates overshoot, crossover and then the beam continues moving beyond 5 volts to stop, for example, at the 0.5 volts position.

2. Balancing beam and controller hardware systems

2. The controller compares the current position of beam (0.5 volts) with the desired position (4.9 volts) and then produces a higher voltage, and accelerates the beam even faster to compensate the error of 4.4 volts.
3. Since the beam is moved faster, the moment inertia is bigger and so, the beam will crossover again.

As a result, the beam will incessantly rotate around in one direction and the system becomes totally uncontrollable.

The second problem is that if the joystick is moved to crossover from 5 to 0 volts (5 degrees from previous reference point), instead of moving backwards 5 degrees, the controller thinks that there is an error of 5 volts and moves the beam in the opposite direction from 5 volts towards 0 volts.

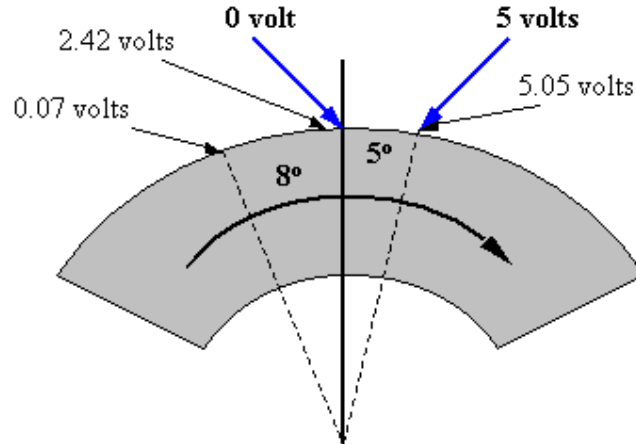


Figure 2.3.2: Non-linearity at crossover region (0-5 volts)

In addition, the crossover region of the potentiometer is in fact non-linear, i.e. it is not ideally as indicated by the bold values (**0** and **5 volts**) in Figure 2.3.2. Although the output voltage is calibrated from 0 to 5 volts, the actual minimum voltage is 0.07 volts, after that it jumps to 2.42 volts within a section of about 8 degrees, before transiting to

5.05 volts. The potentiometer's behaviour can only be considered as linear outside this crossover region (i.e. between 5.05 volts and 0.07 volts).

Note that the crossover problem does not exist with the optical encoder (on horizontal axis) since its counter can keep track of the crossover simply by remembering the desired position and increasing the voltage over 5 volts. After the overshoot created by sudden movement of the joystick (e.g. from 2 volts to 4.9 volts), the controller will move the beam back to the desired position. However, the price of an optical encoder is rather high, therefore, one can only be installed on the horizontal axis and three potentiometers are used for joysticks and the vertical axis.

The problems with the potentiometer crossover were successfully solved by software developed in 1999. It simply keeps track of the zero crossing and the respective direction of movement within the crossover region. The non-linearity is also overcome by fixing the voltage at 0.07 volts or 5.05 volts depending on the movement direction. After crossing that region, the error occurring during the transition will be compensated for by the controller.

2.3.3 Tachometer

This device acts as a motor tracking device, measures the motor speed and then produces an output voltage linearly proportional with that speed. This voltage is then used as negative feedback to the amplifier circuit for preventing the motor from runaway acceleration, which may damage the driving motors.

2.4 Control driving hardware

2.4.1 Description

In the system developed, each beam axis has its own control driving hardware, which includes a driving motor connected to a dual-blade propeller. The variation of motor shaft rotation speed will vary the propeller speed and the torque. The design purpose is to produce a suitable signal to control the motor speed so that the beam will be adjusted to the desired position.

The model of the motor is 2326.945-52.236-200, with rating power of 6 Watts at nominal voltage of 18 volts. It should not be operated at voltage larger than this nominal voltage (18V) and so a power amplifier will be utilized to control the motor power.

The control signal from the computer is passed through DAC PC-30 and an amplifier circuit, to drive the motor and then the propeller. The tangential force created by the fan will move the axis to its desired position. In fact, the relation between input motor voltage and the corresponding propeller force is non-linear and it creates a dead-zone when the axis stands still.

2.4.2 Dead-zone effect

The dead-zone is created by the effect of static Coulomb friction (when the beam axis is at rest) and the differences in the directional efficiencies of the propeller blade. Due to the presence of a larger coefficient of static friction, shifting a static object always requires a larger force than maintaining the motion of a moving object. That is, when

the beam is at stationary state, a voltage larger than a certain limit (enough to win the static friction force) must be applied to move it out.

Experimental data on the horizontal axis gives the non-linear relationship between the propeller force and the motor driving voltage as shown in Figure 2.4.1. It can be seen that:

- Within the dead-zone (from -3.2 volts to 2.9 volts), the input voltage is small so the propeller has low rotational speed and produces tangential force insufficient to exceed the static friction force. Therefore, the axis remains stationary in the dead-zone regardless of different values of the motor driving voltage.
- The efficiency (tangential force per input voltage) for positive voltage is higher than that of negative voltage (65mN/V and 55mN/V respectively). Therefore, a higher voltage (absolute value) is required to produce the same amount of tangential force in the negative direction.

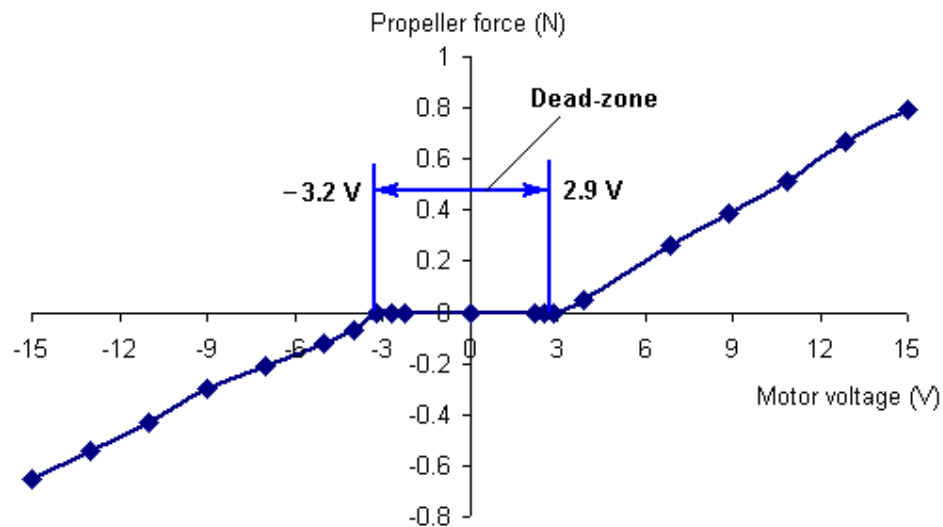


Figure 2.4.1: The propeller force vs. motor driving voltage

Those observations will be taken into account when the controller software is developed. Otherwise, the un-symmetric properties related to moving directions will lead to inaccurate control signal, and so incorrect position.

2.5 Amplifier circuits

Since the output signal power from the computer and required input power of driving motors are different, the amplifier stage is required to amplify the control signal (current, not voltage). It also receives the negative feedback from the tachometer to produce a suitable voltage to the driving motor. There are three main parts in this amplifier stage (more details and circuit diagrams can be seen in Appendix D):

- Summing circuit: to sum the control signal from the computer with negative feedback from the tachometer.
- Power amplifier circuit: to drive the motor connected with driving propellers. The limited bandwidth of this circuit affects the switching performance and leads to poor response for sudden movement of the joystick (high frequency).
- Buffer circuit: to isolate the summing and power amplifier circuits and remove the high frequency noise in the control signal by a low pass filter.

The summing and buffer circuits are built on a prototype board but the power amplifier has its own casing, with separate rectifier circuit connected directly to main power.

3 The involved theories

3.1 Liapunov stability criterion

3.1.1 Introduction

The reason why the stability analysis criterion is presented before all other control theories related to this project is that control means determination of system stability. This is because a system cannot be considered as controllable if its response behaviour is not stable against disturbances and input variance. Theoretically, a stable system is defined as a system with a bounded system response with respect to a bounded input or disturbance.

In most cases, the controlled system dynamics can be described as a set of different equations. For linear, time-invariant controlled systems, the system transfer function can be found using the Laplace transform, and then many approaches can be applied (e.g. Nyquist stability criterion, Routh's stability criterion) for the system stability analysis.

However, the stability analysis for nonlinear and/or time-varying systems is quite difficult or may be impossible and these methods are certainly unsuitable since complex nonlinear-differential equations are required to be solved. For those systems, the second method of Liapunov (or the direct method of Liapunov) has proved to be the most powerful and general analysing tool for system stability.

3.1.2 Stability in the sense of Liapunov

Consider a system defined by:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \quad (3.1)$$

where \mathbf{x} is a state vector (n-vector) and $\mathbf{f}(\mathbf{x}, t)$ is an n-vector whose elements are functions of x_1, x_2, \dots, x_n and the observed time t .

Assuming that at a given initial condition, the system of Equation 3.1 has a unique solution, denote the solution of Equation (3.1) at $\mathbf{x} = \mathbf{x}_0$, $t = t_0$ as:

$$\boldsymbol{\varphi}(t; \mathbf{x}_0, t_0) = \mathbf{x}_0$$

An equilibrium state \mathbf{x}_e for this system is defined as follows:

$$\mathbf{f}(\mathbf{x}_e, t) = \mathbf{0} \quad \text{for all } t \quad (3.2)$$

Define a spherical region of radius k about an equilibrium state \mathbf{x}_e as:

$$\|\mathbf{x} - \mathbf{x}_e\| \leq k$$

where $\|\mathbf{x} - \mathbf{x}_e\|$ is called the Euclidean norm and is defined by:

$$\|\mathbf{x} - \mathbf{x}_e\| = \left[(x_1 - x_{1e})^2 + (x_2 - x_{2e})^2 + \dots + (x_n - x_{ne})^2 \right]^{1/2}$$

Call $S(\delta)$ the region that contains all points such that:

$$\|\mathbf{x} - \mathbf{x}_e\| \leq \delta$$

and $S(\varepsilon)$ the region that contains all points such that:

$$\|\boldsymbol{\varphi}(t; \mathbf{x}_0, t_0) - \mathbf{x}_e\| \leq \varepsilon \quad \text{for all } t \geq t_0$$

An equilibrium state \mathbf{x}_e of the system of Equation (3.1) is said to be stable in the sense of Liapunov if, corresponding to each (arbitrarily chosen) $S(\varepsilon)$, there is an $S(\delta)$ such that trajectories starting in, do not leave $S(\varepsilon)$ as t increases indefinitely. The real number δ depends on ε and often depends on t_0 . If δ does not depend on t_0 , the equilibrium state is said to be uniformly stable [1].

3.1.3 Asymptotic stability in the large & uniform stability

An equilibrium state \mathbf{x}_e of the system of Equation (3.1) is said to be asymptotically stable if it is stable in the sense of Liapunov and if every solution starting within $S(\delta)$ converges, without leaving $S(\varepsilon)$, to \mathbf{x}_e as t increases indefinitely.

However, this is a local concept and so the asymptotic stability does not assure stable responses in the system. It leads to the definition of asymptotic stability in the large as follows:

“If asymptotic stability holds for all states (all points in the state space) from which trajectories originate, the equilibrium state is said to be asymptotically stable in large. That is, the equilibrium state \mathbf{x}_e ... is said to be asymptotically stable in the large if it is stable and if every solution converges to \mathbf{x}_e as t increases indefinitely” [1].

These concepts will be easier to understand with the help of graphical representations shown in the following figures.

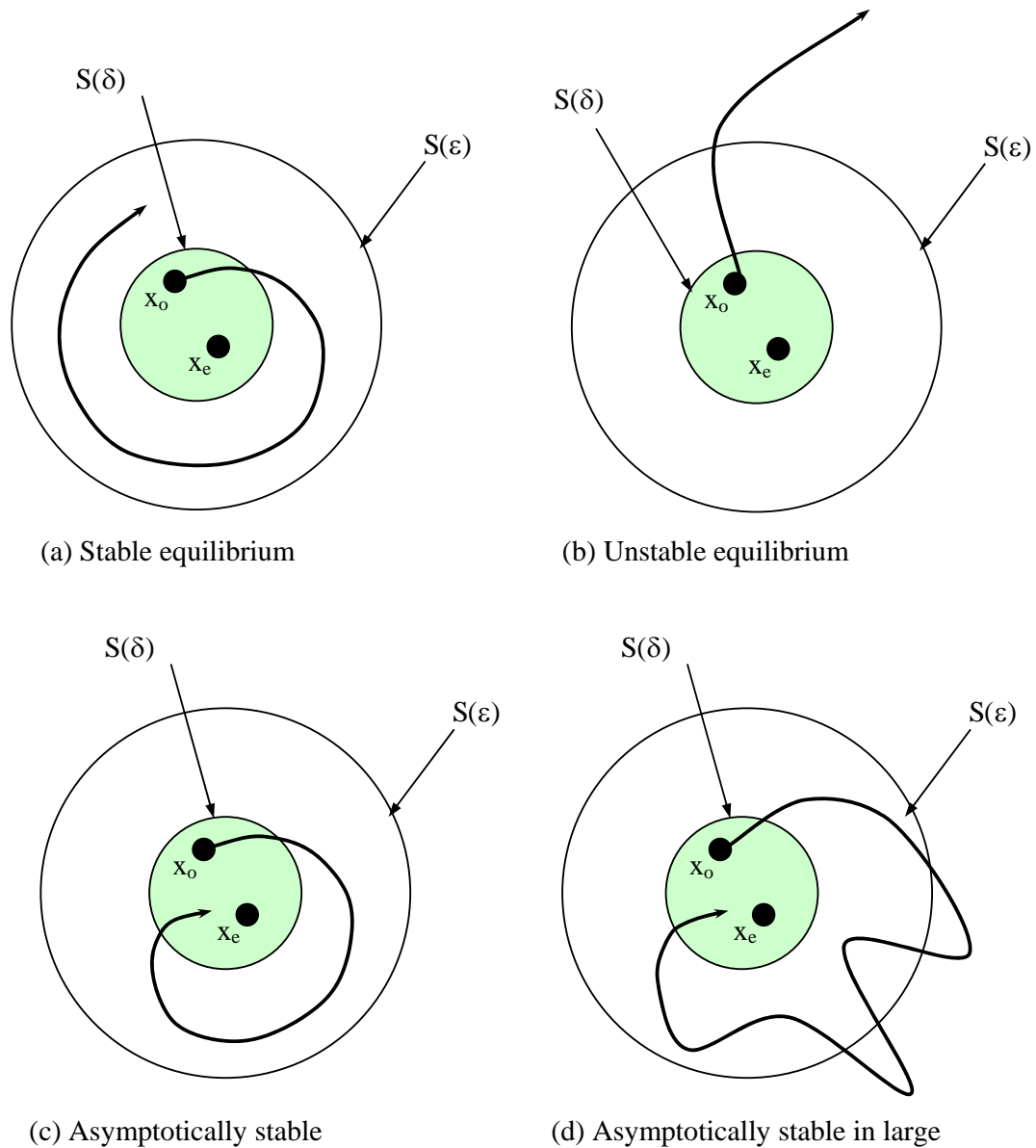


Figure 3.1.1: Equilibrium states and their representative trajectories

The previous stability definitions are dependent on the initial time t_0 so the region of attraction is also varied with different initial time. If the system of Equation (3.1) is a time-independent system, the term “uniformly” will be added.

In control design, the aim is to prove that the system has asymptotic stability in the large or if not, to determine the largest region of asymptotic stability. However, it is often

quite complex and for practical situations, it is sufficient to determine a region of asymptotic stability large enough so that no disturbance will exceed it.

3.1.4 Second method of Liapunov

In 1892, A.M. Liapunov presented two methods (called the first and second methods) for determining the stability of dynamic systems described by ordinary differential equations.

- The first method consists of all analyzing procedures based on the explicit form of the solutions of the differential equations (so inapplicable for nonlinear systems).
- The second method, on the other hand, does not require the solutions of the differential equations. Therefore, it is quite advantageous since the stability of a system can be determined without solving the system (differential) state equations, which are often nonlinear, time-dependant and so very difficult to solve.

The idea of Liapunov's direct method comes from the classical mechanics theory that: A vibratory system is stable if its energy (a positive-definite function¹) is continuously

¹A scalar function $V(\mathbf{x})$ is said to be positive definite in a region Ω (which includes the origin of the state space) if $V(\mathbf{x}) > 0$ for all nonzero state \mathbf{x} in the region Ω and $V(\mathbf{0}) = 0$.

A time-varying function $V(\mathbf{x}, t)$ is said to be positive definite in a region Ω (which includes the origin of the state space) if it is bounded from below by a time-invariant positive-definite function. That is, there exists a positive-definite function $V(\mathbf{x})$ such that:

$$\begin{aligned} V(\mathbf{x}, t) &> V(\mathbf{x}) \\ V(\mathbf{0}, t) &= 0 \end{aligned} \quad \text{for all } t \geq t_0$$

decreasing (i.e. the time derivative of the total energy must be negative-definite¹) until an equilibrium state is reached. That is, if a system has an asymptotically stable equilibrium state, then the stored energy of the system displaced within the domain of attraction decays with increasing time until it finally assumes its minimum value at the equilibrium state.

By generalizing this idea, Liapunov introduced the Liapunov function $V(\mathbf{x}, t)$, which somehow related to the energy function of a system, but is more generally and more widely applicable. Liapunov's main stability theorem is stated as follows [1]:

Suppose that a system is described by

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$$

where

$$\mathbf{f}(\mathbf{0}, t) = \mathbf{0} \text{ for all } t$$

If there exists a scalar function $V(\mathbf{x}, t)$ having continuous, first partial derivatives and satisfying the conditions:

- 1. $V(\mathbf{x}, t)$ is positive definite.*
- 2. $\dot{V}(\mathbf{x}, t)$ is negative definite.*

Then the equilibrium state at the origin² is uniformly asymptotically stable (or $V(\mathbf{x}, t) \rightarrow \mathbf{0}$ as $t \rightarrow \infty$).

¹ A scalar function $V(\mathbf{x})$ is said to be negative definite if $-V(\mathbf{x})$ is positive definite.

² Since any isolated equilibrium state (that is, isolated from each other) can be shifted to the origin of the coordinates, or $\mathbf{f}(\mathbf{0}, t) = \mathbf{0}$, by a translation of coordinates, it can be assumed that the equilibrium state of the system is at the origin without losing the generality.

It should be noted that if $\dot{V}(\mathbf{x}, t)$ is not negative definite, but only negative semi-definite¹, then the trajectory can become tangential to some particular surface $V(\mathbf{x}, t) = C$. If $\dot{V}(\mathbf{x}, t)$ does not vanish identically along any trajectory except at the origin, it can move towards the system origin as t tends to infinity. In that case, the system asymptotic stability can also be achieved.

3.1.5 Discussion of Liapunov stability

Although the Liapunov stability analysis is quite powerful and much easier than other methods for nonlinear systems since the solving of complex system state equations is not required, the stability analysis for these problems is still complicated.

The main problem in Liapunov's second method is how to determine the Liapunov function. There is no rule for choosing a Liapunov function and it is highly dependent on the experience, ingenuity and even intuition. The only clue is that Liapunov function somehow relates to the system's energy (so it can be guessed based on the observed system behaviour). However, that relation is fictitious and guessing is possible for simple systems, but for complex industrial systems, finding a suitable function will be quite difficult.

¹ A scalar function $V(\mathbf{x})$ is said to be negative semi-definite if $-V(\mathbf{x})$ is positive semi-definite.

A scalar function $V(\mathbf{x})$ is said to be positive semi-definite negative if it is positive at all states in the region Ω except at the origin and at certain other states, where it is zero.

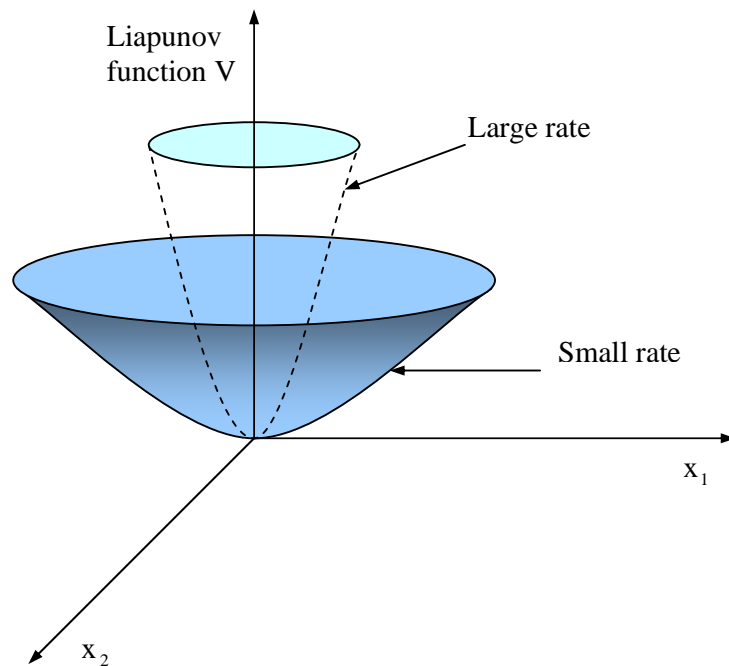


Figure 3.1.2: Liapunov functions with different negative derivative

The absolute value of the negative derivative of Liapunov function $\dot{V}(\mathbf{x}, t)$ is an important factor in the performance of the controlled system. It is proportional to the rate at which the system converges to the stable equilibrium point. That is, the larger the magnitude, the higher the rate of convergence, the faster the controller drives systems into the stable equilibrium state.

If sliding mode control is applied, the stable equilibrium state is the surface where the sliding variable is zero. The error begins to converge towards zero only after the system has reached this surface (see 3.3). Therefore, the higher the absolute value of $\dot{V}(\mathbf{x}, t)$, the better the system's transient response will be. More discussion about the effects of this value on the system performance will be presented in section 4.5.

In this project, the Liapunov stability criterion will be used in conjunction with the sliding mode control scheme (see 3.3) to prove the stability of the system.

3.2 Adaptive control

3.2.1 Introduction

For classical controllers (e.g. using state feedback laws or pole-placement methods), it is required that the system parameters should be known or measurable in advance. The control gains will then be defined based on that specific information. Therefore, whenever the system conditions are changed rapidly or over a wide magnitude, the controller signal will be unable to drive the system into its desired stable state. If the variation of system parameters is small and slow, the performance of classical controllers can be acceptable. However, most practical systems are quite complex with high non-linearities, inherently instability and many parameter uncertainties due to external environmental changes. Therefore, an adaptive control scheme (self-tuning, [2]) is applied to improve the controller performance.

The main purpose of the adaptive controller is to force the controlled system to its optimum performance according to some desired criterion, even under the changing of environmental conditions. It is achieved by adaptively updating (estimating) the system parameters (bounds) for modifying the control signal so that it will be suitable in new operating and environmental conditions. The control signal in turn will vary the system parameters and then the results will be feed back to adjust the adaptive laws. By using this scheme, the effects of system uncertainty parameters will be removed.

The adaptive rules do not create any control signal but are only used to learn the system parameters and vary the control parameters. That is, they must be used in conjunction with other existing control techniques, to help the controller achieve optimal performance corresponding to system conditions at each instant.

3.2.2 Definition of adaptive control system

Ideally, an adaptive control system is considered as a system that can adjust its own structure and parameters to accomplish a specified purpose, without prior knowledge of the environment in which the system will operate [3].

The adaptation approach may be direct control where controller parameters are adjusted using the output error, indirect control where the plant parameters are estimated on-line and used to determine the controller parameters, or the modified variable structure method which uses discontinuous control and high-gain feedback of the output error. The variety in the adaptation approaches leads to many different definitions and classifications of adaptive controller systems. However, a comprehensive and generally accepted definition of an adaptive control system is given by Ogata [1] as follows:

“An adaptive control system is one that continuously and automatically measures the dynamic characteristics (such as the transfer function or state equation) of the plant, compares them with the desired dynamic characteristics, and uses the difference to vary adjustable system parameters (usually controller characteristics) or to generate an actuating signal so the optimal performance can be maintained regardless of the environmental changes; such a system may measure its own performance

according to a given performance index and modify, if necessary, its own parameters so as to maintain optimal performance regardless of the environmental changes.”

That is, any controller capable of adjusting its parameters on-line to achieve optimal performance (within specific system conditions over the time interval of interest) can be considered as an adaptive controller. The main difference between an adaptive control system and a conventional feedback control is that an adaptive scheme is aimed to eliminate the effects of structural disturbance (e.g. parameter variations) while the latter is oriented to eliminate the effects of state disturbance upon the system performance [4].

3.2.3 Adaptive control structure and algorithm

In general, the algorithm for an adaptive controller (self-tuner) is as follows:

1. Online estimate the system parameters and other required variables by recursive least-squares method¹.
2. Substitute the obtained estimation values into the adaptive rules to adjust the controller parameters.
3. Update the control signal based on its new parameters.

All three of the above steps will be repeated at each sampling period. A general structure of an adaptive controller is shown in Figure 3.2.1.

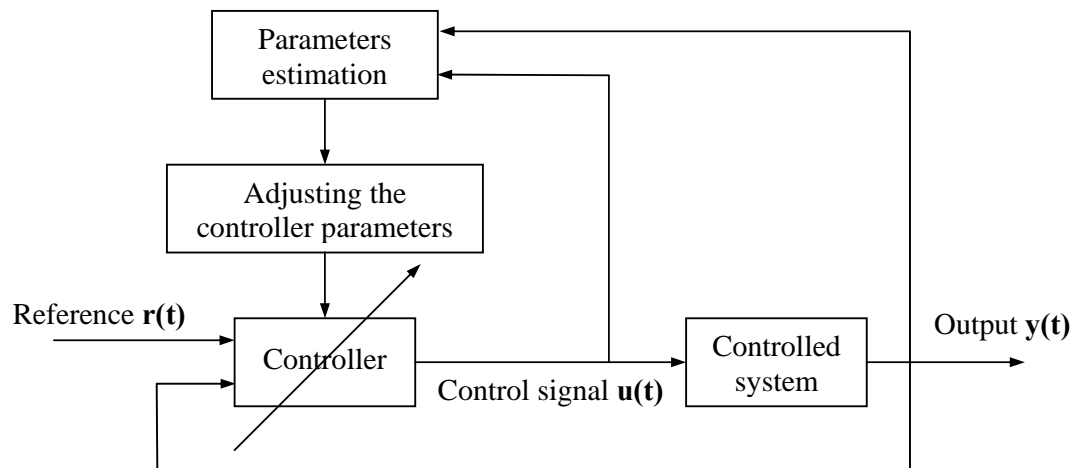


Figure 3.2.1: Block diagram of an adaptive control system

¹ The main idea of least-squares estimation is to minimize the sum of the squares of errors between measured values and predicted values. The recursive least-squares method estimates based on historical observations, particularly, it adjusts the estimated value $\hat{\theta}(n+1)$ by adding the previous estimate $\hat{\theta}(n)$ with a correction term, which is proportional to the error between measured output $y(n+1)$ and its prediction value. More details can be found in [5].

In fact, there are various configurations for an adaptive control system depending on the particular application and the basis of the control theory used. The above model is presented largely because it is quite suitable for the controller system in this project.

3.2.4 Discussion of adaptive control

3.2.4.1 Advantages of adaptive control

As mentioned above, the adaptive control scheme has a lot of superior characteristics over other classical control schemes. Some of its important advantages are:

- **Strong robustness:** the controlled system performance has high immunity with environmental changes, uncertainties and moderate modelling errors since system parameters are online estimated and adjusted.
- **Performance oriented:** the estimates of system parameters are adaptively adjusted not to obtain their exact values, but to make the system output converge to its desired reference.
- **High reliability and self-repair:** the system can compensate the failures of minor system components or internal parameters.

Due to these advantages, adaptive control schemes are widely used in control systems that require high performance over a wide range of operating conditions, especially in tracking systems. Some examples are aerospace vehicles, robotic systems, autopilots for ship navigation, and radar target tracking applications.

3.2.4.2 Disadvantages of adaptive control

Despite such elegant and promising characteristics, some matters should be considered in applying the adaptive control system. Some disadvantages of the adaptive control scheme are briefly shown below.

- **High cost:** Firstly, the adaptive control scheme requires a lot measurement of system parameters. The amount of required sensory equipment as well as its cost is not always affordable.

Secondly, the adaptive laws to update controller parameters are differential equations in nature, and therefore integrators are necessary to manipulate the required data. The integrators can be a micro-controller or implemented by circuits. The number of required integrators increases dramatically with the order and complexity of the controlled plant. Of course, the overall cost for the controller system will be increased radically.

- **Difficult to apply:** An adaptive control scheme can only be applied to systems whose parameters can be continuously measured. It is not always possible to obtain all the needed information.
- **Complex design:** The design of an adaptive control system often involves a large number of mathematical calculations. Mathematical proof of stability may be very challenging, and even so, stability is not always adequately analyzed. In addition, some complex and high-performance systems may require additional designs (or installations) for estimators to enable parameter identification.

The required efforts and associated costs in implementing an adaptive control system are the main obstacles for its application in reality. Therefore, an adaptive control

system is only a good investment when there is no simpler control scheme to satisfy the required controller performance or the operating environment of the system is highly unsettling.

3.2.5 Why should adaptive control be used for this project?

As mentioned before, this project required real-time tracking control for the balancing beam system with high non-linearities, uncertain dynamics and parameter variations. In addition, the number of parameters required for online measurement is limited so not much sensor equipment is required. The controller is implemented in the computer, i.e. no complex circuits have to be used as integrators or computing units. Therefore, the application of an adaptive control scheme for the system in this project is both necessary and possible.

For the adaptive controller, the adaptive rate is one important parameter that will affect the system performance. Moreover, notice should be taken since a digital computer is used as the system controller (e.g. round-off effects, sampling time, etc). The effects of those parameters will be considered and discussed in the design section (see 4.5).

3.3 Sliding mode control

3.3.1 Introduction

The sliding mode control (SMC) scheme (or Variable structure control, [6]) was first introduced and developed in the Soviet Union by Emilyanov in 1966. It then attracted a lot of attention from researchers not only in Soviet literature but also around the world, in both theoretical and applied aspects.

The purpose of sliding mode control is to maintain the system behaviour within a pre-specified trajectory (sliding mode state). Whenever the system dynamics go out of the preferred state, the controller varies the control gain so that the plant's state trajectory comes back to the sliding mode surface. It will create rapidly switching operations within the neighbourhood of the desired trajectory (sliding manifold).

By applying this scheme, the controller can drive the system from an arbitrary initial state to the sliding mode surface along a pre-specified trajectory. Once the system has reached that surface, the system dynamics behaviour will totally follow the pre-defined characteristics and be insensitive to non-linearities, parameter variations and bounded disturbances.

3.3.2 The principle of sliding mode control

Consider an n -dimensional system and a $(n-1)$ -dimensional hyperplane (switching boundary) $S: \mathbb{R}^n \rightarrow \mathbb{R}^1$ given by $s(\mathbf{x}) = 0$ where \mathbf{x} is the system state vector. By using a control signal of the form:

$$\mathbf{u}(\mathbf{x}) = \begin{cases} \mathbf{u}^+(\mathbf{x}) & \text{if } s(\mathbf{x}) > 0 \\ \mathbf{u}^-(\mathbf{x}) & \text{if } s(\mathbf{x}) < 0 \end{cases}$$

the system will become of the form:

$$\dot{\mathbf{x}} = \begin{cases} \mathbf{f}_+(\mathbf{x}) & \text{if } s(\mathbf{x}) > 0 \\ \mathbf{f}_-(\mathbf{x}) & \text{if } s(\mathbf{x}) < 0 \end{cases}$$

where \mathbf{f}_+ and \mathbf{f}_- are smooth functions from $\mathbb{R}^n \rightarrow \mathbb{R}^n$ and the vector fields represented by \mathbf{f}_+ and \mathbf{f}_- should be such that near S , they point towards S on both sides of S (negative and positive side).

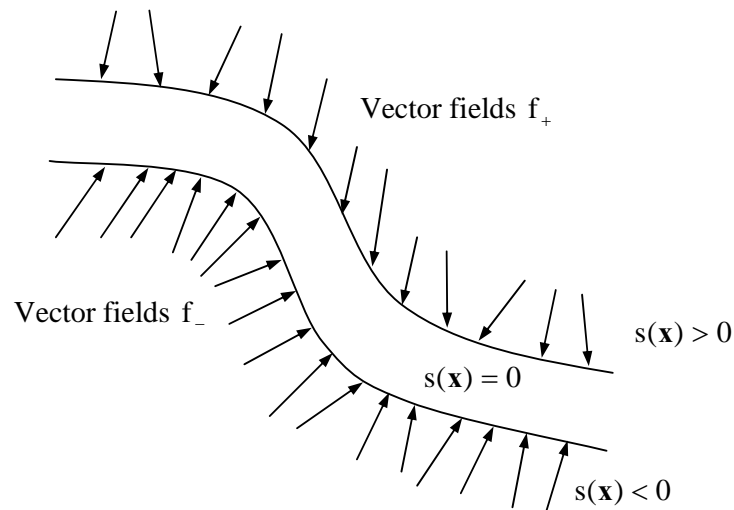


Figure 3.3.1: A graphical representation of sliding mode control

A sliding mode is defined on the surface S and once the system reaches to S , it remains there with reduced-order dynamics determined by the system itself and the switching surface $s(\mathbf{x}) = 0$. From then on, the system behaviour will not be affected by non-linearities, parameter variations and other bounded disturbances. Quite often, the sliding variable is chosen as the (tracking) error so that the system error will asymptotically converge to zero in the sliding mode state.

The condition for sliding can be expressed as follows:

$$s \cdot \dot{s} < 0 \quad \text{near } s(\mathbf{x}) = 0$$

3.3.3 Discussion of sliding mode control

3.3.3.1 Advantages of sliding mode control

The most prominent feature of the sliding mode control scheme is its **strong robustness**. From any given initial point in the state space, the sliding mode controller can quickly drive the system behaviour into the sliding mode surface and constrain it with the desired characteristics. In the sliding mode, the closed loop control system is completely insensitive to non-linearities, uncertain dynamics and totally rejects bounded input disturbances [7].

Due to that superior characteristic, the sliding mode control scheme is highly appropriate for applications with non-linearities and requiring fast response, anti-disturbance but not focused on a smooth trajectory. Some examples are for military systems such as missile guidance systems and autopilot navigator devices.

3.3.3.2 Disadvantages of sliding mode control – Chattering effect

Although the sliding mode controller possesses advantageous attributes such as strong robustness and swift response, there are some unavoidable problems with its design as well as the control system performance, as follows:

- **Unknown converge time:** The sliding mode controller can provide only asymptotic convergence of system dynamics (e.g. tracking error). That is, it only assures that the system will reach to the stable state as time tends to infinity but cannot provide a specific, finite value of the convergence time.
- **Complex design:** The design requires prior knowledge of the bounds of system uncertainties. The more complex the system structure, the more difficult the task the designer has to cope with. However, an adaptive control scheme may be added in to overcome this problem.
- **Demanding calculations:** It requires knowledge about the full state vector of the system during the control process. For complex systems, it is not always possible to measure all required parameters and estimation has to be employed. It means more calculations are required and the control performance may be slowed down.
- **Chattering effect:** Due to the discontinuous nature of the control signal (in attempting to maintain $s(\mathbf{x})$ at zero), the fast switching around the sliding mode surface is inevitable. If the switching frequency is infinite, no problem will be encountered. However, the imperfections of actual switching devices (e.g. small time constants, hysteretic) limit the switching frequency to a finite value. It not only reduces the sliding accuracy and but more severely, makes

the system oscillate around the sliding mode surface. This phenomenon is referred to as chattering.

Another reason of chattering effect is the interacting between some high-frequency dynamics, which was neglected during the design (e.g. un-modelled structure modes, neglected time delays), with the switching operations. These interactions may cause non-decaying oscillatory components with finite amplitude and frequency and may eventually make the system unstable.

This is the most undesirable outcome of the sliding mode control scheme. Some methods have been proposed to reduce or even eliminate its effects. Detailed discussion of these methods (boundary layer control, fuzzy logic tuning) will be shown in section 3.4.

3.3.4 Why should sliding mode control be used for this project?

From the above discussion, it can be seen that the sliding mode control technique is quite suitable for systems with high non-linearities, uncertain dynamics and parameter variations, like the balancing beam system in this project. Moreover, the problems associated with this control scheme will be overcome by combining sliding mode control with others techniques. For example, unknown bounds of the system will be estimated by using an adaptive scheme and the chattering effect will be eliminated by an additional fuzzy logic controller.

Since this project's system is a tracking application, the sliding mode will be described in terms of the tracking error (e.g. $\dot{\varepsilon} + \lambda\varepsilon = 0$) to ensure its asymptotic convergence to zero. The error convergence is divided into two steps, as follows:

1. The controller drives the sliding variable s to reach the sliding mode (zero).
2. On the sliding mode surface, the error dynamics are no longer affected by non-linearities, uncertainties and disturbances and are fully determined by the sliding mode dynamics. It will then asymptotically converge to zero as time increases to infinity.

The graphical representation of these steps is shown in Figure 3.3.2.

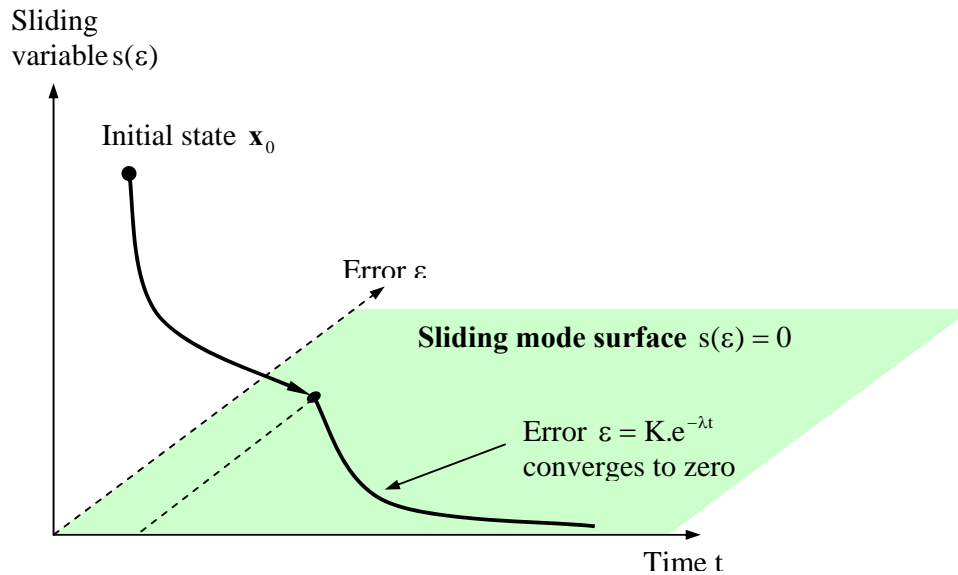


Figure 3.3.2: Convergence processes in sliding mode control

The error convergence rate λ is an important factor affecting the convergence performance of the control system. Whether the error can reach to zero or not, how fast the convergence rate will be, etc highly depend on the chosen value of λ . More detailed discussion about this parameter will be represented in section 4.5.

3.4 Fuzzy logic control

3.4.1 Introduction

As mentioned above (section 3.3.3.2), chattering is an inescapable effect when a sliding mode control scheme is utilized due to finite-frequency switching of the control signal. It reduces the accuracy of sliding control, increases the rate of wear of control motors, etc. The normal harm created by chattering is the reduction in system reliability but extreme chattering may drive the system out of its stable state to unexpected fluctuation. To reduce these objectionable outcomes, a fuzzy logic control scheme may be applied around the sliding mode surface.

The fuzzy controller was first developed by Mamdani and his colleagues in the 1970s, based on Zahed's seminal papers on the linguistic approach and system analysis with fuzzy set theory. Over the years, fuzzy logic theory control has attracted the attention of many researchers and been successfully applied in various industrial processes such as water quality control, automatic train operator systems, nuclear reactor control, etc. Generally, it is quite suitable for complex ill-defined processes that can be controlled with (human) expert experience and practical understanding of the system behaviour without knowing its underlying dynamics [8], [9].

In this section, some basic theory about fuzzy and fuzzy logic controller, which closely relates to the present project, will be briefly presented. Based on this background information, the fuzzy logic approach will be compared with the boundary layer control method in chattering reduction to explain why a fuzzy logic controller is preferred in this project.

3.4.2 Fuzzy set theory

Fuzzy logic, the foundation concept of developed fuzzy control, is a logic form much closer in spirit to human thinking and natural language than traditional logical systems. Proposed and developed by Zadeh from 1965 (Zimmermann, 1991), fuzzy set theory deals with imprecise, ambiguous facts of the real world and presents them in linguistic statements like human intelligence and decision-making.

Fuzzy logic can be considered as an extension of conventional Boolean logic, aiming to handle the concept of partial truths, which lie between "completely true" ("black") and "completely false" ("white"). Imaginably, classical set theory is concerned with "crisp" sets with sharp boundaries whereas fuzzy set theory deals with "fuzzy" sets whose boundaries are "grey". Particularly, in classical set theory, an element (x) can either belong to a set \mathbf{A} or not, i.e. the degree to which the element u belongs to set \mathbf{A} is either 1 or 0. However, the degree of belonging of an element x to a fuzzy set \mathbf{A} is a real number between 0 and 1 (i.e. the transition from "belonging to a set" to "not belonging to a set" is gradual).

3.4.2.1 Definitions of fuzzy set

Following are some basic definitions of fuzzy set theory:

1. A fuzzy set \mathbf{A} of the universe of discourse \mathbf{U} is characterized by a membership function $\mu_{\mathbf{A}} : \mathbf{U} \rightarrow [0,1]$ that associates each element $x \in \mathbf{U}$ with a number $\mu_{\mathbf{A}}(x) \in [0,1]$ represented the grade of membership of x in \mathbf{A} . Formally, a fuzzy set \mathbf{A} in \mathbf{U} is defined as a set of ordered pairs:

$$\mathbf{A} = \{(x, \mu_{\mathbf{A}}(x) \mid x \in \mathbf{U})\}$$

2. The support of a fuzzy set A is the crisp set of all $x \in U$ such that $\mu_A(x) > 0$.
3. A fuzzy set A is called a fuzzy singleton if the support of A contains only one element and its membership is 1.

3.4.2.2 Basic operations on fuzzy set

Since fuzzy sets are defined by membership functions, the operations on fuzzy set are in fact the mathematical calculations in terms of membership functions. Some basic operations on fuzzy set are presented follows:

1. **Complement:** The complement \bar{A} of fuzzy set A is defined by

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad u \in U$$

2. **Intersection:** The intersection of two fuzzy sets A and B is a fuzzy set $C = A \cap B$ such that for each $x \in U$:

$$\begin{aligned} \mu_{A \cap B}(x) &= \min\{\mu_A(x), \mu_B(x)\} \\ &= \mu_A(x) \wedge \mu_B(x) \end{aligned}$$

3. **Union:** The union of fuzzy set A and fuzzy set B , denoted as $D = A \cup B$, is a fuzzy set such that for each $x \in U$:

$$\begin{aligned} \mu_{A \cup B}(x) &= \max\{\mu_A(x), \mu_B(x)\} \\ &= \mu_A(x) \vee \mu_B(x) \end{aligned}$$

4. **Fuzzy relation:** A fuzzy relation R from a set X to a set Y is defined as a fuzzy subset of the Cartesian product $X \times Y$:

$$R = \int_{X \times Y}^{\Delta} \mu_R(x, y) / (x, y) \quad \forall x \in X, y \in Y^1$$

¹ Note that the integration and “/” signs are only markers and do not indicate integration or division.

Alternative definition¹: Assume A and B are two fuzzy subset of X and Y , respectively, their cross (Cartesian) product $A \times B$ is a fuzzy relationship T on the set $X \times Y$, denoted $T = A \times B$ where:

$$T(x, y) = \min[A(x), B(y)]$$

The graphical representations of some operations are shown in Figure 3.4.1.

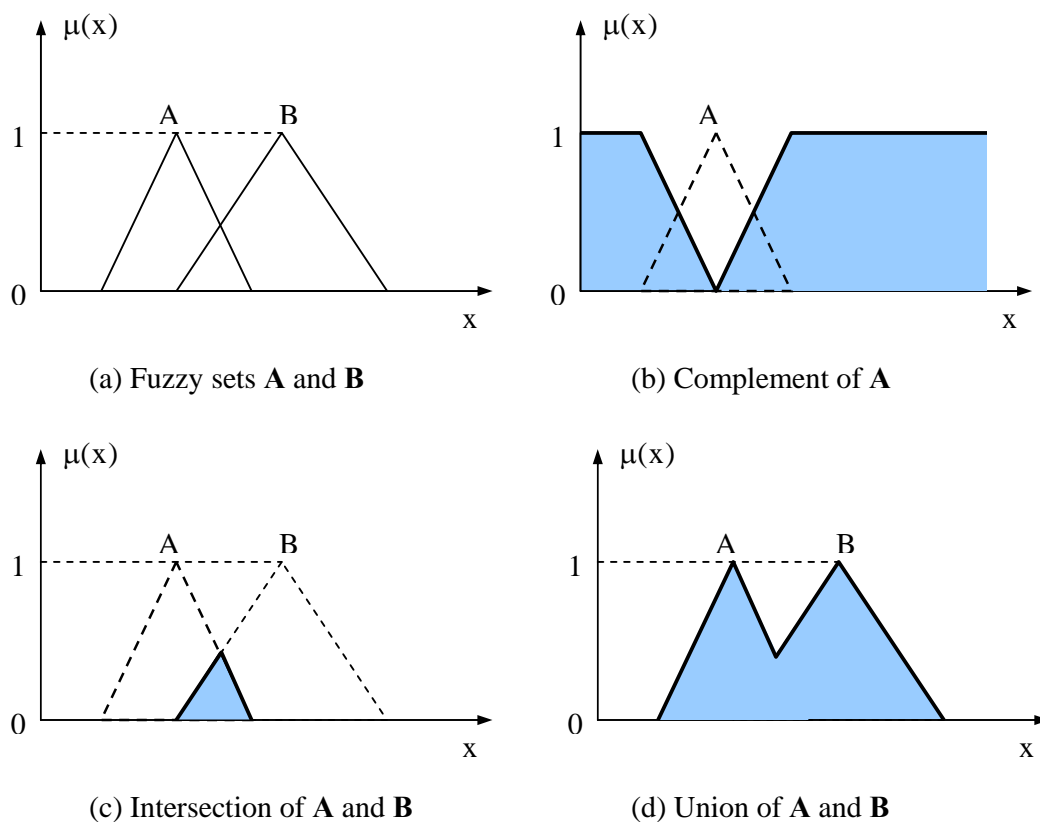


Figure 3.4.1: Basic operations on fuzzy set

¹ This definition is often used to obtain fuzzy relationships in practice [10].

3.4.3 Fuzzy interference and fuzzy reasoning

3.4.3.1 Fuzzy if-then rules

A fuzzy system consists of a collection of fuzzy IF-THEN rules, for which the form for a two-input-single-output system is as below:

$$R_i: \text{If } x \text{ is } A_i \text{ and } y \text{ is } B_i, \text{ then } z \text{ is } C_i$$

where x, y, z are linguistic variables; A_i, B_i, C_i are linguistic values (e.g. “slow”, “fast”, etc) defined by fuzzy sets on universes X, Y and $X \times Y$ respectively.

3.4.3.2 Fuzzy interference

In fuzzy logic, there are two main fuzzy interference rules: generalized modus ponens (GMP) and generalized modus tollens (GMT). Let A, B be linguistic values and x, y be linguistic variables. The GMP has the form as follows:

Fact: x is A

Fuzzy rule: If x is A , then y is B

Consequence: y is B

The general form of the GMT method is as below:

Fact: y is B

Fuzzy rule: If x is A , then y is B

Consequence: x is A

It can be seen that the GMP is a kind of forward inference (data-driven inference) while GMT is backward inference (goal-driven inference). That is, the forward method finds consequence from given input conditions while the latter seeks the reasons for what has

happened. Therefore, the GMP method is often used in fuzzy control while GMT is applied in expert systems.

3.4.3.3 The compositional rule of inference

The compositional rule of inference can be regarded as a GMP (forward) rule, which enables the fuzzy system to infer an output corresponding to an arbitrary input not explicitly covered by the rule base. Therefore, it can be used with approximate and uncertain inputs. For that fact, it can reduce the number of relations required to describe a process because it is not necessary to cover all possible inputs.

The simplest form is as follows:

Fact: x is A'

Fuzzy rule R: If x is A, then y is B

Consequence: y is B'

where A, A', B, B' are fuzzy sets of X, X and Y, Y respectively. Let μ_A , $\mu_{A'}$, μ_R , $\mu_{B'}$ be the membership functions of A, A', fuzzy relation R and B'. The min-max composition (so called Mamdani's minimum operation) gives us the resulting fuzzy set:

$$B' = A' \circ R = A' \circ (A \rightarrow B) \quad \text{or}$$

$$\begin{aligned} \mu_{B'}(y) &= \max_x \min[\mu_{A'}(x), \mu_R(x, y)] \\ &= \vee_x [\mu_{A'}(x) \wedge \mu_R(x, y)] \end{aligned}$$

where \vee , \wedge are the fuzzy intersection (AND) and union (OR) operations, which can be defined by the user. The formula above uses min, max for fuzzy operators AND, OR and these assignments are applied for all other rules from now on.

3.4.3.4 Fuzzy reasoning

Fuzzy reasoning is an inference procedure used to derive conclusions from a set of fuzzy if-then rules and one or more conditions. It is based on the max-min composition presented in the above section.

To illustrate the fuzzy reasoning process, consider a simple example with two rules and one antecedent, which is closely related to the present project as will later become evident. The reasoning process for multiple rules and antecedents is just the multiple repetitions of this process.

Fact: x is A' and y is B'

Fuzzy rule R_1 : If x is A_1 and y is B_1 , then z is C_1

$$(R_1 = A_1 \times B_1 \rightarrow C_1)$$

Fuzzy rule R_2 : If x is A_2 and y is B_2 , then z is C_2

$$(R_2 = A_2 \times B_2 \rightarrow C_2)$$

Consequence: z is C'

We have:

$$\begin{aligned} C' &= (A' \times B') \circ (R_1 \cup R_2) \\ &= [(A' \times B') \circ R_1] \cup [(A' \times B') \circ R_2] \\ &= C'_1 \cup C'_2 \end{aligned}$$

where C'_1, C'_2 are the inferred fuzzy sets for rule R_1 and R_2 , defined as follows:

$$\mu_{C'_i}(x) = \left\{ \bigvee_x [\mu_{A'_i}(x) \wedge \mu_{A_i}(x)] \right\} \wedge \left\{ \bigvee_y [\mu_{B'_i}(x) \wedge \mu_{B_i}(x)] \right\}$$

The fuzzy relations R is defined as:

$$\begin{aligned} \mu_R(x, y, z) &= \mu_{(A \times B) \times C}(x, y, z) \\ &= \mu_A(x) \wedge \mu_B(y) \wedge \mu_C(z) \end{aligned}$$

The graphical representation of this process is shown in Figure 3.4.2.

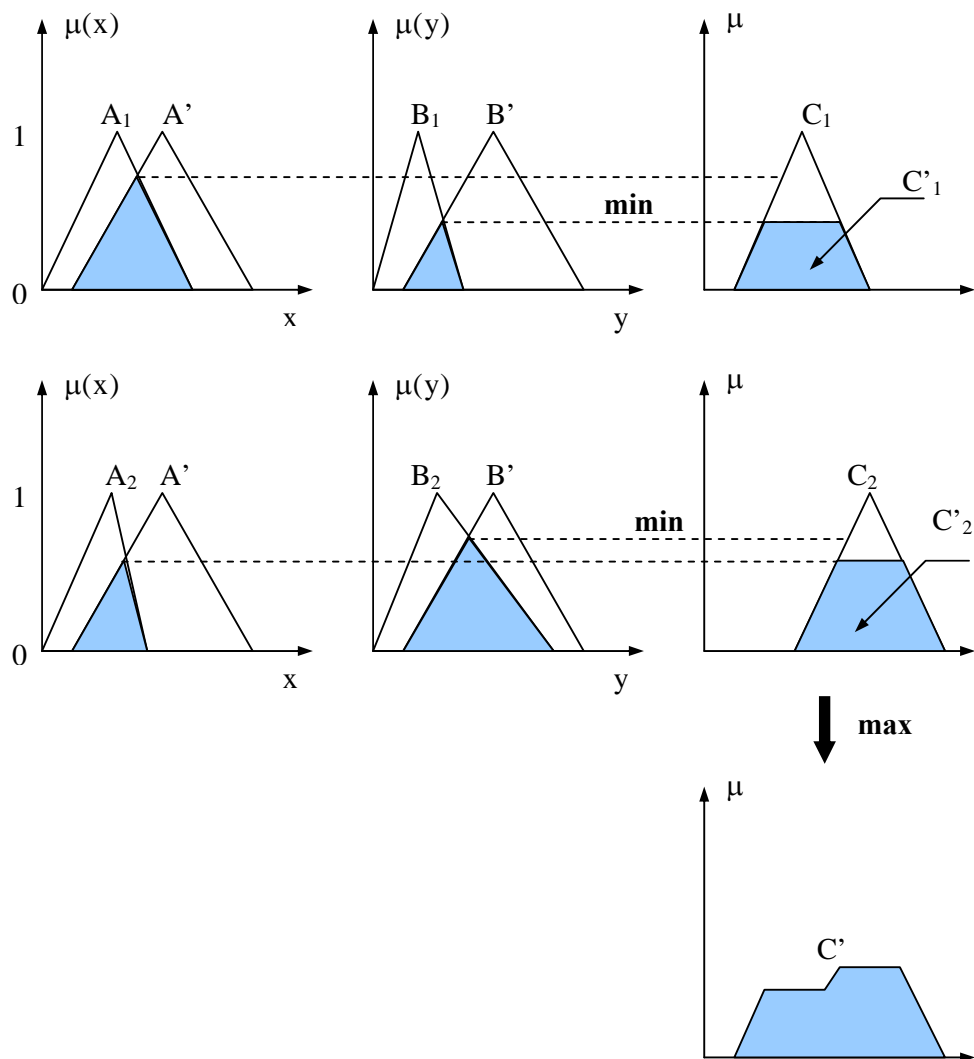


Figure 3.4.2: A simple fuzzy reasoning process

3.4.4 General structure of a fuzzy logic controller

In general, the structure of a fuzzy logic controller is as illustrated in Figure 3.4.3 below [8].

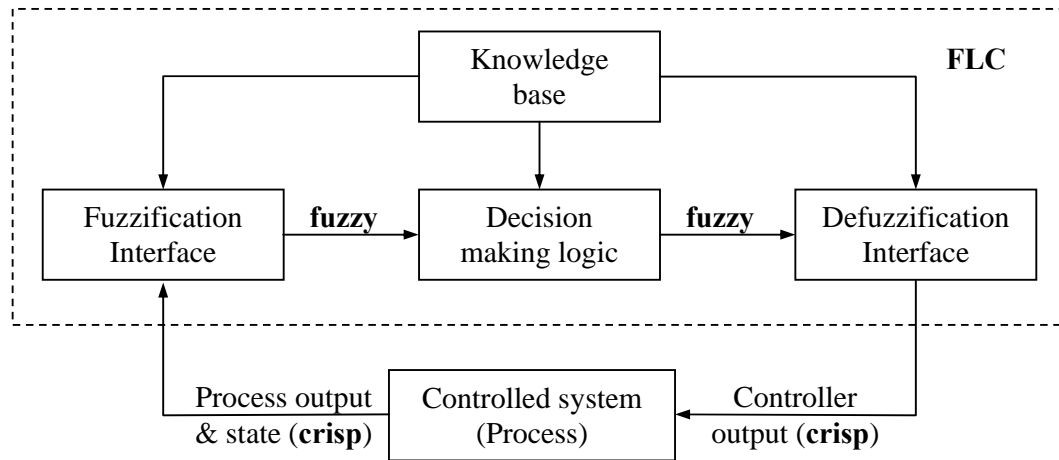


Figure 3.4.3: The general structure of a fuzzy logic controller (FLC)

As can be seen, the four principal components of a fuzzy logic controller are:

1. The fuzzification interface which:
 - Receives and measures the (crisp) values of input variables.
 - Performs a scale mapping that transfers the range of values of input variables into a suitable domain or universe of discourse.
 - Performs fuzzification that converts the input crisp data into suitable linguistic values, which may be viewed as labels of fuzzy sets.
2. The knowledge base comprises knowledge of the application domain and the attendant control goals. Particularly, it consists of a “database” and a “linguistic – fuzzy – control rule base”:
 - The database provides necessary definitions (e.g. about boundaries, fuzzy sets), which are used to define linguistic control rules and fuzzy data manipulation.

- The rule base characterises the control goals and control policy of the domain experts by means of a set of linguistic control rules.
3. The decision-making logic, which is the kernel of a FLC and is able to:
 - Simulate the human decision-making based on fuzzy concepts.
 - Interfere with necessary fuzzy control actions by using fuzzy implication and the rules of inference in fuzzy logic (to determine suitable outputs corresponding to the measured inputs).
 4. The defuzzification interface which:
 - Performs a scale mapping, which converts the range of values of output variables into an appropriate domain or universe of discourse.
 - Performs defuzzification to create a non-fuzzy control action from an inferred fuzzy control action (e.g. in this project, to convert fuzzy output values into corresponding crisp values).

3.4.5 Logic processing in a fuzzy controller

A fuzzy logic controller, like any other kind of controllers, is aimed to produce a suitable (desired) set of outputs corresponding to a given set of input values. In most rule-based fuzzy logic controllers, this goal is achieved through the three following primary steps:

1. Fuzzification
2. Fuzzy reasoning (rule evaluation)
3. Defuzzification

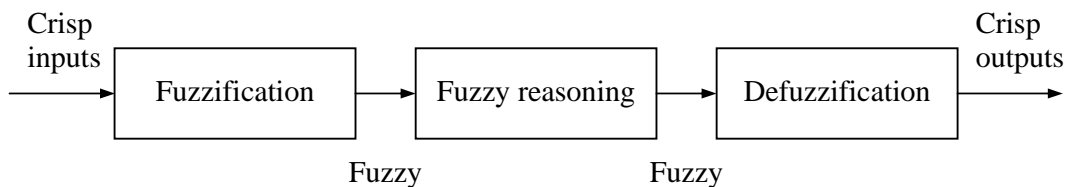


Figure 3.4.4: Fuzzy logic processing in FLC

3.4.5.1 Fuzzification

The fuzzification is used to deal with the vagueness, imprecision and uncertain information in a natural language. This is the first step in the logic processing of a typical FLC, which converts the observed crisp values of input variables into fuzzy linguistic variables in certain universes of discourse.

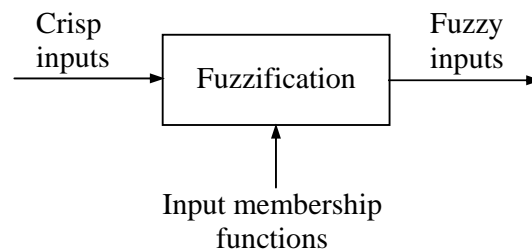


Figure 3.4.5: The fuzzification process

To design the fuzzification rules for a FLC, the following steps are required:

1. Firstly, the range of the complete input universe, the number of fuzzy labels (variables) within that set and their names are defined.
2. Each fuzzy label is then characterised by a membership function¹ to cover a particular sub-range of the input universe. Those membership functions provide the numerical meaning to fuzzy variables via the “degree of membership” (“grade of membership”) concept.

¹ Note that there are several common membership functions such as triangular, trapezoidal, Gaussian distribution curves, bell-shaped function and sigmoidal (opened or closed) functions that can be used for the fuzzification purpose. The first two membership functions are constructed from piecewise straight lines, which are rather simple and therefore, fast calculation can be obtained. Meanwhile, the advantages of other kinds are their smoothness, concise notation, non-zero at all points and sigmoid can even provides asymmetric functions, which are important in certain applications. However, the trade off is that complex and time-consuming computations are required.

The fuzzification process to convert a crisp input value into fuzzy values consists of the following steps:

1. Receives and scales the measured crisp value of input variable.
2. Compares the crisp value with the ranges of all fuzzy labels to determine the range to which this value belongs. When the crisp input falls into a certain range of a fuzzy label, it will be associated with this fuzzy label by a “degree of membership” number, which is found from the value of crisp input and the shape of the membership function of this label.
3. If the crisp input lies in a range that is covered by two or more fuzzy labels, it will be associated with all these fuzzy labels, with degrees of membership are defined as step 2.

3.4.5.2 Fuzzy reasoning (rule evaluation)

Fuzzy reasoning is the essential part of the logic processing within a FLC. In this step, the fuzzy inputs from the fuzzification process are used with the antecedent variables in conjunction with the knowledge base of FLC to evaluate the values of fuzzy outputs (for the next step – defuzzification).

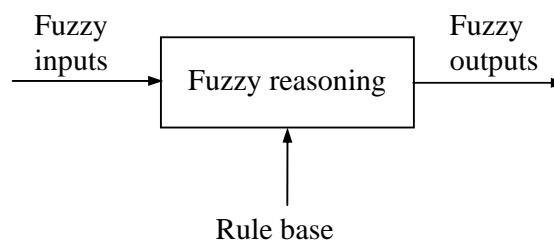


Figure 3.4.6: The fuzzy reasoning (rule evaluation) process

The rule base is a set of **if-then** statements written in terms of fuzzy linguistic labels, which follows the common sense behaviour of the controlled system to describe the suitable control actions in response to various fuzzy inputs.

For the convenience of the present design, consider fuzzy controllers with two inputs and one output. Assuming that the designed rule base of that kind of FLC has m rules and is represented in the following format (this rule base can be represented in the matrix format, as will later be used in the project's design):

$$\left\{ \begin{array}{l} \text{Rule 1: IF } (u_1 \text{ is } B_{11}) \text{ AND } (u_2 \text{ is } B_{12}) \text{ THEN } (v \text{ is } D_1) \\ \dots \\ \text{Rule } i: \text{ IF } (u_1 \text{ is } B_{i1}) \text{ AND } (u_2 \text{ is } B_{i2}) \text{ THEN } (v \text{ is } D_i) \\ \dots \\ \text{Rule } m: \text{ IF } (u_1 \text{ is } B_{m1}) \text{ AND } (u_2 \text{ is } B_{m2}) \text{ THEN } (v \text{ is } D_m) \end{array} \right.$$

In which

- u_1, u_2 and v are input and output fuzzy variables.
- x_1, x_2 and y are input and output crisp variables, corresponding to fuzzy variables u_1, u_2 and v .
- B_{i1}, B_{i2} and D_i are fuzzy sets with membership functions $B_{i1}(x_1), B_{i2}(x_2)$ and $D_i(y)$ respectively.

Assume the crisp inputs are x_1^*, x_2^* , the rule evaluation process (using the Mamdani method) to find out the fuzzy output of the about knowledge base from those inputs consists of the following steps:

1. Calculate the “degree of firing” τ_i (DOF) of each rule R_i with respect to the input values $u_1 = x_1^*$ and $u_2 = x_2^*$. This value represents the truthfulness (or relevance) of the antecedent part of the i^{th} rule.

$$\tau_i = B_{i1}(x_1^*) \wedge B_{i2}(x_2^*) = \min(B_{i1}(x_1^*), B_{i2}(x_2^*)) \quad (3.4.1)$$

Note that $B_{i1}(x_1^*), B_{i2}(x_2^*)$ are numbers and so τ_i is.

2. Find the “rule strength” (or “degree of truth”, fuzzy implication) F_i of each rule R_i by the following formula:

$$F_i(Y) = \tau_i \wedge \mathbf{D}_i(Y) = [\min(\tau_i, D_i(y_1)), \dots, \min(\tau_i, D_i(y_n))] \quad (3.4.2)$$

Note that in the practical situation, the output universe Y has already been equidistantly discretized into n small ranges as $[y_1, y_2, \dots, y_j, \dots, y_n]$. Then the degrees of membership of reference fuzzy set (output fuzzy label) D_i at all y_j are calculated to represent the membership function of D_i by the array:

$$\mathbf{D}_i(Y) = [D_i(y_1), D_i(y_2), \dots, D_i(y_j), \dots, D_i(y_n)]$$

So the rule strength will be represented in the form of an array as follows:

$$F_i(Y) = [F_i(y_1), F_i(y_2), \dots, F_i(y_j), \dots, F_i(y_n)]$$

3. Calculate the fuzzy output of the overall knowledge base (from the fuzzy outputs of each rule obtained in step 2)

$$\begin{aligned} F(Y) &= \vee_i F_i(Y) \\ &= \vee_i (\tau_i \wedge \mathbf{D}_i(Y)) \\ &= \left[\max\{F_i(y_1), i = 1..m\}, \dots, \max\{F_i(y_j), i = 1..m\}, \dots \right] \\ &\quad \dots, \max\{F_i(y_n), i = 1..m\} \end{aligned} \quad (3.4.3)$$

In fact, this formula has the spirit of a weighted aggregation of the individual rule consequents, in which the weights are determined by the degree of firing τ_i of each rule ($i = 1, 2, \dots, m$).

The Figure 3.4.7 illustrates the reasoning process for a simple rule base with two rules.

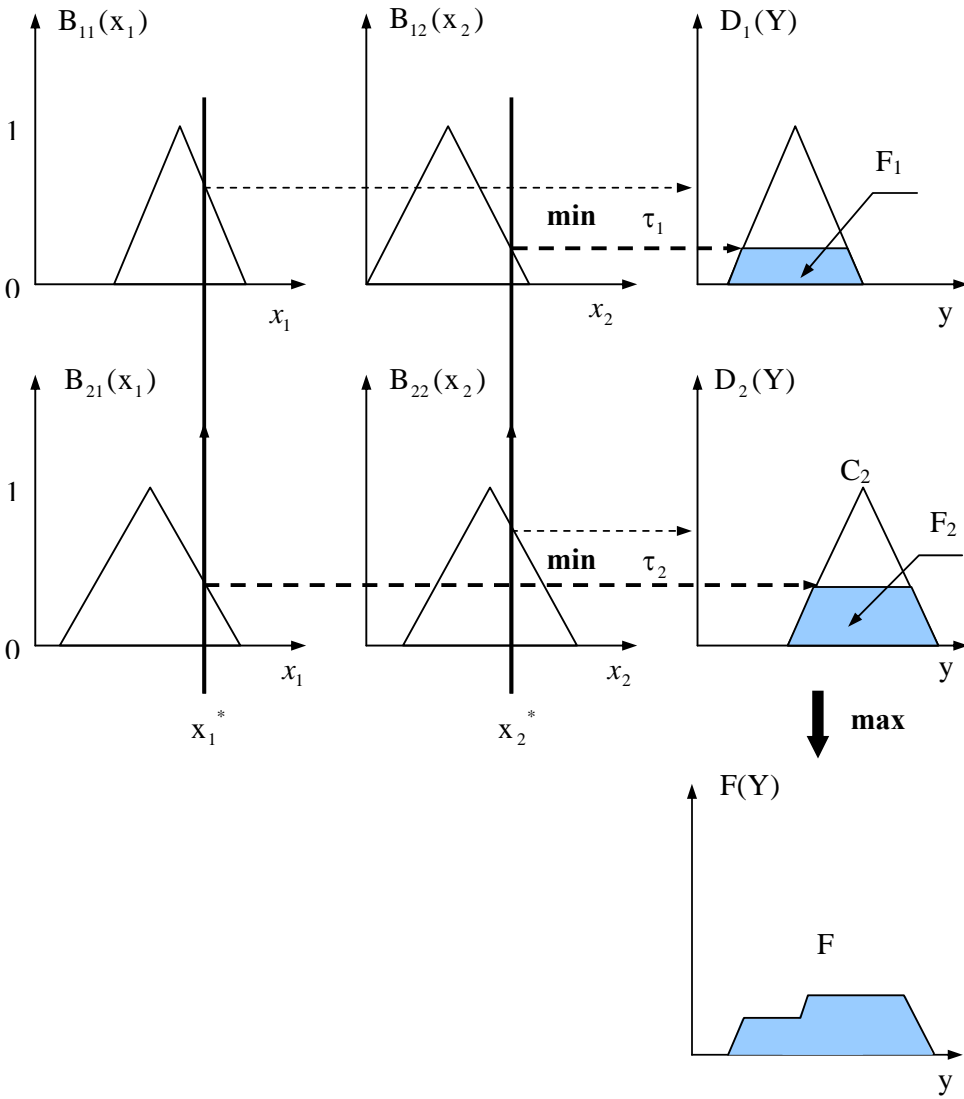


Figure 3.4.7: The fuzzy reasoning (rule evaluation) process in FLC

3.4.5.3 Defuzzification

This is the final step to convert the fuzzy output of the fuzzy controller to a single crisp value for the controlled system. It aims to produce a non-fuzzy control action that best represents the possibility distribution of an inferred fuzzy control action.

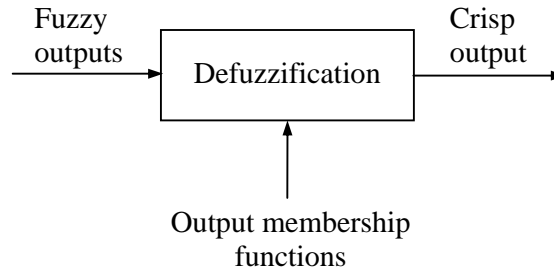


Figure 3.4.8: The defuzzification process

Although there is no systematic procedure to determine which is the best defuzzification strategy for which kind of problem, several common methods for defuzzification such as the max criteria, Mean of Maximum (MOM) method and Center of Area (COA) method are currently employed.

The first technique produces the point at which the possibility distribution of the control action reaches a maximum value. Meanwhile, the MOM method defines the crisp output as a mean of all local control actions whose membership functions reach the maximum.

However, the most widely used technique is COA, which is applied in the present project and determines the crisp output value of a fuzzy set F by its centre of gravity of the possibility distribution. The defuzzification formula is shown below:

$$y^* = \frac{\int_Y F(y) \cdot y dy}{\int_Y F(y) dy} \quad \text{or}$$

$$y^* = \frac{\sum_{j=1}^n F(y_j) \cdot y_j}{\sum_{j=1}^n F(y_j)} \quad (3.4.5)$$

3.4.6 Design method of a fuzzy logic controller

Designing a fuzzy logic controller requires the consideration of many factors such as the choosing of fuzzification strategy, setting up the database (partition of the universes, choose membership functions), determining the rule base, selecting a decision-making logic mechanism and the defuzzification strategy. In general, the design process of a fuzzy controller follows the following steps:

1. Analyse and partition the control system:

- The behaviour of the system should be analyzed in the early stages to decompose the system into individual functional sub-systems if necessary. These sub-systems should be evaluated as to whether they can be modelled using fuzzy logic or not.

2. Define input & output fuzzy universes

- Measure and analyze the system crisp inputs and outputs to determine their suitable universes of discourse (the fuzzy input, output ranges). Sometimes, scaling factors are applied to obtain better fuzzification and defuzzification results.
- If a non-linear relationship exists between the desired system input and the output of a sensing detector, a conversion process may be required. Two popular options are: (1) Represent the membership functions in look-up tables to compensate for the non-linearity of the sensor; (2) Scale the membership functions to the units of implementation and tune these functions while trying to maintain the shape of scaled functions.
- The number of membership functions is often an odd number in the range from 3 to 9. As a rule of thumb, the greater the control required the greater the membership function density in that input region.
- The degree of overlap between membership functions (distribution of membership functions) should be carefully chosen since inappropriate overlap will impact on the system performance and lead to casual behaviour.

- The shapes of membership functions do not take important impacts as their distribution. As discussed in section 3.4.5.1, it is better to use triangular and trapezoidal shapes to achieve equitable (short) calculation time and accommodate low-cost implementation hardware.

3. Construct the rule base

- Define the control rules based on the system characteristics, the expert's experience about the system and the desired model behaviour.

4. Observe model behaviour; verify and fine tune as needed

- The controller should be tried with various combinations of inputs and careful observation of the system output should be made to identify any unexpected or undesired behaviour. This is a really important part in the design process of any fuzzy controller since fuzzy control is the reflection of human knowledge and experience, which is only obtained after numerous trials and errors.
- Choosing suitable distribution, shapes of membership functions in step 2 and defining an appropriate rule base in step 3 are essential factors in the success of the fuzzy controller. Therefore, great effort should be taken in the fine-tuning of these parameters to reflect the desired output more accurately. Once the parameters are well chosen, the response of the system will have very good time domain characteristics.

5. Optimise system for target platform

- In practical implementation, memory requirements and the system's response time are two important issues that should be considered.
- When the real-time calculation is too complex and heavy so that the system hardware (processors, memory) cannot afford it, alternative implementation schemes such as a look-up table¹ or active inference² may be applied.

¹ The outputs for all possible inputs are found off-line and then stored in a look-up table.

² The fuzzification and defuzzification processes are performed in real time on the target system.

3.4.7 Discussion of the fuzzy logic controller

3.4.7.1 Advantages of the fuzzy logic controller

Experience shows that in many applications, fuzzy logic controllers have provided better results than those obtained by conventional control algorithms. Among numerous superior characteristics, the two following features are the significant advantages of FLC:

- When the processes are too complex (e.g. non-linear) to analyse by conventional quantitative methods or the system model is unknown, and systems are mostly controlled by experience, and expert knowledge, FLC can provide an algorithm, which can convert that linguistic control strategy into an automatic control strategy.
- When the available source of information about system behaviour is interpreted qualitatively, inexact or uncertainly, even heavy and sophisticated mathematical approximations of non-fuzzy controllers cannot guarantee an acceptable system performance. In these cases, FLC is an easier and more applicable alternative which can provide better results due to its approximate reasoning capability. Once a fuzzy controller is given, the whole system can actually be considered as a deterministic system.

In fact, fuzzy control logic may be considered as “a step forwards the rapprochement between conventional precise mathematical control and human-like decision making” (Gupta, from [8]).

3.4.7.2 Disadvantages of the fuzzy logic controller

Despite its advantages, there are some problems related with the design as well as the application of the fuzzy logic controller, for example:

- Up to this time, there is no systematic procedure for the design and verification of a fuzzy logic controller but trial-and-error method and human practical knowledge.
- While the decision making of human experts decides the performance level of FLC, it is often informal and unstructured. Therefore, the process of transferring expert knowledge into a usable knowledge base of FLC (knowledge acquisition) is time-consuming and nontrivial.
- In some cases, the computing time of FLC is much longer than that of a mathematical-based controller due to the complex manipulation required in fuzzification, defuzzification and especially in the reasoning process.
- The fuzzy modelling of a process is still not well understood due to the difficulties in modelling its linguistic structure and obtaining data from complex industrial processes. Therefore, the controllability and stability of fuzzy logic controllers are still difficult, or sometimes impossible, to prove.

4 Design controller for the balancing beam system

4.1 Introduction

In this section, the theories discussed above will be applied to design a control scheme for the balancing beam (tracking) system, which is a non-linear system with high uncertainties and disturbance. The system controller is composed of two sub-controllers:

1. An adaptive sliding mode controller (ASMC) to drive the system into the sliding mode surface.
2. A fuzzy logic controller (FLC) to reduce the chattering effects around the sliding mode surface.

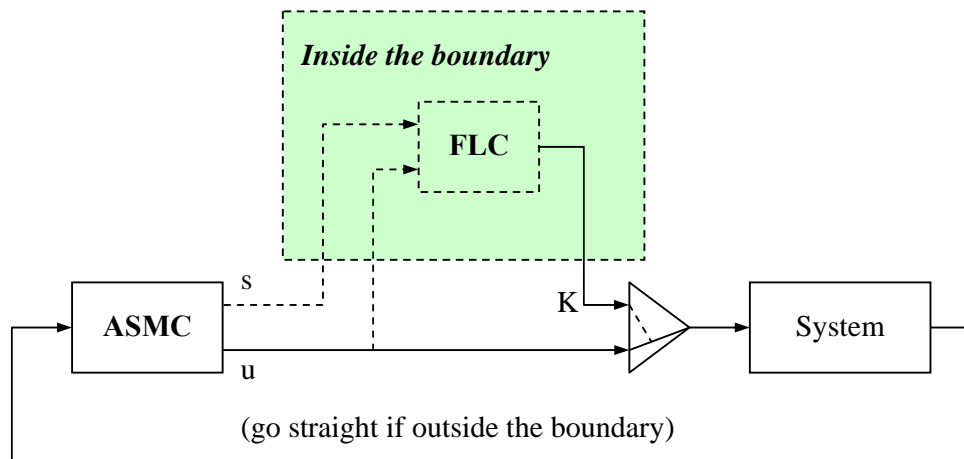


Figure 4.1.1: General structure of the controller

Outside a certain boundary (near the sliding mode surface), only ASMC operates. However, once the system comes inside the boundary, FLC will be activated to smooth the control signal of ASMC (via the variation of control gain). That is, the FLC will be used in conjunction with the ASMC inside that region to reduce the effects of switching operations of the ASMC.

4.2 Adaptive sliding mode controller

The design of ASMC contains two main parts. The first part is to choose the switching surface s to describe the desired error dynamic of the control system. The second part is the design of discontinuous control signal and adaptive rules to drive the system into sliding mode $s = 0$ and remain in it forever. In that state, the dynamic characteristics of the system are independent of the variations of system parameters and disturbance.

In detail, the design process follows some small steps:

1. Define the sliding mode to describe the desired error dynamics of the controlled system.
2. Select a Liapunov candidate function $V(x, t)$.
3. Determine the control law.
4. Determine the adaptive laws to eliminate the uncertainties and disturbances.
5. Prove the stability of the system by using Liapunov criteria.

4.2.1 Control signal and stability proof

The nonlinear dynamics equation of the balancing beam is shown in Equation 2.2.2 (section 2.2) as follows:

$$\ddot{\theta}(t) = -f(\dot{\theta}, \theta) + b \cdot u(t)$$

in which f and b are unknown parameters and are bounded by f_o , b_o as follows:

- $f(\dot{\theta}, \theta) \leq f_o$
- $0 < b < b_o$

4. Design controller for the balancing beam system

The objective of this controller is to force the plant state vector $\theta(t)$ to follow a specified desired trajectory $r(t)$. To do that, the error between actual and reference trajectory is defined as follows:

$$\varepsilon(t) = \theta(t) - r(t) \quad (4.1.1)$$

The system output will follow the input $r(t)$ when the system error asymptotically converges to zero as time tends to infinity.

Using the SMC approach, a time varying sliding surface is defined as

$$s = \dot{\varepsilon} + \lambda\varepsilon \quad (4.1.2)$$

$$\therefore \dot{s} = \ddot{\varepsilon} + \lambda\dot{\varepsilon} = (\ddot{\theta} - \ddot{r}) + \lambda\dot{\varepsilon}$$

$$\therefore \dot{s} = (-f(\dot{\theta}, \theta) + b.u - \ddot{r}) + \lambda\dot{\varepsilon} \quad (4.1.3)$$

Consider the Liapunov function candidate

$$V = \frac{1}{2}s^2 + \frac{b}{2}(\theta_1 - \hat{\theta}_1)^2 + \frac{b}{2}(\theta_2 - \hat{\theta}_2)^2 > 0 \quad (4.1.4)$$

where

- $\theta_1 = \frac{f_0}{b_0}$
- $\theta_2 = \frac{1}{b_0}$
- $\hat{\theta}_1, \hat{\theta}_2$ are the estimates of θ_1, θ_2 respectively.

Differentiating V with respect to t gives

$$\dot{V} = s\dot{s} + b(\theta_1 - \hat{\theta}_1)\dot{\hat{\theta}}_1 + b(\theta_2 - \hat{\theta}_2)\dot{\hat{\theta}}_2$$

Substituting it in Equation 4.1.3 to obtain

$$\begin{aligned} \dot{V} &= s(-f(\dot{\theta}, \theta) + b.u - \ddot{r} + \lambda\dot{\varepsilon}) - b\theta_1\dot{\hat{\theta}}_1 + b\hat{\theta}_1\dot{\hat{\theta}}_1 - b\theta_2\dot{\hat{\theta}}_2 + b\hat{\theta}_1\dot{\hat{\theta}}_2 \\ &= -s.f(\dot{\theta}, \theta) + s.b.u - s.\ddot{r} + s.\lambda.\dot{\varepsilon} - b.\theta_1.\dot{\hat{\theta}}_1 + b.\hat{\theta}_1.\dot{\hat{\theta}}_1 - b.\theta_2.\dot{\hat{\theta}}_2 + b.\hat{\theta}_1.\dot{\hat{\theta}}_2 \end{aligned} \quad (4.1.5)$$

Choosing the control signal:

$$u = -\hat{\theta}_1.\text{sgn}(s) - \hat{\theta}_2.\text{sgn}(s)|\ddot{r}| - \hat{\theta}_2.\text{sgn}(s).\lambda|\dot{\varepsilon}| \quad (4.1.6)$$

where

$$\text{sgn}(s) = \begin{cases} +1 & \text{for } s > 0 \\ -1 & \text{for } s < 0 \end{cases}$$

Substituting it in Equation 4.1.5 results in the derivative of Liapunov function:

$$\begin{aligned} \dot{V} &= -s.f(\dot{\theta}, \theta) + s.b(-\hat{\theta}_1.\text{sgn}(s) - \hat{\theta}_2.\text{sgn}(s)|\ddot{r}| - \hat{\theta}_2.\text{sgn}(s).\lambda|\dot{\varepsilon}|) \\ &\quad - s.\ddot{r} + s.\lambda.\dot{\varepsilon} - b.\theta_1.\dot{\hat{\theta}}_1 + b.\hat{\theta}_1.\dot{\hat{\theta}}_1 - b.\theta_2.\dot{\hat{\theta}}_2 + b.\hat{\theta}_1.\dot{\hat{\theta}}_2 \end{aligned}$$

$$\begin{aligned} \dot{V} &= -s.f(\dot{\theta}, \theta) - s.\ddot{r} + s.\lambda.\dot{\varepsilon} - |s|.b.\hat{\theta}_1 - |s|.b.\hat{\theta}_2|\ddot{r}| - |s|.b.\hat{\theta}_2.\lambda|\dot{\varepsilon}| \\ &\quad - b.\frac{f_o}{b_o}\dot{\hat{\theta}}_1 + b.\hat{\theta}_1.\dot{\hat{\theta}}_1 - \frac{b}{b_o}\dot{\hat{\theta}}_2 + b.\hat{\theta}_2.\dot{\hat{\theta}}_2 \\ &\leq |s|. \left| f(\dot{\theta}, \theta) \right| + |s|.|\ddot{r}| + |s|.|\lambda.\dot{\varepsilon}| - |s|.b.\hat{\theta}_1 - |s|.b.\hat{\theta}_2|\ddot{r}| - |s|.b.\hat{\theta}_2.\lambda|\dot{\varepsilon}| \\ &\quad - b.\frac{f_o}{b_o}\dot{\hat{\theta}}_1 + b.\hat{\theta}_1.\dot{\hat{\theta}}_1 - \frac{b}{b_o}\dot{\hat{\theta}}_2 + b.\hat{\theta}_2.\dot{\hat{\theta}}_2 \end{aligned}$$

Rearranging the above equation gives:

$$\begin{aligned} \dot{V} \leq & \left(|s| \left| f(\dot{\theta}, \theta) \right| - b \frac{f_o}{b_o} \dot{\hat{\theta}}_1 \right) + \left(|s| |\ddot{r}| - |s| b \hat{\theta}_2 |\ddot{r}| \right) + \left(|s| \lambda |\dot{\varepsilon}| - \frac{b}{b_o} \dot{\hat{\theta}}_2 \right) \\ & + \left(b \hat{\theta}_1 \dot{\hat{\theta}}_1 - |s| b \hat{\theta}_1 \right) + \left(b \hat{\theta}_2 \dot{\hat{\theta}}_2 - |s| b \hat{\theta}_2 \lambda |\dot{\varepsilon}| \right) \end{aligned} \quad (4.1.7)$$

Define the adaptation laws as follows:

- $\dot{\hat{\theta}}_1 = |s|$
- $\dot{\hat{\theta}}_2 = |s| \lambda |\dot{\varepsilon}|$

With $\dot{\hat{\theta}}_1 = |s|$:

- $|s| \left| f(\dot{\theta}, \theta) \right| - \frac{b}{b_o} f_o \dot{\hat{\theta}}_1 = |s| \left| f(\dot{\theta}, \theta) \right| - \frac{b}{b_o} f_o |s| < 0$ since $\frac{b}{b_o} > 1$
- $-b |s| \hat{\theta}_1 + b \hat{\theta}_1 \dot{\hat{\theta}}_1 = -b |s| \hat{\theta}_1 + b |s| \hat{\theta}_1 = 0$

With $\dot{\hat{\theta}}_2 = |s| \lambda |\dot{\varepsilon}|$:

- $-b |s| \lambda |\dot{\varepsilon}| \hat{\theta}_2 + b \hat{\theta}_2 \dot{\hat{\theta}}_2 = -b |s| \lambda |\dot{\varepsilon}| \hat{\theta}_2 + b |s| \lambda |\dot{\varepsilon}| \hat{\theta}_2 = 0$
- $|s| \lambda |\dot{\varepsilon}| - \frac{b}{b_o} \dot{\hat{\theta}}_2 = \left(1 - \frac{b}{b_o} \right) |s| \lambda |\dot{\varepsilon}| < 0$
- $|s| |\ddot{r}| - b |s| |\ddot{r}| \hat{\theta}_2 \leq |s| |\ddot{r}| \times \left(1 - \frac{b}{b_o} \right) < 0$ since $-\hat{\theta}_2 \leq -\theta_2 = \frac{-1}{b_o}$

Therefore, the inequality Equation 4.1.7 becomes:

$$\dot{V} < 0 \quad (4.1.8)$$

4. Design controller for the balancing beam system

The chosen control signal and adaptive rules lead to inequality equations 4.1.4 and 4.1.8, which mean that the system is stable in the Liapunov sense. Therefore, the sliding variable s asymptotically converges to zero. At sliding mode, the system dynamics will be fully determined by the switching manifold dynamics:

$$\begin{aligned} s &= \dot{\varepsilon} + \lambda\varepsilon = 0 \\ \Rightarrow \dot{\varepsilon} &= -\lambda\varepsilon \\ \Rightarrow \varepsilon &= e^{-\lambda t} \end{aligned}$$

With a given positive λ , the error ε will asymptotically converge to zero; the system is stable and the system state vector $\theta(t)$ will follow the specified desired trajectory $r(t)$.

Note that the discontinuity of the control signal is caused by the presence of discontinuous function $\text{sgn}(s)$ in the control law. This is the reason for the switching phenomenon, which in turn leads to the chattering effect. The design of a fuzzy logic controller in the next section will try to eliminate this undesired outcome.

Outside the boundary, the system stability is mathematically guaranteed by the ASMC. Inside the boundary, the fuzzy logic controller will be combined with the ASMC to eliminate the chattering effects. Whenever the system goes outside that boundary, the ASMC is applied again to drive it back and so the system is constrained to be near the sliding mode. Moreover, the boundary region is very small and therefore, the system can be considered as stable in all states (outside and inside the boundary)¹.

¹ In fact, the stability of the fuzzy controller is really difficult to mathematically prove due to its vague, linguistic nature. In this case, the system stability is mainly assured by ASMC with Liapunov criteria.

4.3 Fuzzy controller to reduce chattering effects

4.3.1 The boundary layer control approach in brief

The problem with sliding mode control is that there is high frequency switching around the sliding surface $s = 0$ to assure the system dynamics stay within the sliding mode. In 1983, Stoline and Sastry proposed a ‘boundary layer’ approach to reduce the chattering effects by replacing the discontinuous $\text{sgn}(s)$ function in the control signal u by a continuous factor s/δ within the boundary layer. It can be applied in this case as follows:

$$u = \begin{cases} \text{sgn}(s) \times (-\hat{\theta}_1 - \hat{\theta}_2 |\ddot{r}| - \hat{\theta}_2 \lambda |\dot{\epsilon}|) & \text{for } |s| \geq \delta \\ \frac{s}{\delta} \times (-\hat{\theta}_1 - \hat{\theta}_2 |\ddot{r}| - \hat{\theta}_2 \lambda |\dot{\epsilon}|) & \text{for } |s| \leq \delta \end{cases}$$

However, the layer thickness δ is a fixed number therefore the reduction of chattering is highly dependent on its value. The larger the value of layer thickness δ , the more the system dynamic behaves as if no sliding mode control is taken.

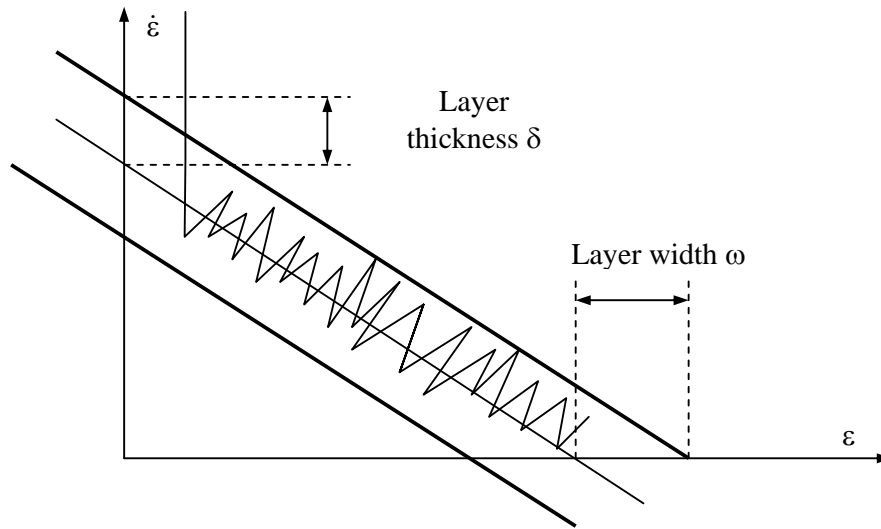


Figure 3: The switching (chattering) effects and boundary layer

4.3.2 Fuzzy logic approach and the FLC design

It can be seen that the chattering effect is determined by factor K in the control signal:

$$u = K \times (-\hat{\theta}_1 - \hat{\theta}_2 |\ddot{x}| - \hat{\theta}_2 \lambda |\dot{x}|)$$

From the fuzzy logic viewpoint, the sliding mode control with and without boundary layer can be seen as follows:

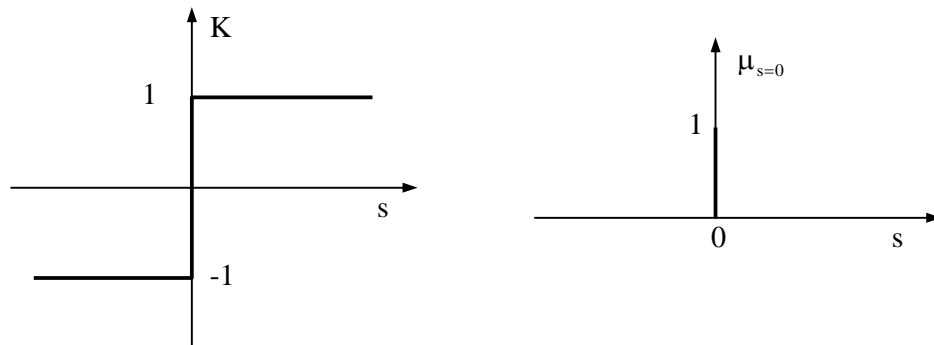


Figure 4.3.1: Fuzzy level of pure sliding mode (in switching surface)

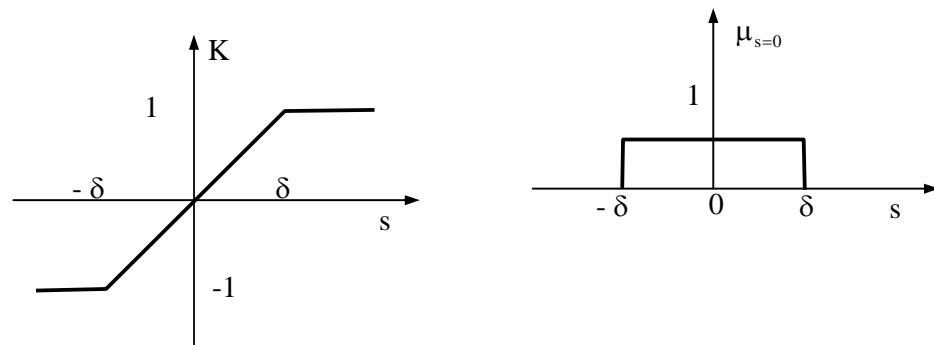


Figure 4.3.2: Fuzzy level of sliding mode with boundary layer (in switching surface)

It can be seen that the variation of K (discontinuous or continuous, how smooth it is) affects the efficiency of chattering control. In the other words, the higher the fuzziness level of K, the better the chattering reduction.

4. Design controller for the balancing beam system

This idea is extended to two dimensions, in which the value of K is dependent not only on the value of sliding mode variable s but also on the change of s (the derivative \dot{s}) with the following strategy:

IF further from the sliding mode surface

THEN larger factor K (control gain)

(i.e. IF s is small THEN u is small; IF s is large THEN u is large)

The diagram of the additional fuzzy controller used in conjunction with the adaptive sliding mode controller is shown in Figure 4.2.3:

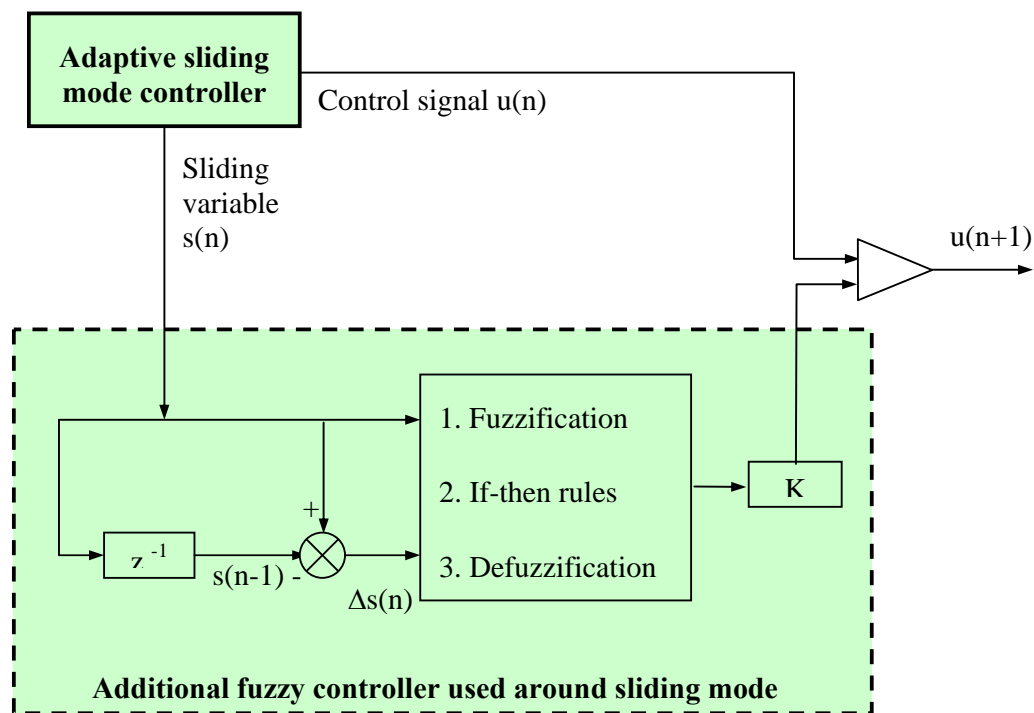


Figure 4.3.3: The block diagram of the fuzzy controller in conjunction with ASMC

The design of the additional fuzzy controller is as follows:

1. Define two linguistic input variables **S**, **DeltaS** and one linguistic output variable **FuzzyK** corresponding to two crisp inputs s , \dot{s} and output K .
2. Define their membership functions in **general ranges (independent of actual data)** and the rule base for fuzzy controller.
3. Observe the values of sliding variable s and its derivative \dot{s} (from ASMC) and scale their values to fit with the pre-defined range of fuzzy variables:

$$S = s \times \text{ScaleS}$$

$$\text{DeltaS} = \dot{s} \times \text{ScaleDeltaS}$$

4. The output FuzzyK can also be scaled and the control signal becomes:

$$u = \begin{cases} \text{sgn}(s) \times \left(-\hat{\theta}_1 - \hat{\theta}_2 |\ddot{r}| - \hat{\theta}_2 \lambda |\dot{e}| \right) & \text{for } |s| \geq \delta \\ \text{FuzzyK} \times \text{ScaleK} \times \left(-\hat{\theta}_1 - \hat{\theta}_2 |\ddot{r}| - \hat{\theta}_2 \lambda |\dot{e}| \right) & \text{for } |s| \leq \delta \end{cases}$$

5. Fine-tune the membership functions, rule base and scaling factors from practical trials.

In this case, the output FuzzyK set contains 7 linguistic variables: **Negative Big (NB)**, **Negative Medium (NM)**, **Negative Small (NS)**, **Zero (Z)**, **Positive Big (PB)**, **Positive Medium (PM)** and **Positive Small (PS)** with the membership functions as below:

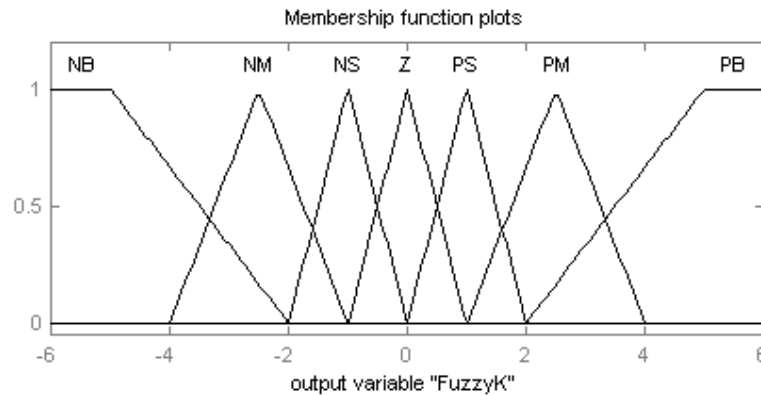


Figure 4.3.4: The membership functions of output variable FuzzyK

4. Design controller for the balancing beam system

Since **K** is aimed to replace the $\text{sgn}(s)$ function and acts as a control gain, the general **range of FuzzyK is chosen as [-6, 6]** and the **ScaleK is equal to 1/6**. This is similar to the option that **FuzzyK range is [-1, 1]** and the **ScaleK is equal to 1**. The reason why FuzzyK is not represented in [-1,1] range is to make the fine-tuning of membership functions easier (range [-1,1] is too small).

In the current project, the parameters for output FuzzyK are fixed as above. Therefore, depending on the number of membership functions of two input variables (and so the corresponding rule base), there are different options for the fuzzy controller. Following are two investigated configurations:

- Both input sets have 3 linguistics variables: **Negative (N)**, **Zero (Z)** and **Positive (P)** in the range of [-6,6]. This configuration is called Fuzzy controller 3-3-7.
- Both input sets have 5 linguistics variables: **Negative Big (NB)**, **Negative Small (NS)**, **Zero (Z)**, **Positive Big (PB)** and **Positive Small (PS)** in the range of [-6,6]. This configuration is called Fuzzy controller 5-5-7.

As mentioned before, the distribution and the shape of membership functions have important effects on system performance. In the current design, various combinations and enormous fine-tuning had been carried out and it would be unfeasible to present again those hundreds of options. Therefore, represented here are the best membership functions configurations that have been observed.

4. Design controller for the balancing beam system

The scaling factors of input variables also affect the FLC performance but there is no systematic way for tuning it and the relationship is not very clear [11]. Therefore, the trial-and-error method is applied again until a satisfactory result is obtained. Similarly, the best scaling values are presented here.

4.3.2.1 Fuzzy controller 3-3-7

The membership functions of the input S are shown below:

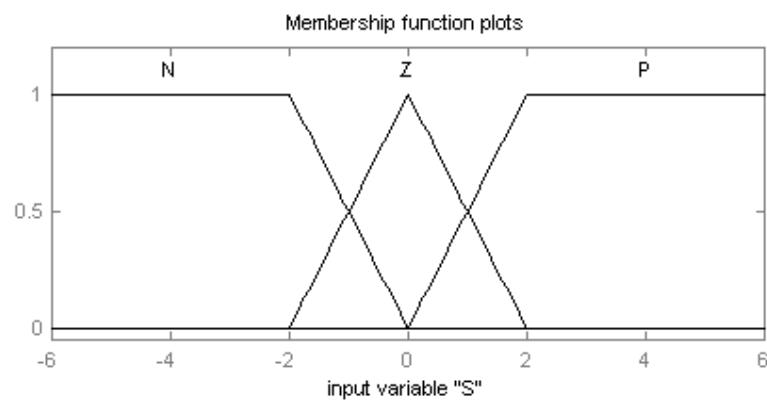


Figure 4.3.5: The membership functions of input S (3-3-7 configuration)

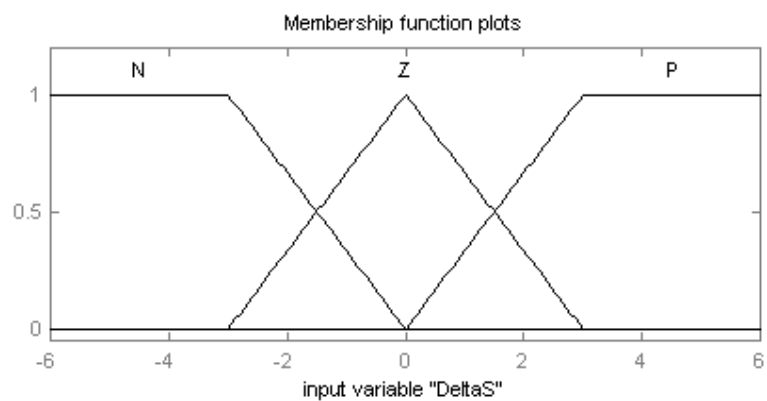


Figure 4.3.6: The membership functions of input DeltaS (3-3-7 configuration)

The rule base can be presented either in the if-then format or in the rule matrix. While the first method gives more natural linguistic statements, the latter format is more concise and will be used in this project.

4. Design controller for the balancing beam system

The conversion from a set of if-then rules to a rule matrix is very simple. For example, the highlighted cell in the below matrix corresponds to the rule:

IF (S is Z) AND (DeltaS is Z) THEN (FuzzyK is Z)

Following this idea, the matrix format of the linguistic rule base is obtained below.

		DeltaS		
		N	Z	P
S	N	NB	NM	NS
	Z	NS	Z	PS
	P	PS	PM	PB

The control surface, which describes the relationship between input and output variables after the rule evaluation (with above rule base) and the defuzzification process (COA method), is shown below:

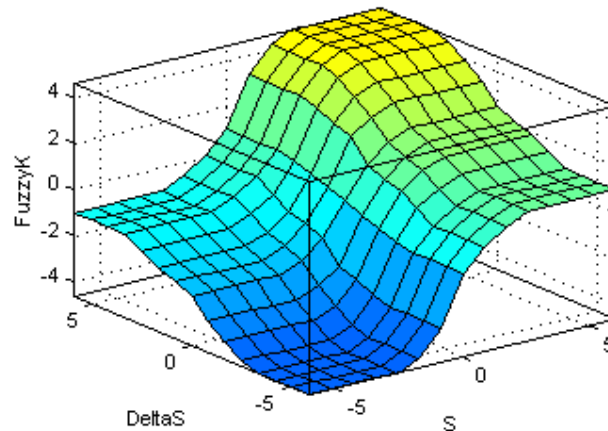


Figure 4.3.7: Fuzzy surface of the proposed rule base (3-3-7 configuration)

The best parameters of this configuration are:

- ScaleS = 2.5 ScaleDeltaS = 1/64
- Boundary thickness $\delta = 0.2$

4.3.2.2 Fuzzy controller 5-5-7

For this option, the membership functions of the input S are as below:

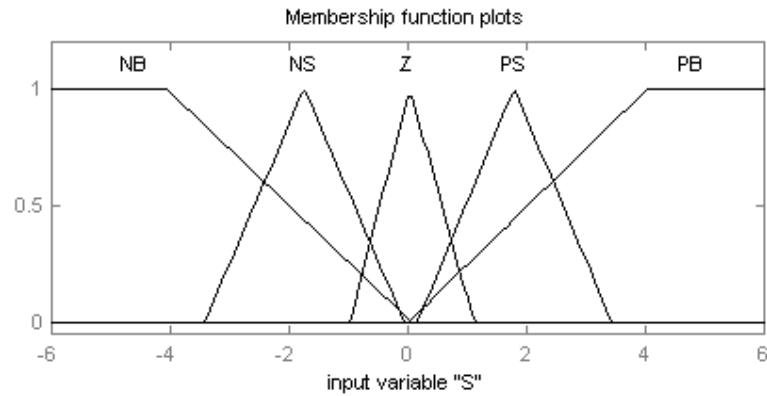


Figure 4.3.8: The membership functions of input S (5-5-7 configuration)

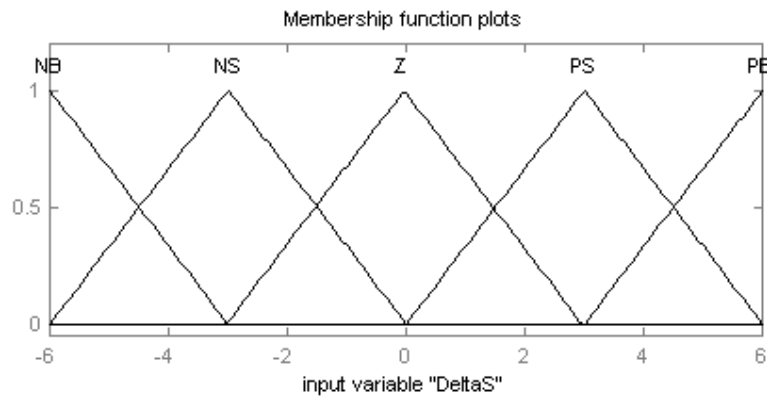


Figure 4.3.9: The membership functions of input DeltaS (5-5-7 configuration)

Similar to FLC 3-3-7, the rule base is presented in the matrix format in which some dominant rules are highlighted. These rules contain linguistic variables Zero of S and DeltaS, i.e. nearest to the sliding mode surface. They are most frequently fired during the control process and therefore, are essential for the success of system performance. More discussion about the number of rules in the knowledge base will be presented in section 4.5.

4.4 Summary

In this section, the Fuzzy Adaptive Sliding Mode tracking Controller (FASMC) is successfully designed for the balancing beam system. The combination of an Adaptive Sliding Mode tracking Controller (ASMC) with a Fuzzy Logic Controller (FLC) gives the best of both worlds.

The application of adaptive rules allows the controller to estimate the unknown system parameters, therefore enabling it to perform optimally under a wide range of operating conditions with high uncertainties and disturbance.

The sliding mode control technique can drive the system from any initial state into the sliding mode, where the system response is fully determined by the pre-designed characteristics. That is, it provides a fast tracking performance.

Meanwhile, the inevitable chattering effects associated with the switching operation of the ASMC were effectively reduced by the utilization of the FLC. It can be seen that in fuzzy logic control, the relation of K and sliding mode variable s , its derivative \dot{s} is much better than other cases without a fuzzy controller. In fact, the big boundary layer in the boundary layer approach is divided into many smaller sub-layers. Therefore, with the smoothing change of K , the change of control signal is smoothed and chattering is effectively reduced.

4. Design controller for the balancing beam system

One noticeable advantage of utilizing fuzzy logic is that it does not care about how system dynamics will be driven into sliding mode (by sliding control, adaptive sliding control or finite sliding control, etc.). The fuzzy controller just bases decisions on how far the output is away from the desired reference (related to s) and how fast it reaches that state (related to \dot{s}) to adjust the control gain.

Therefore, the overall controller provides very good system performance with quick tracking, uncertainties immunity and a very stable response.

In summary, the FASMC is designed with two steps: firstly an ASMC and then a FLC. The control scheme of the ASMC for this project is implemented by the following steps:

1. **Set the error rate:** λ
2. **Measure inputs from system:**
 - Reference signal (from joystick or software): $r(t)$
 - Output position: $\theta(t)$
3. **Calculate necessary parameters:**
 - Calculate the tracking error: $\varepsilon(t) = \theta(t) - r(t)$
 - Differentiate to obtain the 1st derivative of error: $\dot{\varepsilon}(t)$
 - Calculate the sliding variable: $s = \dot{\varepsilon} + \lambda\varepsilon$
 - Calculate the adaptive parameter: $\hat{\theta}_1 = |s|$
 - Calculate the adaptive parameter: $\hat{\theta}_2 = |s|\lambda|\dot{\varepsilon}|$
 - Integrate to determine boundary estimation: $\hat{\theta}_1$

- Integrate to determine boundary estimation: $\hat{\theta}_2$
- Differentiate to obtain the 1st derivative of reference: $\dot{r}(t)$
- Differentiate to obtain the 2nd derivative of reference: $\ddot{r}(t)$

4. Calculate the output control signal:

$$u = -\hat{\theta}_1 \cdot \text{sgn}(s) - \hat{\theta}_2 \cdot \text{sgn}(s) |\dot{r}| - \hat{\theta}_2 \cdot \text{sgn}(s) \lambda |\dot{e}|$$

Steps 2 to 4 will be performed in every sampling period to update the control law and drive system into the desired state.

The differentiation and integration can be implemented either by hardware or by software. Hardware implementation will provide faster results, however, software calculation will be applied in this project. Due to the complexity and amount of computational demand, the computer processor as well as sensor devices must be sufficiently fast. Otherwise, the controllability and stability of the system will not be successfully achieved.

The data obtained from ASMC will then be used to determine the input range for FLC. In fact, the design of FLC is a practical process with steps shown in section 4.3.2; in which numerous trials, careful observation and understanding of the system behaviour are the main factors for success. Therefore, the optimum membership functions, rule base and other controller parameters are only obtained after much practice.

4.5 The effects of controller parameters

Vicious friction of the system: B_0

- Increasing the value of B_0 means more force (so motor torque) required to move the beam. That is, the control signal magnitude has to be larger.

Dimension: the amount of historical data used in differentiation

- Increasing the number of points using in Least Square Method calculation will make the results more accurate and may reduce the control signal magnitude.
- However, the longer calculating time means the longer response time, i.e. the controller is less sensitive to the change of input signal. In conjunction with that is a lower error convergence rate.

Adaptive gain:

- Adaptation rates determine how rapidly the controller parameters are updated. Increasing the adaptive gain will increase the adaptive rate, that is, the controller will estimate (learn) the optimal parameters faster.
- However, the use of excessively high adaptive gains (which controls the adaptation rates) may degrade the overall robustness of the control system.

Error gain of sliding variable: λ (lamda)

- To ensure the error will converge to zero, λ should be a positive number.

- Increasing λ means increasing error convergence rate. That is, in the sliding mode, the error will converge to zero faster and tracking time is shorter.
- However, the control signal magnitude also is increased along with the increment of λ . It then causes overshoot and slight oscillation occurs. Excessive error gain leads to prominent chatter and prolonged settling time.

Sampling time:

- To identify the plant parameters, a control signal is imposed on the plant and the system response is analyzed. The identification of plant parameters must be measured continuously or at the least very frequently. Such a requirement should be accomplished without affecting the normal operation of the control system. For effective adaptation, identification must not take too long since if it does, further variations of the plant parameters may occur. This translates to relatively high sampling frequency to the rate of environmental changes.
- For online estimation, the greater the sampling period (smaller sampling frequency), the worse parameters estimation is (and the longer the time to obtain a reasonable estimation). This is because when the system is sampled less frequently, the observed data is not enough to re-construct the real system behaviour. That eventually leads to inexact estimation results.
- However, the smaller the sampling time, the heavier the manipulation required. If the sampling period were smaller than the time required for estimation calculating, the system would be uncontrollable. Normally, the sampling frequency must be greater than 2 times the largest signal frequency of the real system.

Width of the boundary layer in which the FLC is applied: δ

- Increasing the boundary width will reduce the chattering effect within the boundary (and create a smaller control signal and overshoot). However, it will increase the steady-state error, i.e. reduce the tracking performance of the controller, particularly in systems where dead-zones exist.
- Increasing the boundary width also degrades the ability of the controller to reject disturbances. The sensitivity of the system to the changing rate of input signal will be reduced also.

The sharpness of membership functions:

- The sharper the membership functions are, the smaller the output the controller produces. Therefore, the membership functions are often set to be wider outside and narrower near the centre of the range. By this way, large error or chattering will be reduced faster by larger control signal. Inversely, when the system is in stable state with small error or chattering, small or even no control signal will be produced to keep the system stable.

The number of linguistic variables (fuzzy sub-sets) of fuzzy input variables:

- The larger the number of linguistic variables, the more stable and smoother the system response. However, it also means the system sensitivity is reduced.
- The smaller the number of linguistic variables, the faster the system response will be (shorter tracking time). Nevertheless, the trade-off is that large overshoot may occur.

The number of rules in the fuzzy rule base:

- Without the dominant rules (which contain linguistic variables Zero of S and DeltaS), the fuzzy controller does almost nothing and so cannot produce the desired reason.
- Increasing the number of rules will make the system respond faster (since less fuzzy approximation is required). However, too many rules means heavier calculation and therefore, the system response may be even longer.

5 Testing by Matlab simulation

5.1 Introduction

Before the actual implementation in the physical system, the designed controller is tested in a simulation environment. This step enables re-checking of the performance of the controller via system behaviour (e.g. stability, tracking error and chattering reduction, etc.). The obtained information can then be used to fine-tune the design, especially for the fuzzy logic controller.

Due to its power, flexibility and high-performance for technical computing, MATLAB is used for simulation purposes. In this integrated environment (of computation, visualization, and programming), the current design can be easily expressed in mathematical notation. Therefore, the simulation program is in fact the core algorithm for later software development in C++ language.

The simulation of fuzzy controller is implemented with the substantial help of the Matlab Fuzzy Logic Toolbox. By providing a systematic framework for computing with fuzzy rules and graphical user interfaces, the building and fine-tuning of the FLC becomes very easy and convenient. In fact, the way to define the membership function shape in the real system of this project is learnt from the Matlab Fuzzy Toolbox.

5.1.1 Simulation settings

The simulation is carried out with various strategic and parameters options. Among them are the following significant options:

- Simulation without noise, disturbance, time delay, un-modelled dynamics (ideal and unrealistic conditions).
- Simulation with no chattering control (pure ASMC), Boundary layer control for chattering effects (last year's design) and with Fuzzy logic controller for chattering effects (this year design).
- Simulation with noise, delay and disturbance (practical conditions).

The simulation model for of the project's system with disturbance, noise and control delay is shown in Figure 5.1.1:

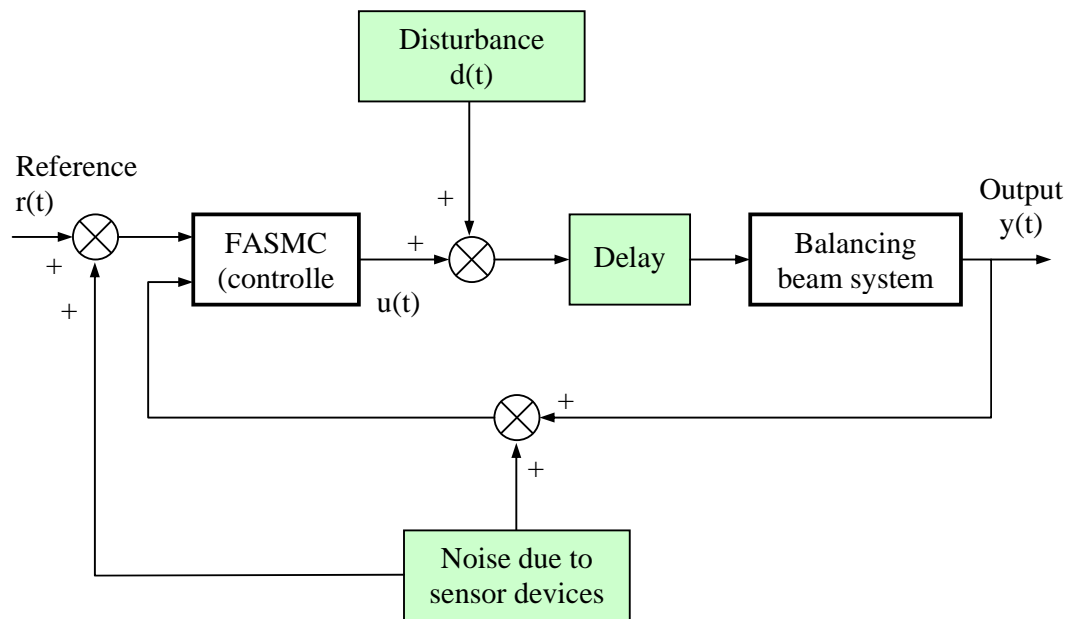


Figure 5.1.1: Simulation model for noisy conditions

All of the following facts were taken into setting the simulation:

- The parametric uncertainty and imperfection of plant (in sensor devices, conversion instruments, connections, etc.) are simulated by a random noise (Gaussian distribution) with zero mean and peak values of 0.2 volts.
- The limitation of control driving motors: the dead-zone, where the beam remains stationary with control signal from -3.2 volts to 2.9 volts, and the lower gain of propeller (0.85, see section 2.4.2) in the negative (reverse) direction are also simulated.
- Disturbance due to external environment is up to 2 volts.
- A time delay of 0.03s or 30ms is used to simulate the delay occurring when the control signal passes through PC card and amplifier.

However, the slew rates and the limitation in amplifier bandwidth were not considered in the simulation.

Due to the large number of combinations and enormous fine-tuning which had been carried out, not all those results can be presented and only the best configurations will be shown here. As mentioned in section 4.3.2, the optimum design parameters for the FLC 3-3-7 are:

- ScaleS = 2.5 ScaleDeltaS = 1/64
- Boundary thickness $\delta = 0.2$

For the FLC 5-5-7, they are:

- ScaleS = 3 ScaleDeltaS = 1/64
- Boundary thickness $\delta = 0.2$

5.1.2 Simulation reference signals

The chosen reference signals are triangular and sine wave for the following reasons:

- The triangular reference signal enables observation of the system response when a fixed velocity movement in reference signal is applied. The sharp and swift changing in direction of the triangular reference signal also provides a study of how the controller will behave in such circumstances.
- The sine wave reference signal allows the investigation of the system behaviour when the velocity of reference signal is gradually changed. It also provides a chance to observe the tracking performance when the system follows a smooth reference trajectory.

5.1.3 Simulation outputs

The simulation program is shown in Appendix A.1. The output from this simulation consists of four key figures, in which:

- The first figure shows the reference and actual position of the system. The tracking performance will be roughly observed from this result.
- The second figure plots the accurate tracking error between reference and actual position.
- The third figure displays the control signal and from that, the switching operation (chattering) can be investigated.
- The last figure shows the sliding variable and its first derivative, which is really important to determine performance of the fuzzy controller in chattering reduction.

5.2 Simulation results under ideal conditions

5.2.1 Triangular reference signal

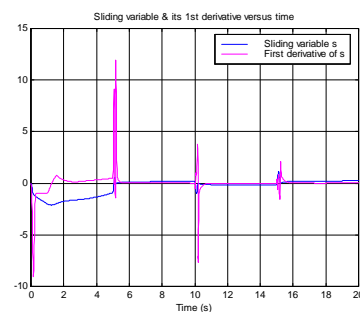
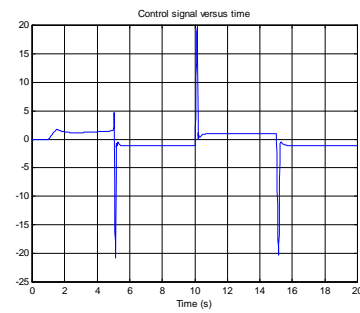
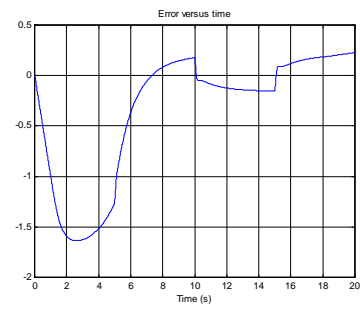
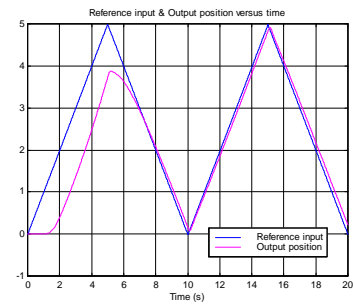
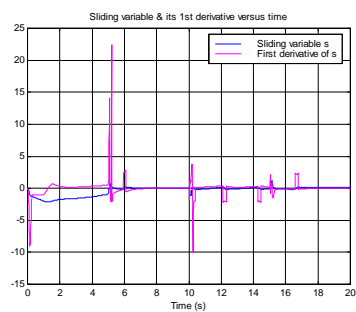
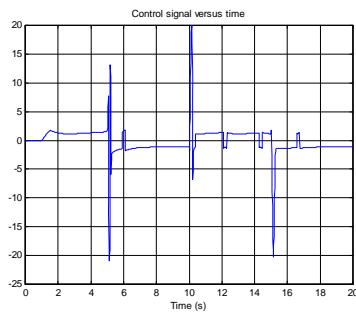
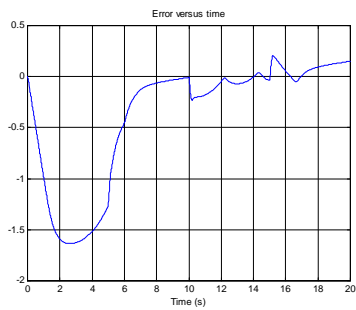
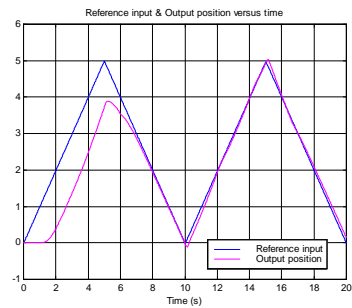


Figure 5.2.1: Without chattering control

Figure 5.2.2: With FLC 3-3-7

5.2.2 Sine-wave reference signal

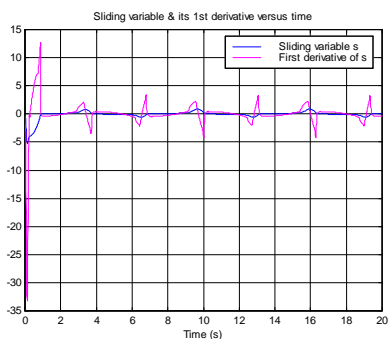
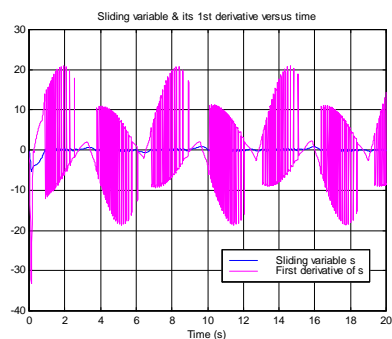
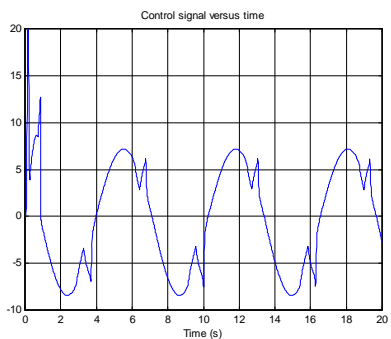
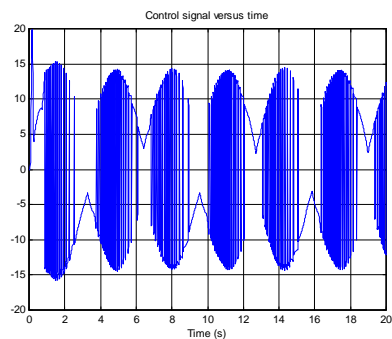
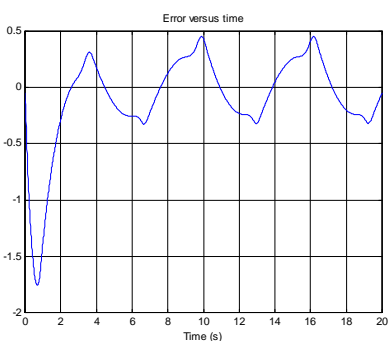
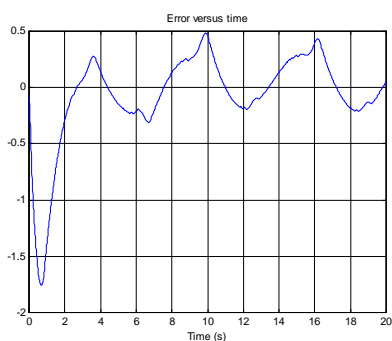
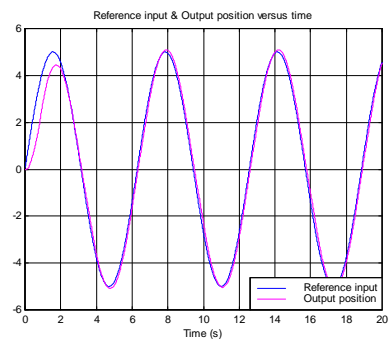
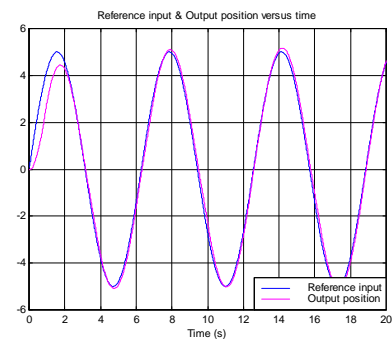


Figure 5.2.3: Without chattering control

Figure 5.2.4: With FLC 3-3-7

5.3 Simulation results with noise, disturbances and delay

5.3.1 Triangular reference signal

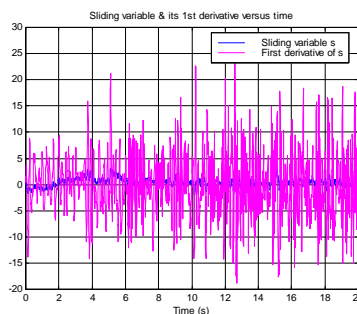
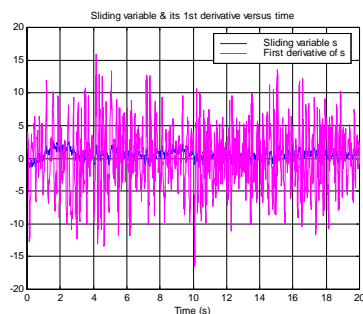
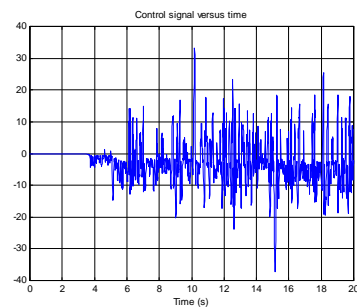
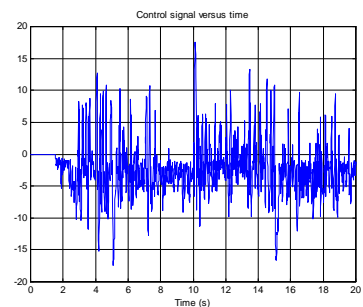
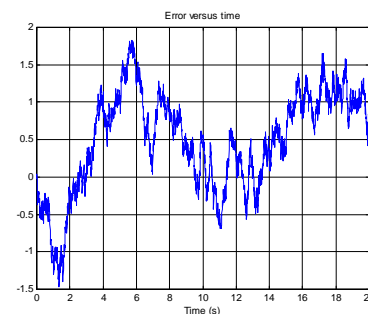
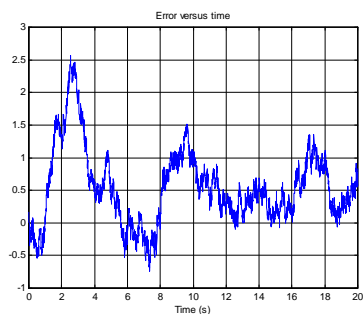
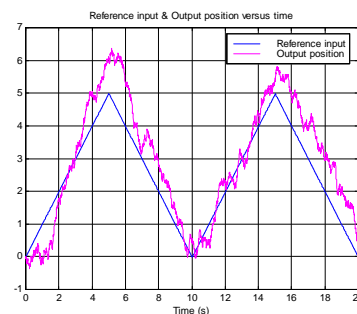
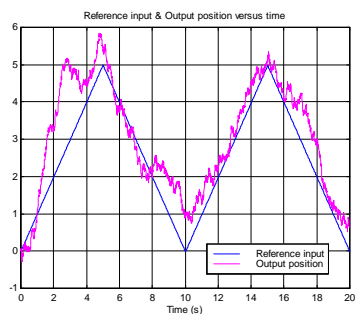


Figure 5.3.1: With boundary layer

Figure 5.3.2: With FLC 3-3-7

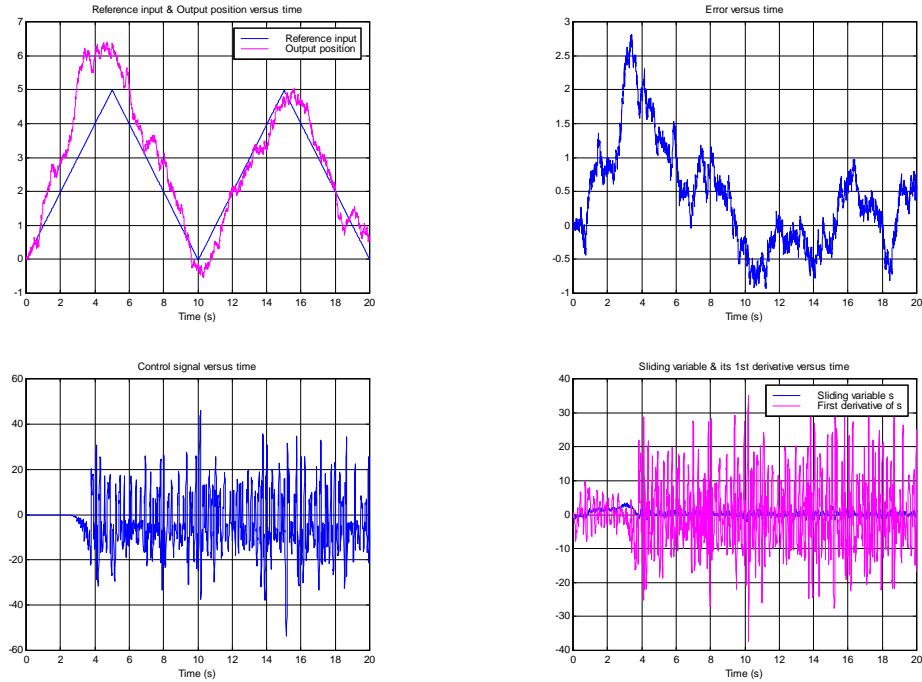


Figure 5.3.3: Without chattering control

5.3.2 Sine-wave reference signal

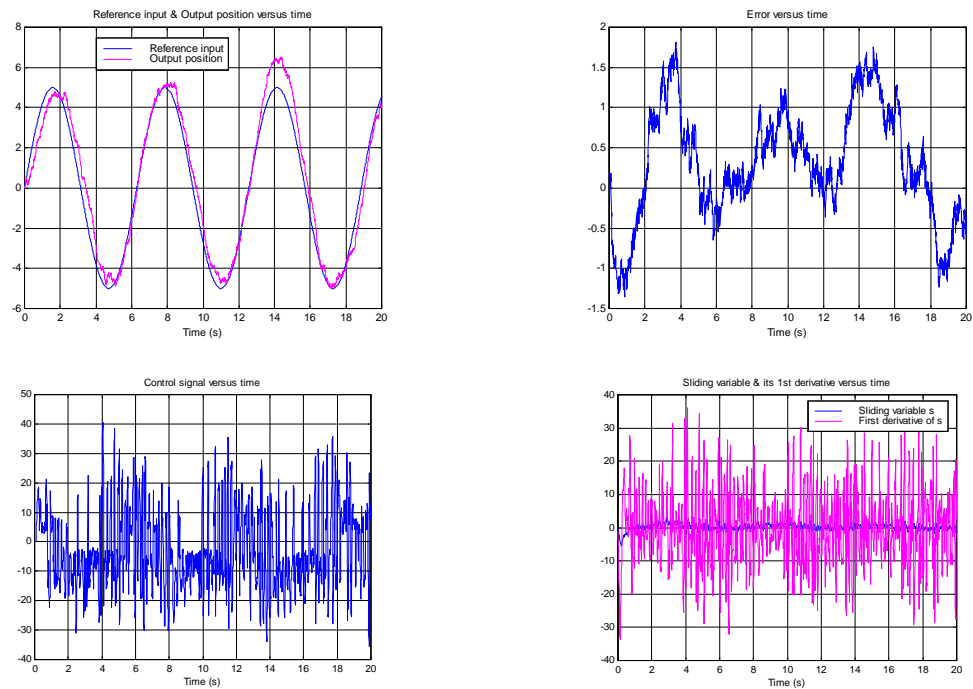


Figure 5.3.4: Without chattering control

5. Testing by Matlab simulation

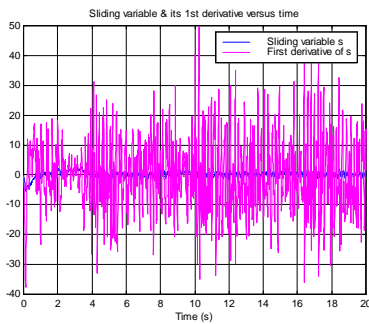
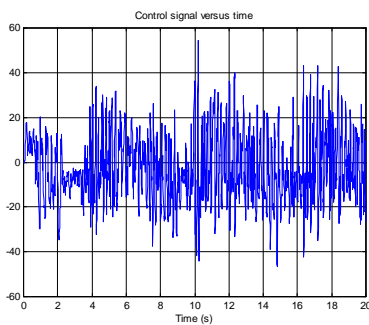
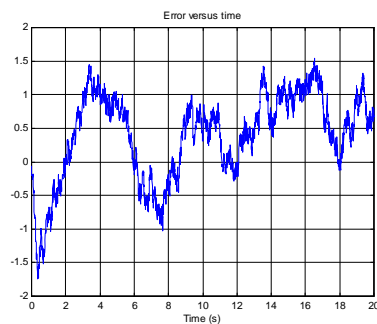
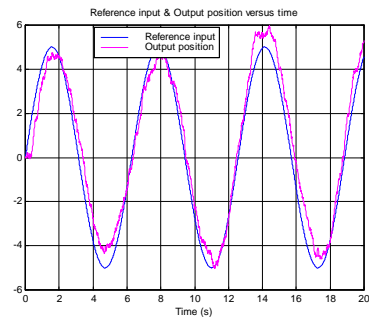


Figure 5.3.5: With boundary layer

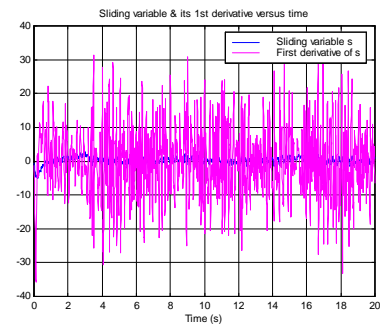
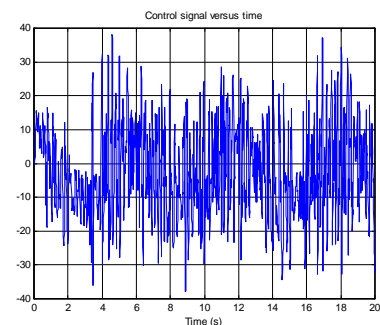
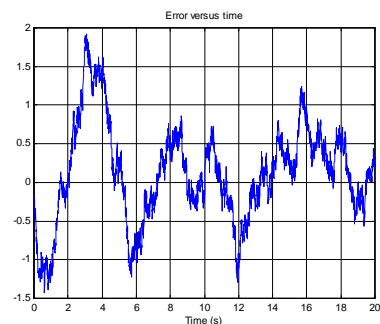
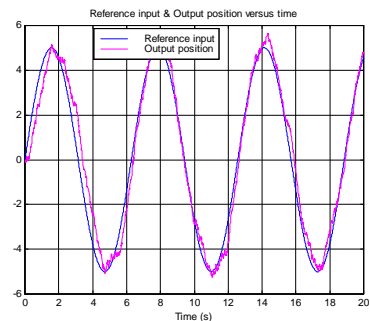


Figure 5.3.6: With FLC 3-3-7

5.4 Analysis of simulation results

From the simulation results, the following observations were obtained:

- Under ideal conditions without noise, disturbance and delay, the ASMC with boundary layer control and the FASMC with FLC 3-3-7 have the same performance (so only FLC's results are shown), which is certainly superior to what was produced by pure ASMC without any chattering control. Particularly, the first two controllers perform the same tracking error, fast response but the chattering is significantly reduced (Figures 5.2.3 and 5.2.4) compared to the pure ASMC case.
- However, under noisy conditions with disturbance and delay, the FASMC has shown its advanced performance with smaller tracking error, small overshoot with triangular reference (see Figure 5.3.2). In the case of sine wave reference, its chattering reduction is obviously better than that of ASMC with boundary control with lower magnitude of control signal and sliding variable (Figure 5.3.6). The conclusion is that very strong robustness has been achieved by the FASMC designed in this project.
- In fact, tracking error and chattering reduction are somehow mutually exclusive. A good performance for this factor is only obtained in favour of the other one. That is, in order to obtain less oscillatory effects (chatter), one has to accept a certain level of steady state error (tracking performance) and vice versa. This is because the approximation of the control law with a continuous signal prevents the system reaching the ideal sliding mode.

- A swift change in the reference direction will create a large peak in the first derivative of error. It then leads to a corresponding jump in the sliding variable (which is defined in terms of error ε and $\dot{\varepsilon}$). This in turn causes large spikes in the control signal. Therefore, the higher the rate of change in reference signal will require a larger control signal and so generate more severe chatter.
- The FASMC has effectively learnt the adaptive parameters (see the sliding variable graph since the designed adaptive parameters are closely related to that variable). In most cases, the controller takes about 2 seconds to achieve the optimum estimation¹ (stable), and so optimal system performance.
- The transient performance can be improved by increasing the adaptive gain and the magnitude of the control signal can be reduced with a higher amount of historic data. However, some trade-offs should be considered (see 4.5).

The simulation has proved that the designed controller is capable of driving the system into stable state with very good tracking performance and excellent chattering reduction. Even under intense conditions when the system is subjected to noise, delay and disturbance (at the same time), the system still performs with very strong robustness and closely follows the reference signal. That is, the success of practical implementation is almost assured.

¹ Note that the initialized values of adaptive parameters should be large enough to generate sufficient control signal for moving the beam. If they are too small, the controller takes a longer time to increase them to optimal values. However, this is not a problem on an inherently stable system like the balancing beam in this project.

6 Practical implementation

6.1 Software development

Initially, there was an intention is to build up a control program running on Windows with **graphics interface, online plotting, system parameters customisation**, etc. The software was intended to develop on C++ Builder environment, which is very powerful programming language in dealing with hardware control, heavy computational program, data transaction via network, etc.

For that reason, a software component was firstly created to realise the additional fuzzy logic controller (called FLC component). Overall, this component was developed as an independent, re-usable, “plug-and-play” module so that it can easily combine with the existing ASMC software as well as other future software. Therefore, its inputs and outputs were designed so that sufficient flexibility could be obtained along with the interface simplicity.

The internal implementation of the FLC component is structured in the spirit of object orientation. Particularly, object-oriented design (OOD) is applied in this process. Based on the general structure of a fuzzy controller (see 3.4.4), it was determined which objects (“class” in terms of C++ language) were needed to implement the FLC and how those objects interact with each other to perform the required goals. From the reasoning process within FLC (see 3.4.5), it was determined what attributes the objects need to

6. Practical implementation

have and what behaviours these objects need to exhibit (member functions). In this way, the FLC component naturally reflects its real structure.

Specifically, the two following classes are constructed: Class **Fuzzy** aims to create a general fuzzy variable (label) by which the user can define its membership function shape and range (as in Matlab Fuzzy Toolbox). Class **FuzzyController** is then built up based on **Fuzzy** class, in which users can define the type of FLC, the range of a variable, the range of its derivative and of the output value to use some pre-defined controllers there (the designed FLC 3-3-7 and FLC 5-5-7). Users can even construct their own type of FLC based on the **Fuzzy** class with their own membership functions and fuzzy labels.

The simplified structure of these classes and how they are related to implement this project's FLC controller is shown in Figure 6.1.1. The code for these classes is shown in Appendix B.1.

After that, the designed FLC component is combined with the existing ASMC software from 1999 to form a FASMC. The modified code is shown in Appendix B.2.

As mentioned above, it was intended to come up with a Windows version of the FASMC. However, the testing with the FASMC combined from new FLC with old ASMC in DOS environment showed that the computer configuration (in the lab) was only sufficient for that FASMC. It was not powerful enough for graphic interface and other burdens that were to be created by the Windows FASMC. Therefore, although

6. Practical implementation

some work had been carried out to implement the Windows FASMC, this option had to be cancelled.

The final software was then fully developed in Turbo C++ environment.

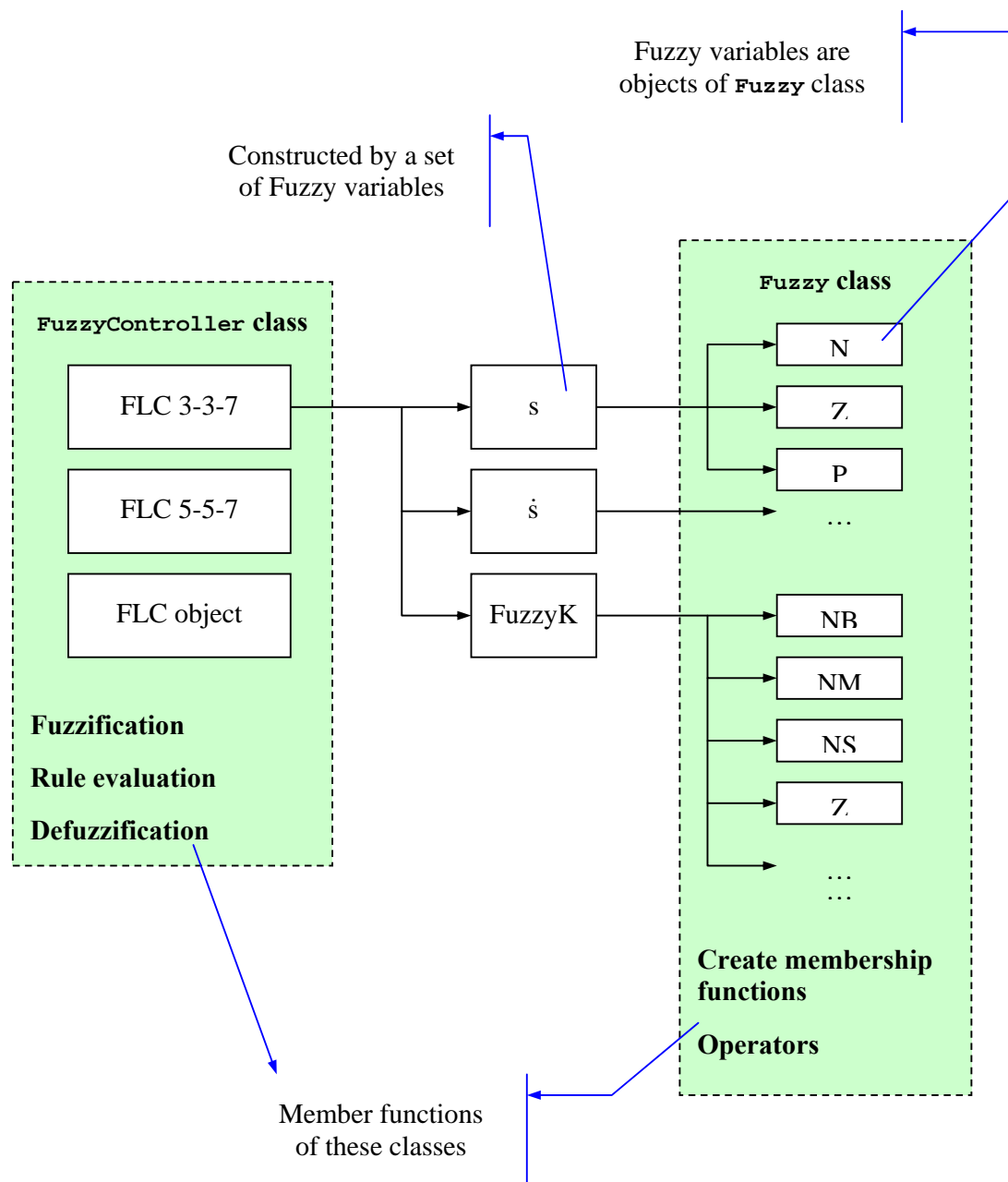


Figure 6.1.1: C++ classes to construct FLC

6.2 Practical results

6.2.1 Introduction

As observed from the simulation testing, the FASMC and ASMC with boundary layer control give the same performance in ideal conditions. However, in noisy and disturbed conditions, FASMC provides superior results. For all conditions, these two controllers always have better performance than pure ASMC without any chattering control. Therefore, in the practical system only these two were considered.

For each running section in data-logging mode, the experimental data is collected by software and stored in files. Those data then are plotted by Matlab (see Appendix A.2) to provide four key figures as in the simulation.

6.2.2 Square wave reference signal

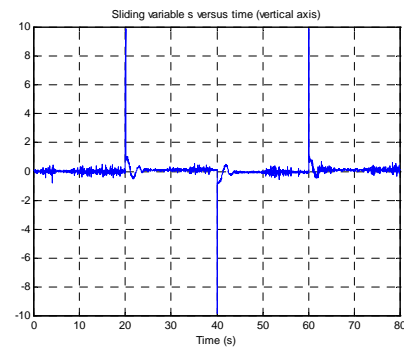
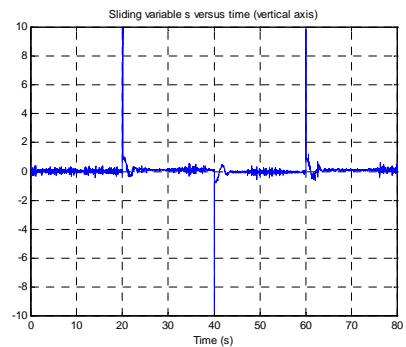
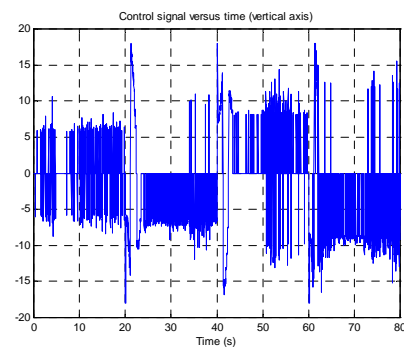
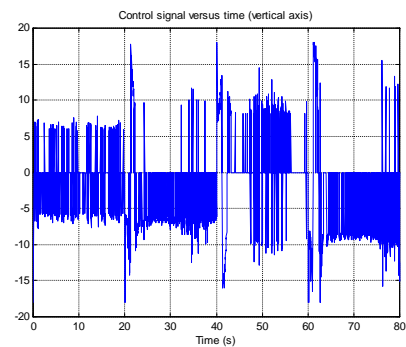
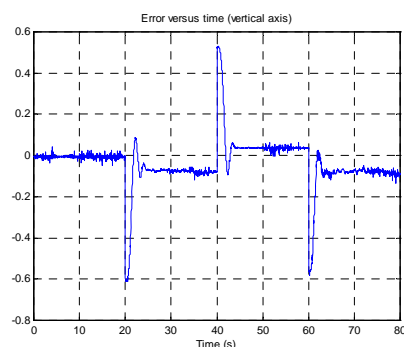
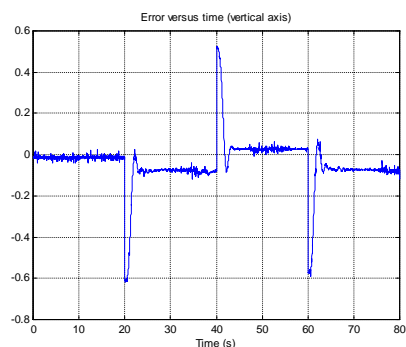
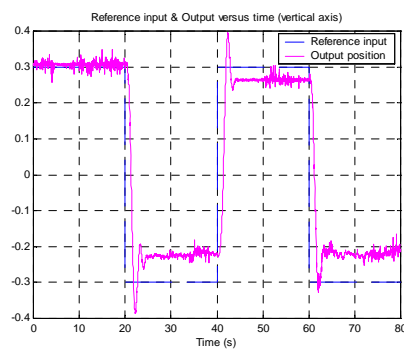
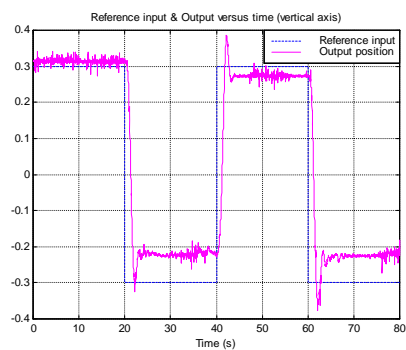


Figure 6.2.1: With boundary control

Figure 6.2.2: With FLC 3-3-7

6.2.3 Sine wave reference signal

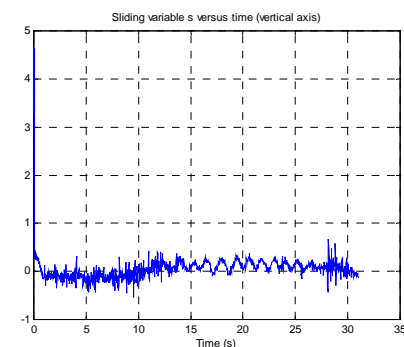
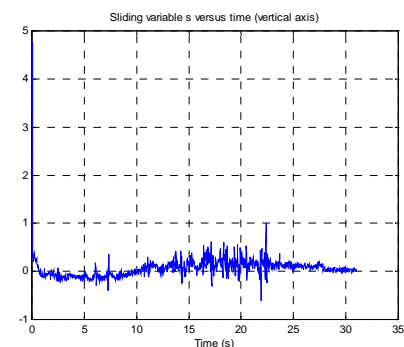
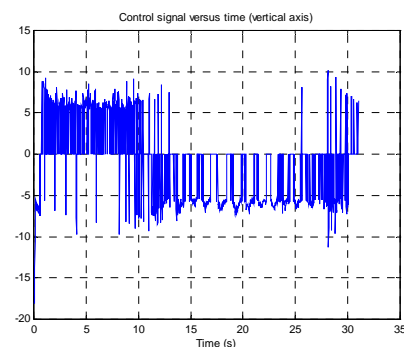
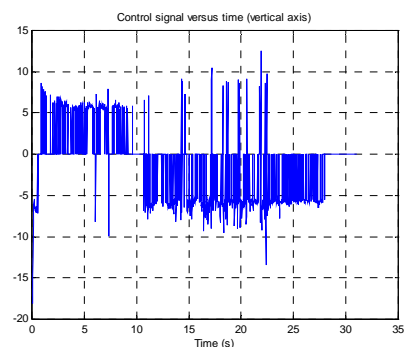
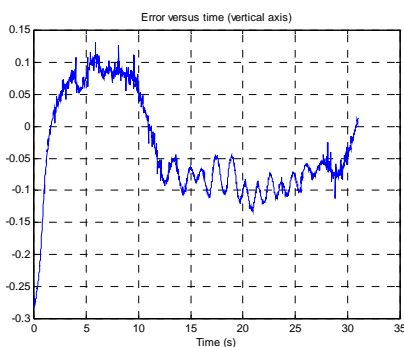
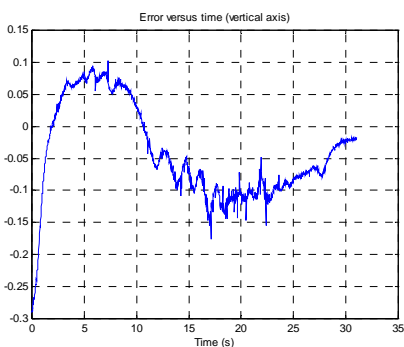
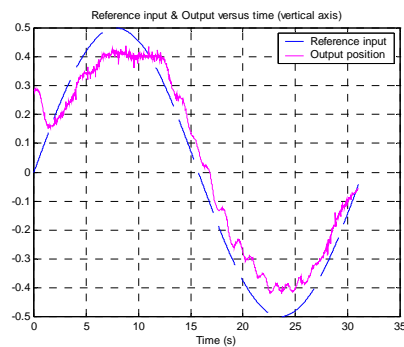
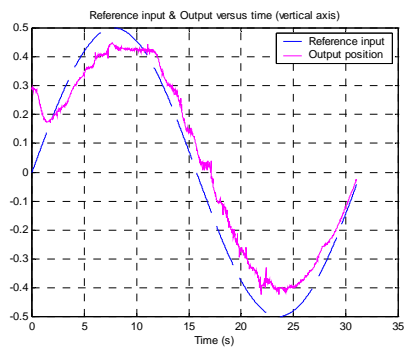


Figure 6.2.3: With boundary control

Figure 6.2.4: With FLC 3-3-7

6. Practical implementation

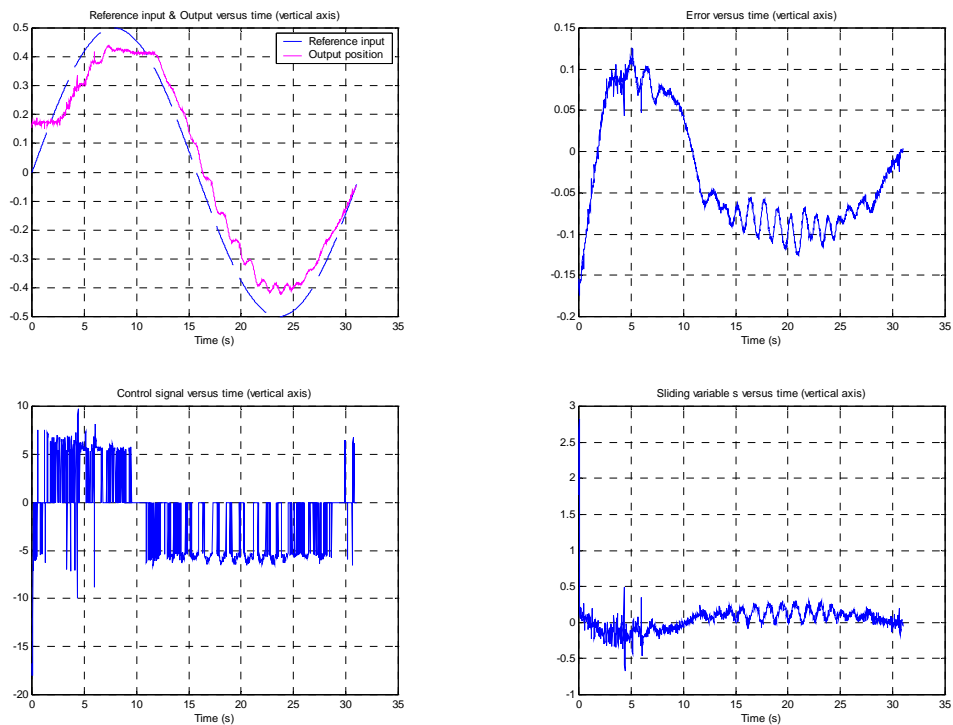


Figure 6.2.5: With FLC 5-5-7

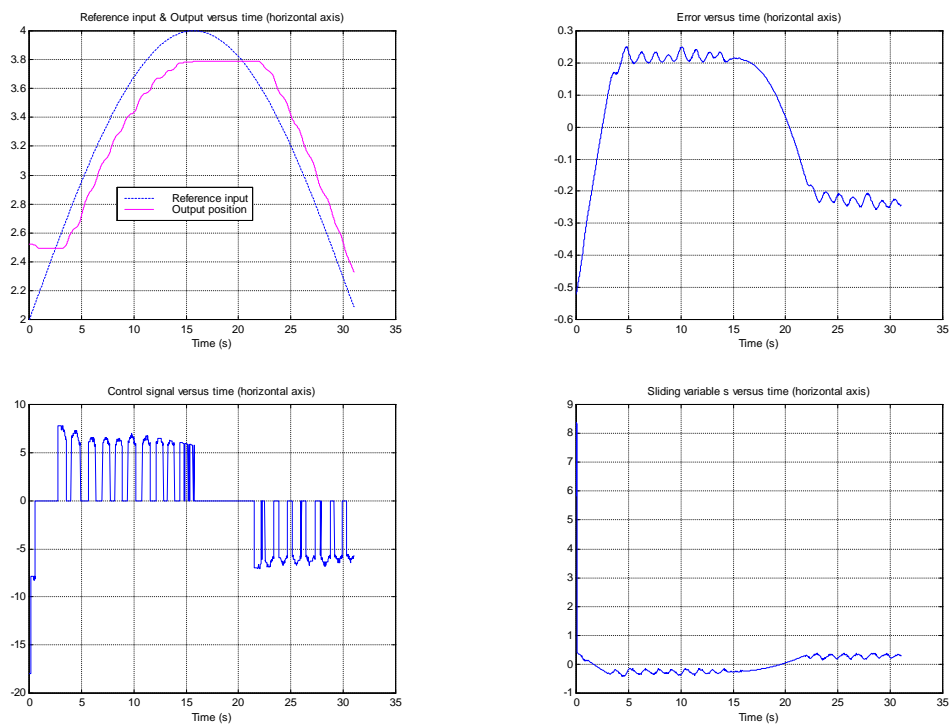


Figure 6.2.6: With FLC 3-3-7

6.2.4 Joystick reference signal

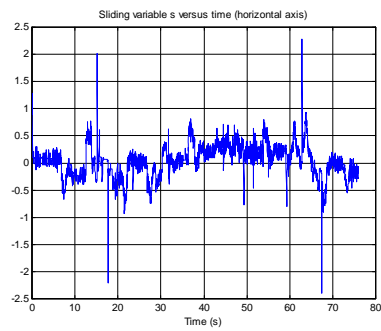
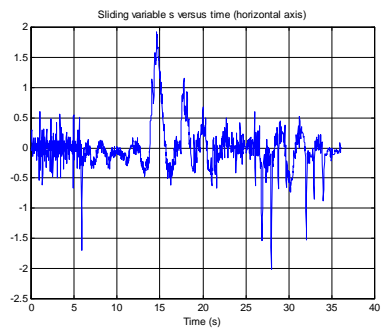
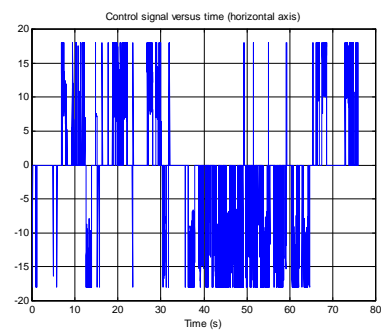
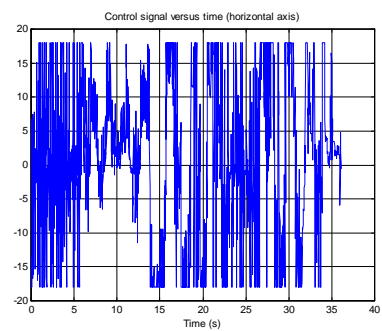
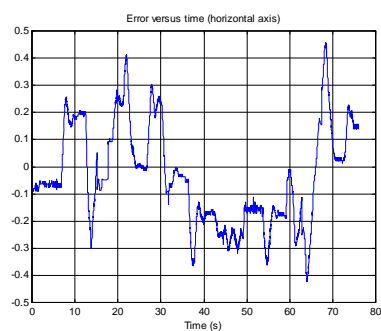
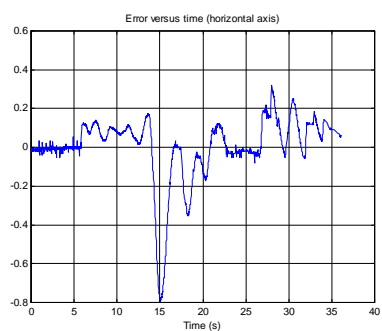
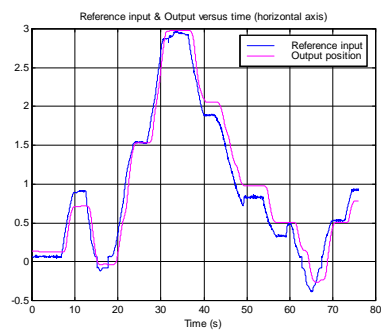
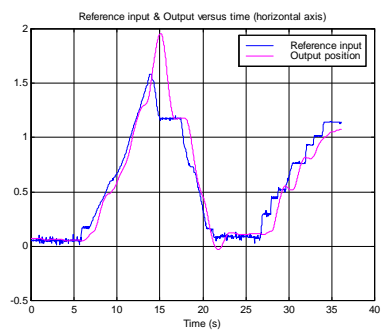


Figure 6.2.7: With boundary control

Figure 6.2.8: With FLC 3-3-7

6. Practical implementation

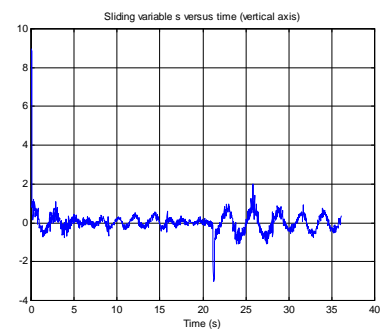
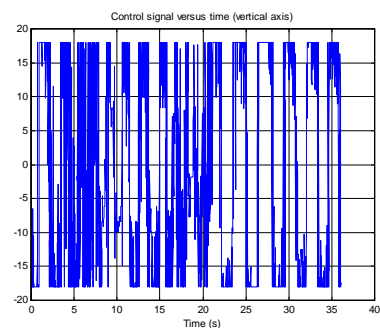
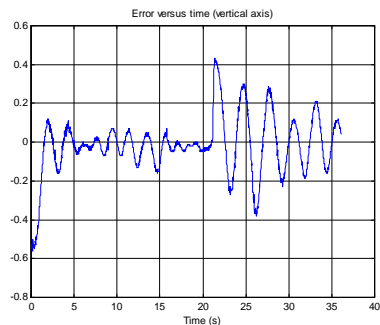
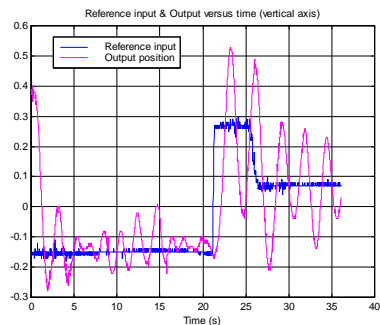


Figure 6.2.9: With boundary control

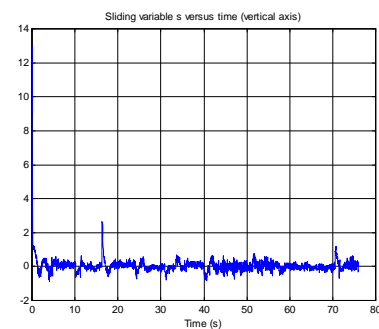
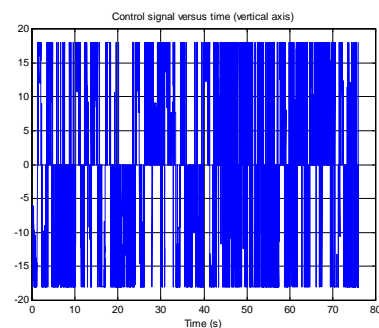
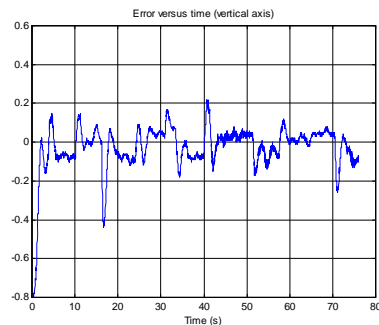
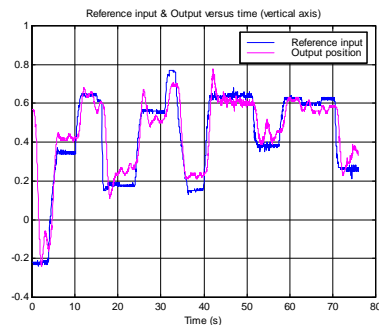


Figure 6.2.10: With FLC 3-3-7

6.3 Analysis of practical results

The implementation of the FASMC in the physical system has drawn the following observations. In the case of pre-defined reference signal, the following observations were obtained:

- With square wave reference signal, the FASMC (with FCL 3-3-7) and the ASMC from 1999 (with boundary layer control) gives almost the same performance except a small improvement in tracking error by FASMC (see Figures 6.2.1 and 6.2.2).
- However, when a sine wave reference is applied, FASMC 3-3-7 has provided enhanced chattering reduction compared to the old ASMC (refer control signal plots in Figures 6.2.3 and 6.2.4) while still keeping the same performance in terms of tracking error and response time.
- In that case, the implementation of FASMC with FLC 5-5-7 remarkably reduced the chattering effects. From Figure 6.2.5, it can be seen that the tracking error is still the same while the control signal and sliding variable are greatly reduced compared with the old ASMC and FASMC 3-3-7. That is, increasing the number of fuzzy variables (finer segmentation on the boundary layer) can remarkably improve the chattering control.
- The controller performance on the horizontal axis is better than that on the vertical axis (see Figure 6.2.6). It is obvious when looking at the control signal and sliding variable plots. At first sight, the tracking error seems to be much worse than on the vertical axis but it is not true. Careful scrutiny shows that the magnitude of the reference signal in the horizontal case is much larger (4 volts) than that for the vertical case (0.5 volts). Meanwhile, its maximum error

is around 0.25 (only $0.25/4 = 6.25\%$ of the reference signal) and error in the vertical axis is 0.1, i.e. 20% of the reference signal. The reason is that the vertical axis has higher non-linearities on inertia and viscous friction due to the effect of gravity.

The analysis with pre-defined, simple reference signal has proved that the FAMSC performance is better in chattering control while maintaining the same error tracking, robustness, etc. performance compared to the ASMC in 1999's project.

However, the main purpose of the system in this project is to follow the joystick reference signal. This kind of signal is totally random, complex and unpredictable. Since the reference signals in different running sessions are quite dissimilar (in shape, magnitude, changing speed, etc.), it is rather difficult to quantitatively compare the performances of this project's FAMSC and the old ASMC. Therefore, qualitative comments based on observations and common sense are given in most cases and quantitative remarks are presented only if possible (e.g. relative tracking error – actual error compared to reference magnitude).

Observations from the system behaviour show that:

- From Figures 6.2.7 and 6.2.8, it can be seen that the magnitude of control signals and sliding variables are the same for ASMC and FAMSC while **chatter is significantly reduced by FAMSC**. Meanwhile, the relative errors in 2 cases are almost the same (around 13%, $0.2/1.5$ for ASMC and $0.4/3$ for FAMSC). This is an extremely advanced performance when noting that the reference signal in the FAMSC case is much more complex and 2 times greater than that in the ASMC case.

- Figures 6.2.9 and 6.2.10 show the **superior performance of FASMC in tracking error**. Although the reference signal in the ASMC case is very simple and 2 times smaller compared to that in the FASMC case, the system still swings a lot around the reference trajectory while FASMC gives very stable tracking. The error of ASMC is even worse: 133.3% ($0.4/0.3$) while FASMC error is 66.7% ($0.4/0.6$). Note that the relative error here is calculated from extreme error and the largest signal magnitude to provide a general idea about the error performance. Considering the tracking errors on the whole process (stable state), the performance of FASMC is much better than that of the ASMC in 1999.

Through the testing with joystick reference signals, the outstanding performance of FASMC in terms of tracking error, system response, settling time, stability and robustness has been strongly confirmed. Despite the uncertainties of system parameter and variant, unknown disturbances from the external environment, the designed FASMC is always successful in driving the system to closely follow the reference signal.

While the old ASMC was good at reducing the chattering effects with simple, pre-defined and small reference signals, it failed to do so when complex, completely random and large reference signals were applied (from the joystick). The designed FASMC has effectively solved this problem while still maintaining the same quality for other performance criteria (e.g. tracking error, system response).

In conclusion, the practical implementation on the real system has provided specific and reliable evidence about the success of the designed FASMC in controllability, stability, robustness and chattering reduction.

6.4 Some additional remarks

6.4.1 Limitations of the physical plant

In this section, we will summarize and discuss some physical limitations of the balancing beam that affect on the performance of the controller and how the designed controller deals with them.

Noise:

- Since the controller uses error derivatives for its calculation, it is rather sensitive to noise (like PID – proportional integral derivative controller). Large and spike noise in the reference and output signal (due to sensor devices, connections) will create sudden change in tracking error, then sliding variable, and so control signal. The final result is severe system fluctuation.
- The problem can be partly reduced by increasing the amount of historic data used in estimating the error derivative (to smoothen the derivative), the number of linguistic variables, etc. However, system response will be lower and larger steady-state error may occur (see 4.5).
- Therefore, it is better to directly tackle at its source by eliminating as much noise in the system as possible. That is, additional budget is required to obtain better sensor devices, converters and connections.

Un-symmetrical properties of driving propellers:

- As mentioned in 2.4.2, the propellers produce larger tangential force when they rotate in the positive direction. Therefore, the error can converge to zero

faster (like a greater error gain is set) but overshoot and chatter are also higher (see results with square wave reference, Figures 6.2.1 and 6.2.2).

Disturbance from propellers:

- When a propeller rotates around the beam axis, it generates a centrifugal force, which in turns exerts an external disturbance on the other axis aside from normal disturbances. However, this project's controller can successfully deal with such disturbance and maintain the beam at the specified position due to its strong robustness (by driving it back whenever it is moved out).

Dead-zone of driving motors:

- The existence of a large dead-zone in the operating range of motors increases the tracking error in that region and then, larger chattering when the system goes out of that dead-zone. This is because within that area, the variation in control signal cannot move the beam and when the control signal exceeds the dead-zone limits, the beam will be suddenly shifted.
- The old ASMC failed to solve this problem but the implementation of the new FASMC has considerably overcome it. The beam now is gradually moved out of the dead-zone region.

System uncertainties:

- As mentioned before, the moment inertia J and viscous friction coefficient B are non-linear and vary with the position and speed of the beam. Other uncertainties are friction from motors and the external environment, which have not been taken into the system model. However, the FASMC has dealt

quite well with those parametric non-linearities and uncertainties to provide excellent tracking performance and stability.

Small bandwidth of amplifier:

- When the joystick is moved too fast with high frequency over the limited bandwidth of amplifier, the high frequency noise make the controller think that the reference signal is making numerous tiny movements at high speed. The old ASMC fails to follow and the beam oscillates a lot. The newly designed FASMC has significantly improved the reduction of chatter in this case, however, the limited bandwidth still needs to be physically solved.

Limited computer resources:

- In each sampling time, the controller has to perform a lot of manipulation; therefore, sufficient sampling time should be set. However, smaller sampled time (high sampling frequency) is preferred since system parameters will be updated more often and the generated control signal will be more accurate. In 1999, the sampling time of ASMC was 10ms and it functioned well with the 486 DX processor. This year, a 133MHz Pentium processor (faster calculation speed) was used so the sampling time could be reduced. However, the additional FLC requires much more computation in the fuzzification, reasoning and defuzzification process (more than the whole ASMC computation). Therefore, the sampling time cannot be decreased.
- As mentioned above, the initial attempt was to develop the controller software in Windows with on-line plotting and graphical user-interface. However, the limitations of processor speed, memory and graphic card prevented this implementation.

6.4.2 How to choose optimum controller parameters

From the comments about the effects of controller parameters in section 4.5 and the simulation and practical implementation experience, it can be seen that choosing an optimum set of parameters is in fact making a compromise between various factors. A good performance in this aspect is often obtained by the sacrifice of other aspects.

For example, a higher error gain can be set to make the system error converge towards zero faster but the trade-off is larger steady-state error, more oscillation and longer settling time. Similarly, increasing the boundary width or the number of fuzzy variables tends to produce larger steady-state error although a smaller control signal (and overshoot) and less chattering can be obtained.

In military applications such as missile guiding, fast response is the most important requirement so high error gain may be preferred. Meanwhile, civilian applications that are related to humans (ship, plane control, etc.), stability and minimum oscillation are more important than accurate tracking error and fast system response. For such cases, larger boundary width can be chosen.

In other words, it is really dependent on the application's purpose to define which performance criteria are important and then determine which values of controller parameters will be suitable. Different objectives will lead to different "optimum" parameters.

7 Conclusion and future improvements

This project introduced a combination of fuzzy logic controller (FLC) and conventional adaptive sliding controller (ASMC) to control and reduce the chattering for the 2-degree balancing beam tracking system with non-linearities and uncertainties. The Fuzzy Adaptive Sliding mode controller (FASMC) was successfully designed, tested by Matlab simulation and implemented in the physical system to provide excellent performance results.

Compared to the old ASMC, the following advantages of the FASMC over ASMC can be observed:

- All good properties of ASMC such as strong robustness, fast response, etc. remain and are even improved.
- The chattering, which is unavoidable in ASMC, is effectively reduced in FASMC and the control signal magnitude becomes smaller.
- The tracking performance is better since the error between output and reference signal is smaller.
- The overall performance is significantly improved in the case of random, unpredictable and noisy reference signals (from the joystick). In these situations, the new FASMC can drive the system closely following the desired trajectory with an evidently smaller chattering and error.

7. Conclusion and future improvements

The fuzzy controller is rather independent of the ASMC; therefore, in reality one can reduce the chattering of an existing system just by adding an additional fuzzy component. Further improvement in the rule base and membership functions of the controller will give better tracking and system response.

Although the project's FASMC is currently the best control scheme for the balancing beam system, some further improvements can still be made. For example, FASMC assures that the error will asymptotically converge to zero along with increasing time. The problem is that it cannot specify how long it will take the error to reach zero (maybe infinity) and the closer to zero, the slower the convergence.

Using a new control theory – Finite time sliding mode control – instead of conventional sliding mode control, can solve this problem. This theory was recently proposed by Dr. Zhihong Man and Dr. X.H. Yu and could be applied to provide exact convergence time with superior robust characteristics.

Another possible improvement is to control through a communication network. That is, the user of a computer with a joystick can control the distant system, which is connected to another computer, by transmitting data via the network. A rough configuration for this idea is shown in Figure 7.1.

The successful implementation of these suggestions will further improve the performance of the system's controller.

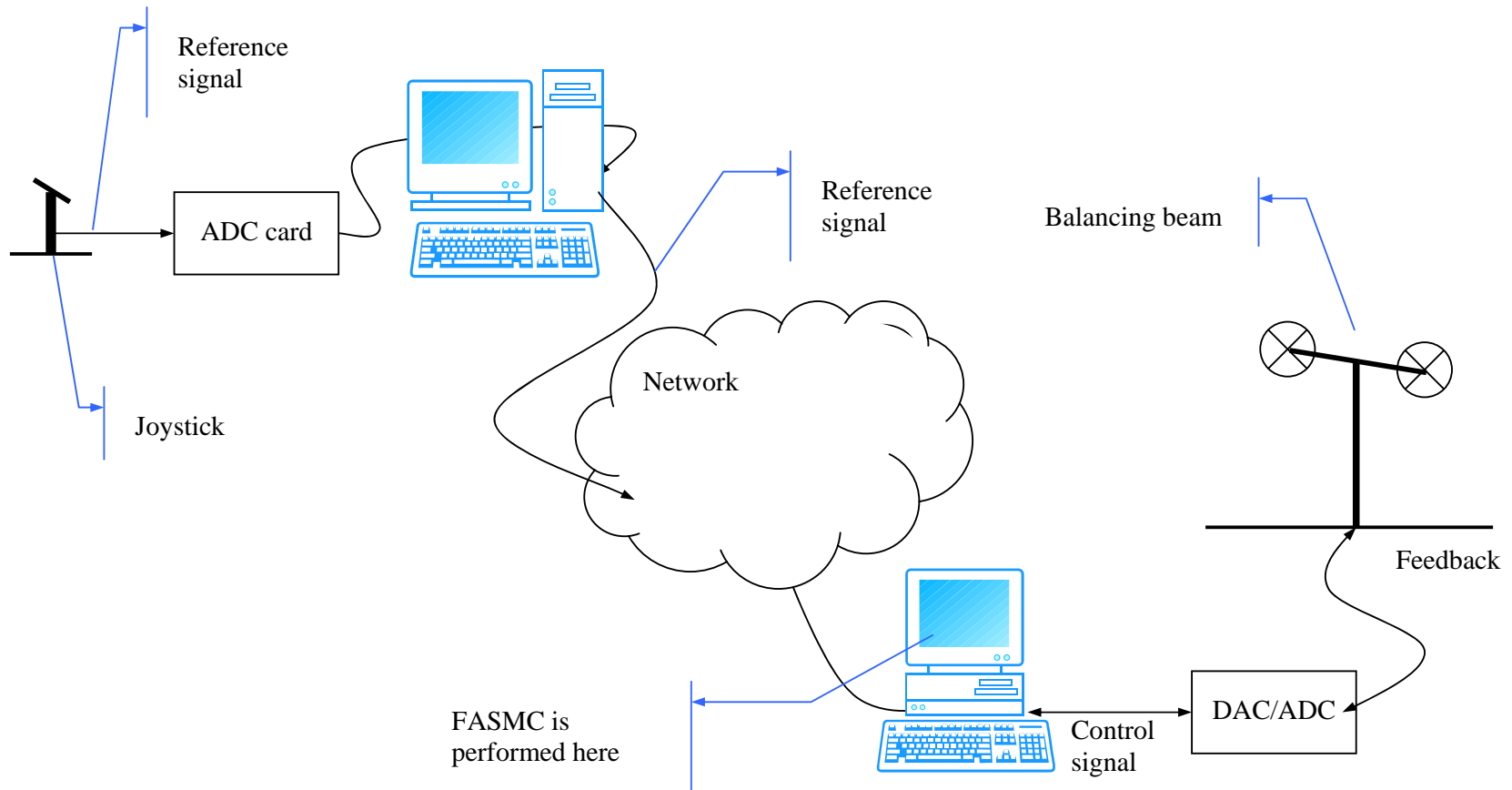


Figure 7.6.4.1: Proposal for control over network

References & Bibliography

- [1]. Ogata, K., *Modern control engineering*, 3rd ed., Prentice-Hall, 1997.
- [2]. Chalam, V.V., “Adaptive control system”, *Techniques and application*, 1987.
- [3]. Gupta, Madan M. (ed. & Chen, Chi-Hau, *Adaptive methods for control system design*, IEEE Press, 1986.
- [4]. Laudau, Yoan D., *Adaptive control*, Marcel Dekker, 1979, pp.42-43.
- [5]. Franklin, Gene F., Powell, J. David & Workman, Michael L., *Digital control of dynamic systems*, 2nd ed., Addison-Wesley, 1992.
- [6]. Hsu, Liu & Costa, Ramon R., “Variable structure model reference adaptive control using input and output measurements”, *International Journal of Control*, Vol. 46, No. 2, 1992.
- [7]. Zhihong Man & Xinghuo Yu, “Adaptive terminating sliding mode tracking control for rigid robotic manipulators with uncertain dynamics”, *JSMA International Journal*, Vol. 40, No.3, Series C, September 1997.
- [8]. Lee, Chuen Chien, “Fuzzy logic in control systems: Fuzzy logic controller – Part I, II”, *IEEE Transactions on systems, man, and cybernetics*, Vol. 20, No. 2, March/April 1990.
- [9]. Lin, Sinn-Cheng & Kung, Chung-Chun, “A linguistic fuzzy-sliding mode controller”, *ACC*, 1992.
- [10]. Yager, R.R.& Filer, D.P, *Essentials of fuzzy modeling and control*, John Wiley & Sons, 1994.

- [11]. Li, H.X., Gatland, H.B. & Green, A.W., "Fuzzy variable structure control", *IEEE Transactions on systems, man, and cybernetics – Part B: Cybernetics*, Vol. 27, No. 2, April 1997.
- [12]. Young, K. David, Utkin, Vadim I. & Umit Orguzer, "A control engineer's guide on sliding mode control", *IEEE Transactions on control technology*, Vol. 7, No.3, May 1999.
- [13]. Zhihong Man, Wu, H.R. & Palaniswami, M., "An adaptive tracking controller using neural networks for a class of nonlinear systems", *IEEE Transactions on neural networks*, Vol. 9, No.5, September 1998.
- [14]. Decarlo, Raymond A., Zak, Stanislaw H. & Matthews, Gregory P., "Variable structure control of non-linear multivariable systems: a tutorial", *Proceedings IEEE*, Vol. 76, No. 3, March 1998.
- [15]. Xinghuo Yu, Zhihong Man & Baolin Wu, "Design of fuzzy sliding mode control systems", May 1995.
- [16]. Mahfouf M., Kee C.H., "Fuzzy logic-based anti-sway control design for overhead cranes", *Neural computing & applications*, Vol. 9, 2000, pp. 38-43.
- [17]. Liu, T.S. & Wu, J.C., "A model for a rider-motorcycle system using fuzzy control", *IEEE Transactions on systems, man, and cybernetics*, Vol. 23, No. 1, January/February 1993.
- [18]. Zhen Yu Zhao, Masayoshi Tomizuka & Setsuo Sagara, "A fuzzy tuner for fuzzy logic controllers", *ACC/TP3*, 1992, pp. 2268-2272.

- [19]. Mounir Ben Ghalia, "Modelling and robust control of uncertain dynamical systems using fuzzy set theory", *Int. J. Control*, Vol. 68, No. 6, 1997, pp. 1367-1395.
- [20]. Mei Fei, Zhihong Man, Nguyen Thong & Yu Xinghuo, "A robust tracking control scheme for a class of nonlinear systems with fuzzy nominal model", *Appl. Math. and Comp. Sci.*, Vol. 8, No. 1, 1998, pp. 145-158.
- [21]. Ha, Q.P., Rye, D.C. & Durrant-Whyte, H.F., "Robust sliding mode control with application", *Int. J. Control*, Vol. 72, No. 12, 1999, pp. 1087-1096.
- [22]. Lee Kok Long Anthony, *Adaptive sliding mode control of two-degree of freedom balancing beam*, Honours thesis of Bachelor of Engineering, University of Tasmania, 1999.
- [23]. Jim, Jee Gah, *An RBF neural network based controller for a 5-link rigid robotic manipulator*, Honours thesis of Bachelor of Engineering, University of Tasmania, 1999.
- [24]. Do Khac Cuong, *Fuzzy logic controller for water tank system*, Honours thesis of Bachelor of Engineering, University of Tasmania, 1998.
- [25]. Banyasz, Cs. (ed.), *Adaptive systems in control and signal processing 1995: a postscript volume from the 5th IFAC Symposium, Budapest, Hungary, 14-16 June 1995*, Pergamon, 1995.
- [26]. Mandan M. Gupta (ed.), *Adaptive methods for control system design*, IEEE Press, 1986.
- [27]. Zhihong Man, *Lecture notes in Control Engineering*, University of Tasmania, 1999.

- [28]. Zhihong Man, *Lectures notes in Engineering Systems*, University of Tasmania, 2000.
- [29]. *Simulink 1.3 Release Notes*, The MathWorks, Inc., January 1995.
- [30]. *Simulink User's Guide*, The MathWorks, Inc., November 1994.
- [31]. *Using Simulink*, The MathWorks, Inc., January 1998.
- [32]. *Simulink 2.1 New Features*, The MathWorks, Inc., May 1997.
- [33]. *Fuzzy logic toolbox User's Guide*, The MathWorks, Inc., January 1998.
- [34]. *Application Program Interface Guide*, The MathWorks, Inc., January 1998.
- [35]. Deitel, H.M & Deitel, P.J., *C++ How to program*, Prentice Hall, 1994.
- [36]. Reisdorph, Kent (et al.), *Borland[®] C++Builder[™]4 (Unleashed)*, SAMS, 1999.
- [37]. Julius T. Tou, *Optimum design of digital control systems*, Academic Press, United States, 1963.
- [38]. Sage, Andrew P. & White III, Chelsea C., *Optimum systems control*, Prentice Hall, United States, 1997.
- [39]. Bitmead, Robert R., Gervers Michel, Vincent Wertz, *Adaptive optimal control: The thinking man's GPC*, Prentice Hall, Australia, 1990.
- [40]. Cox Earl, *The fuzzy systems handbook: a practitioner's guide to building, using and maintaining fuzzy systems*, Academic Press, 1994, United States.

Appendix A: Matlab programs for simulation & off-line data plotting

A.1 Simulation program

```
%=====
% Honours thesis - Fuzzy Adaptive Sliding Mode tracking Controller
% University of Tasmania, Australia
%
% Nguyen Vu Thanh
% Email address: n_vuthanh@yahoo.com
% 3:52am, Tuesday, 17.10.2000
%=====

%===== INITIALIZATION =====

delay = 3;
disturbance = 2;
dimension = 15; %number of iterations for Least Square Method (LSM)

for i = 1:dimension
    r(i) = 0;
    rd(i) = 0;
end

boundary = 0.2
limit = 2;
rdd = 0;

y = 0; %initial condition of system output
yd = 0; %initial condition of y'
ydd = 0; %initial condition of y''

e = 0; %initial value of the tracking error; e=x-xm
ed = 0; %initial value of derv of error, e'=x'-xm'

%=====
edd = 0; % ADDITIONAL
sd = 0;
%=====

Bo = 1;
Jo = 1; %u increase when Jo increase
fo = 1; %u increase when fo increase

gain = 1; %amplifier gain in motor, K
lamda = 1; %steady state error affected. error increase when
lamda small
adaptive_gain = 1;

thetal = 0; %Adaptive Law
thetal_d = 0; %estimate of adaptive law
```

Appendix A: Matlab programs for simulation and data plotting

```
theta2 = 0;      %Adaptive Law
theta2_d = 0;    %estimate of adaptive law

u = 0;          %the control input signal is initialized as zero
dt = 0.01;      %the sampling interval is 0.01s
z = 0;          %variable for triangular signal
TimeLimit = 20;
NoLimit = TimeLimit/dt;

%===== MAIN SIMULATION =====

for i = 1:NoLimit

    %Sine function
    %r(1) = 5*sin(i*dt);
    %=====

    %Step function
    %if (i <= NoLimit/4)
    %    r(1) = 0;
    %else
    %    r(1) = 5;
    %end
    %=====

    %Square function
    if (i < NoLimit/4)
        r(1) = 0;
    elseif (i > NoLimit/4 & i <= 0.5*NoLimit)
        r(1) = 5;
    elseif (i > 0.5*NoLimit & i <= 0.75*NoLimit)
        r(1) = 0;
    else
        r(1) = 5;
    end

    %=====

    %Misc function
    %if (i <= NoLimit/4)
    %    r(1) = 0.01*i + sin(i*dt);
    %else (i > NoLimit/4 & i <= 0.75*NoLimit)
    %    r(1) = 2*sin(i*dt) + 3*sin(2*i*dt) + 4*sin(3*i*dt) - i*0.001;
    %end
    %=====

    %Triangular function
    %=====
    %r(1) = 0.01*z;
    %if (i < NoLimit/4)
    %    z = z+1;
    %elseif (i > NoLimit/4 & i <= 0.5*NoLimit)
    %    z = z-1;
    %elseif (i > 0.5*NoLimit & i <= 0.75*NoLimit)
    %    z = z+1;
    %else
    %    z = z-1;
    %end

    r1(i) = r(1);
```

```

%=====
% adding noise to signal
r(1) = r(1) + (rand(1)/5 - 0.1);

%=====
%LSM to obtain the first derv of ref signal
a = 0;
b = 0;
aa = 0;
ab = 0;

for k = 1:dimension
    a = a + (k+1)*dt;
    aa = aa + (k+1)*dt*(k+1)*dt;
    b = b + r(k);
    ab = ab + (k+1)*dt*r(k);
end

rd(1) = -((a*b-ab*dimension)/(a*a-aa*dimension));
rdl(i) = rd(1);

%LSM again to obtain the second derv of ref signal
a = 0;
b = 0;
aa = 0;
ab = 0;

for k = 1:dimension
    a = a +(k+1)*dt;
    aa = aa + (k+1)*dt*(k+1)*dt;
    b = b + rd(k);
    ab = ab + (k+1)*dt*rd(k);
end

rdd = -((a*b-ab*dimension)/(a*a-aa*dimension));
rddl(i) = rdd;
%=====

%swapping loop to save last (dimension) of ref signal data points
for k = dimension:-1:2
    r(k) = r(k-1);
    rd(k) = rd(k-1);
end

%=====
%Calculations for sliding mode control

bo = gain/Jo;           %calculate b

ydd = (u*bo) -((Bo*yd)/Jo); %calculate y''
yddl(i) = ydd;

yd = yd + dt*ydd;      %compute y'
ydl(i) = yd;

y = y + dt*yd;         %compute y
yl(i) = y;
% add noise
y = y + (rand(1)/5 - 0.1);

```

```

e = y(1) - r(1);           %calculate the tracking error
el(i)= e;

ed = yd - rd(1);         %calculate e'
edl(i) = ed;
%=====
% calculalte 1st devriavtiv of e
edd = ydd - rdd(1);
edd1(i) = edd;
%=====

%=====
sd = edd + lamda*ed;      %calculate 1st derivative S'
sdl(i)= sd;
%=====
s = ed + lamda*e;        %calculate sliding variable S
sl(i) = s;
%=====

if abs(s) > limit;
    thetal_d = adaptive_gain*abs(s); %adaptive law 1
    thetal = thetal + dt*thetal_d; %compute thetal_hat'

    theta2_d = adaptive_gain*abs(s)*lamda*abs(ed); %adaptive law 2
    theta2 = theta2 + dt*thetal_d; %compute theta2_hat'
else
    thetal = thetal;
    theta2 = theta2;
end

if thetal > 20; thetal = 20; end;
if theta2 > 20; theta2 = 20; end;

%=====
% Fuzzy part
a = readfis('paper337');
%=====

if abs(s) >= boundary;
    if (s<0)
        sign_s = -1;
    else
        sign_s = 1;
    end
else
    sign_s = s/boundary;
%=====
% In rule base, range of s is -6 -> 6
% range of sd is -6 -> 6

% s*2 ==> same as last year with Boundary
% s*3 ==> better but some has a bit chattering
% b = evalfis([s*2.5 sd/64],a);
% sign_s = b; % + sign_s;
% sign_sl(i) = sign_s;
%=====
end;

u = -sign_s*(thetal + theta2*abs(rdd) + theta2*lamda*abs(ed));
%calculate control signal

```

```

u1(i) = u;

%=====
u = u + disturbance;
u2(i+delay) = u;
u = u2(i);
%=====

% deadzone effect
if (u > 3.2 & u < -2.8) u =0; end
if (u < 0) u = u*0.85; end

end % of for loop

%===== PLOTTING =====
t = dt:dt:TimeLimit;

figure(1);
plot(t,r1,'b--',t,y1,'m');
legend('b--','Reference input','m','Output position');
xlabel('Time (s)');
title('Reference input & Output position versus time'), grid;

figure(2);
plot(t,e1,'b'); % For error dot : t,ed1,'m--'
%legend('b','Error','m--','First derivative of error');
xlabel('Time (s)');
title('Error versus time'), grid;

figure(3);
plot(t,u1,'');
%legend('b','Control signal');
xlabel('Time (s)');
title('Control signal versus time'), grid;

figure(4);
plot(t,s1,'b', t,sd1,'m--');
legend('b','Sliding variable s','m--','First derivative of s');
xlabel('Time (s)');
title('Sliding variable & its 1st derivative versus time'), grid;

figure(5);
subplot(2,2,1), plot(t,r1,'b--',t,y1,'m'),
legend('b--','Reference','m','Output');
%xlabel('Time (s)');
title('Reference & Output versus time'), grid;

subplot(2,2,2), plot(t,e1,'b');
%legend('b','Error','m--','First derivative of error');
%xlabel('Time (s)');
title('Error versus time'), grid;

subplot(2,2,3), plot(t,u1,''),
xlabel('Time (s)');
title('Control signal versus time'), grid;

subplot(2,2,4), plot(t,s1,'b',t,sd1,'m--'),
legend('b','s','m--','s dot');
xlabel('Time (s)');
title('Sliding variable s versus time'), grid;

```

A.2 Programs for off-line data plotting

A.2.1 For both horizontal and vertical axes

```
load outref.dat;
load outs.dat;
load outu.dat;
load outy.dat;
load outerr.dat;

load voutref.dat;
load vouts.dat;
load voutu.dat;
load vouty.dat;
load vouterr.dat;

Reference = outref(:,1);
Output    = outy(:,1);
Error     = outerr(:,1);
Sliding   = outs(:,1);
Control   = outu(:,1);

vReference = voutref(:,1);
vOutput    = vouty(:,1);
vError     = vouterr(:,1);
vSliding   = vouts(:,1);
vControl   = voutu(:,1);

dt = 0.02;
HorTimeLimit = length(outref)*dt;
VerTimeLimit = length(voutref)*dt;

t_hor = dt:dt:HorTimeLimit;
t_ver = dt:dt:VerTimeLimit;

figure(1);
plot(t_hor,Reference,'b--',t_hor,Output,'m');
legend('b--','Reference input','m','Output position');
xlabel('Time (s)');
title('Reference input & Output versus time (horizontal axis)'), grid;

figure(2);
plot(t_hor,Error,'b'); % For error dot : t,ed1,'m--'
%legend('b','Error','m--','First derivative of error');
xlabel('Time (s)');
title('Error versus time (horizontal axis)'), grid;

figure(3);
plot(t_hor,Control,'');
xlabel('Time (s)');
title('Control signal versus time (horizontal axis)'), grid;

figure(4);
plot(t_hor,Sliding,'b'); %, t_hor,SlidingDot,'m--');
```

```

%legend('b','Sliding variable s','m--','First derivative of s');
xlabel('Time (s)');
%title('Sliding variable & its derivative versus time (horizontal
axis)'), grid;
title('Sliding variable s versus time (horizontal axis)'), grid;

figure(5);
subplot(2,2,1), plot(t_hor,Reference,'b--',t_hor,Output,'m'),
legend('b--','Reference','m','Output');
title('Horizontal axis - Reference & Output versus time'), grid;
subplot(2,2,2), plot(t_hor,Error,'b');
title('Error versus time'), grid;
subplot(2,2,3), plot(t_hor,Control,''),
xlabel('Time (s)');
title('Control signal versus time'), grid;
subplot(2,2,4), plot(t_hor,Sliding,'b'); %, t_hor,SlidingDot,'m--'),
%legend('b','s','m--','s dot');
xlabel('Time (s)');
title('Sliding variable s versus time'), grid;

%=====
figure(6);
plot(t_ver,vReference,'b--',t_ver,vOutput,'m');
legend('b--','Reference input','m','Output position');
xlabel('Time (s)');
title('Reference input & Output versus time (vertical axis)'), grid;

figure(7);
plot(t_ver,vError,'b'); % For error dot : t,ed1,'m--'
%legend('b','Error','m--','First derivative of error');
xlabel('Time (s)');
title('Error versus time (vertical axis)'), grid;

figure(8);
plot(t_ver,vControl,'');
xlabel('Time (s)');
title('Control signal versus time (vertical axis)'), grid;

figure(9);
plot(t_ver,vSliding,'b'); %, t_ver,vSlidingDot,'m--');
%legend('b','Sliding variable s','m--','First derivative of s');
xlabel('Time (s)');
%title('Sliding variable & its derivative versus time (horizontal
axis)'), grid;
title('Sliding variable s versus time (vertical axis)'), grid;

figure(10);
subplot(2,2,1), plot(t_ver,vReference,'b--',t_ver,vOutput,'m'),
legend('b--','Reference','m','Output');
title('Vertical axis - Reference & Output versus time'), grid;
subplot(2,2,2), plot(t_ver,vError,'b');
title('Error versus time'), grid;
subplot(2,2,3), plot(t_ver,vControl,''),
xlabel('Time (s)');
title('Control signal versus time'), grid;
subplot(2,2,4), plot(t_ver,vSliding,'b'); %, t_ver,vSlidingDot,'m--');
%legend('b','s','m--','s dot');
xlabel('Time (s)');
title('Sliding variable s versus time'), grid;

```

A.2.2 For horizontal axis

```

load outref.dat;
load outs.dat;
load outu.dat;
load outy.dat;
load outerr.dat;

Reference = outref(:,1);
Output    = outy(:,1);
Error     = outerr(:,1);
Sliding   = outs(:,1);
Control   = outu(:,1);

dt = 0.02;
TimeLimit = length(outref)*dt;
t = dt:dt:TimeLimit;

figure(1);
plot(t,Reference,'b--',t,Output,'m');
legend('b--','Reference input','m','Output position');
xlabel('Time (s)');
title('Reference input & Output versus time (horizontal axis)'), grid;

figure(2);
plot(t,Error,'b'); % For error dot : t,ed1,'m--'
%legend('b','Error','m--','First derivative of error');
xlabel('Time (s)');
title('Error versus time (horizontal axis)'), grid;

figure(3);
plot(t,Control,'');
xlabel('Time (s)');
title('Control signal versus time (horizontal axis)'), grid;

figure(4);
plot(t,Sliding,'b'); %, t,SlidingDot,'m--');
%legend('b','Sliding variable s','m--','First derivative of s');
xlabel('Time (s)');
%title('Sliding variable & its derivative versus time (horizontal axis)'), grid;
title('Sliding variable s versus time (horizontal axis)'), grid;

figure(5);
subplot(2,2,1), plot(t,Reference,'b--',t,Output,'m'),
legend('b--','Reference','m','Output');
title('Horizontal axis - Reference & Output versus time'), grid;

subplot(2,2,2), plot(t,Error,'b');
title('Error versus time'), grid;

subplot(2,2,3), plot(t,Control,''),
xlabel('Time (s)');
title('Control signal versus time'), grid;

subplot(2,2,4), plot(t,Sliding,'b'); %, t,SlidingDot,'m--'),
%legend('b','s','m--','s dot');
xlabel('Time (s)');
title('Sliding variable s versus time'), grid;

```

A.2.3 For vertical axis

```
load voutref.dat;
load vouts.dat;
load voutu.dat;
load vouty.dat;
load vouterr.dat;

Reference = voutref(:,1);
Output    = vouty(:,1);
Error     = vouterr(:,1);
Sliding   = vouts(:,1);
Control   = voutu(:,1);

dt = 0.02;
TimeLimit = length(voutref)*dt;
t = dt:dt:TimeLimit;

figure(1);
plot(t,Reference,'b--',t,Output,'m');
legend('b--','Reference input','m','Output position');
xlabel('Time (s)');
title('Reference input & Output versus time (vertical axis)'), grid;

figure(2);
plot(t,Error,'b'); % For error dot : t,ed1,'m--'
%legend('b','Error','m--','First derivative of error');
xlabel('Time (s)');
title('Error versus time (vertical axis)'), grid;

figure(3);
plot(t,Control,'');
xlabel('Time (s)');
title('Control signal versus time (vertical axis)'), grid;

figure(4);
plot(t,Sliding,'b'); %, t,SlidingDot,'m--');
%legend('b','Sliding variable s','m--','First derivative of s');
xlabel('Time (s)');
title('Sliding variable s versus time (vertical axis)'), grid;

figure(5);
subplot(2,2,1), plot(t,Reference,'b--',t,Output,'m'),
legend('b--','Reference','m','Output');
title('Vertical axis - Reference & Output versus time'), grid;

subplot(2,2,2), plot(t,Error,'b');
title('Error versus time'), grid;

subplot(2,2,3), plot(t,Control,''),
xlabel('Time (s)');
title('Control signal versus time'), grid;

subplot(2,2,4), plot(t,Sliding,'b'); %, t,SlidingDot,'m--'),
%legend('b','s','m--','s dot');
xlabel('Time (s)');
title('Sliding variable s versus time'), grid;
```

Appendix B: The controller's software

B.1 Fuzzy controller

B.1.1 The header file of fuzzy class

```
//-----  
// Honours thesis - Fuzzy Adaptive Sliding Mode tracking Controller  
// University of Tasmania, Australia, October 2000  
//  
// Nguyen Vu Thanh  
// Email address: n_vuthanh@yahoo.com  
//-----  
  
#ifndef __FuzzyH  
#define __FuzzyH  
  
//----- For Fuzzy -----  
#include <string.h>  
#include <time.h>  
  
#define MAX(A,B) ((A)>(B) ? (A):(B))  
#define MIN(A,B) ((A)<(B) ? (A):(B))  
#define INFINITY 100000.0  
#define ZERO 1e-15  
#define StepSize 0.01  
#define NumOfParas 5  
//-----  
typedef enum  
{  
    ftTriangle = 0,  
    ftTrapezoidal  
} TMemberFuncType;  
  
//-----  
class Fuzzy  
{  
// class to define the fuzzy set  
    friend class FuzzyController;  
  
    // those operators are between 2 single degrees  
    friend Fuzzy operator & (const Fuzzy&,const Fuzzy&);  
    friend Fuzzy operator & (double,const Fuzzy&);  
    friend Fuzzy operator & (const Fuzzy&, double);  
  
    friend Fuzzy operator | (const Fuzzy&,const Fuzzy&);  
    friend Fuzzy operator | (double,const Fuzzy&);  
    friend Fuzzy operator | (const Fuzzy&, double);  
  
    friend Fuzzy operator ! (const Fuzzy&);  
  
    friend Fuzzy CreateNewFuzzy();  
    friend Fuzzy CreateNewFuzzy(double);
```

```

        friend Fuzzy CreateNewFuzzy(const Fuzzy&);
        friend Fuzzy CreateNewFuzzy(double, double,
                                    TMemberFuncType,
                                    double, double,
                                    double, double = 0.0);
public:
    Fuzzy(); // constructor
    Fuzzy(const Fuzzy&);
    Fuzzy(double);

    // constructor to define the membership funtion
    // Point i are to determine the shape of membership funtion
    // MsfShape is to describe the shape of membership funtion
    Fuzzy(double, double, TMemberFuncType,
           double, double, double, double);

    // Destructor Fuzzy Class
    ~Fuzzy();

    // find the Degree of Membership y from Crisp Input x
    virtual double MemberDegree(const double&);

    Fuzzy& operator = (const Fuzzy&);
    // return the Degree of Membership of input tx
    // Fuzzy operator [] (const double&);
    Fuzzy& DegreeOfMember(const double&);

    // MIN between 1 number & a array of degrees of 2 obj
    Fuzzy& operator && (const Fuzzy&);

    // MAX items of 2 arrays of degrees of 2 object
    Fuzzy& operator || (const Fuzzy&);

    void GenerateMemberValues();
    double Defuzzification();

private:
    TMemberFuncType MemberFuncType;
    double InputCrisp, MsfDegree;
        // degree of membership function for SINGLE input
    double RangeLowerLimit;
        // lower limit of the whole fuzzy range
    double RangeUpperLimit;
        // lower limit of the whole fuzzy range
    double* MsfParas;
        // points to construct the membership function
    int NoOfPoints;
    double* MsfValues;
        // for array of degree

    // to construct the shape of membership function
    void MsfTriangular(const double&, const double&,
                      const double&);
    void MsfTrapezoid(const double&, const double&,
                     const double&, const double&);
};

//=====
class FuzzyController
{

```

```
// friend class Fuzzy;

public:
    FuzzyController(int, double, double, double);
    ~FuzzyController();
    void SetController(int, double, double, double);
    double CrispOutput(const double&, const double&);

private:
    double InputCrisp, OutputCrisp;
    double VarLimit, VarRateLimit, ResultLimit;
    Fuzzy NB_Re, NM_Re, NS_Re, Z_Re, PS_Re, PM_Re, PB_Re;
    Fuzzy NB_V, NM_V, NS_V, Z_V, PS_V, PM_V, PB_V;
    Fuzzy NB_VR, NM_VR, NS_VR, Z_VR, PS_VR, PM_VR, PB_VR;

    int Status;
        // Status = 1: all membership func of Controller is set
        // Status = 0: Controller is not set yet
    int NumberOfVars;
        // Number of linguistics variables of controller

    void SetController5(double, double,double);
    void SetController7(double, double,double);
    double CrispOutput5(const double&, const double&);
    double CrispOutput7(const double &, const double&);

    void SetController337(double, double,double);
    double CrispOutput337(const double&, const double&);

    void SetController557(double, double,double);
    double CrispOutput557(const double&, const double&);

};

//=====

#endif
```

B.1.2 The implemented code of fuzzy class in C++

```
//-----  
// Honours thesis - Fuzzy Adaptive Sliding Mode tracking Controller  
// University of Tasmania, Australia, October 2000  
//  
// Nguyen Vu Thanh  
// Email address: n_vuthanh@yahoo.com  
//-----  
  
#include "Fuzzy.h"  
//-----  
  
//-----  
Fuzzy::Fuzzy()  
// define the membership function parameters  
{  
    RangeLowerLimit = 0.0;  
    RangeUpperLimit = 0.0;  
    InputCrisp       = 0.0;  
    MsfDegree        = 0.0;  
    MemberFuncType   = ftTriangle;  
    NoOfPoints       = 0;  
    MsfValues        = NULL;  
    MsfParas         = new double [NumOfParas];  
    for (int i = 0; i < NumOfParas; i++)  
        MsfParas[i] = 0.0;  
    return;  
}  
  
//-----  
Fuzzy::Fuzzy(double Degree)  
// define the membership function parameters  
{  
    RangeLowerLimit = 0.0;  
    RangeUpperLimit = 0.0;  
    InputCrisp      = 0.0;  
    MsfDegree       = Degree;  
    MemberFuncType   = ftTriangle;  
    MsfValues       = NULL;  
    NoOfPoints      = 0;  
    MsfParas        = new double [NumOfParas];  
    for (int i = 0; i < NumOfParas; i++)  
        MsfParas[i] = 0.0;  
    return;  
}  
  
//-----  
Fuzzy::Fuzzy(const Fuzzy& __Source)  
// define the membership function parameters  
{  
    *this = __Source;  
    return;  
}  
  
//-----  
Fuzzy::Fuzzy(double LowerLimit, double UpperLimit,  
              TMemberFuncType MsfShape,  
              double Point1, double Point2,
```

```

        double Point3, double Point4)
// define the membership function parameters
{
    RangeLowerLimit = LowerLimit;
    RangeUpperLimit = UpperLimit;
    InputCrisp      = 0.0;
    MsfDegree       = 0.0;
    MemberFuncType  = MsfShape;
    MsfValues       = NULL;
    MsfParas       = new double [NumOfParas];

    switch (MemberFuncType)
    {
        case ftTriangle:
            MsfParas[1] = Point1;    // left
            MsfParas[2] = Point2;    // mean
            MsfParas[3] = Point3;    // right
            MsfParas[0] = MsfParas[4] = 0;
            break;
        case ftTrapezoidal:
            MsfParas[1] = Point1;    // Lowleft
            MsfParas[2] = Point2;    // HighLeft
            MsfParas[3] = Point3;    // HighRight
            MsfParas[4] = Point4;    // LowRight
            MsfParas[0] = 0;
            break;
        default:
            // Error code
            break;
    };
    GenerateMemberValues();
    return;
}

//-----
Fuzzy::~~Fuzzy()
{
    if (MsfValues != NULL)
        delete [] MsfValues;
    delete [] MsfParas;
}

//-----
void Fuzzy::GenerateMemberValues()
// to calculate the membership values of points
// & store in array MsfValues[NoOfPoints]
// ONLY for this object, with specific member func
{
    double x;

    NoOfPoints = int((RangeUpperLimit - RangeLowerLimit)/StepSize);
    NoOfPoints = (NoOfPoints < 0) ? (- NoOfPoints) : (NoOfPoints);
    if (MsfValues != NULL)
        delete [] MsfValues;
    MsfValues = new double[NoOfPoints];

    if (MsfValues != NULL)
    {
        for (int i = 0; i < NoOfPoints; i++)
        {
            x = RangeLowerLimit + i*StepSize;

```

```

        MsfValues[i] = MemberDegree(x);
    }
}
return;
}

//-----
Fuzzy CreateNewFuzzy()
// define the membership function parameters
{
    return Fuzzy();
}

//-----
Fuzzy CreateNewFuzzy(double Degree)
// define the membership function parameters
{
    return Fuzzy(Degree);
}

//-----
Fuzzy CreateNewFuzzy(const Fuzzy& __Sour)
// define the membership function parameters
{
    return Fuzzy(__Sour);
}

//-----
Fuzzy CreateNewFuzzy(double LowerLimit,
                    double UpperLimit,
                    TMemberFuncType MsfShape,
                    double Point1, double Point2,
                    double Point3, double Point4)
// define the membership function parameters
{
    return Fuzzy(LowerLimit, UpperLimit, MsfShape,
                Point1, Point2, Point3, Point4);
}

//=====
//----- Define the class member functions -----
//=====
Fuzzy operator & (const Fuzzy &a,const Fuzzy &b)
{
    return Fuzzy(MIN(a.MsfDegree,b.MsfDegree));
};

//-----
Fuzzy operator & (double a,const Fuzzy &b)
{
    return Fuzzy(MIN(a,b.MsfDegree));
};

//-----
Fuzzy operator & (const Fuzzy &a, double b)
{
    return Fuzzy(MIN(a.MsfDegree,b));
};

//-----
Fuzzy operator | (const Fuzzy &a,const Fuzzy &b)

```

```

{
    return Fuzzy(MAX(a.MsfDegree,b.MsfDegree));
};

//-----
Fuzzy operator | (double a,const Fuzzy &b)
{
    return Fuzzy(MAX(a,b.MsfDegree));
};

//-----
Fuzzy operator | (const Fuzzy &a, double b)
{
    return Fuzzy(MAX(a.MsfDegree,b));
};

//-----
Fuzzy operator ! (const Fuzzy &a)
{
    return Fuzzy(1 - a.MsfDegree);
};

//-----
Fuzzy& Fuzzy::operator = (const Fuzzy &r)
{
    MemberFuncType = r.MemberFuncType;
    InputCrisp      = r.InputCrisp;
    MsfDegree       = r.MsfDegree;
    RangeLowerLimit = r.RangeLowerLimit;
    RangeUpperLimit = r.RangeUpperLimit;
    NoOfPoints      = r.NoOfPoints;
    if (MsfParas != NULL)
        delete [] MsfParas;
    MsfParas        = new double [NumOfParas];
    if (MsfParas != NULL)
    {
        for (int i = 0; i < NumOfParas; i++)
            MsfParas[i] = r.MsfParas[i];
    }
    if (MsfValues != NULL)
        delete [] MsfValues;
    MsfValues = new double[NoOfPoints];

    if (MsfValues != NULL)
    {
        for (int i = 0; i < NoOfPoints; i++)
            MsfValues[i] = r.MsfValues[i];
    }
    return *this;
};

//-----
Fuzzy& Fuzzy::operator && (const Fuzzy &a)
// calculate Fi(y), Di(y): MIN between a sigle & and array of degree
{
    for (int i = 0; i < NoOfPoints; i++)
        MsfValues[i] = MIN(a.MsfDegree, MsfValues[i]);
    return *this;
}

//-----

```

```
Fuzzy& Fuzzy::operator || (const Fuzzy &ObjArray2)
// from Fi(y) or Di(y), cal F(y) or D(y)
{
    for (int i = 0; i < NoOfPoints; i++)
        MsfValues[i] = MAX(MsfValues[i], ObjArray2.MsfValues[i]);
    return *this;
}

//-----
Fuzzy& Fuzzy::DegreeOfMember (const double &CrispInput)
// return the Degree of Membership of input tx
{
    InputCrisp = CrispInput;
    MsfDegree = MemberDegree(InputCrisp);
    return *this;
};

//-----
double Fuzzy::MemberDegree(const double &CrispInput)
// find the Degree of Membership y from crisp input x
{
    double y;

    switch (MemberFuncType)
    {
        case ftTriangle: // left, mean, right = MsfParas[1,2,3]
            // outside the triangle ==> Degree = 0
            if ((CrispInput < MsfParas[1]) ||
                (CrispInput > MsfParas[3]))
                y = 0;

            if (MsfParas[2] > CrispInput) // line is Up
            {
                if ((MsfParas[2]-MsfParas[1]) != 0)
                    y = (CrispInput - MsfParas[1])/(MsfParas[2]-
MsfParas[1]);
            }
            else if (MsfParas[2] < CrispInput) // line is Down
            {
                if ((MsfParas[3]-MsfParas[2]) != 0)
                    y = (MsfParas[3] - CrispInput)/(MsfParas[3]-
MsfParas[2]);
            }
            else // input = mean of triangle
                y = 1;
            break;

        case ftTrapzoidal: // LowLeft, HighLeft,HighRight,LowRight =
MsfParas[1,2,3,4]
            // outside the trap. ==> Degree = 0
            if ((CrispInput <= MsfParas[1]) ||
                (CrispInput >= MsfParas[4]))
                y = 0;

            if ((MsfParas[2] <= CrispInput) &&
                (CrispInput <= MsfParas[3]))
                y = 1;

            if (MsfParas[2] > CrispInput) // line is Up
            {
                if ((MsfParas[2]-MsfParas[1]) != 0)
```

```

        y = (CrispInput - MsfParas[1])/(MsfParas[2]-
MsfParas[1]);
    }
    else if (MsfParas[3] < CrispInput) // line is Down
    {
        if ((MsfParas[4]-MsfParas[3]) != 0)
            y = (MsfParas[3] - CrispInput)/(MsfParas[4]-
MsfParas[3]);
    }
    break;
default:
    // Error Code
    break;
};
if (y < 0) y = 0;
if (y > 1) y = 1;
return y;
}

//-----
double Fuzzy::Defuzzification()
{
    double x;
    double numerator = 0.0;
    double denominator = 0.0;

    for (int i = 0; i < NoOfPoints; i++)
    {
        x = RangeLowerLimit + i*StepSize;
        numerator = numerator + x*MsfValues[i];
        denominator = denominator + MsfValues[i];
    }

    if (denominator != 0)
        return (numerator/denominator);
    else
        return 0.0;
}

//=====
//----- DEFINE THE CONTROLLER AND FIND OUTPUT -----
//----- in class FuzzyController -----
//=====
FuzzyController::FuzzyController(int NumberOfVariables,
                                double VarRange,
                                double VarRateRange,
                                double ResultRange)
{
    SetController(NumberOfVariables, VarRange,
                  VarRateRange, ResultRange);
};

//-----
FuzzyController::~FuzzyController()
{
};

//-----
void FuzzyController::SetController(int NumberOfVariables,
                                    double VarRange,

```

```

        double VarRateRange,
        double ResultRange)
    {
        Status          = 0;
        NumberOfVars    = NumberOfVariables;
        VarLimit        = VarRange;
        VarRateLimit    = VarRateRange;
        ResultLimit     = ResultRange;

// initialize the membership function
        switch (NumberOfVars)
        {
            case 5:
                SetController5(VarLimit, VarRateLimit,
ResultLimit);
                break;
            case 7:
                SetController7(VarLimit, VarRateLimit,
ResultLimit);
                break;
            case 337:
                SetController337(VarLimit, VarRateLimit,
ResultLimit);
                break;
            default:
                break;
        };
        return;
    };

//-----
double FuzzyController::CrispOutput(const double &Variable,
                                   const double &VariableRate)
{
    switch (NumberOfVars)
    {
        case 5:
            return(CrispOutput5(Variable, VariableRate));
        case 7:
            return(CrispOutput7(Variable, VariableRate));
        default:
            return 0.0;
    };
}

//-----
void FuzzyController::SetController5(double VarLimit,
                                     double VarRateLimit,
                                     double ResultLimit)
{
    // Define membership function for Result (Delta U)
    // The ARRAY MsfValues[] of each var. is filled in this step
    // and should be kept unchanged
    double Step = ResultLimit/5.0;
    NB_Re = CreateNewFuzzy(-ResultLimit, ResultLimit,
                           ftTrapezoidal, -INFINITY, -INFINITY,
                           -4*Step, -2*Step);
    NS_Re = CreateNewFuzzy(-ResultLimit,

```

```

ftTriangle,
    Z_Re = CreateNewFuzzy(-ResultLimit,
ResultLimit,
-4*Step, -2*Step, 0);
ftTriangle,
    PS_Re = CreateNewFuzzy(-ResultLimit,
ResultLimit,
-3*Step, 0, 3*Step);
ftTriangle,
    PB_Re = CreateNewFuzzy(-ResultLimit,
ResultLimit,
0, 2*Step, 4*Step);
ftTrapzoidal,
    INFINITY, INFINITY);
    2*Step, 4*Step,

//Define membership function for Variable
Step = VarLimit/5.0;
NB_V = CreateNewFuzzy(-VarLimit,
ftTrapzoidal,
    VarLimit,
    -INFINITY, -INFINITY,
    -4*Step, -3*Step);
    NS_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    -4*Step, -2*Step, 0);
    Z_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    -2*Step, 0, 2*Step);
    PS_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    0, 2*Step, 4*Step);
    PB_V = CreateNewFuzzy(-VarLimit,
ftTrapzoidal,
    VarLimit,
    3*Step, 4*Step,
    INFINITY, INFINITY);

//Define membership function for Variable Rate
Step = VarRateLimit/5.0;
NB_VR = CreateNewFuzzy(-VarRateLimit,
ftTrapzoidal,
    VarRateLimit,
    -INFINITY, -INFINITY,
    -4*Step, -3*Step);
    NS_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    -4*Step, -2.5*Step, 1*Step);
    Z_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    -2*Step, 0, 2*Step);
    PS_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    1*Step, 2.5*Step, 4*Step);
    PB_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTrapzoidal,
    3*Step, 4*Step,
    INFINITY, INFINITY);

```

```

    Status = 1; // all memberships is set
}

//-----
double FuzzyController::CrispOutput5(const double &Variable,
                                     const double &VariableRate)
// Controller 1 with 5 linguistic variables
// return OutputCrisp
{
    Fuzzy* Output;

// Step 1 : Find Degree of member
// Calculate SINGLE member degree for Variable
NB_V.DegreeOfMember(Variable);
NS_V.DegreeOfMember(Variable);
Z_V.DegreeOfMember(Variable);
PS_V.DegreeOfMember(Variable);
PB_V.DegreeOfMember(Variable);

// Calculate SINGLE member degree for Variable Rate
// don't care about array MsfValues[]
NB_VR.DegreeOfMember(VariableRate);
NS_VR.DegreeOfMember(VariableRate);
Z_VR.DegreeOfMember(VariableRate);
PS_VR.DegreeOfMember(VariableRate);
PB_VR.DegreeOfMember(VariableRate);

// Step 2,3: Cal MF of rules and MF of Output fuzzy
// Find array MsfValues[] of Output (F(y)) by following steps:
// 1. For each rules, find the DOF by operator & (single value
MsfDegree)
// 2. Find Fi(y) by && (DOF and array MsfValues[] of that rule's
output)
// 3. Find F(y) by operator || (arrays of all MsfValues of Fi(y))
// The F(y) is the array MsfValues[] of Output
// Note: Dominant rules will be listed fisrt
Output = new Fuzzy (
    ((Z_V & NB_VR) && NB_Re)
    || ((Z_V & NS_VR) && NS_Re)
    || ((Z_V & Z_VR) && Z_Re)
    || ((Z_V & PS_VR) && PS_Re)
    || ((Z_V & PB_VR) && PB_Re)
    || ((Z_VR & NB_V) && NB_Re)
    || ((Z_VR & NS_V) && NS_Re)
    || ((Z_VR & PS_VR) && PS_Re)
    || ((Z_VR & PB_VR) && PB_Re) // 9 dominant rules

    || ((NB_V & NB_VR) && NB_Re)
    || ((NB_V & NS_VR) && NB_Re)
    || ((NB_V & PS_VR) && Z_Re)
    || ((NB_V & PB_VR) && Z_Re)

    || ((NS_V & NB_VR) && NB_Re)
    || ((NS_V & NS_VR) && NS_Re)
    || ((NS_V & PS_VR) && Z_Re)
    || ((NS_V & PB_VR) && PS_Re)

    || ((PS_V & NB_VR) && NS_Re)
    || ((PS_V & NS_VR) && Z_Re)
    || ((PS_V & PS_VR) && PS_Re)
    || ((PS_V & PB_VR) && PB_Re)

```

```

        ||      ((PB_V & NB_VR) && Z_Re)
        ||      ((PB_V & NS_VR) && Z_Re)
        ||      ((PB_V & PS_VR) && PB_Re)
        ||      ((PB_V & PB_VR) && PB_Re)
    );// end of 25 rules

// Step 4: Defuzzificate to calculate the crisp output of Output fuzzy
OutputCrisp = Output->Defuzzification();
delete Output;
return(OutputCrisp);
}

//-----

void FuzzyController::SetController7(double VarLimit,
                                    double VarRateLimit,
                                    double ResultLimit)
{
    // Define membership function for Result (Delta U)
    // The ARRAY MsfValues[] of each var. is filled in this step
    // and should be kept unchanged
    double Step = ResultLimit/6.0;
    NB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          -INFINITY, -INFINITY,
                          -5*Step, -2*Step);
    NM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -4*Step, -2.5*Step, -1*Step);
    NS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -2*Step, -1*Step, 0);
    Z_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -1*Step, 0, 1*Step);
    PM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          0, 1*Step, 2*Step);
    PS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          1*Step, 2.5*Step, 4*Step);
    PB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          2*Step, 5*Step,
                          INFINITY, INFINITY);

    //Define membership function for Variable
    Step = VarLimit/6.0;
    NB_V = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          -INFINITY, -INFINITY,
                          -5*Step, -2*Step);
    NM_V = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -4*Step, -2.5*Step, -1*Step);
    NS_V = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -2*Step, -1*Step, 0);
    Z_V = CreateNewFuzzy(-ResultLimit,

```

```

        ResultLimit, ftTriangle,
        -1*Step, 0, 1*Step);
PM_V = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        0, 1*Step, 2*Step);
PS_V = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        1*Step, 2.5*Step, 4*Step);
PB_V = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTrapzoidal,
        2*Step, 5*Step,
        INFINITY, INFINITY);

//Define membership function for Variable Rate
Step = VarRateLimit/6.0;
NB_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTrapzoidal,
        -INFINITY, -INFINITY,
        -5*Step, -2*Step);
NM_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        -4*Step, -2.5*Step, -1*Step);
NS_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        -2*Step, -1*Step, 0);
Z_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        -1*Step, 0, 1*Step);
PM_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        0, 1*Step, 2*Step);
PS_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTriangle,
        1*Step, 2.5*Step, 4*Step);
PB_VR = CreateNewFuzzy(-ResultLimit,
        ResultLimit, ftTrapzoidal,
        2*Step, 5*Step,
        INFINITY, INFINITY);

    Status = 1; // all memberships is set
}

//-----
double FuzzyController::CrispOutput7(const double &Variable,
        const double &VariableRate)
// Controller 2 with 7 linguistic variables
// return OutputCrisp
{
    Fuzzy* Output;

// Step 1 : Find Degree of member
// Calculate SINGLE member degree for Variable
NB_V.DegreeOfMember(Variable);
NM_V.DegreeOfMember(Variable);
NS_V.DegreeOfMember(Variable);
Z_V.DegreeOfMember(Variable);
PS_V.DegreeOfMember(Variable);
PM_V.DegreeOfMember(Variable);
PB_V.DegreeOfMember(Variable);

// Calculate SINGLE member degree for Variable Rate

```

```

// don't care about array MsfValues[]
NB_VR.DegreeOfMember(VariableRate);
NM_VR.DegreeOfMember(VariableRate);
NS_VR.DegreeOfMember(VariableRate);
Z_VR.DegreeOfMember(VariableRate);
PS_VR.DegreeOfMember(VariableRate);
PM_VR.DegreeOfMember(VariableRate);
PB_VR.DegreeOfMember(VariableRate);

// Step 2,3: Cal MF of rules and MF of Output fuzzy
// Find array MsfValues[] of Output (F(y)) by following steps:
// 1. For each rules, find the DOF by operator & (single value
MsfDegree)
// 2. Find Fi(y) by && (DOF and array MsfValues[] of that rule's
output)
// 3. Find F(y) by operator || (arrays of all MsfValues of Fi(y))
// The F(y) is the array MsfValues[] of Output
// Note: Dominant rules will be listed fisrt
Output = new Fuzzy (
    ((Z_V & NB_VR) && NB_Re)
    || ((Z_V & NM_VR) && NM_Re)
    || ((Z_V & NS_VR) && NS_Re)
    || ((Z_V & Z_VR) && Z_Re)
    || ((Z_V & PS_VR) && PS_Re)
    || ((Z_V & PM_VR) && PM_Re)
    || ((Z_V & PB_VR) && PB_Re)

    || ((Z_VR & NB_V) && NB_Re)
    || ((Z_VR & NS_V) && NS_Re)
    || ((Z_VR & NM_V) && NM_Re)
    || ((Z_VR & PS_V) && PS_Re)
    || ((Z_VR & PM_V) && PM_Re)
    || ((Z_VR & PB_V) && PB_Re) // 13 dominant rules

    || ((NB_V & NB_VR) && NB_Re)
    || ((NB_V & NM_VR) && NB_Re)
    || ((NB_V & NS_VR) && NB_Re)
    || ((NB_V & PS_VR) && Z_Re)
    || ((NB_V & PM_VR) && PM_Re)
    || ((NB_V & PB_VR) && Z_Re)

    || ((NM_V & NB_VR) && NB_Re)
    || ((NM_V & NM_VR) && NM_Re)
    || ((NM_V & NS_VR) && NS_Re)
    || ((NM_V & PS_VR) && Z_Re)
    || ((NM_V & PM_VR) && Z_Re)
    || ((NM_V & PB_VR) && PB_Re)

    || ((NS_V & NB_VR) && NS_Re)
    || ((NS_V & NM_VR) && NS_Re)
    || ((NS_V & NS_VR) && NS_Re)
    || ((NS_V & PS_VR) && Z_Re)
    || ((NS_V & PM_VR) && PS_Re)
    || ((NS_V & PB_VR) && PS_Re)

    || ((PS_V & NB_VR) && NS_Re)
    || ((PS_V & NM_VR) && NS_Re)
    || ((PS_V & NS_VR) && Z_Re)
    || ((PS_V & PS_VR) && PS_Re)
    || ((PS_V & PM_VR) && PS_Re)
    || ((PS_V & PB_VR) && PS_Re)

```

```

        || ((PM_V & NB_VR) && NB_Re)
        || ((PM_V & NM_VR) && Z_Re)
        || ((PM_V & NS_VR) && Z_Re)
        || ((PM_V & PS_VR) && PS_Re)
        || ((PM_V & PM_VR) && PM_Re)
        || ((PM_V & PB_VR) && PB_Re)
        || ((PB_V & NB_VR) && Z_Re)
        || ((PB_V & NM_VR) && NM_Re)
        || ((PB_V & NS_VR) && Z_Re)
        || ((PB_V & PS_VR) && PB_Re)
        || ((PB_V & PM_VR) && PB_Re)
        || ((PB_V & PB_VR) && PB_Re)
    );// end of 49 rules

// Step 4: Defuzzificate to calculate the crisp output of Output fuzzy
OutputCrisp = Output->Defuzzification();
return(OutputCrisp);
}

//-----
//----- 4:50m, Thursday, 14 Sept 2000 -----
//----- New fuzzy controller definition -----
//--- 2 inputs var with 3 membership func, 1 output with 7 member ----
//-----
void FuzzyController::SetController337(double VarLimit,
                                     double VarRateLimit,
                                     double ResultLimit)
{
    // Define membership function for Result (Delta U)
    // The ARRAY MsfValues[] of each var. is filled in this step
    // and should be kept unchanged
    double Step = ResultLimit/6.0;
    NB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          -INFINITY, -INFINITY,
                          -5*Step, -2*Step);
    NM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -4*Step, -2.5*Step, -1*Step);
    NS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -2*Step, -1*Step, 0);
    Z_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -1*Step, 0, 1*Step);
    PM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          0, 1*Step, 2*Step);
    PS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          1*Step, 2.5*Step, 4*Step);
    PB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          2*Step, 5*Step,
                          INFINITY, INFINITY);

    //Define membership function for Variable
    Step = VarLimit/6.0;

```

```

        NB_V = CreateNewFuzzy(-VarLimit,
                                VarLimit,
ftTrapzoidal,
                                -INFINITY, -INFINITY,
                                -2*Step, 0);

        Z_V = CreateNewFuzzy(-VarLimit,
                                VarLimit, ftTriangle,
                                -2*Step, 0, 2*Step);

        PB_V = CreateNewFuzzy(-VarLimit,
                                VarLimit,
ftTrapzoidal,
                                0, 2*Step,
                                INFINITY, INFINITY);

        //Define membership function for Variable Rate
        Step = VarRateLimit/6.0;
        NB_VR = CreateNewFuzzy(-VarRateLimit,
                                VarRateLimit,
ftTrapzoidal,
                                -INFINITY, -INFINITY,
                                -3*Step, 0);
        Z_VR = CreateNewFuzzy(-VarRateLimit,
                                VarRateLimit, ftTriangle,
                                -3*Step, 0, 3*Step);
        PB_VR = CreateNewFuzzy(-VarRateLimit,
                                VarRateLimit, ftTrapzoidal,
                                0, 3*Step,
                                INFINITY, INFINITY);

        Status = 1; // all memberships is set
    }

//-----
double FuzzyController::CrispOutput337(const double &Variable,
                                        const double &VariableRate)
// Controller 3 with 3-3-7 linguistic variables
// return OutputCrisp
{
    Fuzzy* Output;

    // Step 1 : Find Degree of member
    // Calculate SINGLE member degree for Variable
    NB_V.DegreeOfMember(Variable);
    Z_V.DegreeOfMember(Variable);
    PB_V.DegreeOfMember(Variable);

    // Calculate SINGLE member degree for Variable Rate
    // don't care about array MsfValues[]

    NB_VR.DegreeOfMember(VariableRate);
    Z_VR.DegreeOfMember(VariableRate);
    PB_VR.DegreeOfMember(VariableRate);

    // Step 2,3: Cal MF of rules and MF of Output fuzzy
    // Find array MsfValues[] of Output (F(y)) by following steps:
    // 1. For each rules, find the DOF by operator & (single value
MsfDegree)
    // 2. Find Fi(y) by && (DOF and array MsfValues[] of that rule's
output)
    // 3. Find F(y) by operator || (arrays of all MsfValues of Fi(y))
    // The F(y) is the array MsfValues[] of Output

```

```

// Note: Dominant rules will be listed first
Output = new Fuzzy (
    ((NB_V & NB_VR) && NB_Re)
    || ((NB_V & Z_VR) && NM_Re)
    || ((NB_V & PB_VR) && NS_Re)
    || ((Z_V & NB_VR) && NS_Re)
    || ((Z_V & Z_VR) && Z_Re)
    || ((Z_V & PB_VR) && PS_Re)
    || ((PB_V & NB_VR) && PS_Re)
    || ((PB_V & Z_VR) && PM_Re)
    || ((PB_V & PB_VR) && PB_Re)
); // end of 9 rules

// Step 4: Defuzzificate to calculate the crisp output of Output fuzzy
OutputCrisp = Output->Defuzzification();
delete Output;
return(OutputCrisp);
}

//-----
//----- 5:15pm, Thursday, 14 Sept 2000 -----
//----- New fuzzy controller definition -----
//--- 2 inputs var with 5 membership func, 1 output with 7 member ---
//-----
void FuzzyController::SetController557(double VarLimit,
                                     double VarRateLimit,
                                     double ResultLimit)
{
    // Define membership function for Result (Delta U)
    // The ARRAY MsfValues[] of each var. is filled in this step
    // and should be kept unchanged
    double Step = ResultLimit/6.0;
    NB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          -INFINITY, -INFINITY,
                          -5*Step, -2*Step);
    NM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -4*Step, -2.5*Step, -1*Step);
    NS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -2*Step, -1*Step, 0);
    Z_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          -1*Step, 0, 1*Step);
    PM_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          0, 1*Step, 2*Step);
    PS_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTriangle,
                          1*Step, 2.5*Step, 4*Step);
    PB_Re = CreateNewFuzzy(-ResultLimit,
                          ResultLimit, ftTrapezoidal,
                          2*Step, 5*Step,
                          INFINITY, INFINITY);

    //Define membership function for Variable
    Step = VarLimit/6.0;
    NB_V = CreateNewFuzzy(-VarLimit,

```

```

ftTrapzoidal,
    VarLimit,
    -INFINITY, -INFINITY,
    -4*Step, 0);
    NS_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    -4*Step, -2*Step, 0);
    Z_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    -1*Step, 0, 1*Step);
    PS_V = CreateNewFuzzy(-VarLimit,
    VarLimit, ftTriangle,
    0, 2*Step, 4*Step);
    PB_V = CreateNewFuzzy(-VarLimit,
    VarLimit,
ftTrapzoidal,
    0, 4*Step,
    INFINITY, INFINITY);

    //Define membership function for Variable Rate
    Step = VarRateLimit/6.0;
    NB_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit,
ftTrapzoidal,
    -INFINITY, -INFINITY,
    -6*Step, -3*Step);
    NS_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    -6*Step, -3*Step, 0);
    Z_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    -3*Step, 0, 3*Step);
    PS_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTriangle,
    0, 3*Step, 6*Step);
    PB_VR = CreateNewFuzzy(-VarRateLimit,
    VarRateLimit, ftTrapzoidal,
    3*Step, 6*Step,
    INFINITY, INFINITY);

    Status = 1; // all memberships is set
}

//-----
double FuzzyController::CrispOutput557(const double &Variable,
    const double &VariableRate)
// Controller 3 with 5-5-7 linguistic variables
// return OutputCrisp
{
    Fuzzy* Output;

    // Step 1 : Find Degree of member
    // Calculate SINGLE member degree for Variable
    NB_V.DegreeOfMember(Variable);
    NS_V.DegreeOfMember(Variable);
    Z_V.DegreeOfMember(Variable);
    PS_V.DegreeOfMember(Variable);
    PB_V.DegreeOfMember(Variable);

    // Calculate SINGLE member degree for Variable Rate
    // don't care about array MsfValues[]

```

```

NB_VR.DegreeOfMember(VariableRate);
NS_VR.DegreeOfMember(VariableRate);
Z_VR.DegreeOfMember(VariableRate);
PS_VR.DegreeOfMember(VariableRate);
PB_VR.DegreeOfMember(VariableRate);

// Step 2,3: Cal MF of rules and MF of Output fuzzy
// Find array MsfValues[] of Output (F(y)) by following steps:
// 1. For each rules, find the DOF by operator & (single value
MsfDegree)
// 2. Find Fi(y) by && (DOF and array MsfValues[] of that rule's
output)
// 3. Find F(y) by operator || (arrays of all MsfValues of Fi(y))
// The F(y) is the array MsfValues[] of Output
// Note: Dominant rules will be listed fisrt
Output = new Fuzzy (
    ((NB_V & NB_VR) && NB_Re)
    || ((NB_V & NS_VR) && NB_Re)
    || ((NB_V & Z_VR) && NS_Re)
    || ((NB_V & PS_VR) && Z_Re)
    || ((NB_V & PB_VR) && Z_Re)

    || ((NS_V & NB_VR) && NB_Re)
    || ((NS_V & NS_VR) && NB_Re)
    || ((NS_V & Z_VR) && NS_Re)
    || ((NS_V & PS_VR) && PS_Re)
    || ((NS_V & PB_VR) && PS_Re)

    || ((Z_V & NB_VR) && NM_Re)
    || ((Z_V & NS_VR) && Z_Re)
    || ((Z_V & Z_VR) && Z_Re)
    || ((Z_V & PS_VR) && PS_Re)
    || ((Z_V & PB_VR) && PM_Re)

    || ((PS_V & NB_VR) && NS_Re)
    || ((PS_V & NS_VR) && Z_Re)
    || ((PS_V & Z_VR) && PS_Re)
    || ((PS_V & PS_VR) && PB_Re)
    || ((PS_V & PB_VR) && PB_Re)

    || ((PB_V & NB_VR) && NS_Re)
    || ((PB_V & NS_VR) && Z_Re)
    || ((PB_V & Z_VR) && PS_Re)
    || ((PB_V & PS_VR) && PB_Re)
    || ((PB_V & PB_VR) && PB_Re)
); // end of 25 rules

// Step 4: Defuzzificate to calculate the crisp output of Output fuzzy
OutputCrisp = Output->Defuzzification();
delete Output;
return(OutputCrisp);
}

```

B.2 Full FASMC software

```

////////////////////////////////////
////////////////////////////////////
// Header //
////////////////////////////////////
// Standard header
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
// #include <graphics.h>
#include <iomanip.h>
#include <process.h>
#include <dos.h>

#include "Fuzzy.h"

// External declaration header
extern "C"
{
#include "pc30io.h"
#include "pcl833.h"
};

////////////////////////////////////
////////////////////////////////////
// Global constant, File, Array & Struct declarations //
////////////////////////////////////
const double sample_time=0.02;
const int dimension=5;
float limit=0.6,ver_boundary=0.1,hor_boundary=0.25;
double hor_lamda=1.2,ver_lamda=1.5;
double vmax,vmin;

FuzzyController Controller5V(5,2,8,1), Controller5H(5,2,8,1);
FuzzyController Controller7V(7,2,8,1), Controller7H(5,2,8,1);
FuzzyController Controller337V(5,2,8,1), Controller337H(5,2,8,1);
FuzzyController Controller557V(5,2,8,1), Controller557H(5,2,8,1);

//ver_boundary=0.1,hor_boundary=0.25;
//hor_lamda=1.0,ver_lamda=1.5;

ofstream reference("outref.dat",ios::out);
ofstream controller_output("outu.dat",ios::out);
ofstream sliding_variable("outs.dat",ios::out);
ofstream D_sliding_variable("outs_d.dat",ios::out);
ofstream error("outerr.dat",ios::out);
ofstream position("outy.dat",ios::out);

ofstream vreference("voutref.dat",ios::out);
ofstream vcontroller_output("voutu.dat",ios::out);
ofstream vsliding_variable("vouts.dat",ios::out);
ofstream D_vsliding_variable("vouts_d.dat",ios::out);
ofstream verror("vouterr.dat",ios::out);
ofstream vposition("vouty.dat",ios::out);

```

```
typedef double matrix[dimension];

struct system_type
{
    matrix r;
    matrix r_dot;
    matrix r_dot_dot;
    matrix y;
    matrix y_dot;
    matrix y_dot_dot;
    double offset;
    double u;
    double theta2_dot;
    double theta2;
    double theta1_dot;
    double theta1;
    double sliding_variable;
    double D_sliding_variable;
    double average;
    int dir;

    double boundary; // NEW
}horizontal, vertical;

////////////////////////////////////
// Prototype functions //
////////////////////////////////////
int main(void);
int exit(void);

system_type initialise(system_type plant);
system_type swap(system_type plant);
system_type hor_controller(system_type plant);
system_type ver_controller(system_type plant);
system_type integrate(system_type plant);
system_type calibrate_vertical(system_type plant);
system_type calibrate_horizontal(system_type plant);
double ReadVertical_InputPot(system_type vertical);
double ReadVertical_OutputPot(system_type vertical);
double derivative(double);
double convert(double);
double ReadHorizontalPot(void);

void Manual_postion(system_type horizontal, system_type vertical, int
axis);
void Manual_motor_output(int axis);

void display_data(system_type hor, system_type ver, int offset);
void display_table(int offset, int mode);
void display_menu(int menu);

void rectangular(system_type hor, system_type ver, int axis);
void triangular (system_type hor, system_type ver, int axis);
void save_data(system_type plant, int axis);
void Change_system_parameters(void);
void controller(system_type horizontal, system_type vertical, int
savedata);
void Manual_Entry(void);
```

```

////////////////////////////////////
////////////////////////////////////
// Initialise members of the struct, system_type //
////////////////////////////////////
system_type initialise(system_type plant)
{

    plant.u = 0.0;
    for(int i = 0; i < dimension; i++)
    {
        plant.r[i] = 0.0;
        plant.r_dot[i] = 0.0;
        plant.r_dot_dot[i] =0.0;
        plant.y[i] = 0.0;
        plant.y_dot[i] = 0.0;
        plant.y_dot_dot[i] = 0.0;
    };

    plant.theta1_dot=0.01;
    plant.theta1=5;
    plant.theta2_dot=0.01;
    plant.theta2=5;
    plant.sliding_variable=0;
    plant.D_sliding_variable=0;

    // Controller5.SetController(5, plant.boundary, 8, 1);
    // Controller7.SetController(7, plant.boundary, 8, 1);

    return plant;
};

////////////////////////////////////
////////////////////////////////////
// Function like a shift register, which store all the history data in
the array //
////////////////////////////////////
system_type swap(system_type plant)
{
    int i;

    for(i = (dimension-1); i > 0; i--)
    {
        plant.r[i] = plant.r[i-1];
        plant.r_dot[i] = plant.r_dot[i-1];
        plant.r_dot_dot[i] = plant.r_dot_dot[i-1];
        plant.y[i] = plant.y[i-1];
        plant.y_dot[i]= plant.y_dot[i-1];
        plant.y_dot_dot[i]= plant.y_dot_dot[i-1];
    };

    return plant;
};

////////////////////////////////////
////////////////////////////////////
// Returns the derivative value base on the input value (Least square
method) //
// i.e. return quantity dot base on quantity
//
////////////////////////////////////
double derivative(double *quantity)
{
    double x, y, xx, xy;

```

```

int i;

x = 0.0;
y = 0.0;
xx = 0.0;
xy = 0.0;

for(i = 0; i < dimension; i++)
{
x += (i+1)*sample_time;
xx += (i+1)*sample_time*(i+1)*sample_time;
y += quantity[i];
xy += (i+1)*sample_time*quantity[i];
}

return -((x*y-xy*dimension)/(x*x-xx*dimension));
};

////////////////////////////////////
// Returns the integrated value of theta1 and theta2 //
// Using runge kutta method //
//
////////////////////////////////////
system_type integrate_theta(system_type plant)
{
float k1,k2,k3,k4;

k1=plant.theta1_dot;
k2=plant.theta1_dot +(0.5*k1*sample_time);
k3=plant.theta1_dot +(0.5*k2*sample_time);
k4=plant.theta1_dot +(0.5*k3*sample_time);

plant.theta1=(sample_time*(k1+(2*k2)+(2*k3)+k4))/6;

k1=plant.theta2_dot;
k2=plant.theta2_dot +(0.5*k1*sample_time);
k3=plant.theta2_dot +(0.5*k2*sample_time);
k4=plant.theta2_dot +(0.5*k3*sample_time);

plant.theta2 =(sample_time*(k1+(2*k2)+(2*k3)+k4))/6;

return plant;
};

////////////////////////////////////
//
// Main controller loop //
////////////////////////////////////
//
void controller(system_type horizontal, system_type vertical, int
savedata)
{
do
{
horizontal.r[0] = double(int(ReadHorizontalPot()*100))/100;
horizontal.y[0] = CounterValueInVoltage(1);
horizontal.y[0] += horizontal.offset;
horizontal.y[0] = horizontal.y[0];
horizontal = hor_controller(horizontal);
}
}

```

```

    dacout(0, horizontal.u);
    horizontal = swap(horizontal);
    if (savedata) save_data(horizontal,1);

    vertical.r[0] =
double(int(ReadVertical_InputPot(vertical)*100))/100;
    vertical.y[0] =
double(int(ReadVertical_OutputPot(vertical)*100))/100;
    vertical = ver_controller(vertical);
    dacout(1, vertical.u);
    vertical = swap(vertical);
    if (savedata) save_data(vertical,0);

    endsmpl();
    display_data(horizontal,vertical,0);

}while(!kbhit());
getch();
}

//////////////////////////////////////
// Horizontal axis controller design based on the derived theory //
//////////////////////////////////////
system_type hor_controller(system_type horizontal)
{
    double r_dot_dot;
    double error, error_dot, sign_s ,error_dot_dot;
    double abs_r_dot_dot, abs_error_dot;

    horizontal.r_dot[0] = derivative(horizontal.r);
    horizontal.y_dot[0] = derivative(horizontal.y);
    horizontal.y_dot_dot[0] = derivative(horizontal.y_dot);
    r_dot_dot = derivative(horizontal.r_dot);
    abs_r_dot_dot= fabs(r_dot_dot);           // absolute value

    error = horizontal.y[0]-horizontal.r[0];
    error_dot = (horizontal.y_dot[0]-horizontal.r_dot[0]);
    error_dot_dot = (horizontal.y_dot_dot[0]-
horizontal.r_dot_dot[0]);
    abs_error_dot=fabs(error_dot);

    horizontal.sliding_variable = error_dot + error * hor_lamda;
    horizontal.D_sliding_variable = error_dot_dot + error_dot *
hor_lamda;
    if (fabs(horizontal.sliding_variable)>limit)
        {
            horizontal.theta1_dot=fabs(horizontal.sliding_variable);
            horizontal.theta2_dot=fabs(horizontal.sliding_variable)*
hor_lamda*abs_error_dot;
            horizontal.theta1= horizontal.theta1 +
horizontal.theta1_dot*sample_time;
            horizontal.theta2= horizontal.theta2 +
horizontal.theta2_dot*sample_time;
        }
    else
        {
            horizontal.theta1 = horizontal.theta1;
            horizontal.theta2 = horizontal.theta2;
        }
}

```

```

    if (horizontal.theta1>20) horizontal.theta1=2.5;
    if (horizontal.theta2>20) horizontal.theta2=2.5;

//-----
    double SlidingScale, D_SlidingScale;
    SlidingScale = 2;
    D_SlidingScale = 1/64;

    if (fabs(horizontal.sliding_variable)>=hor_boundary)
        if (horizontal.sliding_variable<0)
            {
                sign_s = -1;
                deltaU = 0;
            }
        else
            {
                sign_s = 1;
                deltaU = 0;
            }
        else
            {
                sign_s = horizontal.sliding_variable/hor_boundary;
// with Controller5H, smooth but error is a bit large
//          sign_s =
Controller5H.CrispOutput(horizontal.sliding_variable,
//
                horizontal.D_sliding_variable);
// with Controller337H
//          sign_s =
Controller337H.CrispOutput(horizontal.sliding_variable*SlidingScale,
//
                horizontal.D_sliding_variable*D_SlidingScale);
            };

    horizontal.u = -
sign_s*(horizontal.theta1+horizontal.theta2*abs_r_dot_dot
        +horizontal.theta2*hor_lamda*abs_error_dot);
//-----

    if (horizontal.u>18) horizontal.u=18;
    if (horizontal.u<-18) horizontal.u=-18;

    return horizontal;
};

////////////////////////////////////
// Vertical axis controller design base on the derived theory //
////////////////////////////////////
system_type ver_controller(system_type vertical)
{
    double r_dot_dot;
    double error, error_dot, sign_s , error_dot_dot;
    double abs_error_dot, abs_r_dot_dot;

    vertical.r_dot[0] = derivative(vertical.r);
    vertical.y_dot[0] = derivative(vertical.y);
    vertical.y_dot_dot[0] = derivative(vertical.y_dot);
    r_dot_dot = derivative(vertical.r_dot);
    abs_r_dot_dot= fabs(r_dot_dot);

```

```

error = vertical.y[0]-vertical.r[0];
error_dot = (vertical.y_dot[0]-vertical.r_dot[0]);
error_dot_dot = (vertical.y_dot_dot[0]-vertical.r_dot_dot[0]);
abs_error_dot=fabs(error_dot);

vertical.sliding_variable = error_dot + ver_lamda*error;
vertical.D_sliding_variable = error_dot_dot +
ver_lamda*error_dot;

if(fabs(vertical.sliding_variable)>limit)
{
vertical.theta1_dot=fabs(vertical.sliding_variable);

vertical.theta2_dot=fabs(vertical.sliding_variable)*ver_lamda*ab
s_error_dot;
vertical.theta1= (vertical.theta1 +
vertical.theta1_dot*sample_time);
vertical.theta2= (vertical.theta2 +
vertical.theta2_dot*sample_time);
}
else
{
vertical.theta1 = vertical.theta1;
vertical.theta2 = vertical.theta2;
}

if (vertical.theta1>10) vertical.theta1=10;
if (vertical.theta2>10) vertical.theta2=10;

//-----
double SlidingScale, D_SlidingScale;
SlidingScale = 2;
D_SlidingScale = 1/64;

if (fabs(vertical.sliding_variable)>=ver_boundary)
if (vertical.sliding_variable<0)
{
sign_s = -1;
deltaU = 0;
//
}
else
{
sign_s =1;
deltaU = 0;
//
}
else
{
sign_s = vertical.sliding_variable/ver_boundary;
//
deltaU =
Controller5V.CrispOutput(vertical.sliding_variable,
//
vertical.D_sliding_variable);
// with Controller5V, smooth but error is a bit large
//
sign_s =
Controller5V.CrispOutput(vertical.sliding_variable,
//
vertical.D_sliding_variable);
// with Controller337V
//
sign_s =
Controller337V.CrispOutput(vertical.sliding_variable*SlidingScale,

```

```

//          vertical.D_sliding_variable*D_SlidingScale);
        };

        vertical.u = -
sign_s*(vertical.thetal+vertical.theta2*abs_r_dot_dot

        +vertical.theta2*ver_lamda*abs_error_dot);
//-----

        if (vertical.u>18) vertical.u=18;
        if (vertical.u<-18) vertical.u=-18;

        return vertical;
    };

////////////////////////////////////
////
// Sets up a display table according to the current operation Mode
//
////////////////////////////////////
////
void display_table(int offset,int mode)
{
    clrscr();

    gotoxy(20,1+offset);
    if (mode==1)
    {
        cout<<"Current MODE : External Joystick Control"<<endl;
        gotoxy(40,23);
        cout<<"Press any key to return to main menu";
    }
    if (mode==2)        cout<<"Current MODE : Manual Motor Output
Entry"<<endl;
    if (mode==3)        cout<<"Current MODE : Manual Position
Entry"<<endl;
    if (mode==4)        cout<<"Current MODE : Data-Logging"<<endl;

    offset+=2;
    gotoxy(31,2+offset);
    cout<<"Horizontal Axis"<<endl;
    gotoxy(61,2+offset);
    cout<<"Vertical Axis"<<endl;

    gotoxy(3,4+offset);
    cout<<"Reference Position";
    gotoxy(3,6+offset);
    cout<<"Axis Position  ";
    gotoxy(3,8+offset);
    cout<<"Error          ";

    gotoxy(3,10+offset);
    cout<<"Thetal          ";
    gotoxy(3,12+offset);
    cout<<"Theta2          ";

    gotoxy(3,14+offset);
    cout<<"Sliding Variable  ";
    gotoxy(3,16+offset);
    cout<<"Controller Output  ";
}

```

```

    for (int i=1;i<17;i++)
    {
        gotoxy(23,i+offset);
        cout<<"|";
        gotoxy(53,i+offset);
        cout<<"|";
        gotoxy(1,i+offset);
        cout<<"|";
        gotoxy(80,i+offset);
        cout<<"|";
    }

    for(i=1;i<81;i++)
    {
        gotoxy(i,1+offset);
        cout<<"-";
        gotoxy(i,3+offset);
        cout<<"-";
        gotoxy(i,9+offset);
        cout<<"-";
        gotoxy(i,13+offset);
        cout<<"-";
        gotoxy(i,17+offset);
        cout<<"-";
    }

    gotoxy(1,3+offset);
    cout<<"|";
    gotoxy(1,9+offset);
    cout<<"|";
    gotoxy(1,13+offset);
    cout<<"|";
    gotoxy(80,3+offset);
    cout<<"|";
    gotoxy(80,9+offset);
    cout<<"|";
    gotoxy(80,13+offset);
    cout<<"|";
}

////////////////////////////////////
// Displays the data of the controller onto prepared table
//
////////////////////////////////////
void display_data(system_type hor, system_type ver,int offset)
{
    //horizontal controller
    offset+=2;
    gotoxy(34,4+offset);
    cout<<hor.r[0];
    gotoxy(34,6+offset);
    cout<<hor.y[0];
    gotoxy(34,8+offset);
    cout<<hor.r[0]-hor.y[0];
    gotoxy(34,10+offset);
    cout<<setw(5)<<hor.thetal;
    gotoxy(34,12+offset);
}

```

```
    cout<<setw(5)<<hor.theta2;
    gotoxy(34,14+offset);
    cout<<hor.sliding_variable<<"\t\t";
    gotoxy(34,16+offset);
    cout<<hor.u<<"\t\t";

//vertical controller
    gotoxy(64,4+offset);
    cout<<ver.r[0];
    gotoxy(64,6+offset);
    cout<<ver.y[0];
    gotoxy(64,8+offset);
    cout<<(ver.y[0]-ver.r[0]);
    gotoxy(64,10+offset);
    cout<<setw(5)<<ver.theta1;
    gotoxy(64,12+offset);
    cout<<setw(5)<<ver.theta2;
    gotoxy(64,14+offset);
    cout<<ver.sliding_variable<<"\t\t";
    gotoxy(64,16+offset);
    cout<<ver.u<<"\t\t";
}

////////////////////////////////////
// VIP sub-routine in program.
//
// Keeps track of the potentiometer cross-overs, and monitors if the
//
// pot is in the transition zone. Returns appropriate output to cont.
//
////////////////////////////////////
double ReadHorizontalPot(void)
{
    double Current_Reading; //=0
    int    Direction=0;
    static double LastValidReading=2.5;
    static int In_Transition=0,revs=0;

    Current_Reading =(5-rdadc(15));    //refer pc30io.C

    if (LastValidReading<0.08 && Current_Reading>0.2)
        {
            In_Transition=1;
            Direction=-1;
        }

    if (LastValidReading>5.04 && Current_Reading<4.9)
        {
            In_Transition=1;
            Direction=1;
        }

    if (In_Transition)
        {
            if (LastValidReading<0.08 && Current_Reading>5)
                In_Transition=0;
            if (LastValidReading>5.04 && Current_Reading<0.1)
                In_Transition=0;
        }
}
```

```

    if (!In_Transition && Direction==-1) revs--;
    if (!In_Transition && Direction==1) revs++;

    if (!In_Transition) LastValidReading=Current_Reading;
    return LastValidReading+(revs*5.05)+(revs*0.08);
};

////////////////////////////////////
// Returns the corrected reference position of vertical joystick
//
////////////////////////////////////
double ReadVertical_InputPot(system_type vertical)
{
    // endsmpl();
    return vertical.dir*(rdadc(14)-vertical.average);
};

////////////////////////////////////
// Returns the corrected position of vertical beam
//
////////////////////////////////////
double ReadVertical_OutputPot(system_type vertical)
{
    // endsmpl();
    return rdadc(13)-vertical.average;
};

////////////////////////////////////
// Calibrates the horizontal axis
//
////////////////////////////////////
system_type calibrate_horizontal(system_type horizontal)
{
    clrscr();
    gotoxy(1,5);
    cout<<"-----HORIZONTAL AXIS CALIBRATION-----";
    -----";
    gotoxy(7,10);
    cout<<"The current position of the horizontal input is "
        <<(5-rdadc(15))*72<<" degrees";
    horizontal.offset=(5-rdadc(15));
    gotoxy(7,12);
    cout<<"Please move the beam to the same location as the input";
    gotoxy(20,15);
    cout<<"Press any key to continue";
    getch();

    return horizontal;
};

////////////////////////////////////
// Calibrates the vertical axis
//
////////////////////////////////////
system_type calibrate_vertical(system_type vertical)
{
    char ans;
    double min,max;
    int check_okay=0;

    do

```

```
{
clrscr();
gotoxy(1,5);
cout<<"-----VERTICAL AXIS CALIBRATION-----"
-----";
gotoxy(4,7);
cout<<"Move Vertical Axis to uppermost position/Highest Value";
do
{
max = rdadc(14);
gotoxy(15,9);
cout<<"Current Position : "<<max;
gotoxy(18,11);
cout<<"Press any key when done";
}
while (!kbhit());
getch();

gotoxy(4,15);
cout<<"Move Vertical Axis to lowermost position/Lowest Value";
do
{
min = rdadc(14);
gotoxy(15,17);
cout<<"Current Position : "<<min;
gotoxy(18,19);
cout<<"Press any key when done";
}
while (!kbhit());
getch();
if (min<max)
check_okay=1;
else
{
clrscr();
gotoxy(15,14);
cout<<"Initialisation Failed, Max > Min, please try
again";
gotoxy(18,16);
cout<<"Press any key when done";
getch();
}
}while (!check_okay);

vertical.average=(max+min)/2;
vmax = max-vertical.average;
vmin = min-vertical.average;

for (long x=0;x<200000;x++)
dacout(1, 2.5);
dacout(1,0);

gotoxy(5,21);
cout<<"Is the vertical axis at the same location? [Y/N]:";
ans=getch();

if (ans=='Y' || ans=='y') vertical.dir=-1; else vertical.dir=1;

return vertical;
```

```

};

/////////////////////////////////////////////////////////////////
// Provides an internal triangular reference signal //
/////////////////////////////////////////////////////////////////
void triangular(system_type horizontal, system_type vertical, int
hor_axis)
{

    display_table(0,4);
    gotoxy(15,22);
    cout<<"Data Logging with Triangular Waveform on ";
    if (hor_axis) cout<<"Horizontal Axis....";
    else cout<<"Vertical Axis...";
    vertical.r[0]=0;
    horizontal.r[0]=1;

    for (int i=0; i<1550;i++)

    {
        //set 1.65 and 0.15
        horizontal.r[0]=2*sin(i*sample_time*0.10)+2;
        vertical.r[0]=0.5*sin(i*sample_time*0.2);
        //set 2 0.22
    /*
        if (i>3200)
        {
            horizontal.r[0] +=0.005;
            vertical.r[0]+=0.0008;
        }
        if (i>2400 && i<=3200)
        {
            horizontal.r[0] -= 0.005;
            vertical.r[0]-=0.0008;
        }
        if (i>1600 && i<=2400)
        {
            horizontal.r[0] +=0.005;
            vertical.r[0]+=0.0008;
        }
        if (i>800 && i<=1600)
        {
            horizontal.r[0] -=0.005;
            vertical.r[0]+=0.0008;
        }
        if (i<=800)
        {
            horizontal.r[0]+=0.005;
            vertical.r[0]-=0.0008;
        }
    */

        if(hor_axis)
        {
            horizontal.y[0] = CounterValueInVoltage(1);
            horizontal.y[0] += horizontal.offset;
            horizontal = hor_controller(horizontal);
            dacout(0, horizontal.u);
            horizontal = swap(horizontal);
            endsmpl(); // refer pc30io.C
            display_data(horizontal,vertical,0);
            save_data(horizontal,1);
        }
        else

```

```

        {
            vertical.y[0] =
ReadVertical_OutputPot(vertical);//rdadc(13);
            vertical = ver_controller(vertical);
            dacout(1, vertical.u); //refer pc30io.C
            vertical = swap(vertical);
            display_data(horizontal,vertical,0);
            endsmpl();
            save_data(vertical,0);
        }
    }

};

////////////////////////////////////
// Provides an internal rectangular reference signal //
////////////////////////////////////
void rectangular(system_type horizontal, system_type vertical, int
hor_axis)
{

    display_table(0,4);
    gotoxy(15,22);
    cout<<"Data Logging with Rectangular Waveform on ";
    if (hor_axis) cout<<"Horizontal Axis....";
    else cout<<"Vertical Axis...";
    for (int i=0; i<4000;i++)

    {
        if (i>3000)
        {
            horizontal.r[0] = 1;
            vertical.r[0] = -0.3; // vertical axis cannot move
to 0.4 value
        }
        if (i>2000 && i<=3000)
        {
            horizontal.r[0] = 3;
            vertical.r[0] = +0.3;
        }
        if (i>1000 && i<=2000)
        {
            horizontal.r[0] = 1;
            vertical.r[0] = -0.3;
        }
        if (i<=1000)
        {
            horizontal.r[0] = 3;
            vertical.r[0] = +0.3;
        }
    }
    if(hor_axis)
    {
        horizontal.y[0] = CounterValueInVoltage(1);
        horizontal.y[0] += horizontal.offset;
        horizontal = hor_controller(horizontal);
        dacout(0, horizontal.u);
        horizontal = swap(horizontal);
        endsmpl(); // refer pc30io.C
        display_data(horizontal,vertical,0);
        save_data(horizontal,1);
    }
}

```

```

        else
        {
            vertical.y[0] =
ReadVertical_OutputPot(vertical);//rdadc(13);
            vertical = ver_controller(vertical);
            dacout(1, vertical.u); //refer pc30io.C
            vertical = swap(vertical);
            display_data(horizontal,vertical,0);
            endsmpl();
            save_data(vertical,0);
        }
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Enables user-specified motor voltage output                                     //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Manual_motor_output(int hor_axis)
{
    float voltage;

    display_table(0,2);
    display_data(horizontal,vertical,0);
    gotoxy(15,21);
    if (hor_axis)
        cout<<"Enter horizontal motor output (-18 to +18 volts):
";
    else cout<<"Enter vertical motor output (-18 to +18 volts): ";
    cin>>voltage;
    if (voltage<-18 || voltage >18)
    {
        cout<<"\n\t\t Error. Value outside limits!!";
        cout<<"\n\t\t\t\t Press any key to return to sub-menu.";
        getch();
        return;
    }

    do
    {
        if (hor_axis) dacout(0,voltage); else dacout(1,voltage);
        horizontal.r[0] = ReadHorizontalPot();
        horizontal.y[0] = CounterValueInVoltage(1);
        horizontal.y[0] += horizontal.offset;
        vertical.r[0] = ReadVertical_InputPot(vertical);
        vertical.y[0] = ReadVertical_OutputPot(vertical);//rdadc(13);
        display_data(horizontal,vertical,0);
        endsmpl();
    }while(!kbhit());
    getch();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Moves beam to user-specified location using keyboard entry                   //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void Manual_position(system_type horizontal, system_type vertical,int
hor_axis)
{
    double position;

    display_table(0,3);
    display_data(horizontal,vertical,0);

```

```

if (hor_axis)
{
    gotoxy(15,21);
    cout<<"Enter horizontal position (0 to 360 degrees): ";
    cin>>position;
    position=(position/360)*5;
    if (position<0 ||position >5)
    {
        cout<<"\n\t\t Error. Value outside limits!!";
        cout<<"\n\t\t\t\t\t Press any key to return to sub-
menu.";
        getch();
        return;
    }
    else horizontal.r[0]=position;
}
else
{
    gotoxy(1,21);
    cout<<"\t\tMax Vertical Position : "<<(vmax/5)*360<<endl;
    cout<<"\t\tMin Vertical Position : "<<(vmin/5)*360<<endl;
    cout<<"\n\t\tEnter vertical position : ";
    cin>>position;
    position=(position/360)*5;          //convert to volts
    if (position<vmin || position >vmax)
    {
        cout<<"\n\t\t Error. Value outside limits!!";
        cout<<"\n\t\t\t\t\t Press any key to return to sub-
menu.";
        getch();
        return;
    }
    else vertical.r[0]=position;
}

display_table(0,3);

do
{
    if (!hor_axis)    horizontal.r[0] = ReadHorizontalPot();
    horizontal.y[0] = CounterValueInVoltage(1);
    horizontal.y[0] += horizontal.offset;

    horizontal = hor_controller(horizontal);
    dacout(0, horizontal.u);
    horizontal = swap(horizontal);

    if (hor_axis)    vertical.r[0] =
ReadVertical_InputPot(vertical);
    vertical.y[0] =
ReadVertical_OutputPot(vertical);//rdadc(13);
    vertical = ver_controller(vertical);
    dacout(1, vertical.u); //refer pc30io.C
    vertical = swap(vertical);

    display_data(horizontal,vertical,0);

}while(!kbhit());
getch();
}

```



```

    }
}

////////////////////////////////////
// Saves controller data to respective files //
////////////////////////////////////
void save_data(system_type plant, int horizontal)
{
    if (horizontal)
    {
        reference<<plant.r[0]<<endl;
        controller_output<<plant.u<<endl;
        sliding_variable<<plant.sliding_variable<<endl;
        D_sliding_variable<<plant.D_sliding_variable<<endl;
        error<<(plant.r[0]-plant.y[0])<<endl;
        position<<plant.y[0]<<endl;
    }

    if (!horizontal)
    {
        vreference<<plant.r[0]<<endl;
        vcontroller_output<<plant.u<<endl;
        vsliding_variable<<plant.sliding_variable<<endl;
        D_vsliding_variable<<plant.D_sliding_variable<<endl;
        verror<<(plant.r[0]-plant.y[0])<<endl;
        vposition<<plant.y[0]<<endl;
    }
}

////////////////////////////////////
// Manual Entry Loop (menu) //
////////////////////////////////////
void Manual_Entry(void)
{
    int choice,e;

    do
    {
        display_menu(3);
        choice=getch();

        switch(choice)
        {
            case '1': Manual_position(horizontal,vertical,1);
                      dacout(0,0);
                      dacout(1,0);
                      break;

            case '2': Manual_motor_output(1);
                      dacout(0,0);
                      dacout(1,0);
                      break;

            case '3': Manual_position(horizontal,vertical,0);
                      dacout(0,0);
                      dacout(1,0);
                      break;
        }
    }
}

```

```

        case '4': Manual_motor_output(0);
                dacout(0,0);
                dacout(1,0);
                break;

        case '5' : e=1;
                break;
    }
}while (choice!='5' || e==0);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Data-Logging Loop (Menu) //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void data_logging(system_type horizontal,system_type vertical)
{
    int choice,e;

    do
    {

        display_menu(2);
        choice=getch();

        switch(choice)
        {
            case '1': rectangular(horizontal,vertical,1);
                    dacout(0,0);
                    dacout(1,0);
                    break;

            case '2': triangular(horizontal,vertical,1);
                    dacout(0,0);
                    dacout(1,0);
                    break;

            case '3': rectangular(horizontal,vertical,0);
                    dacout(0,0);
                    dacout(1,0);
                    break;

            case '4': triangular(horizontal,vertical,0);
                    dacout(0,0);
                    dacout(1,0);
                    break;

            case '5': display_table(0,4);
                    controller(horizontal,vertical,1);
                    dacout(0,0);
                    dacout(1,0);
                    break;

            case '6' : e=1;
                    break;
        }
    }while (choice!='6' || e==0);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```
// Allows user to change system parameters used in the program.
////////////////////////////////////////////////////////////////////
void Change_system_parameters(void)
{
    char ans;

    clrscr();
    cout<<"\n\n\t\tCurrent System Parameters are as follows :";
    cout<<"\n\n\t\tHorizontal Lamda = "<<hor_lamda<<endl;
    cout<<"\t\tVertical Lamda    = "<<ver_lamda<<endl;
    cout<<"\n\n\t\tHorizontal Boundary Layer = "<<hor_boundary<<endl;
    cout<<"\t\tVertical Boundary Layer    = "<<ver_boundary<<endl;
    cout<<"\n\n\t\tChange System parameters [Y/N] : ";
    ans=getch();
    if (ans=='y' || ans=='Y')
    {
        cout<<"\n\n\t\tEnter new System parameters. "<<endl;
        cout<<"\n\n\t\tHorizontal Lamda = ";        cin>>hor_lamda;
        cout<<"\t\tVertical Lamda    = ";        cin>>ver_lamda;
        cout<<"\n\n\t\tHorizontal Boundary Layer =
";cin>>hor_boundary;
        cout<<"\t\tVertical Boundary Layer    = ";
cin>>ver_boundary;
    }
}

////////////////////////////////////////////////////////////////////
// Confirms user wishes to exit program //
////////////////////////////////////////////////////////////////////
int exit(void)
{
    char confirm;

    clrscr();
    gotoxy(15,12);
    cout<<"Confirm that you wish to terminate the program? [Y/N] ";
    confirm=getch();

    if (confirm=='y' || confirm=='Y')
        return 1;

    else return 0;

}

////////////////////////////////////////////////////////////////////
// Main program which provide sequential processes/steps //
////////////////////////////////////////////////////////////////////
int main(void)
{
    int i, PC30ok, e;
    char choice;

//-----
    horizontal.boundary = hor_boundary;
    vertical.boundary = ver_boundary;

Controller5V.SetController(5, vertical.boundary, 6, 1);
Controller7V.SetController(7, vertical.boundary, 8, 1);

```

```
Controller5H.SetController(5, horizontal.boundary, 8, 1);
Controller7H.SetController(7, horizontal.boundary, 8, 1);

Controller337V.SetController(337, vertical.boundary, 6, 1);
Controller337H.SetController(337, horizontal.boundary, 6, 1);

//-----

    cout.precision(3);
    cout.setf(ios::showpos|ios::fixed|ios::showpoint);

    PC30ok=initpc30(sample_time*1000); // refer pc30io.C
    horizontal = initialise(horizontal);
    horizontal = calibrate_horizontal(horizontal);
    PCL833_Init();
    vertical = initialise(vertical);
    vertical = calibrate_vertical(vertical);

    if (PC30ok==0)
    {
        do
        {
            display_menu(1);
            choice=getch();

            switch(choice)
            {
                case '1': horizontal = initialise(horizontal);
                    horizontal =
calibrate_horizontal(horizontal);
                    PCL833_Init();
                    vertical = initialise(vertical);
                    vertical =
calibrate_vertical(vertical);
                    break;

                case '2' : Change_system_parameters();
                    break;

                case '3': data_logging(horizontal,vertical);
                    dacout(0, 0.0);
                    dacout(1, 0.0);
                    break;

                case '4' : Manual_Entry();
                    dacout(0, 0.0);
                    dacout(1, 0.0);
                    break;

                case '5': display_table(0,1);
                    controller (horizontal,vertical,0);
                    dacout(0, 0.0);
                    dacout(1, 0.0);
                    break;

                case '6': e=exit();
            }
        }while (choice !='6' || e==0);
```

```
    }
    else
    {
        cout<<"\n\n\t\tPC30 Card missing or malfunctioned ";
        cout<<"\n\n\t\tPress any key to exit....";
        getch();
    }

    reference.close();
    controller_output.close();
    sliding_variable.close();
    D_sliding_variable.close();
    error.close();
    position.close();

    vreference.close();
    vcontroller_output.close();
    vsliding_variable.close();
    D_vsliding_variable.close();
    verror.close();
    vposition.close();

    dacout(0, 0.0);
    dacout(1, 0.0);
    return 0;
}
```

////////////////////////////////////

B.3 Device driver for PC-30 interface card

The PC-30 driver software is supplied with PC-30 board. It is a set of real time device drivers for use with a wide variety of software. The device driver is written in C (source code). It can support the following languages:

- Microsoft C (version 6.0 or later)
- Microsoft Fortran (version 5 or later)
- Microsoft QuickBasic (version 4.5 or later)
- Microsoft QuickC (version 2 or later)
- TURBO C (version 1.5 or later)
- Borland C++ (version 2.00 or later)
- TURBO PASCAL (version 5.0 or later). The TURBO PASCAL driver does not support interrupts.

Almost all other compiled languages can also be used in conjunction with one of the other object modules. Further details of the software commands and functions can be found in the User Manual (PC-30 Driver Software Package).

The device driver source code (PC30IO.c)

```
#include <stdio.h>
#include <math.h>
#define tcc
#include "pc30.h"

void dacout(int chan,double val)    // send analog voltage to DAC
channel 'chan'
                                     // with value 'val'
{
    int err;
    if (val<-5.0) val=-5.0;
    if (val>5.0) val=5.0;
    val=(5.0-val)/10*4095;           // for 12 bits D/A resolution
    err=da_out(chan,(int)val);
```

```
        if (err!=ok_30) fprintf(stderr,"Pc30-error  Da_out\n");
        return;
    }

void dout(int chan,double val)
{
    d_out(chan,(int)val);
    return;
}

void endsmpl(void)                // wait for say 10ms (do nothing)
                                  // before another sample begin
{
    unsigned old_val,new_val;
    dout(0,0);
    old_val=cntr_read();
    new_val=cntr_read();
    while (new_val<=old_val)
    {
        old_val=new_val;
        new_val=cntr_read();
    }
    dout(0,1);
    return;
}

int initpc30(double wait_samples)    // initialise clock to 200
KHz
{
    #define two_MHz_to_10KHz 200
    #define ten_KHz_to_1KHz 10
    #define pc30_base 0x700

    int err;
    unsigned ws;
    set_base(pc30_base);
    err=diag();
    if (err!=ok_30)
    {
        fprintf(stderr,"Pc30 board not found or bad\n");
        return 1;
    }
    else
    {
        ad_prescaler(two_MHz_to_10KHz);
        ad_clock(ten_KHz_to_1KHz);
        cntr_cfg(2);
        ws=(unsigned) wait_samples;
        cntr_write(ws);
        d_mode(0,0,1);    // set third digital i/o's to input
        d_out(0,1);
        d_out(1,1);
    }
    return 0;
}
```

```
double rdadc(int chan)           // read voltage from ADC channel
'chan'
{
    int    err,in_val;
    float  voltage;
// double voltage;

    err=ad_in(chan,&in_val);
    if (err!=ok_30)
    {
        fprintf(stderr,"Pc30-error rdadc\n");
        voltage=0;
    }
    else voltage=(in_val-2047.5)*10/4095;
    return(voltage);
}

double rdcnt(void)
{
    return ((double)cntr_read());
}
```

B.4 Device driver for PCL833 optical encoder

The PCL-833 driver software has predefined routines that can be set up to control the card. The user specifies an array of values for the card's registers. The driver modifies the registers in the proper order and sends back the data in a second array. To use the driver, a driver function is called with two arguments to be specified. The first holds the name of the driver operation and the second holds any arguments the operation needs. The driver returns an integer value indicating success (0) or failure (non-zero).

The driver is supplied as C source code for Borland and Microsoft compilers. The file *833tc.c* contains the driver source code in Turbo C format and the file *833mc.c* contains the source code in Microsoft C format. The driver arguments are constants defined in the header file *833drive.h*.

B.4.1 The optical encoder device driver (833Drive.c)

```
#include <dos.h>
#include "833drive.h"

int vCh_SetInputMode(int ChannelNo, int option);
int vCh_SetInputMode(int ChannelNo, int option);
int vCh_SetInputMode(int ChannelNo, int option);
int vCh_DefineResetValue(int ChannelNo, int option);
int vCh_DefineResetValue(int ChannelNo, int option);
int vCh_DefineResetValue(int ChannelNo, int option);
int vCh_SetLatchSource(int ChannelNo, int option);
int vCh_SetLatchSource(int ChannelNo, int option);
int vCh_SetLatchSource(int ChannelNo, int option);
int vCh_IfResetOnLatch(int ChannelNo, int option);
int vCh_IfResetOnLatch(int ChannelNo, int option);
int vCh_IfResetOnLatch(int ChannelNo, int option);
int vLatchWhenOverflow(int option);
int vCounterReset(int option);
int vChooseSysClock(int option);
int vSetCascadeMode(int option);
int vSet16C54TimeBase(int option);
int vSetDIorTimerInt(int option);
int vSet16C54Divider(int option);
int vCh_Read(int option);
```



```

        case Initialize833      : return(vInitialize833());
        default                : return(FUNCTION_NUM_ERR);
    }
}

//:
//:
//:

int vInitialize833(){
    int i;

    //Base address must be set by users first
    for(i=0;i<16;i++) OutReg[i]=InReg[i]=0;
    vCh_SetInputMode(ch1, PclDisable);
    vCh_SetInputMode(ch2, PclDisable);
    vCh_SetInputMode(ch3, PclDisable);
    vCh_DefineResetValue(ch1, start);
    vCh_DefineResetValue(ch2, start);
    vCh_DefineResetValue(ch3, start);
    vCh_SetLatchSource(ch1, SwReadLatch);
    vCh_SetLatchSource(ch2, SwReadLatch);
    vCh_SetLatchSource(ch3, SwReadLatch);
    vCh_IfResetOnLatch(ch1, ResetNo);
    vCh_IfResetOnLatch(ch2, ResetNo);
    vCh_IfResetOnLatch(ch3, ResetNo);
    vLatchWhenOverflow(Latch_Ch1);
    vLatchWhenOverflow(Latch_Ch2);
    vLatchWhenOverflow(Latch_Ch3);
    vCounterReset(Reset_Ch1);
    vCounterReset(Reset_Ch2);
    vCounterReset(Reset_Ch3);
    vChooseSysClock(Sys8MHZ);
    vSetCascadeMode(c24bits); //no cascade
    vSet16C54TimeBase(tPoint1ms);
    vSetDI1orTimerInt(TimerInt);
    vSet16C54Divider(0); // 0-255 divider
    return(OK);
}

//:
//:
//:

int vCh_SetInputMode(int ChannelNo ,int option){

    int OutputReg, PortAddress, RegIndex;

    switch(ChannelNo){
        case ch1:  PortAddress= Base + 0;  RegIndex= 0;   break;
        case ch2:  PortAddress= Base + 1;  RegIndex= 1;   break;
        case ch3:  PortAddress= Base + 2;  RegIndex= 2;   break;
        default:   return(CHANNEL_NUM_ERR);
    }

    OutputReg= OutReg[RegIndex] & 0x08;
    switch(option){
        case x1:

```

```
        case x2:
        case x4:
        case PclDisable:
        case TwoPulseIn:
        case OnePulseIn: outportb(PortAddress,OutputReg | option);
break;

        default:          return(PARAMETER_ERR);
    }
    OutReg[RegIndex]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int vCh_DefineResetValue(int ChannelNo ,int option){

    int OutputReg, PortAddress, RegIndex;

    switch(ChannelNo){
        case ch1:  PortAddress= Base + 0;  RegIndex= 0;   break;
        case ch2:  PortAddress= Base + 1;  RegIndex= 1;   break;
        case ch3:  PortAddress= Base + 2;  RegIndex= 2;   break;
        default:   return(CHANNEL_NUM_ERR);
    }

    OutputReg= OutReg[RegIndex] & 0x07;
    switch(option){
        case start:
        case middle:  outportb(PortAddress,OutputReg | option);
break;
        default:      return(PARAMETER_ERR);
    }
    OutReg[RegIndex]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int vCh_SetLatchSource(int ChannelNo ,int option){

    int OutputReg, PortAddress, RegIndex;

    switch(ChannelNo){
        case ch1:  PortAddress= Base + 3;  RegIndex= 3;   break;
        case ch2:  PortAddress= Base + 4;  RegIndex= 4;   break;
        case ch3:  PortAddress= Base + 5;  RegIndex= 5;   break;
        default:   return(CHANNEL_NUM_ERR);
    }

    OutputReg= OutReg[RegIndex] & 0x08;
    switch(option){
        case SwReadLatch:
        case IndexInLatch:
        case DI0Latch:
        case DI1Latch:
```

```

        case TimerLatch:    outportb(PortAddress,OutputReg | option);
                           break;

        default:           return(PARAMETER_ERR);
    }
    OutReg[RegIndex]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int  vCh_IfResetOnLatch(int ChannelNo ,int option){

    int OutputReg, PortAddress, RegIndex;

    switch(ChannelNo){
        case ch1:  PortAddress= Base + 3;  RegIndex= 3;  break;
        case ch2:  PortAddress= Base + 4;  RegIndex= 4;  break;
        case ch3:  PortAddress= Base + 5;  RegIndex= 5;  break;
        default:   return(CHANNEL_NUM_ERR);
    }

    OutputReg= OutReg[RegIndex] & 0x07;
    switch(option){
        case ResetNo:
        case ResetYes:    outportb(PortAddress,OutputReg | option);
    break;

        default:         return(PARAMETER_ERR);
    }
    OutReg[RegIndex]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int  vLatchWhenOverflow(int option){
    switch(option){
        case Latch_Ch1:  OutReg[6] &= 0x06;  break;
        case Latch_Ch2:  OutReg[6] &= 0x05;  break;
        case Latch_Ch3:  OutReg[6] &= 0x03;  break;
        case FreeAll:    OutReg[6] = 0x07;    break;
        default: return(PARAMETER_ERR);
    }
    outportb(Base+6, OutReg[6]);
    return(OK);
}

//:
//:
//:

int  vCounterReset(int option){
    switch(option){
        case Reset_Ch1:  OutReg[7] &= 0x06;
                           OutReg[7] |= 0x01;

```

```
        break;
    case Reset_Ch2:    OutReg[7] &= 0x05;
                      OutReg[7] |= 0x02;
                      break;
    case Reset_Ch3:    OutReg[7] &= 0x03;
                      OutReg[7] |= 0x04;
                      break;
    case NoneReset:    OutReg[7] =0;
                      break;
    default: return(PARAMETER_ERR);
}
outportb(Base+7, OutReg[7]);
return(OK);
}

//:.....
//:
//:.....

int  vChooseSysClock(option){

    int OutputReg;

    OutputReg= OutReg[8] & 0x0c;
    switch(option){
        case Sys8MHZ:
        case Sys4MHZ:
        case Sys2MHZ:  outportb(Base+8, OutputReg | option);  break;

        default:      return(PARAMETER_ERR);
    }
    OutReg[8]= OutputReg | option;
    return(OK);
}

//:.....
//:
//:.....

int  vSetCascadeMode(option){

    int OutputReg;

    OutputReg= OutReg[8] & 0x03;

    switch(option){
        case c24bits:  // no cascade
            outportb(Base+8,OutputReg | option);
            break;

        case c48bits:  // ch1 ch2 cascade
            OutReg[1] |= 0x07; // set ch2 cascade mode
            OutReg[0] &= 0x07; // set ch1 reset value '000000'
            outportb(Base+1,OutReg[1]);
            outportb(Base,OutReg[0]);

            outportb(Base+8,OutputReg | option);
            vCh_SetInputMode(ch2, cascade);
            break;

        default:      return(PARAMETER_ERR);
    }
}
```

```
    }
    OutReg[8]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int vSet16C54TimeBase(option){

    int OutputReg;

    OutputReg= OutReg[9] & 0x08;
    switch(option){
        case tPoint1ms:
        case t1ms:
        case t10ms:
        case t100ms:
        case t1s:    outportb(Base+9, OutputReg | option);    break;

        default:    return(PARAMETER_ERR);
    }
    OutReg[9]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int vSetDI1orTimerInt(option){

    int OutputReg;

    OutputReg= OutReg[9] & 0x07;
    switch(option){
        case DI1Int    :
        case TimerInt : outportb(Base+9, OutputReg | option);    break;

        default:    return(PARAMETER_ERR);
    }
    OutReg[9]= OutputReg | option;
    return(OK);
}

//:
//:
//:

int vSet16C54Divider(option){

    outportb(Base+10, option);
    OutReg[10]= option;
    return(OK);
}

//:
//: READ FUNCTION
```

```
//:.....  
  
int vCh_Read(int ChannelNo){  
  
    switch(ChannelNo){  
  
        case ch1:  
            InReg[2]=inportb(Base + 2);  
            InReg[0]=inportb(Base);  
            InReg[1]=inportb(Base + 1);  
            break;  
  
        case ch2:  
            InReg[6]=inportb(Base + 6);  
            InReg[4]=inportb(Base + 4);  
            InReg[5]=inportb(Base + 5);  
            break;  
  
        case ch3:  
            InReg[10]=inportb(Base + 10);  
            InReg[8]=inportb(Base + 8);  
            InReg[9]=inportb(Base + 9);  
            break;  
  
        default:    return(CHANNEL_NUM_ERR);  
    }  
    return(OK);  
}  
  
//:.....  
//:  
//:.....  
  
int vOverflow_Read(void){  
  
    InReg[3]= inportb(Base + 3);  
    InReg[7]= inportb(Base + 7);  
    InReg[11]= inportb(Base + 11);  
    return(OK);  
}  
  
//:.....  
//:  
//:.....  
  
int vStatus_Read(void){  
  
    InReg[14]= inportb(Base + 14);  
    return(OK);  
}
```

B.4.2 Optical encoder functions for project (PCL833.c)

```
#include "pcl833.h"
#ifdef tcc
#define _inp inp
#endif
#define pi 3.141592653589793
extern int pcl833(int func, int opt);
extern int OutReg[16];
extern int InReg[16];
extern int Base;

int PCL833_Init(void)
{
    Base=0x200;
    if (pcl833(Initialize833,NA)<0) return -1;
    pcl833(Ch1_SetInputMode,x4);
    pcl833(Ch1_DefineResetValue,middle);
    pcl833(Ch1_SetLatchSource,SwReadLatch);
    pcl833(Ch1_IfResetOnLatch,ResetNo);
    pcl833(Ch2_SetInputMode,x4);
    pcl833(Ch2_DefineResetValue,middle);
    pcl833(Ch2_SetLatchSource,SwReadLatch);
    pcl833(Ch2_IfResetOnLatch,ResetNo);
    pcl833(Ch3_SetInputMode,x4);
    pcl833(Ch3_DefineResetValue,middle);
    pcl833(Ch3_SetLatchSource,SwReadLatch);
    pcl833(Ch3_IfResetOnLatch,ResetNo);
    pcl833(LatchWhenOverflow,FreeAll);
    pcl833(CounterReset,Reset_Ch1);
    pcl833(CounterReset,Reset_Ch2);
    pcl833(CounterReset,Reset_Ch3);
    pcl833(ChooseSysClock,Sys2MHZ);
    pcl833(SetCascadeMode,c24bits);

    return 0;
}

int PCL833_CounterMode(int channel, int mode)
{
    // mode 1 -> x1
    // mode 2 -> x2
    // mode 4 -> x4

    int res;

    switch (channel)
    {
    case 1:
        if (mode==1) res=pcl833(Ch1_SetInputMode,x1);
        if (mode==2) res=pcl833(Ch1_SetInputMode,x2);
        if (mode==4) res=pcl833(Ch1_SetInputMode,x4);
        break;
    case 2:
        if (mode==1) res=pcl833(Ch2_SetInputMode,x1);
        if (mode==2) res=pcl833(Ch2_SetInputMode,x2);
```

```
        if (mode==4) res=pcl833(Ch2_SetInputMode,x4);
        break;
    case 3:
        if (mode==1) res=pcl833(Ch3_SetInputMode,x1);
        if (mode==2) res=pcl833(Ch3_SetInputMode,x2);
        if (mode==4) res=pcl833(Ch3_SetInputMode,x4);
        break;
    default:
        return -1;
    }
    return res;
}

unsigned long PCL833_ReadCounter(int channel)
{
    int base = 0x200;
    int addr;
    unsigned long ans,ans1,ans2;

    addr=base+4*(channel-1);

    _inp(addr+2);          // required to latch data successfully
    ans =_inp(addr+2);
    ans1=_inp(addr+1);
    ans2=_inp(addr);

    // printf("Encoder reading: 0x %2X %2X %2X\n",ans,ans1,ans2);
    ans= (ans<<16) + (ans1<<8) + ans2 ;

    return ans;
}

long PCL833_ReadCounterRelative(int channel)
{
    // reads the counter and returns the value it has deviated from
    0x80000
    unsigned long res1;

    res1=PCL833_ReadCounter(channel);
    return (long) ( (long) res1 - 0x800000);
}

long PCL833_GetAngleInt(int channel,int whichway)
{
    // this function returns the angle of the optical encoder.
    // even though the pcl833 encoder's counter is zeroed with the
    // The angle is returned in the
    // range of -999 to 1000, giving a 2000 count range.
    // The variable whichway determines wether a clockwise
    // rotation from the zero position gives a positive or
    // a negative count value. 'whichway=1' gives positive angles
    // for clockwise rotation- of course this is all relative to the
    // actual encoder wiring.

    long count;

    count = PCL833_ReadCounterRelative(channel);
    count=count%2000;          //modulus
    if (count<-999) count=2000+count;          // count now = -999 to
```

```
        if (count>1000) count=-2000+count;          // +1000
                               // straight down.
        if (whichway==0) count=-count;
        return count;
    }

double PCL833_GetEncoderAngle(int channel,int whichway)
{
    long count;
    double radians;

    count=PCL833_GetAngleInt(channel,whichway);
    radians=count*pi/1000.0;
    return radians;
}

double PCL833_CalcFullEncoderAngle(double Theta)
{
    // This procedure keeps track of +/-pi crossings. It requires
    // regular checking of the angle. You could do this with
    // the encoder card directly (it has a 24/48 bit counter which
    // will keep track of many revolutions), but I have chosen to reduce
    // the count value to one revolution and then expand it
    // to keep track full rotations.

    static double LastTheta=0;
    static int FullRotations=0;

    void UpdateFullRevs(int *FullRotations,double old,double neww);

    UpdateFullRevs(&FullRotations,LastTheta,Theta);
    LastTheta=Theta;
    return (Theta+ 2*pi*(double)FullRotations);
}

void UpdateFullRevs(int *FullRotations,double old,double neww)
{
    // check to see if we have crossed the +pi / -pi boundary
    if ((neww>0) && (old<0) && (old<neww-pi)) (*FullRotations)--;
    if ((neww<0) && (old>0) && (old>neww+pi)) (*FullRotations)++;
}

double CounterValueInVoltage (int channel)
{
    // this function will returns a absolute voltage value (0 to 5)
    // for the encoder based on a 2000 count range.

    long v1;
    double v2;

    v1 = PCL833_ReadCounterRelative(channel);
    v2 =-(v1)*0.0025;
    return v2;
}
```

Appendix C: Interfacing card and encoder

C.1 PC-30 interface card

In this project, the PC-30D board is used as the I/O interface between computer and the plant. PC-30D is one of the members of the PC-30 family. These series of boards are full size, low cost, high accuracy analog and digital I/O boards for IBM PC, PC/XT, PC/AT, PS/2 and compatible series of computers. PC-30D is a development of the PC-30C, with A/D throughput of 200 KHz.

C.1.1 A/D subsystem

The A/D subsystem's major component is a monolithic analog to digital converter, which accepts analog voltage inputs from sensors and converts them into 12-bit digital code. This code is transmitted to the host processor, which processes it according to the software in use at the time.

The main features of the A/D subsystem are listed as follows:

- A/D resolution: 12 bits.
- Non-linearity: Less than ± 0.75 LSB.
- A/D full-scale range may be set, by jumper, for uni-polar (0 to 10V) or bipolar ($\pm 5V$ or $\pm 10V$) operation.
- Number of A/D inputs: 16 single ended.
- A/D throughput rate: 200 KHz.

C.1.2 D/A subsystem

Digital outputs are received from the host processor and converted to an analog voltage output required by the application in hand. The four DACs are independent of one another. Their individual features are listed below:

For DAC0 and DAC1

- D/A resolution: 12 bits.
- D/A non-linearity: Within 0.01% full-scale range.
- Full-scale output ranges may be selected, by mini-jumper, for uni-polar (0 to 10V) or bipolar ($\pm 10V$) operation.
- D/A throughput rate: 130 KHz.

For DAC2 and DAC3

- D/A resolution: 8 bits.
- D/A non-linearity: Within 0.38% full-scale range.
- Full-scale output ranges may be selected, by mini-jumper, for unipolar (0 to 10V) or bipolar ($\pm 10V$) operation.
- D/A throughput rate: 130 KHz.

Further details of hardware configurations and specifications can be found in the User Manual for PC-30 B/C/D.

C.2 PCL-833 encoder

In the project, the encoder card allows the PC to perform position monitoring of the plant's horizontal beam position. The PCL-833 is a 3-axis quadrature encoder and counter add-on card for the IBM PC/AT and its compatibles. It provides three 24bit up/down counters as quadrature encoders and a 16-Mhz oscillator time base with wide range multiplier. An on-board interrupt controller handles nine different sources.

Each input includes a decoding circuit for incremental quadrature encoding. Inputs accept either single-ended or differential signals. Quadrature input works with or without index, allowing linear or rotary encoder feedback.

Counters and their modes

The PCL-833 has three independent 24-bit counters. The maximum quadrature input rate is 1.0 MHz, while the maximum input rate on counter mode is 2.4 MHz. Each counter can be individually configured for quadrature encoding, pulse/direction counting or up/down counting. The counters can operate in four modes:

- Quadrature input counter mode: Quadrature input consists of two square wave inputs (A and B), which are 90 out of phase. The PCL-833 counts the square wave transitions and determines the direction by comparing whether channel A is leading channel B or vice-versa.
- 2-pulse mode: In 2-pulse mode the PCL-833 uses two input pulses as counting sources: one for clockwise and the other for counter-clockwise counting. The

counter will decrement whenever a rising edge occurs on channel A and increment whenever a rising edge occurs on channel B.

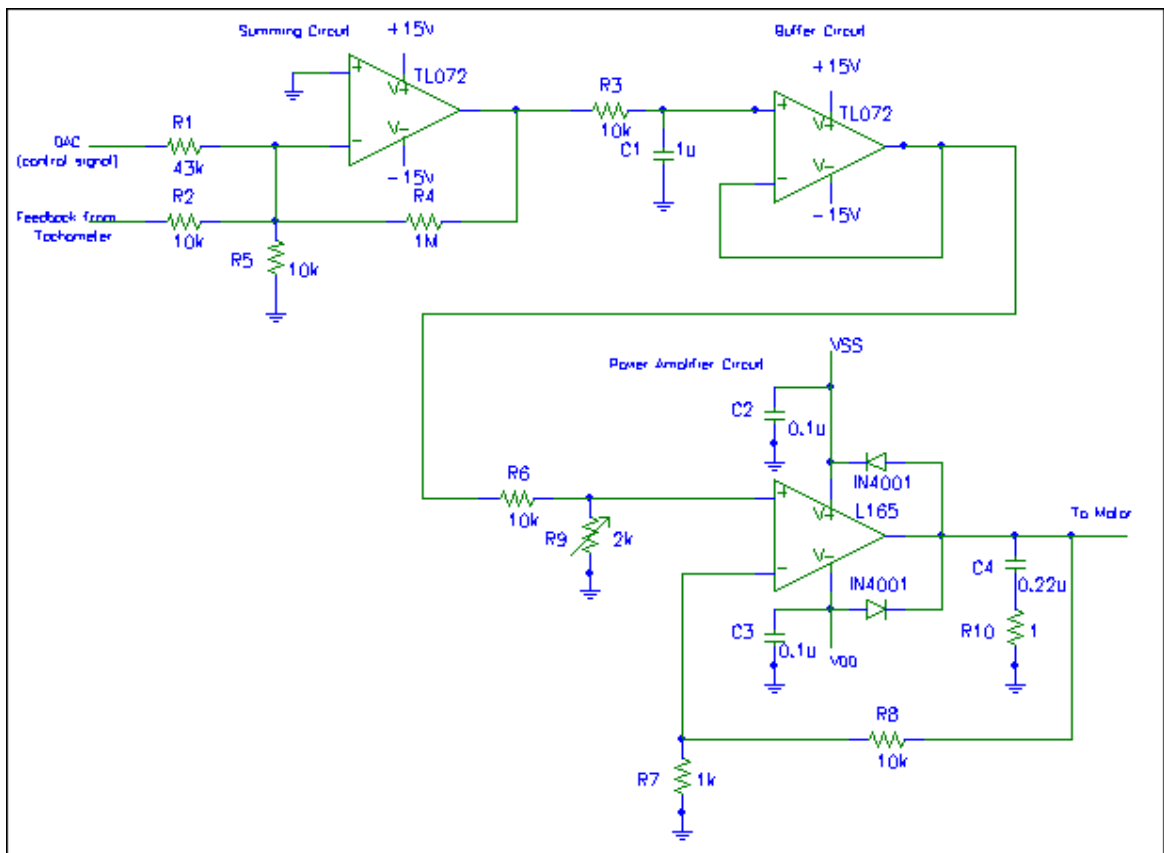
- Pulse/direction mode: In pulse/direction mode the PCL-833 uses one input line (A) for pulse input and one line (B) for direction. If channel B is high (1), the counter will decrement whenever a rising edge occurs on channel A. If channel B is low (0), the counter will increment whenever a rising edge occurs on channel A.
- Disabled mode: PCL-833 will not accept input, but its registers can still be accessed.

Mode selection can be done by programming the card's registers.

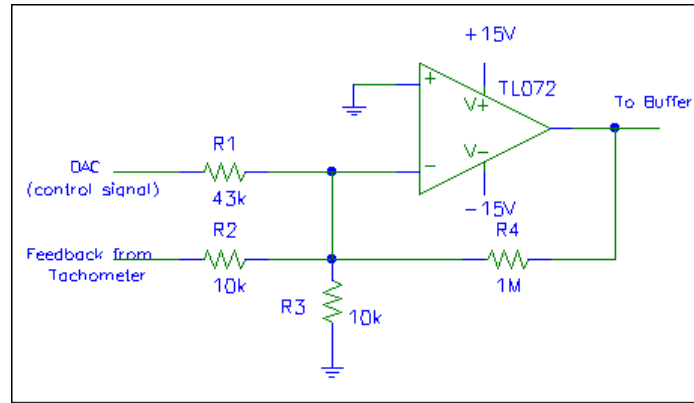
Appendix D: Signal amplification stage

The amplification stage consists of the three following circuits to provide the necessary amplification of the control signal to drive the motors on the plant.

- Summing circuit: to sum the control signal from the computer with negative feedback from the tachometer.
- Power amplifier circuit: to drive the motor connected with driving propellers.
- Buffer circuit: to isolate the summing and power amplifier circuits and remove the high frequency noise in the control signal by a low pass filter.



D.1 Summing amplifier circuit



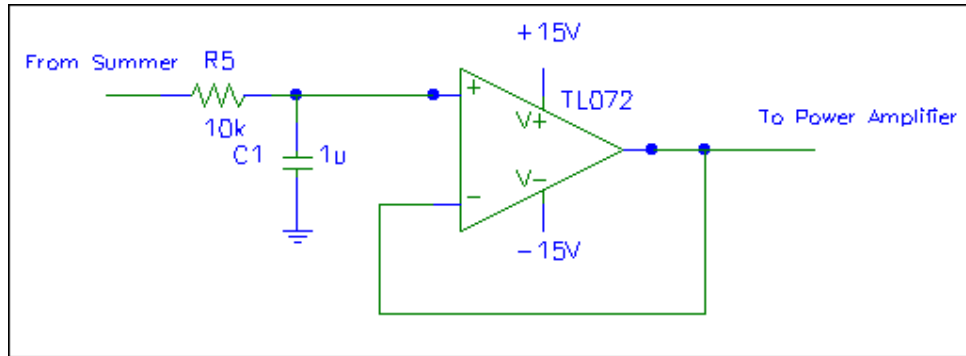
There are two inputs to the summing amplifier circuit:

- The first input is the control signal from the controller. The digital control signal in the computer is converted into the analog signal by means of the PC-30 DAC port. This DAC port provides a maximum output range of $\pm 10\text{V}$.
- The second input is the output from the tachometer. The tachometer provides feedback for the system to determine the speed of the motor thus preventing it from accelerating to damaging speeds. The tachometer output varies proportionally to the voltage applied to the motor. In other words, it increases linearly with respect to the motor speed. The tachometer output is negative when a positive motor voltage is applied. This negative feedback prevents run-away acceleration that could be potentially damaging to the motor. Experiments on the tachometer have shown that respective tachometer voltage is $\mp 2.0\text{V}$.

The summing amplifier circuit provides an output to the buffer circuit as below:

$$V_o = - \left(V_{\text{DAC}} \frac{R_4}{R_1} + V_{\text{Tacho}} \frac{R_4}{R_2} \right)$$

D.2 Buffer circuit



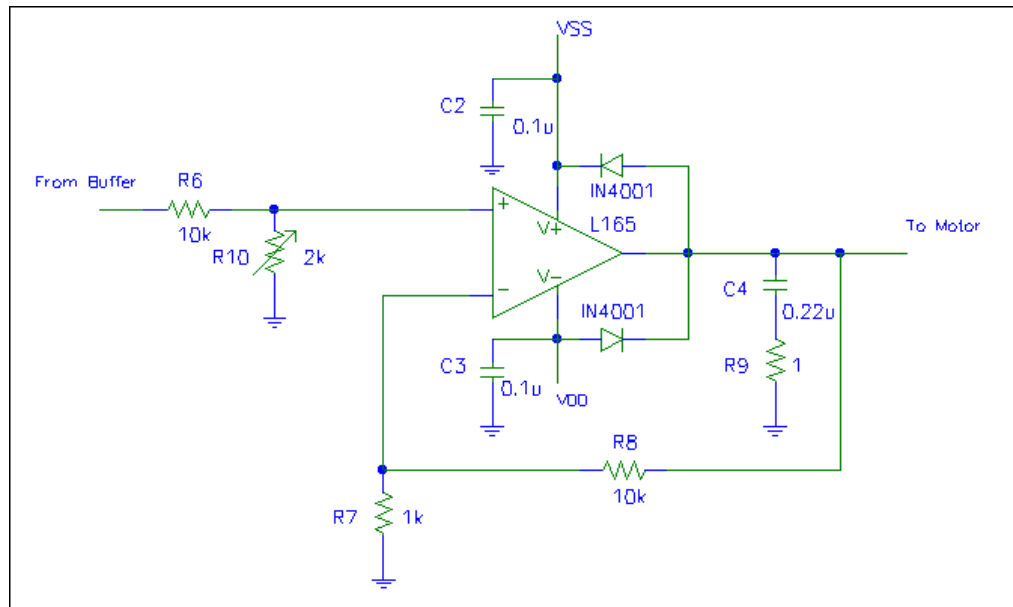
The buffer circuit is used to provide isolation between the summing circuit and the power amplifier circuit and to prevent loading effects from occurring.

To attenuate high frequency noise present in the signal, the buffer circuit is preceded by a first order low-pass filter (LPF). This LPF is designed with a cut-off frequency:

$$f_c = \frac{1}{2\pi.R.C} = 16\text{Hz}$$

The buffer circuit is built by using an operational amplifier in a *voltage follower* configuration. The output of the buffer circuit is directly connected to the Op-Amp inverting input and its input voltage (through a low-pass filter) is at the non-inverting input of the Op-Amp. Since the feedback resistance equals zero, the gain of the follower is 1 (unity gain). That is, the output voltage follows the input voltage with 100% feedback. Op-Amp analysis shows that the voltage difference between the two inputs will always be approximately zero. As such, the circuit provides extremely high input impedance and low output impedance, making it ideal for isolation between circuits.

D.3 Power amplifier circuit



The power amplifier drives the 6W (rated) motors that turn the propellers in a clockwise or anti-clockwise direction to move that plant to the desired position. The L165 is a 3A Power Operational Amplifier, which has high gain and high output power capabilities. However, in this application, the designed gain is 1 and so the output voltage is equal to the input voltage.

On analysis of the circuit, it can be seen that:

- The gain of the non-inverting amplifier is +11: $V_o = \left(1 + \frac{R_8}{R_7}\right) V_{in}$
- Resistor R6 and potentiometer R10 being used as a voltage divider such that the potentiometer is adjusted to give the desired gain of 1. The potentiometer was set to $1k\Omega$ to achieve this.

- Any power supply noise is bypassed to ground via capacitor C4.
- The diodes in the circuit are protection diodes that provide an alternative path for inductive current.
- The capacitors C2 and C3 are to ensure that the power to the operation amplifier remains constant.

In many applications, it is desired that the actuator bandwidth be wide or infinite. However, in the real world, the bandwidth is limited and this is the reason for poor performance when sudden switching is applied. Therefore, the bandwidth of the power amplifier is required to be determined through experiments. For a 1Vp-p input sine wave, the following results were obtained.

Frequency (kHz)	Output voltage (V)	Gain(db)
200	1	0
400	1	0
600	0.96	-0.355
800	0.91	-0.819
1000	0.84	-1.515
1200	0.80	-1.938
1400	0.72	-2.853
1450	0.68	-3.350
1500	0.64	-3.876
1600	0.56	-5.036
1800	0.48	-6.375
2000	0.38	-8.404

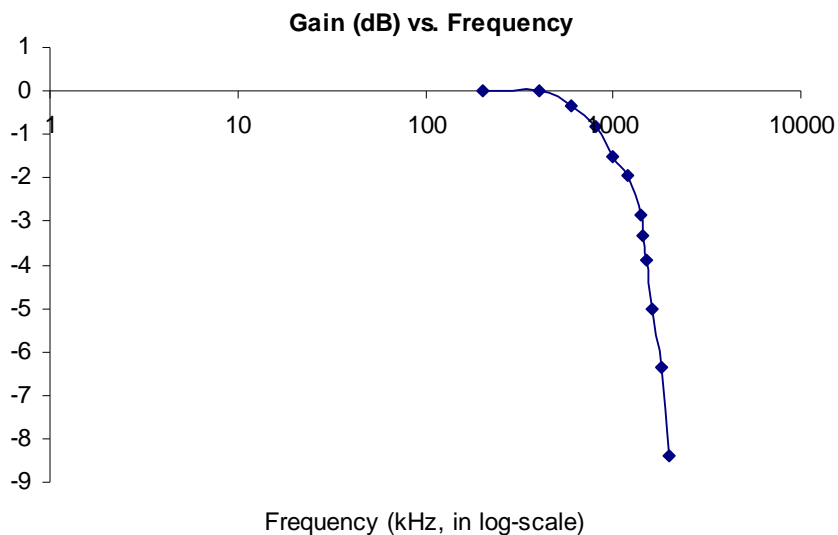


Figure D.1: The gain of the amplifier vs. frequency

The bandwidth of the amplifier is approximated at 400kHz. Within this bandwidth, the output voltage generally follows the shape of the input voltage although it has some ringing as the frequency is increased. The larger the applied frequency, the more apparent the signal distortion will be. Outside the bandwidth, the amplifier circuit cannot provide the desired output signal.