

AUTOMATIC CAMERA CALIBRATION
USING PID AND FUZZY LOGIC
CONTROL

By

BIN WANG

Bachelor of Science

Harbin Institute of Technology

Harbin, China

1988

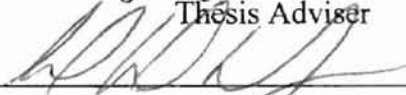
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the degree of
MASTER OF SCIENCE
May, 1999

AUTOMATIC CAMERA CALIBRATION
USING PID AND FUZZY LOGIC
CONTROL

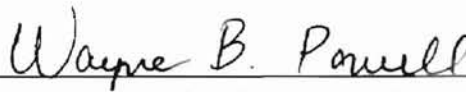
Thesis Approved:



Thesis Adviser







Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to Dr. Lawrence L. Hoberock, my thesis adviser, for his dedication, guidance, encouragement, and endless hours of editing during the course of this work. Many thanks also go to Dr. Eduardo A. Misawa and Dr. Ronald D. Delahoussaye for serving on my graduate committee. Their suggestions and support were very helpful throughout this study.

My deepest thanks go to my wife Hongying Cui and my son Ringmeng Wang. They were the light in my heart that guided me in this research work. Their love and warmth sustained me through the long process of this research.

This work is dedicated to my parents, Liuxiang Li and Zhiyi Wang for their love, encouragement over the years. Special thanks go to my sister, Jie Wang, for all kinds of support.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
Problem Background	1
Machine Vision Review	4
II. INVESTIGATION OF CAMERA PERFORMANCE DYNAMICS.....	7
Machine Vision Hardware Set-Up	7
Camera and Vision System Terminology	7
Minimum and Maximum Gray Levels	9
Camera Performance Dynamics	10
Test 1.....	10
Test 2.....	15
Summary	19
III. DESCRIPTION OF CONTROLLERS	20
Threshold Control.....	22
Gain and Offset Control	24
Proportional Derivative-Integral Control	24
Fuzzy Logic 1 Controller	27
Fuzzy Logic 2 Controller	33
Fuzzy Integral Derivative Controller	40
Fuzzy Integral Controller	47
Summary	56
IV. EXPERIMENTAL RESULTS AND COMPARISONS	57
Testing Methods	58
Transit Response Testing Procedure	58
Steady State Response Testing Procedure	60
Testing Results	61
Fuzzy Logic 1 Controller	61
Fuzzy Logic 2 Controller	67
Fuzzy Integral Controller	73
Fuzzy Integral Derivative Controller	79
Proportional Integral Derivative Controller	85
Summary	91

V. CONCLUSIONS AND RECOMMENDATIONS	95
Conclusions	95
Recommendations	97
REFERENCES	98
APPENDIXES	101
APPENDIX A - COMPOSITE PLOTS	102
APPENDIX B - SOURCE CODE IN V/V+	123
Fuzzy Logic 1 Controller	124
Fuzzy Logic 2 Controller	141
Fuzzy Integral Controller	158
Fuzzy Integral Derivative Controller	179
Proportional Integral Derivative Controller	191
APPENDIX C - SAMPLE DATA	201
Transit Response Test	202
Steady State Response Test	206

LIST OF TABLES

Table	Page
2.1 Initial Value of Camera Control Parameters.	12
2.2 Initial Value of Camera Control Parameters.	15
3.1 Gain Values for Camera Gain Control.	27
3.2 Gain Values for Offset Control.	27
3.3 Linguistic Rules of 1 st Layer of FL1 Controller.	30
3.4 Linguistic Rules of 2 nd Layer of FL1 Controller.	30
3.5 (a) Linguistic Rules of 1 st Layer of FL2 Controller.	36
3.5 (b) Linguistic Rules of 2 nd Layer of FL2 Controller.	37
3.5 (c) Linguistic Rules of 3 rd Layer of FL2 Controller.	38
3.6 Gain Values for the FID Controller.	43
3.7 (a) Linguistic Rules of 1 st Layer of Gain FID Controller.	44
3.7 (b) Linguistic Rules of 2 nd Layer of Gain FID Controller.	44
3.7 (c) Linguistic Rules of 3 rd Layer of Gain FID Controller.	44
3.8 (a) Linguistic Rules of 1 st Layer of Offset FID Controller.	45
3.8 (b) Linguistic Rules of 2 nd Layer of Offset FID Controller.	45
3.8 (c) Linguistic Rules of 3 rd Layer of Offset FID Controller.	45
3.9 Gain Values for the F(E+D)+I Controller.	49
3.10 (a) Linguistic Rules of 1 st Layer of Gain F(E+D)+I Controller.	51
3.10 (b) Linguistic Rules of 2 nd Layer of Gain F(E+D)+I Controller.	52
3.10 (c) Linguistic Rules of 3 rd Layer of Gain F(E+D)+I Controller.	52

3.11 (a)	Linguistic Rules of 1 st Layer of Offset F(E+D)+I Controller.....	53
3.11 (b)	Linguistic Rules of 2 nd Layer of Offset F(E+D)+I Controller.....	54
3.11 (c)	Linguistic Rules of 3 rd Layer of Offset F(E+D)+I Controller.....	54
4.1	I_s for Transit Response Test.....	91
4.2	I_s for Steady State Response Test.....	91
4.3	Peak Control Variable Changes for Transit Response Test.....	92
4.4	Peak Control Variable Changes for Steady State Response Test.....	92

LIST OF FIGURES

Figure		Page
1.1	Machine Vision System Set-UP	2
2.1	Gray Level Histogram of A Vision Target	9
2.2	Vision Target of Varying Grayscale	11
2.3	Minimum Gray Level Change Without Control	13
2.4	Maximum Gray Level Change Without Control.....	14
2.5	Minimum Gray Level Change With Ambient Lighting Disturbances, No Control	17
2.6	Maximum Gray Level Change With Ambient Lighting Disturbances, No Control	18
3.1	Block Diagram for PID Control System.....	24
3.2	Block Diagram for Fuzzy Logic 1 Control System.....	28
3.3 (a)	Fuzzy Set for the First Layer	31
3.3 (b)	Fuzzy Set for the Second Layer	32
3.4	Block Diagram for Fuzzy Logic 2 Control System.....	34
3.5	Fuzzy Set for All Three Layers of FL2C.....	39
3.6	Block Diagram for FID Control System.....	41
3.7	Block Diagram for the F(E+D)+I Control System	48
3.8	Fuzzy Set for $DE_{g(i-1)}$ and $DE_{o(i-1)}$	55
4.1(a)	FL1C Minimum Gray Level Transit Response	62
4.1(b)	FL1C Maximum Gray Level Transit Response.....	62
4.2(a)	FL1C Gain Transit Response.....	63
4.2(b)	FL1C Offset Transit Response	63

4.2(c) FL1C Threshold Transit Response.....	63
4.3(a) FL1C Minimum Gray Level Steady State Response	65
4.3(b) FL1C Maximum Gray Level Steady State Response.....	65
4.4(a) FL1C Gain Steady State Response.....	66
4.4(b) FL1C Offset Steady State Response.....	66
4.4(c) FL1C Offset Steady State Response.....	66
4.5(a) FL2C Minimum Gray Level Transit Response	68
4.5(b) FL2C Maximum Gray Level Transit Response.....	68
4.6(a) FL2C Gain Transit Response.....	69
4.6(b) FL2C Offset Transit Response.....	69
4.6(c) FL2C Threshold Transit Response.....	69
4.7(a) FL2C Minimum Gray Level Steady State Response	71
4.7(b) FL2C Maximum Gray Level Steady State Response.....	71
4.8(a) FL2C Gain Steady State Response.....	72
4.8(b) FL2C Offset Steady State Response	72
4.8(c) FL2C Offset Steady State Response.....	72
4.9(a) F(E+D)+I Minimum Gray Level Transit Response.....	74
4.9(b) F(E+D)+I Maximum Gray Level Transit Response.....	74
4.10(a) F(E+D)+I Gain Transit Response.....	75
4.10(b) F(E+D)+I Offset Transit Response.....	75
4.10(c) F(E+D)+I Threshold Transit Response	75
4.11(a) F(E+D)+I Minimum Gray Level Steady State Response	77
4.11(b) F(E+D)+I Maximum Gray Level Steady State Response.....	77
4.12(a) F(E+D)+I Gain Steady State Response.....	78
4.12(b) F(E+D)+I Offset Steady State Response.....	78

4.12(c) F(E+D)+I Offset Steady State Response.....	78
4.13(a) FID Minimum Gray Level Transit Response.....	80
4.13(b) FID Maximum Gray Level Transit Response.....	80
4.14(a) FID Gain Transit Response.....	81
4.14(b) FID Offset Transit Response.....	81
4.14(c) FID Threshold Transit Response.....	81
4.15(a) FID Minimum Gray Level Steady State Response.....	83
4.15(b) FID Maximum Gray Level Steady State Response.....	83
4.16(a) FID Gain Steady State Response.....	84
4.16(b) FID Offset Steady State Response.....	84
4.16(c) FID Offset Steady State Response.....	84
4.17(a) PID Minimum Gray Level Transit Response.....	86
4.17(b) PID Maximum Gray Level Transit Response.....	86
4.18(a) PID Gain Transit Response.....	87
4.18(b) PID Offset Transit Response.....	87
4.18(c) PID Threshold Transit Response.....	87
4.19(a) PID Minimum Gray Level Steady State Response.....	89
4.19(b) PID Maximum Gray Level Steady State Response.....	89
4.20(a) PID Gain Steady State Response.....	90
4.20(b) PID Offset Steady State Response.....	90
4.20(c) PID Offset Steady State Response.....	90
A.1 Transit Response-Minimum Gray Level (camera 1).....	103
A.2 Transit Response-Minimum Gray Level (camera 2).....	104
A.3 Transit Response-Maximum Gray Level (camera 1).....	105
A.4 Transit Response-Maximum Gray Level (camera 2).....	106

A.5	Transit Response-Gain (camera 1)	107
A.6	Transit Response-Gain (camera 2)	108
A.7	Transit Response-Offset (camera 1)	109
A.8	Transit Response-Offset (camera 2)	110
A.9	Transit Response-Threshold (camera 1)	111
A.10	Transit Response-Threshold (camera 2)	112
A.11	Steady State Response-Minimum Gray Level (camera 1)	113
A.12	Steady State Response-Minimum Gray Level (camera 2)	114
A.13	Steady State Response-Maximum Gray Level (camera 1)	115
A.14	Steady State Response-Maximum Gray Level (camera 2)	116
A.15	Steady State Response-Gain (camera 1)	117
A.16	Steady State Response-Gain (camera 2)	118
A.17	Steady State Response-Offset (camera 1)	119
A.18	Steady State Response-Offset (camera 2)	120
A.19	Steady State Response-Threshold (camera 1)	121
A.20	Steady State Response-Threshold (camera 2)	122

CHAPTER I

INTRODUCTION

Problem Background

Over the past 20 years, machine vision systems have evolved rapidly, with performance increasing and cost decreasing. This has led to significant use of machine vision systems in the food processing industry, automotive industry, and in numerous other manufacturing industries. Machine vision systems have become increasingly powerful, especially when they are combined with robotic and automation systems. With the rapidly developing uses of machine vision systems, it becomes increasingly important to implement continuous automatic camera calibration. Such calibration is especially important in precision measurements by vision systems. An example of this is described by Johnson (1993) and Chen (1996) in the automatic inspection and handling of dishes for a large scale commercial dish washing operation. In this problem, individual dish pieces are to be automatically loaded into a "flight type" dish-belt moving into a dishwasher. After washing, a vision system is used to identify and inspect individual dish pieces at a rate of 2 seconds or less per dish, sending appropriate handling signals to a robot for unloading and placement of the dishes.

Johnson (1993), Chen (1996), and Feng (1992) made use of an experimental set-up to investigate this process, illustrated in Figure 1.1. The important elements in this figure are:

1. ADEPT ONE robot
2. Three CCD (charge coupled device) cameras
3. Three standard targets
4. Central control unit
5. Vision monitor and programming terminal

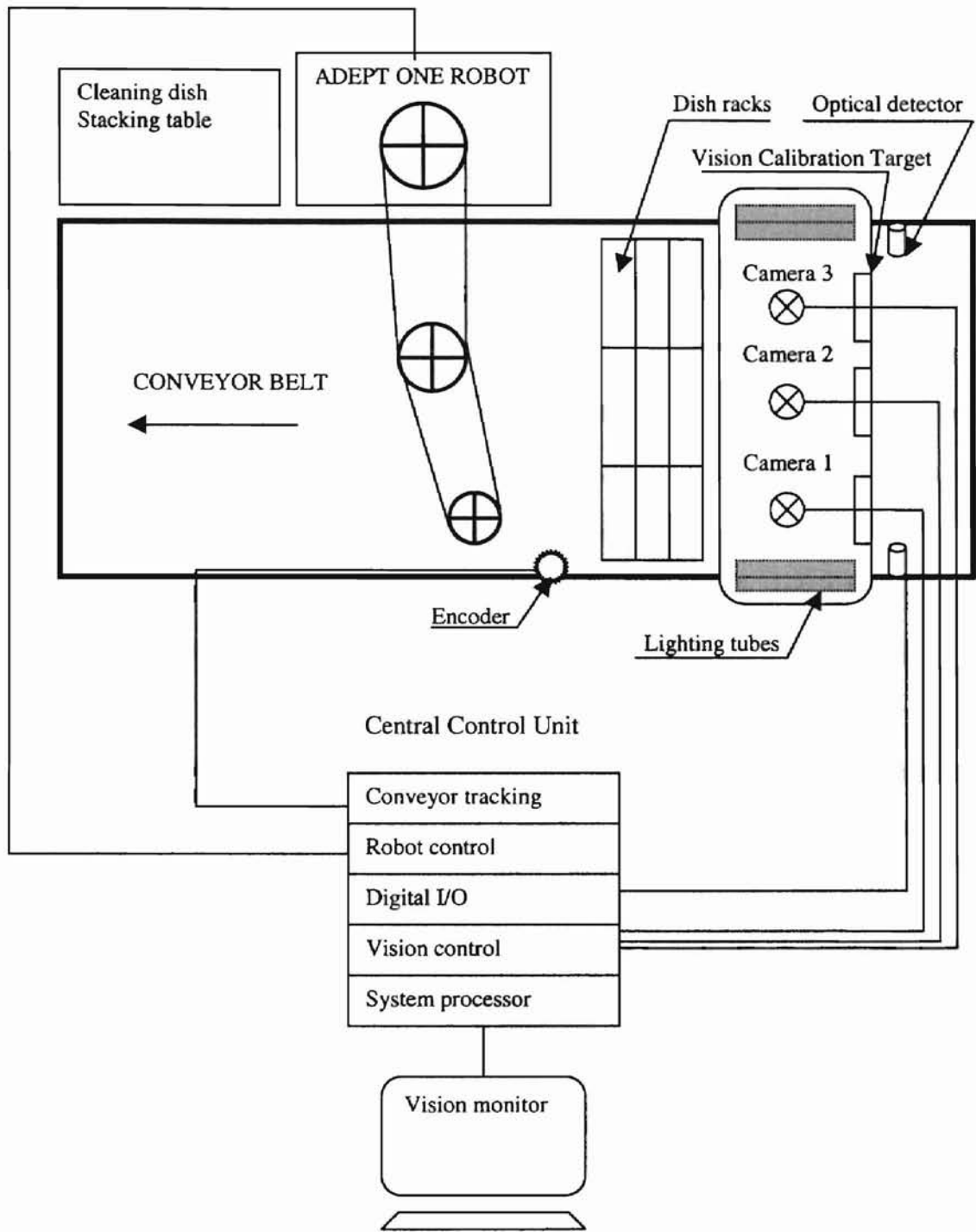


Figure 1.1

Machine Vision System Set-Up
(Adapted From Chen(1996))

6. Conveyor and encoder
7. Dish rack
8. Dish stacking table
9. Optical position detectors
10. Fluorescent lighting tubes

When the loaded dish rack moves into the field of view (FOV) of the cameras, the optical position detectors near the conveyor signal the central control unit (CCU) of the presence of dish pieces, and the cameras are then triggered upon command from the CCU. After an image of a dish is taken and sent to CCU, the CCU analyzes the image data and stores relevant information (such as dish type, dish cleanliness, and dish location on the moving conveyor) into a database. Later, when the dishes enter the robot workspace, relevant information is retrieved and sent to robot, such that the robot will perform “pick” and “place” functions on each individual dish, accordingly.

Normally commercial dish washing machines will run 2 or 3 shifts each day and handle thousands of different dishes. During the process, a machine vision system should maintain high accuracy (recognize each dish correctly, inspect properly for cleanliness, and locate the position precisely) over time. In the proposed dish handling automation system, the CCD camera should continually maintain calibration while securing simultaneously images of numerous (up to 5) dish pieces during shifts lasting up to 5 hours or more. This would allow different dishes to be correctly identified, inspected, and located, such that the ADEPT ONE robot could pick up each dish from its location on conveyor and place it in the correct stacking location. In order to do this, the camera must maintain calibration. However, as will be seen in later chapters, due to changing camera dynamic characteristics, ambient light changes, power supply variations and other environment factors, the calibration of the camera changes, as measured by variations in two quantities, called “minimum gray level” and “maximum gray level”. A standard vision calibration target in the camera FOV is used to identify these changes. Such changes, if not corrected, yield

erroneous results in dish identification and inspection. Accordingly, in the work herein, we investigate means to eliminate or substantially reduce these variations. We will analyze the dynamics of the CCD camera and design controllers to control the camera's parameters of "gain", "offset", and "threshold".

Machine Vision Review

According to Freeman (1989), "Machine vision - for guidance, for inspection, for surveillance, for collision avoidance, and for thousands of materials-handling applications - is likely to be as commonplace eventually as the electric light bulb is today". Jain, et al (1995) stated, "Machine vision is concerned with achieving electronically human visual perception. It involves image sensing, image processing, pattern recognition and image understanding. A machine vision system recovers useful information about a scene from its two-dimensional projections. The good machine vision system is to create a model of the real world from images". These are two recent comments on machine vision, research on which has been in progress for over 40 years. There are several closely related fields that are very important in today's machine vision work. They are: image processing, computer graphics, pattern recognition, artificial intelligence, neural networks and psychophysics (Jain et al, 1995). In early work, due to the high cost of machine vision systems and relatively poor performance, applications were limited to the laboratory. For the past 20 years, with the rapid development of the computer industry, the cost of both hardware and software has decreased, and machine vision functionality, availability, and quality has significantly improved. High performance machine vision systems with acceptable price are available today in the market place and are increasingly in use in industry.

Machine vision innovation has been especially active for the past 10 years. In the manufacturing industry, machine vision is applied to tube inspection (Mohtadi et al, 1992), lamp filament alignment process (Joshi & Sanderson, 1996), screw socket recognition and parts disassembly (Gergenbac et al, 1996), and parts assembly (Nelson et al, 1996). In the food process

industry, machine vision has been applied to food piece recognition for an automatic handling system (Li & Lee, 1996). In the chemical and energy industries, machine vision systems are used to inspect hazardous waste drums (Byler, et al 1995).

With the widespread use of machine vision systems in industries, the need for automatic, continuous camera calibration to maintain camera stability and robustness against disturbances over time becomes especially important. In many applications, it becomes the crucial issue, and determines the overall performance of the vision system. Tappan, et al (1987) identified the camera stability problem when they experimented with a digital image analysis system. Brainard, et al (1989) realized that variation of ambient illumination will cause serious problems in vision system, and they investigated surface reflectance functions to overcome the problem of maintaining a constant image. In 1994, investigators at University of California Irvine developed a model for a CCD camera, and tried to model different sources of variation in real images obtained from video cameras (Healey, et al, 1994). Chang, et al (1996) realized that gray level shift will cause serious problem in color imaging and developed a method using a standard color chart to overcome the stability problem. However none of these investigations produced a robust method to address the problem at hand. Johnson (1993) investigated machine vision for use in an automated dish handling system for commercial dishwashing. This employed machine vision to sort and inspect individual dishes at a rate of 2 seconds per dish. Following this work, Chen (1996) employed neural networks with a vision system for silverware and dish recognition and inspection. Both of these investigations identified the need for continual and automatic camera calibration, the subject of the work herein. While Chen and Johnson each proposed feedback controllers to stabilize camera performance for the dish handling problem, their controllers were not sufficiently robust, non were they optimal in any sense.

In the following chapters, we investigate several new control algorithms to overcome this camera stability problem. In chapter 2, we investigate the performance dynamics of cameras used in the experimental set-up in figure 1.1. In chapter 3, several controllers developed for this

machine vision application are discussed thoroughly. In chapter 4, two extensive experiments are presented. One is a short transit response experiment, which is started at prescribed initial conditions. The other is a longer-term response experiment with 2 different levels of disturbance of ambient lighting applied to the camera FOV. The Integral Square Error is calculated on line during each experiment and used as a performance measure to compare controllers. In Chapter 5, the experiment results are discussed, and conclusion drawn.

CHAPTER II

INVESTIGATION OF CAMERA PERFORMANCE DYNAMICS

Machine Vision Hardware Set-Up

In this chapter we investigate the performance dynamics of the cameras in the experimental set-up described in Chapter 1. The cameras are interfaced with an Adept Technologies AGS vision system. The physical equipment includes:

- Controller equipped with special vision processor boards and a camera multiplexer.
- 2 Pulnix cameras, model TM-540, serial numbers 015207, and 014969 corresponding to Camera 1 and 2 respectively.
- 1 Light box with two (number) inch long fluorescence lighting tubes on each side of the box, oriented parallel with the dishrack, and providing indirect lighting for objects within the camera FOV.
- Graphics terminal that includes: a high-resolution color monitor, standard keyboard, and 3-button mouse.

The Adept AGS vision system controller contains the logic boards, system and vision processor boards, I/O boards, multiplexer, and user-ready software. This hardware system provides an environment for Adept's V+ Operating System and Language that allows user to direct and monitor vision operations. This hardware and software combination is multitasking and contains everything necessary to control up to 8 physical cameras.

Camera and Vision System Terminology

We first define fundamental vision and camera terminology, including pixel, gray level, threshold, offset, gain, edge-strength, binary processing, and grayscale processing.

A "Pixel" is the basic unit of a vision image and is the smallest unit of information a vision system can return to the user. The number of pixels the system can process determines the system's resolution and affects the computer processing time needed to analyze an image. For each pixel there is an integer value of a shade of gray, called the "Gray Level", ranging from 0 to 127.

"Threshold" is an integer parameter ranging from 0 to 127 that sets the gray level value at which the vision software interprets a pixel as either black or white in binary operations. For gray levels equal to or larger than a set threshold value, the pixel is judged white, and less than this level it is judged black.

The parameter "Gain" is used to multiply the gray level of a pixel, and there are 256 integer values of gain used in the AGS vision system. When the gain is set for an incoming video signal, the histogram of scale pixel intensities will be expanded or contracted in proportion to the value of the Gain.

"Edge Strength" is a parameter used for edge recognition in grayscale processing. If the variation in pixel intensity across a region exceeds this parameter, an edge is recognized.

Another parameter called "Offset" can be used together with the Gain to increase the range of grayscale intensities that the system recognizes in objects. A value for Offset is added to the actual pixel grayscale value. When the offset is set for an incoming video signal, the video histogram will be shifted left or right. The Offset has integer values ranging from 0 to 255.

An image of a FOV (Field Of View) returned by the camera can be thought of as a large matrix of pixels, with each pixel in the matrix having a grayscale value. When the software processes image data in the binary mode, each value in the matrix is compared with the value of the threshold parameter. All the pixels with a gray level value equal to or larger than the threshold are considered white, and all pixels with a gray level below this value are considered black. When the software processes the data in the grayscale mode, it looks at a three-by-three section of pixels and compares the difference in gray level values found in the neighboring pixels to the value of

Edge Strength. If the difference found exceeds the value set for Edge Strength, the system considers the three-by-three area to be part of an edge.

There are two different image processing methods, namely binary processing and grayscale processing. Binary processing uses only two states, black and white. However, grayscale processing uses data based on 128 states in the range 0-127 for each pixel, and also has to calculate edges based on 3 by 3 sections of pixels. Accordingly, binary processing requires much less computational time. On the other hand, since binary processing recognizes pixels only as black or white, objects and background should be highly contrasted for high fidelity and best results. In many cases, it is difficult for binary processing to recognize interior features of objects.

Minimum and Maximum Gray Levels

Minimum and maximum values of gray levels produced from a camera image are important quantities for vision system. After an image of the FOV is returned by the camera, the Adept AGS system processes the image data using a set of parameter settings, such as Gain, Offset, and Threshold. Each pixel is assigned a value of gray level ranging from 0 - 127, and sorted into a certain cell of a 128-element array, which is associated with this gray level. Figure 2.1 illustrates a histogram of pixel counts for an image.

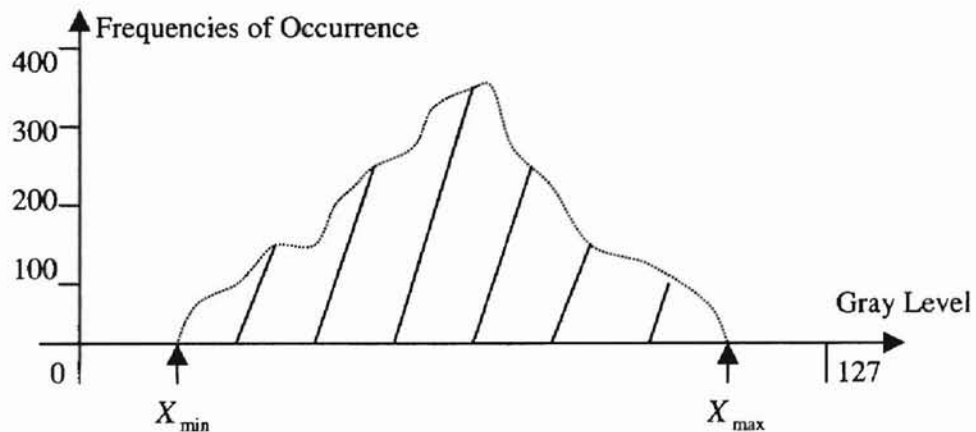


Figure 2.1

Gray Level Histogram of A Vision Target

Observe that there is a lower and an upper bound for the values of gray level found. The lower bound is called “The minimum gray level” designated X_{\min} , while the upper bound is called “The maximum gray level”, designated X_{\max} . The range between these bounds is an indication of the brightness variation over the entire object, such that the bounds are important indications of object brightness range. If while viewing an object over an extended period of time, the camera exhibits performance variations, or if lighting fluctuates or drifts, the minimum and maximum gray levels of this object will shift, such that differing conclusions regarding characteristics of the object can be drawn. Accordingly, monitoring minimum and maximum gray level variations while viewing a standard and constant “Target” is an effective means to check camera and lighting stability.

Camera Performance Dynamics

CCD cameras have as their core a large array of light sensitive semiconductors called “Charge Coupled Devices”. Small deviations in the manufactured quality of these devices yield somewhat different performance characteristics from camera to camera. Environmental factors affecting camera performance include variations in electrical voltage supplying the camera and variations in intensity of lighting of the FOV. Such variations will cause variability in camera performance with time. More over, some CCD cameras exhibit “Drift” in performance, even with constant power and lighting. In what follows, we examine dynamics of camera performance.

As a baseline for our experiments, by trial and error we set desired minimum and maximum gray levels for a standard “target” as:

$$X_{d \min}(1) = 10, \quad X_{d \min}(2) = 10;$$

$$X_{d \max}(1) = 60, \quad X_{d \max}(2) = 60;$$

where: X represents gray level; the parenthetical numbers (1) and (2) represent cameras 1 and 2, respectively; subscripts “min” and “max” indicate minimum and maximum; and subscript “d”

denotes “desired”. For this application we take ± 2 as acceptable variation from desired values of the minimum and maximum gray levels.

A full size reproduction of the standard vision target supplied by Adept Technologies, Inc. is given in Figure 2.2. This reproduction is the actual size of the target. This target is located at the bottom edge of the FOV of the camera, as illustrated in Figure 1.1 of the Introduction. The remainder of the FOV of the camera contains only the black background of the conveyor belt (Figure 1.1). This arrangement is used for all test results reported in this thesis. The target is located at the bottom edge of the camera FOV to provide as much room as possible in the remaining FOV to recognize objects, such as dishes and silverware pieces.

In order to demonstrate dynamic characteristics of the cameras, we present results from two tests. Test 1 is a long-time test of approximately 50 minutes, with no intentional variation of ambient lighting. Test 2 is a short-time test lasting approximately 10 minutes, with intentionally introduced ambient lighting disturbances.

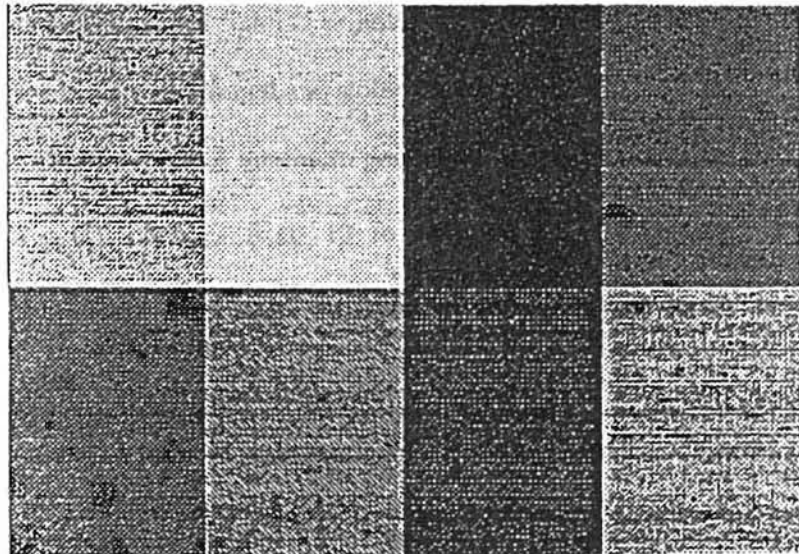


Figure 2.2

Vision Target of Varying Grayscale

Test 1

In order to start the test from known maximum and minimum gray level values, we first apply one of the controllers discussed in later chapters (namely the PID controller). By selecting desired values of $X_{d\min} = 10$ and $X_{d\max} = 60$ for both Cameras 1 and 2, the PID controller found the threshold, gain, and offset values given in Table 2.1 such that both camera maximum and minimum gray levels are forced to the values of 60 and 10, respectively.

Table 2.1 Initial Values of Camera Control Parameters

Camera	Threshold	Gain	Offset
1	51	89	102
2	39	176	65

When these values were reached, the PID controllers for both cameras were simultaneously turned off and the cameras allowed to “drift” or “float” while imaging the target and background for 50 minutes. During this time, the values of threshold, gain, and offset shown in Table 2.1 were held constant, and no intentional lighting or power variations were introduced. The minimum and maximum gray levels were monitored and stored for later analysis.

Figure 2.3 shows plots of minimum gray level variations with time for Cameras 1 and 2. Observe that $X_{\min}(1)$ changes from 10 to 30 and $X_{\min}(2)$ changes from 10 to 5 in approximately 55 minutes. Such large variations of minimum gray levels are unacceptable for quality imaging work.

Figure 2.4 shows plots of maximum gray level variations for Cameras 1 and 2. $X_{\max}(1)$ and $X_{\max}(2)$ change from 60 to 65 and 55, respectively. Such variations in maximum gray level are also unacceptable.

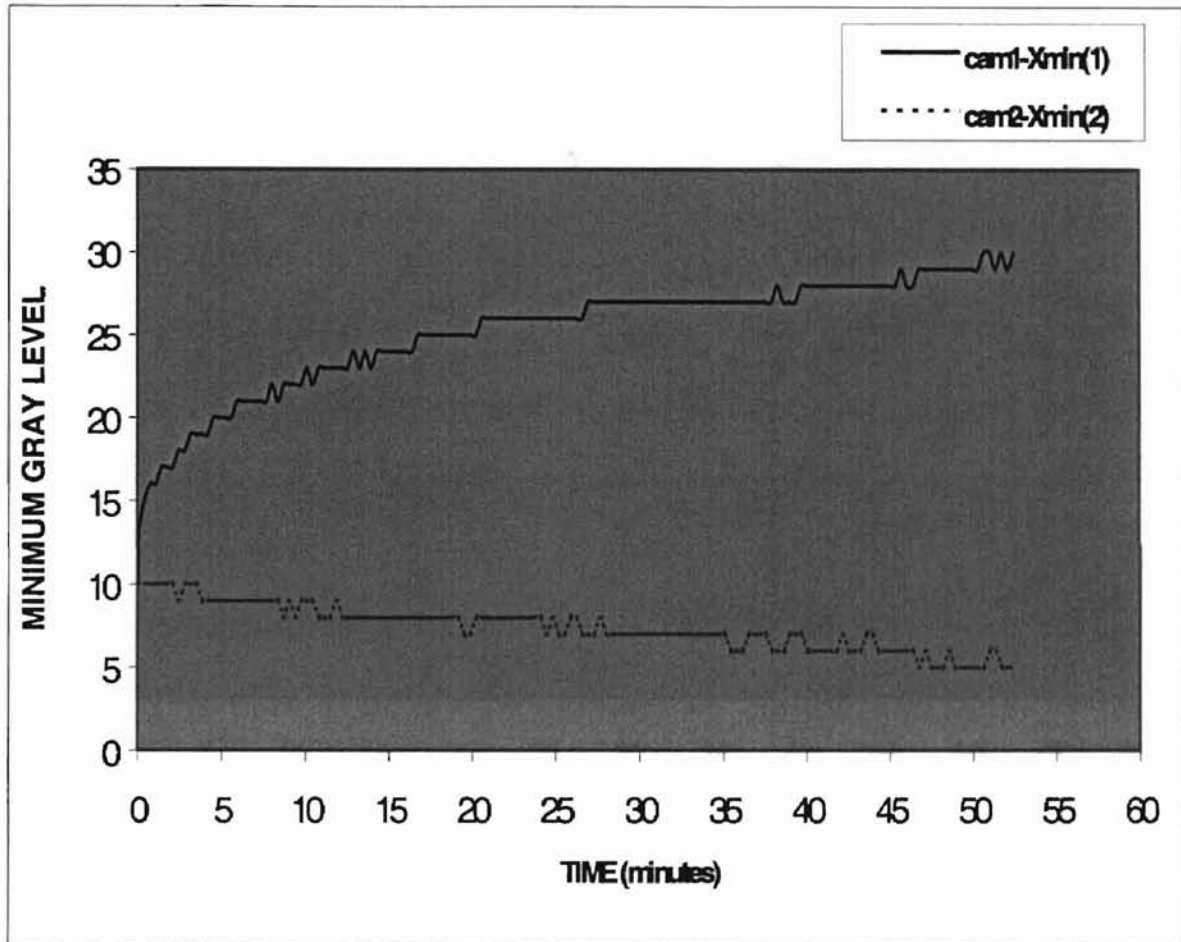


Figure 2.3

Minimum Gray Level Change Without Control

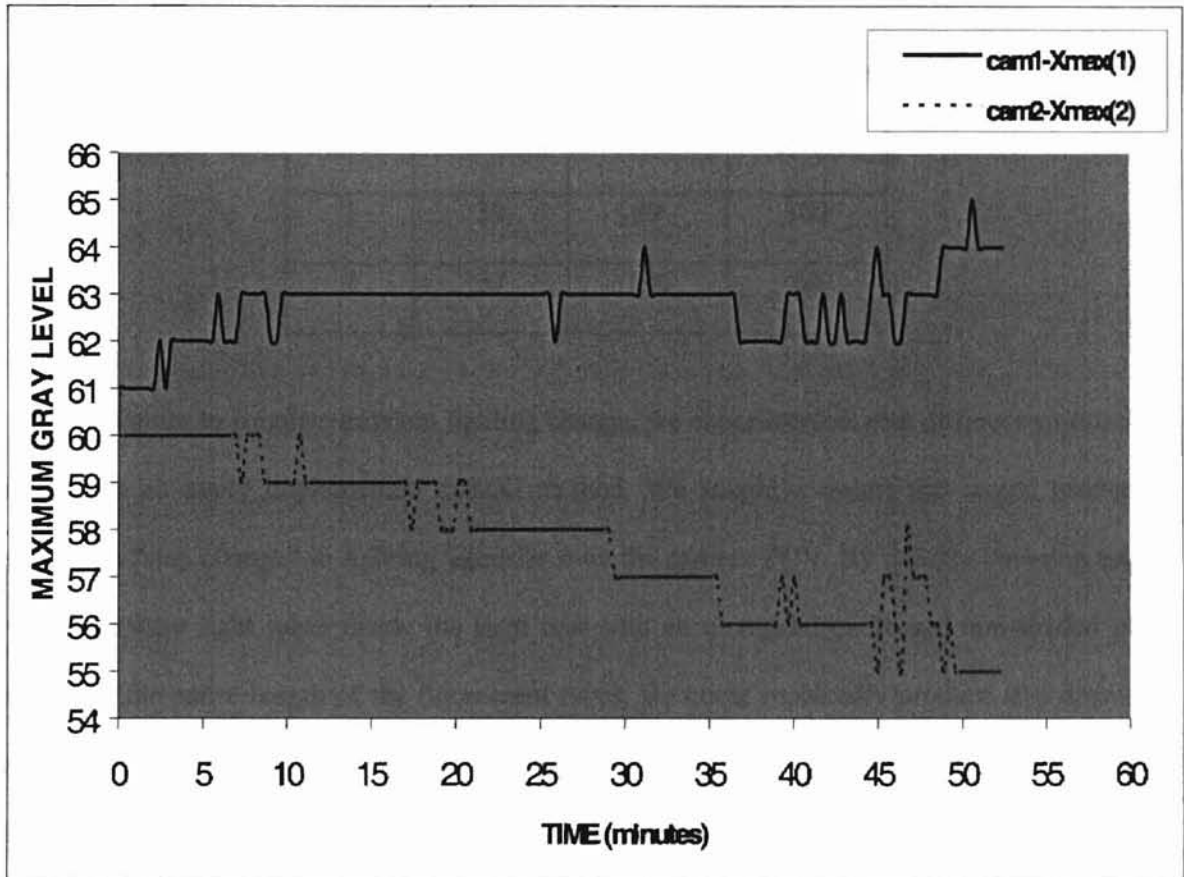


Figure 2.4

Maximum Gray Change Without Control

Test 2

Test 2 was a 10-minute test with an intentional ambient disturbance applied. To initiate the test, we first applied our PID controller as in Test 1 to force minimum and maximum gray levels to 10 and 60, respectively. Table 2.2 lists the values of threshold, gain, and offset found by the PID controller to accomplish this. Note that due to drift in camera performance and/or lighting variations, these values are different from those given in Table 2.1.

Table 2.2 Initial Values of Camera Control Parameters

Camera	Threshold	Gain	Offset
1	39	89	102
2	37	196	60

In order to simulate ambient lighting change, we experimented with different approaches, settling on an easily implemented manual method. We sought a means that would produce a repeatable “step change” in lighting intensity over the camera FOV. By quickly covering one or two fluorescent light tubes inside the light box with an elongated cardboard tent-shaded piece extending the entire length of the fluorescent tubes, we could repeatedly produce step decreases in lighting intensity. Quickly removing this covering produced a repeatable step increase in lighting intensity. We found that use of an on-off switch for the fluorescent tubes would not produce repeatable step changes in light intensity due to transient “flicker” in the fluorescent lights.

Our schedule for lighting changes was as follows:

0 – 2 minutes	no cover over the lighting tubes;
2 – 4 minutes	cover 1 lighting tube out of 2 each side;
4 – 6 minutes	no cover over the lighting tubes;
6 – 8 minutes	cover 2 lighting tubes out of 2 each side;
8 – 10 minutes	no cover over the lighting tubes;

After initially setting the minimum and maximum gray levels to 10 and 60, respectively, we shut off the camera PID controller and operated without control using the constant gain, offset, and threshold values shown in Table 2.2. Then, according to the lighting change schedule above, we covered and uncovered fluorescent tubes in the light box. During the test, both minimum and maximum gray levels were monitored and stored for later analysis.

Figure 2.5 shows plots of $X_{\min}(1)$ and $X_{\min}(2)$ over the 10-minute test period. Observe that with only one tube covered, minimum gray levels for both cameras fell to zero. Also note that the “plateaus” during the uncovered portions of the test shifted upward as the test progressed. After 10 minutes $X_{\min}(1)$ reached 18 while $X_{\min}(2)$ reached 13. Such variation is unacceptable. Moreover, we note that the minimum gray level of both cameras is very sensitive to ambient lighting changes, which can cause severe difficulties in sorting and inspecting dish pieces, as well as in consistent imaging of objects in general.

Figure 2.6 shows plots of maximum gray level changes responding to ambient lighting changes. Observe that during covering of one or more light tubes, the lower “plateaus” in gray level decreased with time, but did not reach zero. Of course, the plateau from 6 to 8 minutes is lower than that from 2 to 4 minutes because more light was blocked. As for minimum gray levels, maximum gray levels showed upper plateaus shifting upward with time during the “uncovered” portions of the test. At the end of 10-minute test, the maximum gray level for Camera 1 reached 65, and that for Camera 2 reached 64, excessive deviations from the desired value of 60. As for minimum gray level, this test demonstrated that the maximum gray level is very sensitive to ambient light disturbances. Of course, the introduced lighting disturbances are substantial and are significantly larger than would be encountered during normal operation. This was intentional to achieve large excursions in camera responses.

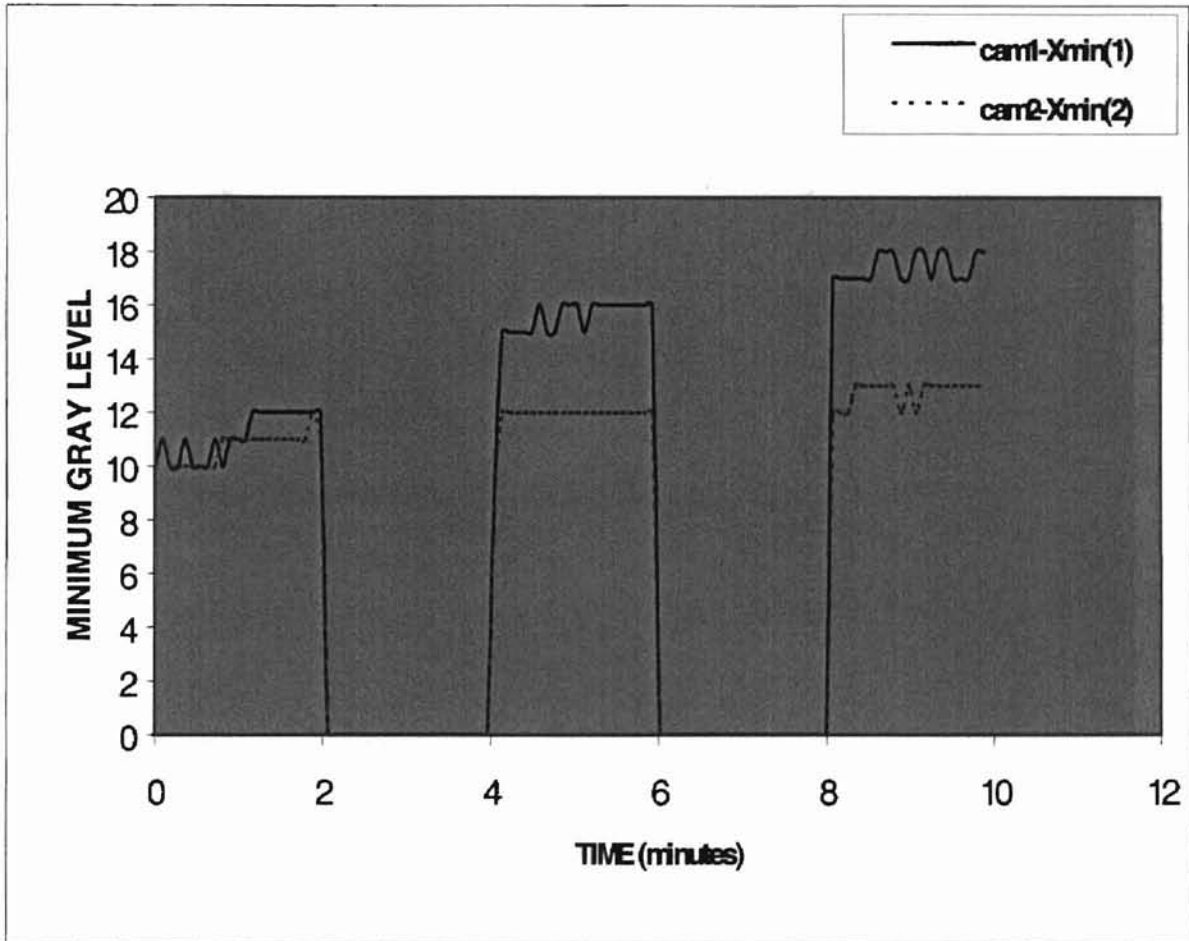


Figure 2.5

Minimum Gray Level Change With Ambient Lighting Disturbances, No Control

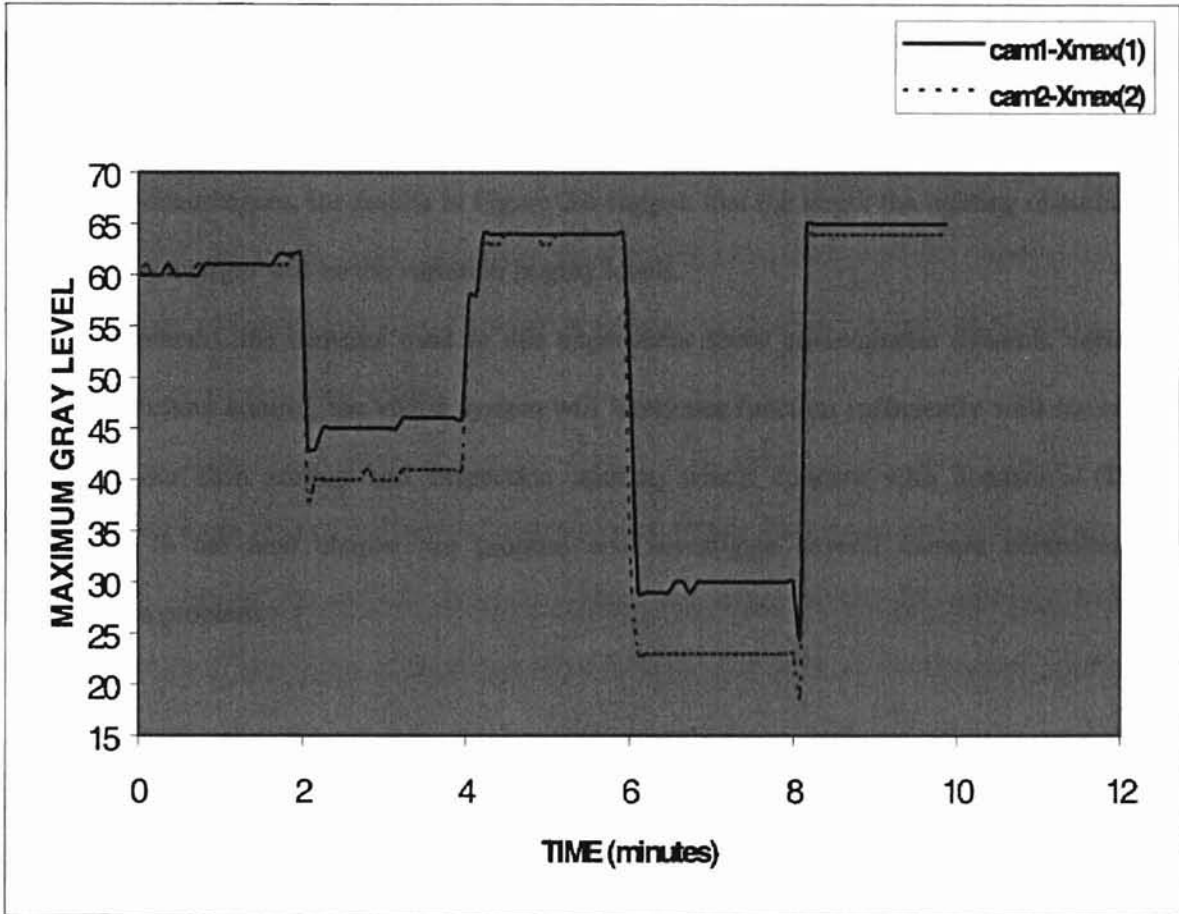


Figure 2.6

Maximum Gray Level Change With Ambient Lighting Disturbances, No Control

Summary

Based on the results of these tests, the following conclusions can be drawn:

- The minimum and maximum gray levels for both cameras shift away from the desired initial setting (in this experiment 10 for minimum and 60 for maximum) over time, even without disturbances. The variations are unacceptable for high quality, repeatable imaging.
- With substantial disturbances in lighting intensity, the minimum and maximum gray levels will change dramatically. While our tests did not cover a wide range of lighting disturbances, the results in Figure 2.6 suggest that the larger the lighting disturbance, the larger will be the variation in gray levels.

In general, the cameras used in this experiment show unacceptable dynamic variation over time. Without control, the vision system will likely not function sufficiently well for object imaging in our dish sorting and inspection system, which concurs with Johnson's (1993) conclusions. In the next chapter we propose and investigate several camera controllers to overcome this problem.

CHAPTER III

DESCRIPTION OF CONTROLLERS

In Chapter 2, experiments on camera performance dynamics demonstrated that over time the minimum and maximum gray levels for both cameras shift away from the desired initial settings. With disturbances in lighting intensity introduced, the minimum and maximum gray levels change dramatically. Such variations are unacceptable for high quality, repeatable imaging. Without control, the vision system will likely not function sufficiently well for object imaging in our dish sorting and inspection system. In this chapter we propose and investigate several camera controllers to overcome this problem.

As mentioned in Chapter 2, for the Adept AGS vision system, there are two image processing methods, binary and grayscale processing. When binary processing is adopted, the parameter "threshold" is used to delineate black from white pixels. By choosing different threshold values, binary pixels of the object show differing distributions. On the other hand, when processing an image in the grayscale mode, the parameters "gain" and "offset" are used together to maximize the range of grayscale intensities that the system recognizes in an image. When different gain values are used for an incoming video signal, the histogram scale of pixel intensities will be expanded or contracted, and the minimum and maximum gray level will be changed accordingly. When different offset values are set for an incoming video signal, the histogram will be shifted left or right, and both minimum and maximum gray levels will also change. By choosing appropriate combinations of values for gain, offset, and threshold, the minimum and maximum gray levels can be forced to desired values while viewing a fixed object. In order to maintain constant minimum and maximum gray levels of an object repeatable, high-

quality imaging over time, the vision parameters gain, offset, and threshold can be controlled in real time.

Since the parameter “threshold” is used in the binary processing mode, while “gain” and “offset” are used in the grayscale processing mode, it is possible to use different algorithms to control threshold than to control gain and offset. This gives considerable flexibility in controller design. After experimenting with different control algorithms, we found a reliable, yet simple and easy to use algorithm for threshold control, which is used in all controller designs discussed herein. Using this algorithm, we then investigated a number of controllers, 5 in all, for gain and offset. These were:

1. Proportional-Integral-Derivative (PID) controller;
2. Fuzzy Logic 1 (FL1) controller;
3. Fuzzy Logic 2 (FL2) controller;
4. Fuzzy-Integral-Derivative (FID) controller;
5. Fuzzy-Integral (F(E+D)+I) controller;

Actually, each composite camera controller consists of two portions, one for threshold, which is the same for all 5 controllers, while the other is for gain and offset.

As we mentioned in Chapter 2, all 5 controllers use the pixel distribution histogram of a multi-gray level target to control gain, offset and threshold to yield a desired minimum and maximum gray level of the target. The target is shown in Figure 2.2. A histogram is a plot versus gray level number of numbers or percentages of pixels having a particular gray level. An example histogram for our target is shown in Figure 2.1. The desired minimum and maximum gray levels are:

$$\begin{aligned} X_{d \min} (1) &= 10, & X_{d \min} (2) &= 10; \\ X_{d \max} (1) &= 60, & X_{d \max} (2) &= 60; \end{aligned}$$

where: X represents gray level; the parenthetical numbers (1) and (2) represent Cameras 1 and 2, respectively; subscripts “min” and “max” indicate minimum and maximum; and subscript “d” denotes “desired”. For this application we take as acceptable variations of the minimum gray level and maximum gray level, ± 2 from the desired values.

Each control implementation cycle requires approximately 0.5 seconds. It starts with taking a picture of standard vision target and sending the image data to the central control unit. The central control unit then analyzes the image data and forms the histogram for this image. This histogram is used to simultaneously update the threshold, gain, and offset in this cycle. Threshold is updated in accordance with a desired black pixel count of the standard vision target, while gain and offset are corrected based on desired minimum and maximum gray levels of the standard vision target simultaneously. After the parameters are updated, this implementation cycle is ended, and the vision system is ready for next cycle.

In the following section of this chapter, the design of the common threshold control section used will be discussed. This will be followed by presentations for the 5 different designs for gain and offset control.

Threshold Control

As discussed in Chapter 2, there is a certain gray level value, an integer ranging from 0 – 127, associated with each pixel of the image returned by the camera. When a vision system works in the binary processing mode, setting a certain threshold value determines whether a given pixel will be seen as black (gray level value below the threshold), or as white (gray level above the threshold value). To improve a vision system’s efficiency and flexibility on recognition and inspection of different dishes, Chen (1996) proposed a dynamic control method to control threshold in real time. In general, the threshold value is adjusted according to a pre-selected desired black pixel count for a standard vision target, which is represented as P_d . This number is

selected by trial and error with different vision objects. In this research, the P_d is set to 3900, since this setting gave better binary image in our dish inspection and recognition experiment.

For each control adjustment cycle, a new picture is taken, and the image of the standard target is returned to the central control unit. After the histogram for this image is formed, the controller accumulates the pixel count P , beginning with the pixels associated with gray level 0 and increasing gray level by 1 at each counting iteration, until the actual pixel count P is not less than desired black pixel count P_d . At this time the gray level value has increased from 0 to a certain value T_f , and all pixels with gray level smaller than T_f would be seen as black if the threshold is equal T_f . As we discussed in the previous chapter, any pixel with gray level smaller than the threshold value is seen as black. At the end of pixel counting process, the final gray level value T_f is chosen as the new threshold. The basic algorithm is due to Chen (1996) and is as follows:

At the i^{th} control implementation cycle, let $H[T]$ be the number of pixels in the image histogram associated with gray level T in the histogram.

Then : $P = H[T], T = 0;$

While ($P < P_d$ and $T < 127$)

$T = T + 1$

$P = P + H[T]$

$T_f = T;$

threshold = T_f ; (3.1)

This algorithm shows that threshold is updated at each control implementation cycle in accordance with the desired black pixel count, using the histogram of the newest image. This works together with the gain and offset control section to adjust vision parameters in each control

implementation cycle, forcing the minimum and maximum gray levels to desired values. We employ this threshold control algorithm for all of the gain and offset controllers discussed in later sections of this chapter.

Gain and Offset Control

Proportional-Integral-Derivative Controller

The PID (Proportional, Integral and Derivative) (Kuo, 1995) control technique is widely used in industry. It is powerful, relatively simple to design, and easy to implement. This technique can yield fast response while eliminating overshoot, and yields zero steady state error for step inputs. For our first controller, we apply conventional PID control for gain and offset, shown by the block diagram in Figure 3.1. Note that this constitutes a double-input, double-output system. The two control inputs are desired minimum gray level $X_{d\min}$ and desired maximum gray level $X_{d\max}$, while the outputs are actual minimum gray level and actual maximum gray level, $X_{\min i}$ and $X_{\max i}$, respectively.

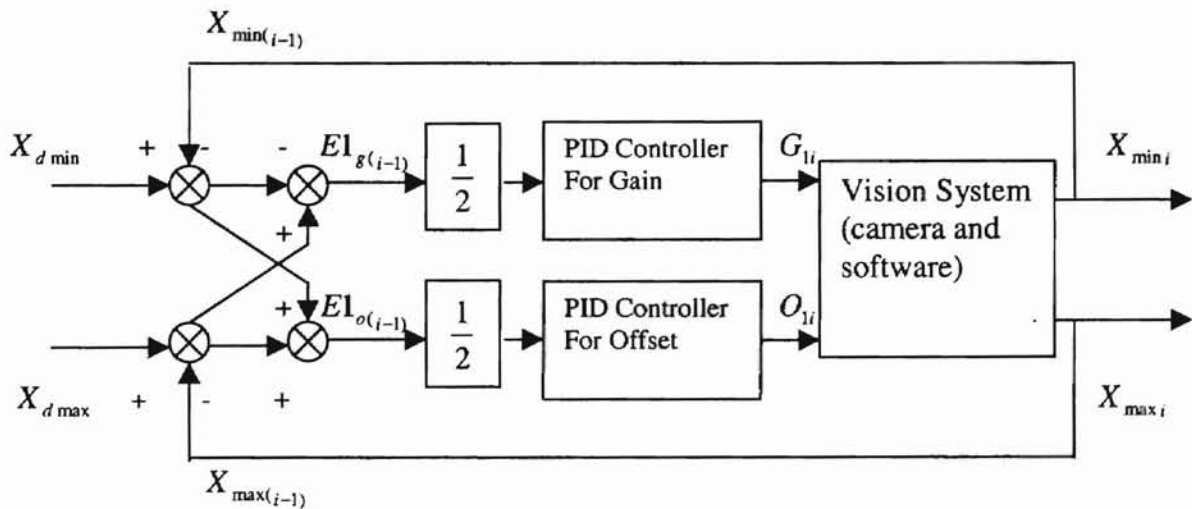


Figure 3.1

Block Diagram for PID Control System

For a certain object, the minimum and maximum gray levels of the image determine the overall characteristics of the pixel gray level distribution, and they are inter-related in an unknown manner. By experimenting with our vision system, we found that the minimum and maximum gray levels would change simultaneously if the parameters gain or offset were adjusted. If we used an error signal based solely on a control input of minimum (or maximum) gray level, conflicts would occur between minimum and maximum gray level control. We experimented with different approaches to form the control input that could combine errors from both minimum and maximum gray levels together. We determined that reasonable results could be obtained using the error signals $E1_{g(i-1)}$ and $E1_{o(i-1)}$, given by:

$$E1_{g(i-1)} = ((X_{\min(i-1)} - X_{d \min}) + (X_{d \max} - X_{\max(i-1)})) / 2; \quad (3.2)$$

$$\text{and } E1_{o(i-1)} = ((X_{d \min} + X_{d \max}) - (X_{\min(i-1)} + X_{\max(i-1)})) / 2; \quad (3.3)$$

in which the subscripts "o" and "g" refer to offset and gain, respectively, and the subscript (j-1) refers to the $(j-1)^{th}$ control adjustment cycle.

Normally, we would expect to have a mathematical model of the camera dynamics before designing a PID controller. However, for the cameras used herein, there is no such model available. Our experiments demonstrated that camera dynamics were time-variant, such that we felt dealing directly with the real cameras would be more productive than identifying a time-varying linear or nonlinear model. Accordingly, we chose to use tuning by trial and error of the PID controller gains operating with the real system, similar to practice in the process control industry (Dorf, 1995). We first set the controller gains of integral and derivative portions in camera gain and offset controllers to zero, and set the proportional gain to a certain value, then run transit response test on this PID control system, which will be discussed in next chapter. According to the transit response testing results, the proportional gains in camera gain and offset controllers were adjusted, and transit response test was re-done, until the optimum proportional

gain was found. After proportional gain was set, the integral portion gain was set in the similar way, and finally the derivative portion gain was tuned.

The outputs of the PID controller are given by:

$$G_{1i} = G_{1(i-1)} + \Delta G_{1i} \quad (3.4)$$

$$O_{1i} = O_{1(i-1)} + \Delta O_{1i} \quad (3.5)$$

where: G_{1i} and O_{1i} are the outputs of the gain and offset controllers, respectively at time step i ; $G_{1(i-1)}$ and $O_{1(i-1)}$ are the corresponding outputs at cycle $i-1$; and ΔG_{1i} and ΔO_{1i} are the change of gain and offset respectively, at cycle i , given by:

$$\Delta G_{1i} = K_{gP} E1_{g(i-1)} + K_{gD} DE_{g(i-1)} + K_{gI} IE_{g(i-1)} \quad (3.6)$$

$$\Delta O_{1i} = K_{oP} E1_{o(i-1)} + K_{oD} DE_{o(i-1)} + K_{oI} IE_{o(i-1)} \quad (3.7)$$

where: K_{gP} , K_{oP} are proportional control constants for gain and offset, respectively;

K_{gD} , K_{oD} are derivative control constants for gain and offset, respectively;

K_{gI} , K_{oI} are integral control constants for gain and offset, respectively;

$E1_{g(i-1)}$, $E1_{o(i-1)}$ are errors defined by (3.2) and (3.3), respectively, at cycle $i-1$.

The quantities $DE_{g(i-1)}$, and $DE_{o(i-1)}$ are approximations to time derivatives of error signals for gain and offset respectively, at cycle $(i-1)$ using the backward derivative method (Gerald, 1994), given by:

$$DE_{g(i-1)} = (E1_{g(i-1)} - E1_{g(i-2)}) / h \quad (3.8)$$

$$\text{and } DE_{o(i-1)} = (E1_{o(i-1)} - E1_{o(i-2)}) / h \quad (3.9)$$

where: $E1_{g(i-2)}$, $E1_{o(i-2)}$ are error signals at cycle $(i-2)$; and h is time interval in seconds. The quantities $IE_{g(i-1)}$, and $IE_{o(i-1)}$ are approximations to time integrals of error signals of gain and

offset, respectively, at cycle (i-1) using the Simpson $\frac{1}{3}$ rule (Gerald, 1994), given by:

$$IE_{g(i-1)} = (E1_{g(i-3)} + 4E1_{g(i-2)} + E1_{g(i-1)}) h / 3 \quad (3.10)$$

$$\text{and } IE_{o(i-1)} = (E1_{o(i-3)} + 4E1_{o(i-2)} + E1_{o(i-1)}) h / 3 \quad (3.11)$$

where: $E1_{g(i-3)}$, $E1_{o(i-3)}$ are error signals of gain and offset respectively, at cycle (i-3); $E1_{g(i-2)}$, $E1_{o(i-2)}$ are error signals of gain and offset respectively, at cycle (i-2); and h is the time interval.

In (3.6) and (3.7), the proportional, derivative, and integral control constants are determined by trial and error. The final values chosen for this PID controller are listed in Tables 3.1 and 3.2.

Table 3.1

Gain Values for Camera Gain Control

K_{gP}	K_{gD}	K_{gI}
3.8	0.05	0.1

Table 3.2

Gain Values for Offset Control

K_{oP}	K_{oD}	K_{oI}
1.0	0.05	0.1

Note that the derivative and integral control constants in table 3.1 and 3.2 are relatively small. If larger values are used, the control system outputs exhibit large overshoot or undershoot. If zero values for these constants are used, the controllers would exhibit very slow control correction, and non-zero steady state error would occur.

Fuzzy Logic 1 Controller

Since 1990, several investigators experimented with different methods to stabilize camera dynamics. Chen (1996) first used a fuzzy logic control technique to control gain and offset. This

controller can adjust the vision parameters to force the minimum and maximum gray levels to desired values, but it requires excessive time to recover from disturbances and yields large values of integral square error. In most applications this is unacceptable. For comparison, we included Chen's fuzzy logic controller in the present investigation, using different testing scenarios than he used to compare with the other 4 controllers designed in this work. We name Chen's controller the FL1 controller (FL1C), described in what follows.

Figure 3.2 shows a block diagram of the FL1C system. Basically it employs a two-layer linguistic rule base for two inputs, producing two outputs, designated the same as in Figure 3.1.

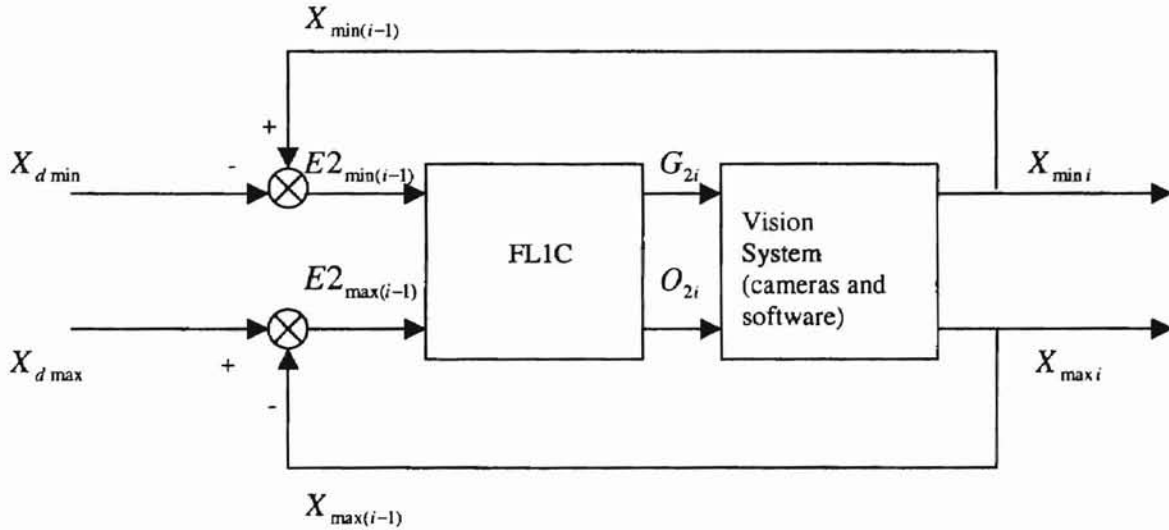


Figure 3.2

Block Diagram of The Fuzzy Logic 1 Control System

The two fuzzy error variables $E2_{\min(i-1)}$ and $E2_{\max(i-1)}$ are error signals of minimum and maximum gray levels, respectively, at cycle (i-1), given by:

$$E2_{\min(i-1)} = X_{\min(i-1)} - X_{d \min} \quad (3.12)$$

$$E2_{\max(i-1)} = X_{d \max} - X_{\max(i-1)} \quad (3.13)$$

where: $X_{\min(i-1)}$, $X_{\max(i-1)}$ are the actual minimum and maximum gray levels, respectively, at cycle (i-1). The outputs G_{2i} and O_{2i} of FL1 controller are new gain and offset values at cycle i, given by:

$$G_{2i} = G_{2(i-1)} + \Delta G_{2i} \quad (3.14)$$

$$\text{and } O_{2i} = O_{2(i-1)} + \Delta O_{2i} \quad (3.15)$$

where: $G_{2(i-1)}$, $O_{2(i-1)}$ are the gain and offset values, respectively, at cycle (i-1); and ΔG_{2i} and ΔO_{2i} are the two fuzzy control variables, change of gain and change of offset, respectively, determined from a fuzzy rule base, described in what follows.

The fuzzy logic controller uses IF-THEN rules to form the two layers of the rule base. Typically, an IF-THEN rule expresses an inference, such that if we know a fact (premise, hypothesis, antecedent), then we can infer, or derive, another fact called a conclusion (consequent) (Ross, 1995). The two layer rule base for two fuzzy error variables and two control variables in our FL1 controller can be represented as:

If $E2_{\min(i-1)}$ and $E2_{\max(i-1)}$ are in layer j,

$$\text{Then, } \{ \text{If } E2_{\min(i-1)} = A_{1k}^j \text{ and } E2_{\max(i-1)} = A_{2k}^j, \text{ Then, } \Delta G_{2i} = B_{1k}^j \text{ and } \Delta O_{2i} = B_{2k}^j \} \quad (3.16)$$

where: subscript $k = 1, 2, \dots, n$; n is the number of total rules; j is layer number with value 1 or 2; A_{1k}^j and A_{2k}^j are linguistic values of fuzzy state variables $E2_{\min(i-1)}$ and $E2_{\max(i-1)}$ in the universe of discourse $U1^j$ and $U2^j$; B_{1k}^j and B_{2k}^j are linguistic values of control variables ΔG_{2i} and ΔO_{2i} in the universe of discourse $V1^j$ and $V2^j$; Table 3.3 and Table 3.4 show the linguistic rules of the first layer and second layer of the FL1 controller, as given by Chen (1996).

Table 3.3

Linguistic Rules of 1st Layer of FL1 Controller (Chen, 1996)

$E2_{\min(i-1)} \backslash E2_{\max(i-1)}$	NL	NM	NS	ZE	PS	PM	PL
NL	GNL OZE	GNM OZE	GNM OPS	GNS OPS	GNS OPM	GZE OPM	GZE OPL
NM	GNM ONS	GNM OZE	GNS OZE	GNS OPS	GZE OPS	GZE OPM	GPS OPL
NS	GNM ONS	GNS ONS	GNS OZE	GNS OZE	GZE OPS	GZE OPM	GPS OPM
ZE	GNS ONM	GNS ONS	GZE ONS	**	GPS OPS	GPS OPS	GPS OPM
PS	GNS ONM	GZE ONM	GZE ONS	GZE ONS	GPS OZE	GPS OZE	GPM OPS
PM	GZE ONL	GZE ONM	GZE ONS	GPS ONS	GPS OZE	GPM OZE	GPM OPS
PL	GZE ONL	GZE ONM	GPS ONM	GPM ONM	GPM ONS	GPM OZE	GPL OZE

Where: $G = \Delta G_{2i}$; $O = \Delta O_{2i}$; N = Negative; P = Positive; L, M, S, ZE represent Large, Medium,

Small, and Zero, respectively; and ** means pass to the second layer, Table 3.4.

Table 3.4

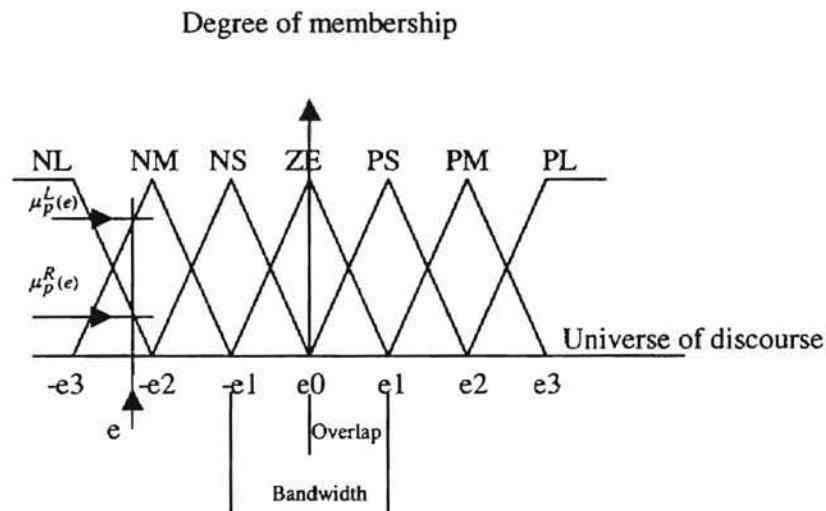
Linguistic Rules of 2nd Layer of FL1 Controller(Chen, 1996)

$E2_{\min(i-1)} \backslash E2_{\max(i-1)}$	NZ	ZE	PZ
NZ	GZE OZE	GNZ OPZ	GZE OPZ
ZE	GNZ ONZ	GZE OZE	GPZ OPZ
PZ	GZE ONZ	GPZ ONZ	GPZ OZE

Where: NZ denotes Negative Zero, and PZ represents Positive Zero.

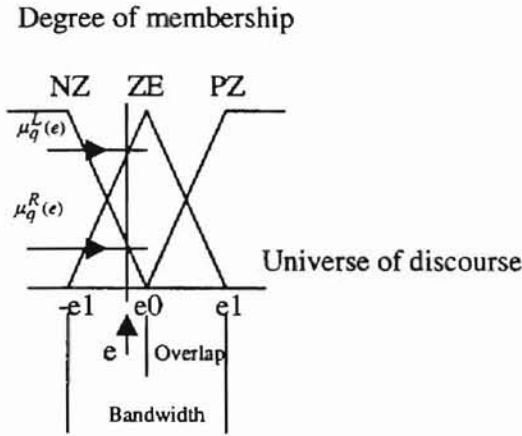
As an example in interpreting table 3.3, consider the entries in the 3rd column, 6th row. This should be read, " IF $E2_{\min(i-1)}$ is negative small (NS) and $E2_{\max(i-1)}$ is positive medium (PM), the change of gain ΔG_{2i} should be set to zero (ZE) and the change of offset ΔO_{2i} should be set to negative small (NS)." If both $E2_{\min(i-1)}$ and $E2_{\max(i-1)}$ are ZERO in the first layer, the control action is passed to the second layer, Table 3.4.

Membership functions are used to transform “crisp” quantities into fuzzy sets in the process of fuzzification, and fuzzy sets back into crisp quantities in the process of defuzzification. An isosceles triangle is chosen for our membership functions, following typical practice (Ross, 1995). A 50% overlap in the membership functions of all the fuzzy sets is used so that at any given point of the universe of discourse, two degrees of membership from 2 membership functions are obtained. Figure 3.3 (a) illustrates this for our first layer. The universe of discourse of inputs and outputs is partitioned into seven membership functions corresponding to seven linguistic variables (NL, NM, NS, ZE, PS, PM, PL) in the first layer as shown in Figure 3.3 (a). Similarly, the universe of discourse of inputs and outputs in the second layer is partitioned into three membership functions, corresponding to three linguistic variables (NZ, ZE, PZ) as illustrated in Figure 3.3 (b). In Figure 3.3, the term “e” denotes $E2_{\min(i-1)}$ or $E2_{\max(i-1)}$ and ΔG_{2i} or ΔO_{2i} .



(a) Fuzzy Sets for the First Layer (Chen, 1996)

Figure 3.3



(b) fuzzy sets for the second layer (Chen, 1996)

Figure 3.3

In Figure 3.3 (a) the degree of membership $\mu_p^L(e)$ for the leftmost fuzzy set in the first layer of the FLC can be calculated by:

$$\mu_p^L(e) = \begin{cases} 1 & p=0 \text{ or } 7 \\ (B(p-3)-e)/B & \text{otherwise} \end{cases} \quad (3.17)$$

$$p = \min[7, \max(0, \text{INT}((e+4B)/B))] \quad (3.18)$$

where: superscript L denotes leftmost; B is half of the bandwidth of the triangular membership functions; INT denotes integer; \min and \max represent minimum and maximum respectively. The degree of membership for rightmost fuzzy set in the first layer, $\mu_p^R(e)$ is the complement of that for the leftmost, given by:

$$\mu_p^R(e) = 1 - \mu_p^L(e) \quad (3.19)$$

The degree of membership for the leftmost fuzzy set in the second layer of the FLC can also be calculated by:

$$\mu_q^L(e) = \begin{cases} 1 & q=0 \text{ or } 3 \\ (B(q-2)-e)/B & \text{otherwise} \end{cases} \quad (3.20)$$

$$q = \min[3, \max(0, \text{INT}((e+3B)/B))] \quad (3.21)$$

The degree of membership for rightmost fuzzy set in the second layer, $\mu_q^R(e)$, is the complement of that for the leftmost, given by:

$$\mu_q^R(e) = 1 - \mu_q^L(e) \quad (3.22)$$

Using Tables 3.3 and 3.4, the FLIC can be represented as two double-input single-output systems. The relevant relationships are:

$$\mu_{R_{1k}^j}(E2_{\min(i-1)}, E2_{\max(i-1)}, \Delta G_{2i}) = \mu_{A_{1k}^j \times A_{2k}^j}(E2_{\min(i-1)}, E2_{\max(i-1)}) \rightarrow \mu_{B_{1k}^j}(\Delta G_{2i}) \quad (3.23)$$

$$\mu_{R_{2k}^j}(E2_{\min(i-1)}, E2_{\max(i-1)}, \Delta O_{2i}) = \mu_{A_{1k}^j \times A_{2k}^j}(E2_{\min(i-1)}, E2_{\max(i-1)}) \rightarrow \mu_{B_{2k}^j}(\Delta O_{2i}) \quad (3.24)$$

where: μ is degree of membership; k is k^{th} rule; and j denotes the j^{th} layer; $A_{1k}^j \times A_{2k}^j$ is a fuzzy set in $U1^j \times U2^j$; and \rightarrow denotes a fuzzy implication. Finally center of sums (COS) method (Hellendoorn, 1993) is used to do the defuzzification (Hellendoorn, 1993), and control variables ΔG_{2i} and ΔO_{2i} are calculated as below:

$$\Delta G_{2i} = \sum_{s=1}^m y_{1s}^j \sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \circ \mu_{R_{1k}^j}(y_{1s}^j) / \sum_{s=1}^m \sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \circ \mu_{R_{1k}^j}(y_{1s}^j) \quad (3.25)$$

$$\Delta O_{2i} = \sum_{s=1}^m y_{2s}^j \sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \circ \mu_{R_{2k}^j}(y_{2s}^j) / \sum_{s=1}^m \sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \circ \mu_{R_{2k}^j}(y_{2s}^j) \quad (3.26)$$

where: y_{1s}^j is the discrete point of ΔG_{2i} in the universe of discourse $V1^j$; y_{2s}^j is the discrete point of ΔO_{2i} in the universe of discourse $V2^j$; m is the number of the discrete points; and n is the number of the rules, here $n = 4$; and \circ denotes the Max-Min composition operator.

Fuzzy Logic 2 Controller

As we discussed in the previous section, the FL1 controller is a two layer fuzzy logic controller. While first layer uses 7 membership functions, the second layer only uses 3 member functions. The range of fuzzy error variables ($E2_{\min(i-1)}$, and $E2_{\max(i-1)}$) and fuzzy control variables (ΔG_{2i} and ΔO_{2i}) covered are relative small, so the controller demonstrates very slow control correction with large integral square error, especially when ambient lighting disturbances occur. In the second layer, there are only 3 membership functions used to implement adjustment

of vision parameters. Such adjustment is not sufficiently fine. In general the FL1 controller can adjust the vision parameters to force the minimum and maximum gray levels to desired values, but it requires excessive time to recover from disturbances, yielding large integral square error. In most applications, this is unacceptable. In order to overcome the disadvantage of the FL1 controller, in this section we design a new fuzzy logic controller, namely the Fuzzy Logic 2 Controller (FL2C). Basically FL2C is a linguistic rule base, three layer, double-input, double-output controller. It uses 3 layers to cover a much wider range of fuzzy error variables and fuzzy control variables than does the FL1 controller, and 9 membership functions in each layer are employed to implement much finer adjustment. Analytically we expect FL2C to function much better than FL1C, which is demonstrated in the next chapter.

Figure 3.4 shows a block diagram of the FL2C system. The two inputs ($X_{d\min}$, $X_{d\max}$) and two outputs ($X_{\min i}$, $X_{\max i}$) are the same as employed for the PID and FL1 controllers.

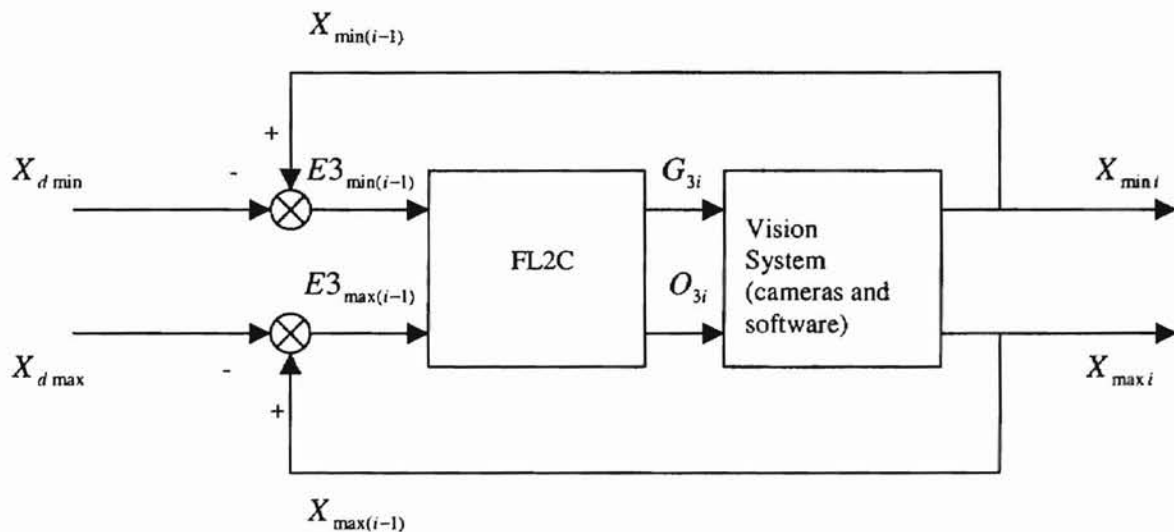


Figure 3.4

Block Diagram for the Fuzzy Logic 2 Control System

The two fuzzy error variables $E3_{\min(i-1)}$ and $E3_{\max(i-1)}$ are error signals of minimum and maximum gray levels at time step (i-1), given by:

$$E3_{\min(i-1)} = X_{\min(i-1)} - X_{d \min} \quad (3.27)$$

$$E3_{\max(i-1)} = X_{\max(i-1)} - X_{d \max} \quad (3.28)$$

The outputs G_{3i} and O_{3i} of FL2 controller are new gain and offset at time step i , given by:

$$G_{3i} = G_{3(i-1)} + \Delta G_{3i} \quad (3.29)$$

$$\text{and } O_{3i} = O_{3(i-1)} + \Delta O_{3i} \quad (3.30)$$

where: $G_{3(i-1)}$, $O_{3(i-1)}$ are the gain and offset, respectively, at cycle $(i-1)$; ΔG_{3i} and ΔO_{3i} are the two fuzzy control variables, change of gain and change of offset, respectively, at cycle i , determined from a fuzzy rule base, described in what follows.

IF-THEN rules are again used to form the three layers of the rule base, represented as:

If $E3_{\min(i-1)}$ and $E3_{\max(i-1)}$ are in layer j ,

$$\text{Then, } \{ \text{If } E3_{\min(i-1)} = A_{1k}^j \text{ and } E3_{\max(i-1)} = A_{2k}^j, \text{ Then, } \Delta G_{3i} = B_{1k}^j \text{ and } \Delta O_{3i} = B_{2k}^j \} \quad (3.31)$$

where: subscript $k = 1, 2, \dots, n$; n is the number of total rules; j is the layer number with value 1, 2 or 3; A_{1k}^j and A_{2k}^j are linguistic values of fuzzy error variables $E3_{\min(i-1)}$ and $E3_{\max(i-1)}$ in the universe of discourse $U1^j$ and $U2^j$; B_{1k}^j and B_{2k}^j are linguistic values of control variables ΔG_{3i} and ΔO_{3i} in the universe of discourse $V1^j$ and $V2^j$;

Table 3.5 (a), (b), and (c) show the linguistic rules of the 1st, 2nd, and 3rd layers of the FL2 controller.

Table 3.5 (a)

Linguistic Rules for 1st Layer of Fuzzy Logic 2 Controller

$E3_{\min(i-1)}$ / $E3_{\max(i-1)}$	NL	NM	NS	NZ	ZE	PZ	PS	PM	PL
NL	GZE OPL	GPZ OPL	GPS OPM	GPS OPM	GPS OPM	GPS OPS	GPM OPS	GPM OPZ	GPL OZE
NM	GNZ OPM	GZE OPM	GPZ OPS	GPS OPS	GPS OPS	GPS OPS	GPS OPZ	GPM OZE	GPM ONZ
NS	GNS OPM	GNZ OPS	GZE OPS	GPZ OPS	GPZ OPS	GPS OPZ	GPS OZE	GPS ONZ	GPM ONS
NZ	GNS OPS	GNS OPS	GZE OPZ	GZE OPZ	GZE OPZ	GZE OZE	GPS ONZ	GPS ONS	GPS ONS
ZE	GNS OPS	GNS OPS	GNZ OPZ	GZE OPZ	GZE OZE	GZE ONZ	GPS ONS	GPS ONS	GPS ONM
PZ	GNS OPS	GNS OPZ	GNS OPZ	GNZ OZE	GZE ONZ	GZE ONZ	GPZ ONZ	GPS ONS	GPS ONM
PS	GNM OPS	GNS OPZ	GNS OZE	GNS ONZ	GNZ ONZ	GNZ ONZ	GZE ONS	GPZ ONS	GPS ONM
PM	GNM OPZ	GNM OZE	GNS ONZ	GNS ONZ	GNS ONS	GNS ONS	GNZ ONS	GZE ONM	GPZ ONL
PL	GNL OZE	GNM ONZ	GNM ONS	GNS ONS	GNS ONS	GNS ONS	GNS ONM	GNZ ONM	GZE ONL

Where: $G = \Delta G_{3i}$; $O = \Delta O_{3i}$; N = Negative; P = Positive; L, M, S, and Z denote Large, Medium, Small, Zero respectively; ZE denotes Zero.

As an example in interpreting Table 3.5 (a), consider the entries in the 2nd column, 3rd row. This rule should be read, " IF $E3_{\min(i-1)}$ is negative medium (NM) and $E3_{\max(i-1)}$ is negative small (NS), the change of gain ΔG_{3i} should be set to negative zero (NZ) and the change of offset ΔO_{3i} should be set to positive small (PS)."

If the variations of minimum or maximum gray levels exceed the range of the 1st layer, then either the 2nd layer (Table 3.5 (b)) or the 3rd layer (Table 3.5 (c)) will be energized, and the appropriate linguistic rule will work in the same manner as in the 1st layer.

Table 3.5 (b)

Linguistic Rules for 2nd Layer of Fuzzy Logic 2 Controller

$E3_{\min(i-1)}$ / $E3_{\max(i-1)}$	NXL	NXM	NXS	NXZ	ZE	PXZ	PXS	PXM	PXL
NXL	GZE OPXL	GPXZ OPXL	GPXS OPXM	GPXZ OPXM	GPXS OPXM	GPXS OPXS	GPXM OPXS	GPXM OPXZ	GPXL OZE
NXM	GNXZ OPXM	GZE OPXM	GPXZ OPXS	GPXS OPXS	GPXS OPXS	GPXS OPXS	GPXS OPXZ	GPXM OZE	GPXM ONXZ
NXS	GNXS OPXM	GNXZ OPXS	GZE OPXS	GPXZ OPXS	GPXS OPXS	GPXS OPXZ	GPXS OZE	GPXS ONXZ	GPXM ONXS
NXZ	GNXS OPXS	GNXS OPXS	GNXZ OPXZ	GZE OPXZ	GPXZ OPXZ	GPXZ OZE	GPXS ONXZ	GPXS ONXS	GPXS ONXS
ZE	GNXS OPXS	GNXS OPXS	GNXS OPXZ	GNXZ OPXZ	GZE OZE	GPXZ ONXZ	GPXS ONXS	GPXS ONXS	GPXS ONXM
PXZ	GNXS OPXS	GNXS OPXZ	GNXS OPXZ	GNXZ OZE	GNXZ ONXZ	GZE ONXZ	GPXZ ONXZ	GPXS ONXS	GPXS ONXM
PXS	GNXM OPXS	GNXS OPXZ	GNXS OZE	GNXS ONXZ	GNXS ONXZ	GNXZ ONXZ	GZE ONXS	GPXZ ONXS	GPXS ONXM
PXM	GNXM OPXZ	GNXM OZE	GNXS ONXZ	GNXS ONXZ	GNXS ONXS	GNXS ONXS	GNXZ ONXS	GZE ONXM	GPXZ ONXL
PXL	GNXL OZE	GNXM ONXZ	GNXM ONXS	GNXS ONXS	GNXS ONXS	GNXS ONXS	GNXS ONXM	GNXZ ONXM	GZE ONXL

Where: X = larger.

Table 3.5 (c)

Linguistic Rules for 3rd Layer of Fuzzy Logic 2 Controller

$E3_{\min(i-1)}$ $E3_{\max(i-1)}$	NXXL	NXXM	NXXS	NXXZ	ZE	PXXZ	PXXS	PXXM	PXXL
NXXL	GZE OPUL	GPUZ OPUL	GPUS OPUM	GPUS OPUM	GPUS OPUM	GPUS OPUS	GPUM OPUS	GPUM OPUZ	GPUL OZE
NXXM	GNUZ OPUM	GZE OPUM	GPUZ OPUS	GPUS OPUS	GPUS OPUS	GPUS OPUS	GPUS OPUZ	GPUM OZE	GPUM ONUZ
NXXS	GNUS OPUM	GNUZ OPUS	GZE OPUS	GPUZ OPUS	GPUS OPUS	GPUS OPUZ	GPUS OZE	GPUS ONUZ	GPUM ONUS
NXXZ	GNUS OPUS	GNUS OPUS	GNUZ OPUZ	GZE OPUZ	GPUZ OPUZ	GPUZ OZE	GPUS ONUZ	GPUS ONUS	GPUS ONUS
ZE	GNUS OPUS	GNUS OPUS	GNUS OPUZ	GNUZ OPUZ	GZE OZE	GPUZ ONUZ	GPUS ONUS	GPUS ONUS	GPUS ONUM
PXXZ	GNUS OPUS	GNUS OPUZ	GNUS OPUZ	GNUZ OZE	GNUZ ONUZ	GZE ONUZ	GPUZ ONUZ	GPUS ONUS	GPUS ONUM
PXXS	GNUM OPUS	GNUS OPUZ	GNUS OZE	GNUS ONUZ	GNUS ONUZ	GNUZ ONUZ	GZE ONUS	GPUZ ONUS	GPUS ONUM
PXXM	GNUM OPUZ	GNUM OZE	GNUS ONUZ	GNUS ONUZ	GNUS ONUS	GNUS ONUS	GNUZ ONUS	GZE ONUM	GPUZ ONUL
PXXL	GNUX OZE	GNUM ONUZ	GNUM ONUS	GNUS ONUS	GNUS ONUS	GNUS ONUS	GNUS ONUM	GNUZ ONUM	GZE ONUL

Where: XX represents extremely large; U = XX.

For purposes of comparison with FL1C, FL2C also uses an isosceles triangle as the membership function with 50% overlap in all the fuzzy sets of the three layers. The universes of discourse of inputs and outputs for all three layers are partitioned into nine membership functions. Correspondingly, the nine linguistic variables are NL, NM, NS, NZ, ZE, PZ, PS, PM, PL in 1st layer; NXL, NXM, NXS, NXZ, ZE, PXZ, PXS, PXM, PXL in the 2nd layer; and NXXL, NXXM, NXXS, NXXZ, ZE, PXXZ, PXXS, PXXM, PXXL in the 3rd layer. The partitions are illustrated in Figure 3.5, where “e” denotes $E3_{\min(i-1)}$ or $E3_{\max(i-1)}$ and ΔG_{3i} or ΔO_{3i} , respectively.

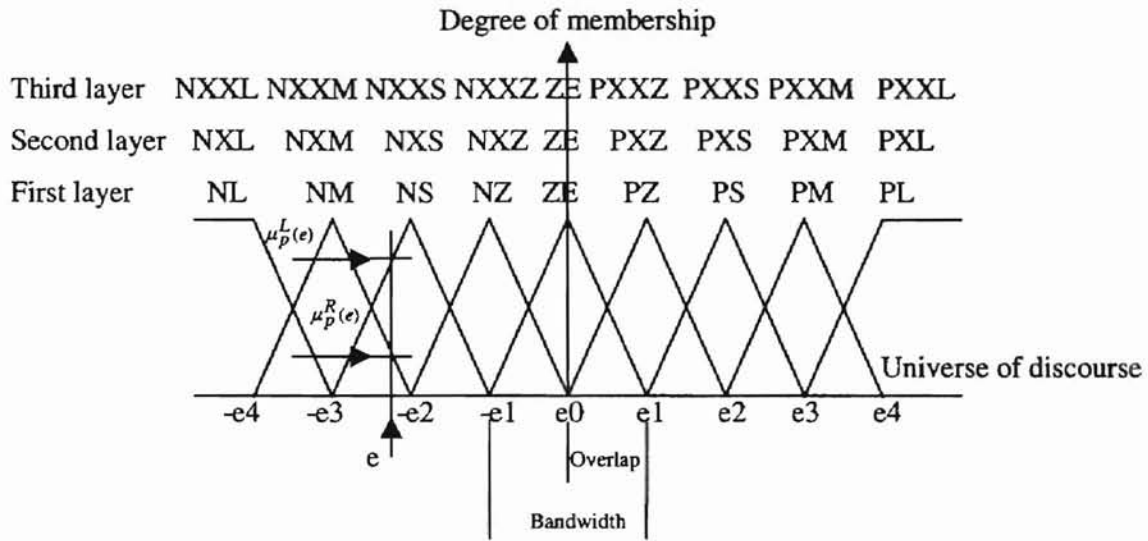


Figure 3.5

Fuzzy Sets for All Three Layers

By choosing different values of bandwidth (support) for fuzzy error variables and fuzzy control variables in the first, second and third layers, the FL2C can respond rapidly to large variation of the error inputs. On the other hand, for small deviation, this controller can make fine adjustments of the fuzzy outputs. In Figure 3.5 the degree of membership $\mu_p^L(e)$ for the leftmost fuzzy set in all three layers of the FL2C can be calculated by:

$$\mu_p^L(e) = \begin{cases} 1 & p \geq 4 \text{ or } \leq -5 \\ ((p+1)B - e) / B & \text{otherwise} \end{cases} \quad (3.32)$$

$$p = \begin{cases} INT(e/B) & e \geq 0 \\ INT(e/B - 1) & \text{otherwise} \end{cases} \quad (3.33)$$

where: L denotes leftmost; INT represents integer; B is half the bandwidth. The degree of membership for rightmost fuzzy set in all three layers, $\mu_p^R(e)$ is the complement of that for the leftmost, given by:

$$\mu_p^R(e) = 1 - \mu_p^L(e) \quad (3.34)$$

Using Table 3.5 (a), (b), and (c), the FL2C can be represented as two double-input, signal-output systems. The relevant relationships are:

$$\mu_{R_{1k}^j} (E3_{\min(i-1)}, E3_{\max(i-1)}, \Delta G_{3i}) = \mu_{A_{1k}^j \times A_{2k}^j} (E3_{\min(i-1)}, E3_{\max(i-1)}) \rightarrow \mu_{B_{1k}^j} (\Delta G_{3i}) \quad (3.35)$$

$$\mu_{R_{2k}^j} (E3_{\min(i-1)}, E3_{\max(i-1)}, \Delta O_{3i}) = \mu_{A_{1k}^j \times A_{2k}^j} (E3_{\min(i-1)}, E3_{\max(i-1)}) \rightarrow \mu_{B_{2k}^j} (\Delta O_{3i}) \quad (3.36)$$

where: μ is degree of membership; k is k^{th} rule; and j denotes j^{th} layer. Finally weighted average method (Ross, 1995) is used for defuzzification, which is reliable, accurate enough and easy to implement requiring less on line computation than that of center of sum method. And ΔG_{3i} and ΔO_{3i} are given by:

$$\Delta G_{3i} = \frac{\sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \cdot y_{1k}^j}{\sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j}} \quad (3.37)$$

$$\Delta O_{3i} = \frac{\sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j} \cdot y_{2k}^j}{\sum_{k=1}^n \mu_{A_{1k}^j \times A_{2k}^j}} \quad (3.38)$$

where: y_{1k}^j is a discrete point of ΔG_{3i} in the universe of discourse $V1^j$; y_{2k}^j is a discrete point of ΔO_{3i} in the universe of discourse $V2^j$; and n is the number of the rules, here $n = 4$;

Fuzzy-Integral-Derivative Controller

To this point, we have presented 3 controller designs, the PID, the FL1, and the FL2 controllers. While the PID controller takes advantages of proportional, integral, and derivative action, yielding fast response and small steady state error, the FL1C and FL2C use fuzzy logic techniques to implement control. With refined fuzzy sets and more layers used, the FL2C yields finer adjustment faster response than the FL1C, as will be shown in chapter 4. In the next two section of this chapter, two more control algorithms are to be investigated, namely Fuzzy-Integral-Derivative (FID) control and Fuzzy-Integral (F(E+D)+I) control. Basically, we combine PID and Fuzzy Logic techniques to take advantage of both, hoping to design a better controller.

In this section we discuss the FID controller design. Figure 3.6 shows a block diagram of the FID controller. Basically it is a double-input, double-output control system. The two inputs are desired minimum and maximum gray levels ($X_{d\min}$ and $X_{d\max}$), and the two outputs are actual minimum and maximum gray levels (X_{\min} and X_{\max}), which are the same as we used in the other 3 control system. There are two FID portions in this control system, one for gain control, and the other for offset control. Each FID portion consists of 3 parallel parts, fuzzy logic, derivative, and integral. The fuzzy logic control part functions just like the proportional part in PID controller. While proportional control gives control action only proportional to the error signal, a linear correction, the fuzzy logic control can apply nonlinear control to the system. This may help us to find improved control performance.

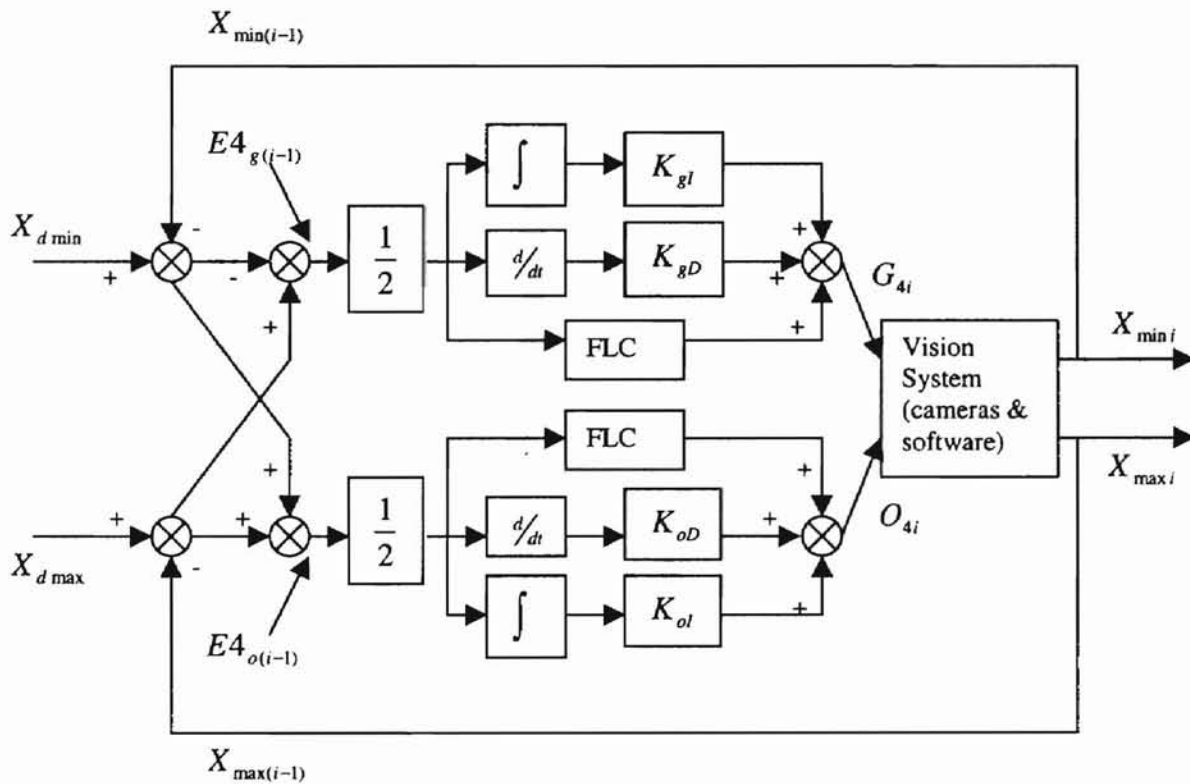


Figure 3.6

Block Diagram for the FID Control System

The FID controller uses similar control input signals to those used in the PID controller, designated as $E4_{g(i-1)}$ and $E4_{o(i-1)}$. The calculation is given by:

$$E4_{g(i-1)} = ((X_{\min(i-1)} - X_{d \min}) + (X_{d \max} - X_{\max(i-1)})) / 2; \quad (3.39)$$

$$\text{and } E4_{o(i-1)} = ((X_{d \min} + X_{d \max}) - (X_{\min(i-1)} + X_{\max(i-1)})) / 2; \quad (3.40)$$

The outputs G_{4i} and O_{4i} of the FID controller are the gain and offset at cycle i , given by:

$$G_{4i} = G_{4(i-1)} + \Delta G_{4i} \quad (3.41)$$

$$\text{and } O_{4i} = O_{4(i-1)} + \Delta O_{4i} \quad (3.42)$$

where: $G_{4(i-1)}$, $O_{4(i-1)}$ are the gain and offset, respectively, at cycle $(i-1)$; and ΔG_{4i} and ΔO_{4i} are change of gain and change of offset, respectively, at cycle i , determined by the following equations:

$$\Delta G_{4i} = \Delta G_{4Fi} + K_{gD} DE_{g(i-1)} + K_{gI} IE_{g(i-1)} \quad (3.43)$$

$$\Delta O_{4i} = \Delta O_{4Fi} + K_{oD} DE_{o(i-1)} + K_{oI} IE_{o(i-1)} \quad (3.44)$$

where: Derivative and integral parts calculation are similar to those used in the PID controller, $DE_{g(i-1)}$, and $DE_{o(i-1)}$ are approximations to time derivatives of error signals of gain and offset at time step $(i-1)$ using the backward derivative method (Gerald, 1994), given by equations similar to (3.8) and (3.9). The quantities $IE_{g(i-1)}$, and $IE_{o(i-1)}$ are approximations to time integrals of error signals of gain and offset at time step $(i-1)$ using the Simpson $\frac{1}{3}$ rule (Gerald, 1994), given by equations similar to (3.10) and (3.11). K_{gD} , K_{oD} are the derivative control constants for gain and offset, respectively, and K_{gI} , K_{oI} are integral control constants for gain and offset, respectively. After trial and error, the constants were chosen as listed in Table 3.6.

Table 3.6

Gain Values for the FID Controller

K_{gD}	K_{gl}	K_{oD}	K_{ol}
0.01	0.1	0.01	0.1

ΔG_{4Fi} and ΔO_{4Fi} are fuzzy outputs from two fuzzy logic controllers, which in fact are fuzzy control variables. The fuzzy logic controllers are described as follows. In general, each FID portion employs a three-layer, single-input, single-output fuzzy logic controller. The fuzzy error variable is $E4_{g(i-1)}$ for gain, and $E4_{o(i-1)}$ for offset. The fuzzy control variables are ΔG_{4Fi} for gain and ΔO_{4Fi} for offset, determined by a fuzzy rule base, described below. IF-THEN rules are used to form the three layers of the rule base. For gain control, the rule base for one fuzzy state variable and one fuzzy control variable can be represented as:

if $E4_{g(i-1)}$ is in layer j,

$$\text{Then, \{if } E4_{g(i-1)} = A_{1k}^j, \text{ Then } \Delta G_{4Fi} = B_{1k}^j \text{ \}}$$
 (3.45)

where $k = 1, 2, \dots, n$; n is the number of total rules; $j = 1, 2, 3$; A_{1k}^j is the linguistic value of fuzzy error variable $E4_{g(i-1)}$ in the universe of discourse $U1^j$; and B_{1k}^j is the linguistic value of control variable ΔG_{4Fi} in the universe of discourse $V1^j$. For offset control, the rule base for one fuzzy state variable and one fuzzy control variable can be represented as:

if $E4_{o(i-1)}$ is in layer j,

$$\text{Then, \{if } E4_{o(i-1)} = A_{2k}^j, \text{ Then } \Delta O_{4Fi} = B_{2k}^j \text{ \}}$$
 (3.46)

where $k = 1, 2, \dots, n$; n is the number of total rules; $j = 1, 2, 3$; A_{2k}^j is the linguistic value of fuzzy state variable $E4_{o(i-1)}$ in the universe of discourse $U2^j$; and B_{2k}^j is linguistic value of control

variable ΔO_{4Fi} in the universe of discourse $V2^j$. Table 3.7 (a), (b), and (c) show the linguistic rules of the 1st, 2nd and 3rd layers of the gain fuzzy controller.

Table 3.7 (a)

Linguistic Rules of 1st Layer of Gain Fuzzy Logic Controller

$E4_{g(i-1)}$	NL	NM	NS	NZ	ZE	PZ	PS	PM	PL
ΔG_{4Fi}	GNL	GNM	GNS	GZE	GZE	GZE	GPS	GPM	GPL

Where: G = ΔG_{4Fi} ; N = Negative; P = Positive; L, M, S, and Z denote Large, Medium, Small, Zero respectively; ZE denotes Zero.

Table 3.7 (b)

Linguistic Rules of 2nd Layer of Gain Fuzzy Logic Controller

$E4_{g(i-1)}$	NXL	NXM	NXS	NXZ	ZE	PXZ	PXS	PXM	PXL
ΔG_{4Fi}	GNXL	GNXM	GNXS	GNXZ	GZE	GPXZ	GPXS	GPXM	GPXL

Where: X denotes larger.

Table 3.7 (C)

Linguistic Rules of 3rd Layer of Gain Fuzzy Logic Controller

$E4_{g(i-1)}$	NUL	NUM	NUS	NUZ	ZE	PUZ	PUS	PUM	PUL
ΔG_{4Fi}	GNUL	GNUM	GNUS	GNUZ	GZE	GPUZ	GPUS	GPUM	GPUL

Where: U = XX, denotes extremely large.

If the value of $E4_{g(i-1)}$ exceeds the range of the 1st layer, then either the 2nd layer (Table 3.7 (b)) or the 3rd layer (Table 3.7 (c)) will be energized, and the appropriate linguistic rule will work in the same way as for the 1st layer.

Table 3.8 (a), (b), and (c) show the linguistic rules of the 1st, 2nd and 3rd layers of the offset fuzzy controller.

Table 3.8 (a)

Linguistic Rules of 1st Layer of Offset Fuzzy Logic Controller

$E4_{o(i-1)}$	NL	NM	NS	NZ	ZE	PZ	PS	PM	PL
ΔO_{4Fi}	ONL	ONM	ONS	ONZ	OZE	OPZ	OPS	OPM	OPL

Where: O = ΔO_{4Fi} ; N = Negative; P = Positive; L, M, S, and Z denote Large, Medium, Small, Zero respectively; ZE denotes Zero.

Table 3.8 (b)

Linguistic Rules of 2nd Layer of Offset Fuzzy Logic Controller

$E4_{o(i-1)}$	NXL	NXM	NXS	NXZ	ZE	PXZ	PXS	PXM	PXL
ΔO_{4Fi}	ONXL	ONXM	ONXS	ONXZ	OZE	OPXZ	OPXS	OPXM	OPXL

Where: X denotes larger.

Table 3.8 (C)

Linguistic Rules of 3rd Layer of Offset Fuzzy Logic Controller

$E4_{o(i-1)}$	NUL	NUM	NUS	NUZ	ZE	PUZ	PUS	PUM	PUL
ΔO_{4Fi}	ONUL	ONUM	ONUS	ONUZ	OZE	OPUZ	OPUS	OPUM	OPUL

Where: U = XX, denotes extremely large.

If the value of $E4_{o(i-1)}$ exceeds the range of the 1st layer, then either the 2nd layer (Table 3.8 (b)) or the 3rd layer (Table 3.8 (c)) will be energized, and appropriate linguistic rule will work in the same way as for the 1st layer.

As for the FL1C and FL2C, an isosceles triangle is chosen as the membership function with 50% overlap in all the fuzzy sets. The universe of discourse of input and output in both gain and offset fuzzy logic controllers are partitioned into nine membership functions. Corresponding

to the nine membership functions, there are nine linguistic variables. They are NL, NM, NS, NZ, ZE, PZ, PS, PM, PL in the 1st layer; NXL, NXM, NXS, NXZ, ZE, PXZ, PXS, PXM, PXL in the 2nd layer; and NXXL, NXXM, NXXS, NXXZ, ZE, PXXZ, PXXS, PXXM, PXXL in the 3rd layer. This is the same strategy used in FL2C, illustrated in Figure 3.5, where “e” denotes $E4_{g(i-1)}$ or $E4_{o(i-1)}$ and ΔG_{4Fi} or ΔO_{4Fi} . Again, this 3 layer rule base design gives us the flexibility of choosing different values of bandwidth (support) for fuzzy input and output variables in the 1st, 2nd and 3rd layers to achieve faster response for large variation, as well as fine adjustment for small deviations. In Figure 3.5 the degree of membership $\mu_p^L(e)$ for the leftmost fuzzy set in all three layers of the gain and offset fuzzy logic controllers is calculated similar to (3.32) and (3.33). The degree of membership for rightmost fuzzy set in all three layers, $\mu_p^R(e)$ is the complement of that for the leftmost, given by a relation similar to (3.34)

Based on Table 3.7 and 3.8, the relevant relationships for single-input, single-output fuzzy logic controllers for gain and offset are given by:

$$\mu_{R_{1k}^j}(E4_{g(i-1)}, \Delta G_{4Fi}) = \mu_{A_{1k}^j}(E4_{g(i-1)}) \rightarrow \mu_{B_{1k}^j}(\Delta G_{4Fi}) \quad (3.47)$$

$$\mu_{R_{2k}^j}(E4_{o(i-1)}, \Delta O_{4Fi}) = \mu_{A_{2k}^j}(E4_{o(i-1)}) \rightarrow \mu_{B_{2k}^j}(\Delta O_{4Fi}) \quad (3.48)$$

where: μ is the degree of membership; k is k^{th} rule; and j denotes j^{th} layer.

Finally the weighted average method (Ross, 1995) is used for defuzzification as for the FL2C. ΔG_{4Fi} and ΔO_{4Fi} are given by:

$$\Delta G_{4Fi} = \frac{\sum_{k=1}^n \mu_{A_{1k}^j} \cdot y_{1k}^j}{\sum_{k=1}^n \mu_{A_{1k}^j}} \quad (3.49)$$

$$\Delta O_{4Fi} = \frac{\sum_{k=1}^n \mu_{A_{2k}^j} \cdot y_{2k}^j}{\sum_{k=1}^n \mu_{A_{2k}^j}} \quad (3.50)$$

where: y_{1k}^j is a discrete point of ΔG_{4Fi} in the universe of discourse $V1^j$; y_{2k}^j is a discrete point of ΔO_{4Fi} in the universe of discourse $V2^j$; and n is the number of the rules, here $n = 2$;

Fuzzy-Integral Controller

In this section we design a final new controller, which combines fuzzy logic and conventional PI control techniques. We name the controller as F(E+D)+I, where F denotes fuzzy logic controller, E represents an error signal, D denotes derivative, and I denotes integral. As we discussed before, FL1C and FL2C are double-input, double-output fuzzy logic controllers, and the FID controller is a combination of fuzzy logic control and PID control, where the fuzzy control part is single-input, single-output. In the F(E+D)+I controller design, a double-input, single-output fuzzy controller is employed, while an integral part is added to this controller to handle steady state error. This design proposes to yield a good comparison with FL1C and FL2C, where a double-input, double-output fuzzy logic controller was used, as well as with the FID controller, where single-input, single-output fuzzy logic controller was used together with integral action.

Figure 3.7 shows a block diagram of F(E+D)+I control system. The inputs ($X_{d\min}$ and $X_{d\max}$) and outputs (X_{\min} and X_{\max}) are similar to those used in the previous four controller designs. There are two portions in this control system, one for gain control and the other for offset control. Each portion consists of 2 parallel parts, a fuzzy logic part and an integral part.

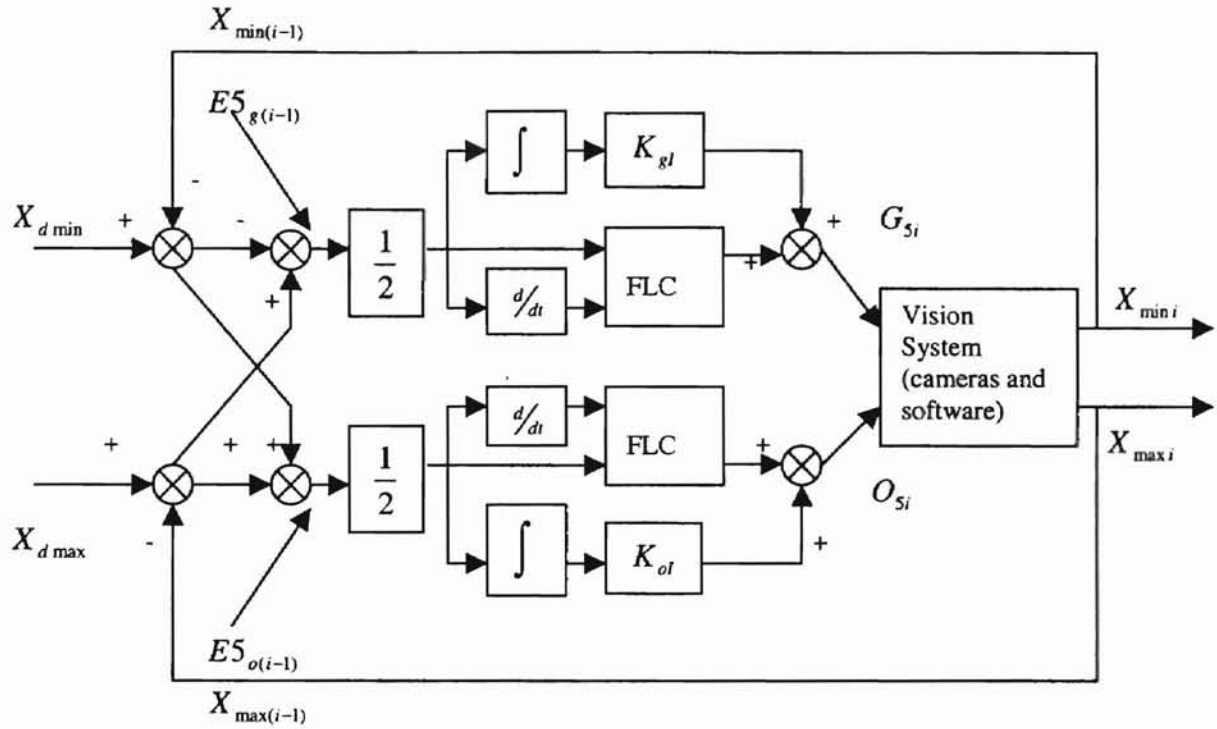


Figure 3.7

Block Diagram for the F(E+D)+I Control System

The F(E+D)+I controller uses similar control input signals as used in PID and FID controllers, designated as $E5_{g(i-1)}$ and $E5_{o(i-1)}$. The calculations are given by:

$$E5_{g(i-1)} = ((X_{\min(i-1)} - X_{d\ \min}) + (X_{d\ \max} - X_{\max(i-1)})) / 2 ; \quad (3.51)$$

$$\text{and } E5_{o(i-1)} = ((X_{d\ \min} + X_{d\ \max}) - (X_{\min(i-1)} + X_{\max(i-1)})) / 2 ; \quad (3.52)$$

The outputs G_{5i} and O_{5i} of the F(E+D)+I controller are the gain and offset at cycle i , given by:

$$G_{5i} = G_{5(i-1)} + \Delta G_{5i} \quad (3.53)$$

$$\text{and } O_{5i} = O_{5(i-1)} + \Delta O_{5i} \quad (3.54)$$

where: $G_{5(i-1)}$, $O_{5(i-1)}$ are the gain and offset, respectively, at cycle $(i-1)$; and ΔG_{5i} and ΔO_{5i} are the change of gain and change of offset, respectively, at cycle i , determined by the following equations:

$$\Delta G_{5i} = \Delta G_{5Fi} + K_{gl} IE_{g(i-1)} \quad (3.55)$$

$$\Delta O_{5i} = \Delta O_{5Fi} + K_{ol} IE_{o(i-1)} \quad (3.56)$$

where: integral parts are calculated using methods similar to that for the PID controller, the quantities $IE_{g(i-1)}$, and $IE_{o(i-1)}$ are approximations to time integrals of error signals of gain and offset at time step (i-1) using the Simpson $\frac{1}{3}$ rule (Gerald, 1994), given by equations similar to (3.10) and (3.11). K_{gl} , K_{ol} are the integral control constants for gain and offset, respectively.

After trial and error, the constants were chosen as listed in Table 3.9.

Table 3.9

Gain Value for F(E+D)+I Controllers

Kg_{int}	Ko_{int}
0.1	0.1

ΔG_{5Fi} and ΔO_{5Fi} are fuzzy outputs from two fuzzy logic controllers, which are fuzzy control variables. The fuzzy logic controllers are described as follows. Basically, each F(E+D)+I portion employs a multi-layer, double-input, single-output fuzzy logic controller. The fuzzy state variables are $E5_{g(i-1)}$ and $DE_{g(i-1)}$ for gain, $E5_{o(i-1)}$ and $DE_{o(i-1)}$ for offset. $DE_{g(i-1)}$, and $DE_{o(i-1)}$ are approximations to time derivatives of error signals of gain and offset, respectively, at cycle (i-1) using the backward derivative method (Gerald, 1994), given by equations similar to (3.8) and (3.9). The fuzzy control variable is ΔG_{5Fi} for gain, and ΔO_{5Fi} for offset is determined by the following fuzzy rule base. A three layer rule base is used for $E5_{g(i-1)}$, $E5_{o(i-1)}$, ΔG_{5Fi} , and ΔO_{5Fi} , such that the controller can cover wide ranges of error signal and fuzzy correction

output. A single-layer rule base is used for derivatives of error signals $DE_{g(i-1)}$ and $DE_{o(i-1)}$, since this was found sufficient to cover the entire range of expected variations.

As before, an IF-THEN rule form is used to form the rule base. The multi-layer rule base for two state variables and one control variable of the gain fuzzy controller can be represented as:

if $E5_{g(i-1)}$ is in layer j,

$$\text{Then, \{if } E5_{g(i-1)} = A_{1k}^j \text{ and } DE_{g(i-1)} = D_{1k} \text{ , Then } \Delta G_{5Fi} = B_{1k}^j \text{ \}} \quad (3.57)$$

where $k = 1, 2, \dots, n$; n is the number of total rules; $j = 1, 2, 3$; A_{1k}^j and D_{1k} are linguistic values of fuzzy error variables $E5_{g(i-1)}$ and $DE_{g(i-1)}$ in the universe of discourse $U1^j$, $W1$; B_{1k}^j is the linguistic value of control variable ΔG_{5Fi} in the universe of discourse $V1^j$. The multi-layer rule base for two state variables and one control variable of the offset fuzzy controller can be represented as:

if $E5_{o(i-1)}$ is in layer j,

$$\text{Then, \{if } E5_{o(i-1)} = A_{2k}^j \text{ and } DE_{o(i-1)} = D_{2k} \text{ , Then } \Delta O_{5Fi} = B_{2k}^j \text{ \}} \quad (3.58)$$

where $k = 1, 2, \dots, n$; n is the number of total rules; $j = 1, 2, 3$; A_{2k}^j and D_{2k} are linguistic values of fuzzy state variables $E5_{o(i-1)}$ and $DE_{o(i-1)}$ in the universe of discourse $U2^j$ and $W2$; and B_{2k}^j is the linguistic values of control variable ΔO_{5Fi} in the universe of discourse $V2^j$. Table 3.10 (a), (b), and (c) show the linguistic rules of the 1st layer, 2nd layer and 3rd layer of the gain fuzzy logic controller. If the value of $E5_{g(i-1)}$ exceeds the range of the 1st layer, then either the 2nd layer (Table 3.10 (b)) or the 3rd layer (Table 3.10 (c)) will be energized, and appropriate linguistic rule will work in the same way as for the 1st layer.

Table 3.10 (a)

Linguistic Rules of 1st Layer of Gain Fuzzy Logic Controller

$E S_{g(i-1)}$ $D E_{g(i-1)}$	NL	NM	NS	NZ	ZE	PZ	PS	PM	PL
NL	GNM	GNS	GNZ	GNZ	GZE	GPS	GPM	GPL	GPL
NM	GNM	GNS	GNS	GNZ	GZE	GPS	GPM	GPL	GPL
NS	GNM	GNM	GNS	GNZ	GZE	GPS	GPM	GPL	GPL
NZ	GNL	GNM	GNS	GNZ	GZE	GPZ	GPS	GPM	GPL
ZE	GNL	GNM	GNS	GNZ	GZE	GPZ	GPS	GPM	GPL
PZ	GNL	GNM	GNS	GNZ	GZE	GPZ	GPS	GPM	GPL
PS	GNL	GNL	GNM	GNS	GZE	GPZ	GPS	GPS	GPM
PM	GNL	GNL	GNM	GNM	GNZ	GPZ	GPS	GPS	GPM
PL	GNL	GNL	GNL	GNM	GNZ	GZE	GPZ	GPS	GPS

Where: $G = \Delta G_{5Fl}$; N = Negative; P = Positive; L, M, S, and Z denote Large, Medium, Small,

Zero respectively; ZE denotes Zero.

Table 3.10 (b)

Linguistic Rules of 2nd Layer of Gain Fuzzy Logic Controller

$E5_{g(i-1)}$ $DE_{g(i-1)}$	NXL	NXM	NXS	NXZ	ZE	PXZ	PXS	PXM	PXL
NL	GNXM	GNXS	GNXZ	GNXZ	GZE	GPXS	GPXM	GPXL	GPXL
NM	GNXM	GNXS	GNXS	GNXZ	GZE	GPXS	GPXM	GPXL	GPXL
NS	GNXM	GNXM	GNXS	GNXZ	GZE	GPXS	GPXM	GPXL	GPXL
NZ	GNXL	GNXM	GNXS	GNXZ	GZE	GPXZ	GPXS	GPXM	GPXL
ZE	GNXL	GNXM	GNXS	GNXZ	GZE	GPXZ	GPXS	GPXM	GPXL
PZ	GNXL	GNXM	GNXS	GNXZ	GZE	GPXZ	GPXS	GPXM	GPXL
PS	GNXL	GNXL	GNXM	GNXS	GZE	GPXZ	GPXS	GPXS	GPXM
PM	GNXL	GNXL	GNXM	GNXM	GNXZ	GPXZ	GPXS	GPXS	GPXM
PL	GNXL	GNXL	GNXL	GNXM	GNXZ	GZE	GPXZ	GPXS	GPXS

Where: X denotes larger.

Table 3.10 (c)

Linguistic Rules of 3rd Layer of Gain Fuzzy Logic Controller

$E5_{g(i-1)}$ $DE_{g(i-1)}$	NUL	NUM	NUS	NUZ	ZUE	PUZ	PUS	PUM	PUL
NL	GNUM	GNUS	GNUZ	GNUZ	GZE	GPUS	GPUM	GPUL	GPUL
NM	GNUM	GNUS	GNUS	GNUZ	GZE	GPUS	GPUM	GPUL	GPUL
NS	GNUM	GNUM	GNUS	GNUZ	GZE	GPUS	GPUM	GPUL	GPUL
NZ	GNUL	GNUM	GNUS	GNUZ	GZE	GPUZ	GPUS	GPUM	GPUL
ZE	GNUL	GNUM	GNUS	GNUZ	GZE	GPUZ	GPUS	GPUM	GPUL
PZ	GNUL	GNUM	GNUS	GNUZ	GZE	GPUZ	GPUS	GPUM	GPUL
PS	GNUL	GNUL	GNUM	GNUS	GZE	GPUZ	GPUS	GPUS	GPUM
PM	GNUL	GNUL	GNUM	GNUM	GNUZ	GPUZ	GPUS	GPUS	GPUM
PL	GNUL	GNUL	GNUL	GNUM	GNUZ	GZE	GPUZ	GPUS	GPUS

Where: U = XX, denotes extremely large.

Table 3.11 (a), (b), and (c) show the linguistic rules of the 1st, 2nd and 3rd layers of the offset fuzzy logic controller.

Table 3.11 (a)

Linguistic Rules of 1st Layer of Offset Fuzzy Logic Controller

$E_{So(i-1)}$ $DE_{o(i-1)}$	NL	NM	NS	NZ	ZE	PZ	PS	PM	PL
NL	ONM	ONS	ONZ	ONZ	OZE	OPS	OPM	OPL	OPL
NM	ONM	ONS	ONS	ONZ	OZE	OPS	OPM	OPL	OPL
NS	ONL	ONM	ONS	ONZ	OZE	OPS	OPM	OPL	OPL
NZ	ONL	ONM	ONS	ONZ	OZE	OPZ	OPS	OPM	OPL
ZE	ONL	ONM	ONS	ONZ	OZE	OPZ	OPS	OPM	OPL
PZ	ONL	ONL	ONM	ONZ	OZE	OPZ	OPS	OPM	OPL
PS	ONL	ONL	ONM	ONZ	OZE	OPZ	OPS	OPS	OPM
PM	ONL	ONL	ONL	ONS	OZE	OPZ	OPZ	OPS	OPM
PL	ONL	ONL	ONL	ONM	ONZ	OZE	OPZ	OPS	OPM

Where: O = ΔO_{SF_i} ; N = Negative; P = Positive; L, M, S, and Z denote Large, Medium, Small,

Zero respectively; ZE denotes Zero.

Table 3.11 (b)
Linguistic Rules of 2nd Layer of Offset Fuzzy Logic Controller

$ES_{o(i-1)}$ $DE_{o(i-1)}$	NXL	NXM	NXS	NXZ	ZE	PXZ	PXS	PXM	PXL
NL	ONXM	ONXS	ONXZ	ONXZ	OZE	OPXS	OPXM	OPXL	OPXL
NM	ONXM	ONXS	ONXS	ONXZ	OZE	OPXS	OPXM	OPXL	OPXL
NS	ONXL	ONXM	ONXS	ONXZ	OZE	OPXS	OPXM	OPXL	OPXL
NZ	ONXL	ONXM	ONXS	ONXZ	OZE	OPXZ	OPXS	OPXM	OPXL
ZE	ONXL	ONXM	ONXS	ONXZ	OZE	OPXZ	OPXS	OPXM	OPXL
PZ	ONXL	ONXL	ONXM	ONXZ	OZE	OPXZ	OPXS	OPXM	OPXL
PS	ONXL	ONXL	ONXM	ONXZ	OZE	OPXZ	OPXS	OPXS	OPXM
PM	ONXL	ONXL	ONXL	ONXS	OZE	OPXZ	OPXZ	OPXS	OPXM
PL	ONXL	ONXL	ONXL	ONXM	ONXZ	OZE	OPXZ	OPXS	OPXM

Where: X denotes larger.

Table 3.11 (c)
Linguistic Rules of 3rd Layer of Offset Fuzzy Logic Controller

$ES_{o(i-1)}$ $DE_{o(i-1)}$	NUL	NUM	NUS	NUZ	ZUE	PUZ	PUS	PUM	PUL
NL	ONUM	ONUS	ONUZ	ONUZ	OZE	OPUS	OPUM	OPUL	OPUL
NM	ONUM	ONUS	ONUS	ONUZ	OZE	OPUS	OPUM	OPUL	OPUL
NS	ONUL	ONUM	ONUS	ONUZ	OZE	OPUS	OPUM	OPUL	OPUL
NZ	ONUL	ONUM	ONUS	ONUZ	OZE	OPUZ	OPUS	OPUM	OPUL
ZE	ONUL	ONUM	ONUS	ONUZ	OZE	OPUZ	OPUS	OPUM	OPUL
PZ	ONUL	ONUL	ONUM	ONUZ	OZE	OPUZ	OPUS	OPUM	OPUL
PS	ONUL	ONUL	ONUM	ONUZ	OZE	OPUZ	OPUS	OPUS	OPUM
PM	ONUL	ONUL	ONUL	ONUS	OZE	OPUZ	OPUZ	OPUS	OPUM
PL	ONUL	ONUL	ONUL	ONUM	ONUZ	OZE	OPUZ	OPUS	OPUM

Where: $U = XX$, denotes extremely large.

As before, an isosceles triangle is chosen as the membership function with 50% overlap in all fuzzy sets. The universe of discourse of input ($E5_{g(i-1)}$ and $E5_{o(i-1)}$) and outputs (ΔG_{5Fi} and ΔO_{5Fi}) is portioned into nine membership functions corresponding to nine linguistic variables. They are NL, NM, NS, NZ, ZE, PZ, PS, PM, PL in the 1st layer; NXL, NXM, NXS, NXZ, ZE, PXZ, PXS, PXM, PXL in the 2nd layer; and NXXL, NXXM, NXXS, NXXZ, ZE, PXXZ, PXXS, PXXM, PXXL in the 3rd layer. This is the same procedure used for the FL2C and FID controller, illustrated in Figure 3.5, where “e” denotes $E5_{g(i-1)}$ or $E5_{o(i-1)}$ and ΔG_{5Fi} or ΔO_{5Fi} .

The universe of discourse for inputs $DE_{g(i-1)}$ and $DE_{o(i-1)}$ is portioned into nine membership functions corresponding to nine linguistic variables (NL, NM, NS, NZ, ZE, PZ, PS, PM, PL), as illustrated in Figure 3.8, where e denotes $DE_{g(i-1)}$ or $DE_{o(i-1)}$.

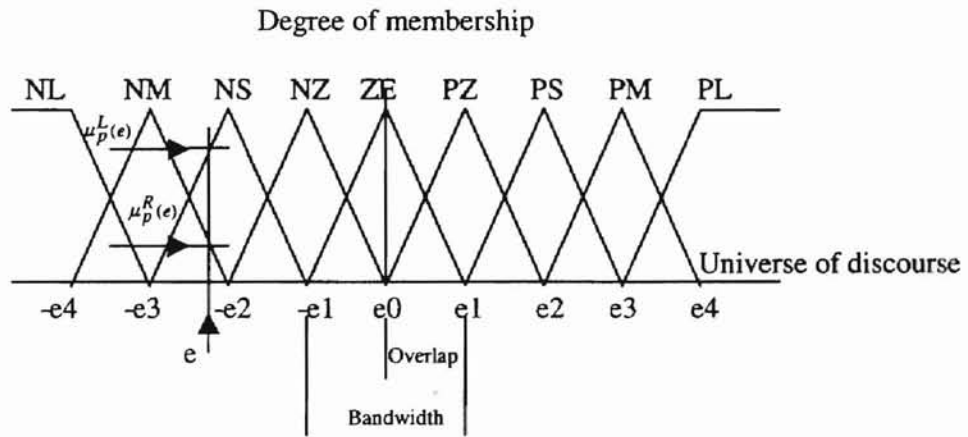


Figure 3.8

Fuzzy Sets for $DE_{g(i-1)}$ and $DE_{o(i-1)}$

In Figure 3.5 and 3.8, the degree of membership $\mu_p^L(e)$ for the leftmost fuzzy set can be calculated using equations similar to (3.32) and (3.33). The degree of membership for rightmost

fuzzy set, $\mu_p^R(e)$ is the complement of that for the leftmost, given by an equation similar to (3.34).

Using Tables 3.10 and 3.11, relevant relationships for the two double-input, single-output fuzzy logic controllers can be written as:

$$\mu_{R_{1k}^j} (E5_{g(i-1)}, DE_{g(i-1)}, \Delta G_{5Fi}) = \mu_{A_{1k}^j \times D_{1k}} (E5_{g(i-1)}, DE_{g(i-1)}) \rightarrow \mu_{B_{1k}^j} (\Delta G_{5Fi}) \quad (3.59)$$

$$\mu_{R_{2k}^j} (E5_{o(i-1)}, DE_{o(i-1)}, \Delta O_{5Fi}) = \mu_{A_{2k}^j \times D_{2k}} (E5_{o(i-1)}, DE_{o(i-1)}) \rightarrow \mu_{B_{2k}^j} (\Delta O_{5Fi}) \quad (3.60)$$

where: μ is degree of membership; k is k^{th} rule; and j denotes the j^{th} layer;

Finally we use the weighted average method (Ross, 1995) as before, for the defuzzification and calculate the fuzzy control outputs ΔG_{5Fi} and ΔO_{5Fi} as:

$$\Delta G_{5Fi} = \frac{\sum_{k=1}^n \mu_{A_{1k}^j \times D_{1k}} \cdot y_{1k}^j}{\sum_{k=1}^n \mu_{A_{1k}^j \times D_{1k}}} \quad (3.61)$$

$$\Delta O_{5Fi} = \frac{\sum_{k=1}^n \mu_{A_{2k}^j \times D_{2k}} \cdot y_{2k}^j}{\sum_{k=1}^n \mu_{A_{2k}^j \times D_{2k}}} \quad (3.62)$$

where: y_{1k}^j is a discrete point ΔG_{5Fi} in the universe of discourse $V1^j$; y_{2k}^j is a discrete point ΔO_{5Fi} in the universe of discourse $V2^j$; n is the number of the rules, here $n = 4$.

Summary

In this chapter we have proposed 5 controllers to control cameras parameters gain, offset, and threshold using conventional PID control technique as well as fuzzy logic techniques. Each controller consists of two portions, one for threshold control, the same for all controllers, and the other for gain and offset control. In the next chapter we present results from tests conducted on each of these 5 controllers.

CHAPTER IV

EXPERIMENTAL RESULTS AND COMPARISONS

In Chapter 2, we investigated performance dynamics of two cameras using two tests. One was a 10 minute test with intentional ambient lighting disturbances applied, while the other was a 55 minute test without intentional disturbances. Results from both show that the minimum and maximum gray levels of both cameras vary with time, and that both minimum and maximum gray levels will change dramatically if lighting disturbances are encountered. Variations, even without intentional disturbances, were unacceptable for repeatable high-quality imaging.

In order to overcome this problem, in Chapter 3 we proposed 5 different controllers to automatically adjust camera parameters gain, offset, and threshold, such that variations of minimum and maximum gray levels would be maintained within an acceptable range, even if relatively large lighting or power disturbances were encountered. Conventional PID control and fuzzy logic control techniques were used for controller design.

In this chapter, we examine camera performance under control of each of the 5 controllers by using two different tests. One test is primarily for short transit response examination, while the other is for longer transient and steady state response evaluation. During both tests, the integral square error (ISE) for both minimum and maximum gray level change is calculated. Plots of minimum and maximum gray level versus time for each control system in each test are presented, while ISE values for each case are evaluated as well. These results provide a good indication on how well each of these 5 controllers performed and which one is the best.

In the following section, the two tests and ISE calculation will be discussed first. Then testing results will be presented and performance of each controller will be investigated. Finally comparisons of the 5 controllers will be made.

Testing Methods

In order to investigate controller performance, two testing procedures are used in this study. One is a transit response test to demonstrate speed of response of the control system in transferring from a prescribed initial state to a desired final state. The other is a steady state response test with ambient lighting disturbances intentionally applied to investigate robustness of each controller. Such tests are widely used in industry to evaluate controller performance. While the plots of transit response and steady state response are good indications of camera general performance, we also employ the ISE in both tests to more carefully quantify control system performance.

Following tests in Chapter 2, we employ the same desired minimum and maximum gray levels, namely:

$$\begin{aligned} X_{d \min}(1) &= 10; & X_{d \min}(2) &= 10; \\ X_{d \max}(1) &= 60; & X_{d \max}(2) &= 60; \end{aligned} \quad (4.1)$$

where again, $X_{d \min}$ represents desired minimum gray level; $X_{d \max}$ represents desired maximum gray level; and parenthetical numbers (1) and (2) denote Camera 1 and Camera 2, respectively.

Transit Response Testing Procedure

Our transit response test consists of 100 cycles of image acquisition and control adjustment started at prescribed initial conditions. We use the same standard vision target (Figure 2.2) used in the tests of Chapter 2. Each testing cycle consists of taking a picture of the vision target, analyzing the image data, calculating actual minimum and maximum gray levels, and updating the control parameters gain, offset, and threshold. Typically, each cycle requires

approximately 0.5 seconds of real time for execution, such that the length of each transient response test is approximately 50 seconds. The testing procedure is as follows:

1. Warm Up.

Due to different performances of controllers and cameras, we allow 100 cycles for each controller-camera combination to warm up, such that each has enough time to reach the same prescribed initial state before the transit response test begins. The parameters specified for the prescribed initial state are:

$$\begin{aligned} X_{i\min}(1) &= 60; & X_{i\min}(2) &= 60; \\ X_{i\max}(1) &= 90; & X_{i\max}(2) &= 90; \end{aligned} \quad (4.2)$$

where: $X_{i\min}$ represents the desired initial minimum gray level; $X_{i\max}$ represents the desired initial maximum gray level; and parenthetical numbers (1) and (2) denote Camera 1 and Camera 2, respectively.

2. Transit from Initial State to Desired Final State

After operating for 100 cycles and the control system having reached the desired initial state, the inputs of desired minimum and maximum gray levels are adjusted as step inputs to the desired final states of 10 and 60, respectively, and the controllers are allowed to bring the actual minimum and maximum gray levels to a final steady state. This period of control action is run for 100 cycles, during which no intentional disturbances are introduced. During this period, the ISE is calculated over the entire time period using the following relationships:

$$I_{\min} = \frac{1}{T_f} \int_0^{T_f} (X_{d\min} - X_{\min})^2 dt \quad (4.3)$$

$$I_{\max} = \frac{1}{T_f} \int_0^{T_f} (X_{d\max} - X_{\max})^2 dt \quad (4.4)$$

$$I_s = I_{\min} + I_{\max} \quad (4.5)$$

where: I_{\min} is the ISE for minimum gray level; I_{\max} is ISE for maximum gray level; I_s is the sum of the integral square error of both minimum and maximum gray levels; $X_{d\min}$ and $X_{d\max}$ represent desired minimum and maximum gray levels; X_{\min} and X_{\max} are the actual minimum and maximum gray levels of the vision image; and T_f is the testing time in minutes. I_s is used as an overall quantitative of measure of control system performance, with the smallest value of I_s being the best.

Steady State Response Testing Procedure

The Steady State Response Test is a 10-minute test with intentional ambient lighting disturbances applied. We use actual time to measure duration of this test, because the intentional ambient lighting disturbances can be easily controlled over time. The ISE is calculated on line through this test as we did in transit response test. To initiate the test, we first apply the controller to the system to force minimum and maximum gray levels to 10 and 60, respectively. After the system reaches steady state, we begin the 10-minute test during which intentional ambient lighting disturbances are introduced. In order to simulate ambient lighting change, we used the same method as in Chapter 2. By quickly covering one or two fluorescent light tubes inside the light box with an elongated cardboard tent-shaded piece extending the entire length of the fluorescent tubes, we could repeatedly produce step decreases in lighting intensity. Quickly removing this covering produced a repeatable step increase in lighting intensity. Our schedule for lighting changes was as follows:

0 – 2 minutes	No cover over the lighting tubes;
2 – 4 minutes	Cover 1 lighting tube out of 2 each side;
4 – 6 minutes	No cover over the lighting tubes;
6 – 8 minutes	Cover 2 lighting tubes out of 2 each side;
8 – 10 minutes	No cover over the lighting tubes;

During the Steady State Test, both minimum and maximum gray levels were monitored and integral square error ISE was calculated according to Equations (4.3 – 4.5).

Testing Results

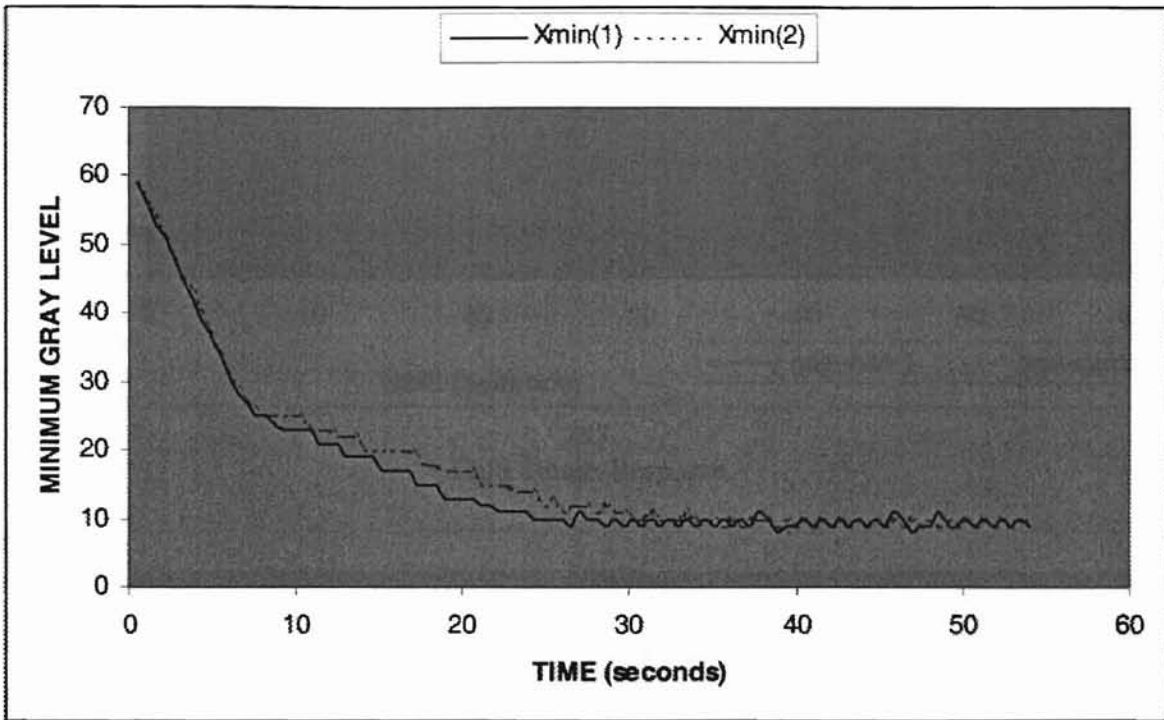
According to the testing procedures discussed in the previous section, each controller was evaluated during transit and steady state response tests while the ISE value was calculated.

Results will be presented in the following order:

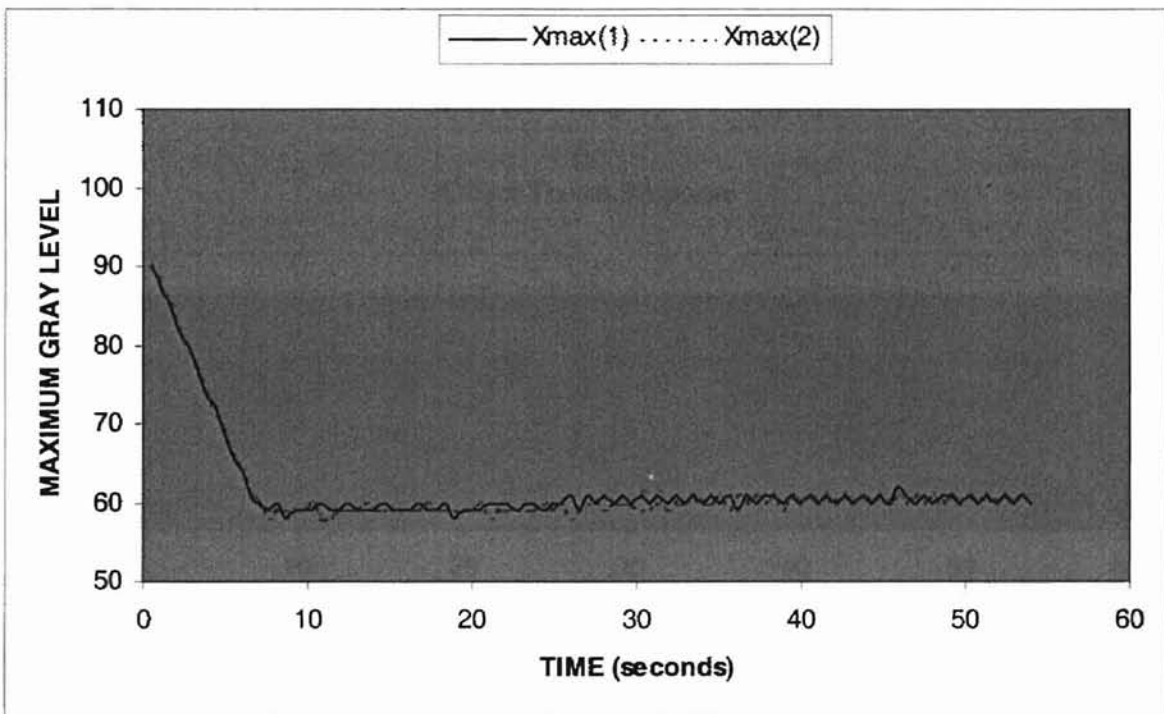
1. Fuzzy Logic 1 Controller (FL1C)
2. Fuzzy Logic 2 controller (FL2C)
3. Fuzzy-Integral Controller (F(E+D)+I)
4. Fuzzy-Integral-Derivative Controller (FID)
5. Proportional-Integral-Derivative Controller (PID)

Fuzzy Logic 1 Controller

As given in Chapter 3, the FL1C is a linguistic rule base, two layer, double-input, double-output fuzzy logic controller (Chen, 1996). Figures 4.1 (a) and (b) show the transit response of minimum and maximum gray levels for both cameras. As we can see, the minimum gray level is forced to the desired value of 10 from the prescribed initial value of 60, while the maximum gray level is forced to desired value of 60 from the prescribed initial state of 90. Figures 4.2 (a), (b), and (c) show the transit response plots of control variables gain, offset, and threshold, respectively. During the transit period for cameras 1 and 2, respectively, gain increased by 99 (64 to 163) and 135 (109 to 244), offset decreased by 72 (134 to 62) and 82 (122 to 40), and threshold decreased by 39 (74-35) and 41 (73-32), values required to bring about these gray level changes. Observe that due to the relatively slow change of control variables shown in Figures 4.2 (a), (b), and (c), the minimum and maximum gray levels also change slowly during transit response period. Approximately 55 testing cycles are required for the minimum gray level to reach 10, while about 15 cycles are required to bring the maximum gray level to 60. Such slow transit

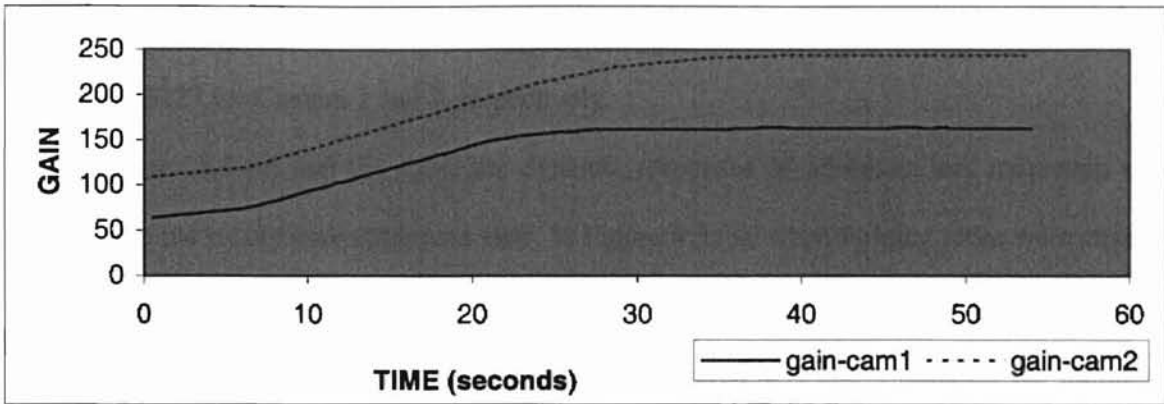


(a)
Minimum Gray Level Transit Response

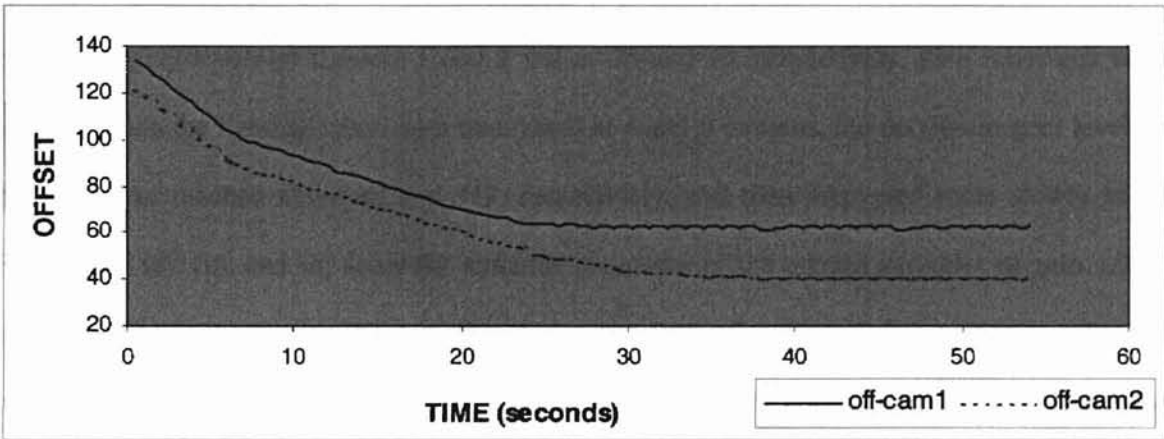


(b)
Maximum Gray Level Transit Response

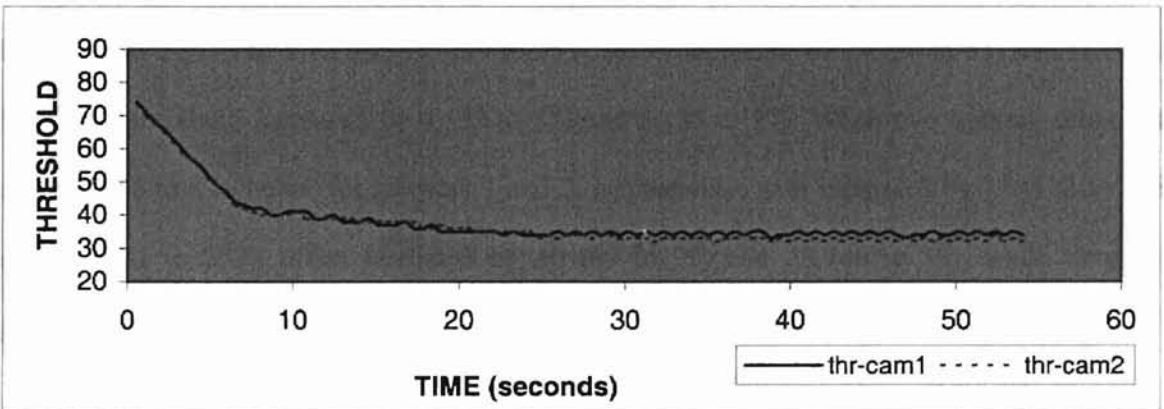
Figure 4.1 FLIC



(a)
Gain Transit Response



(b)
Offset Transit Response



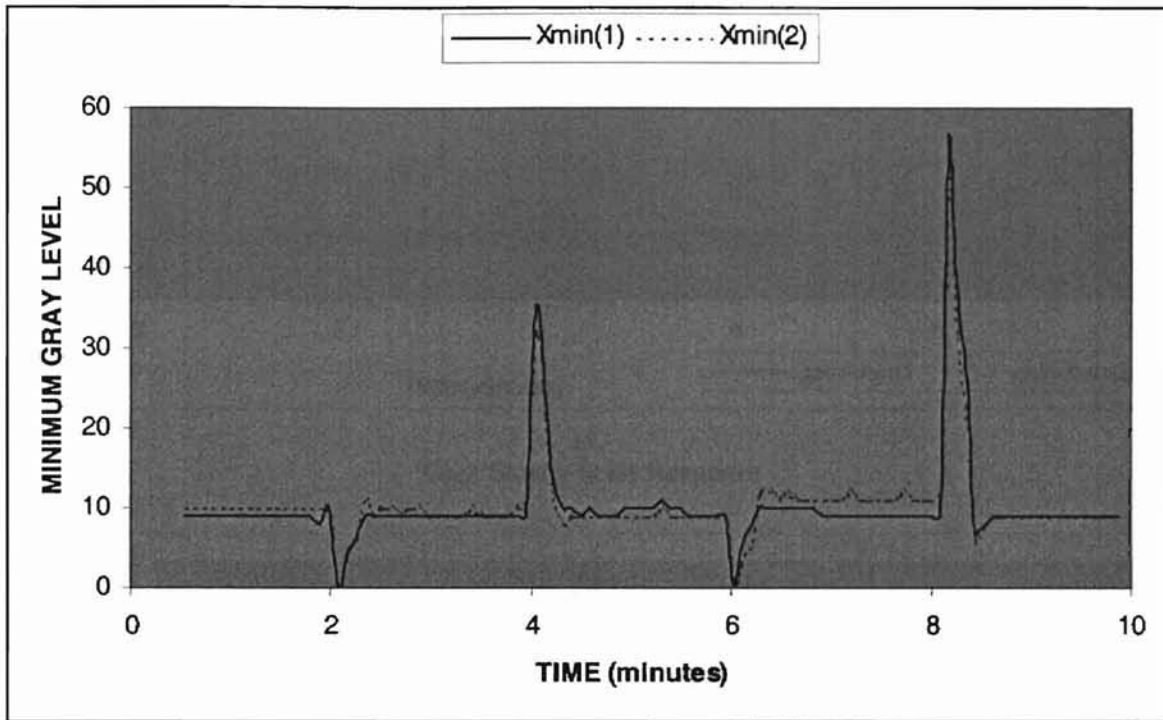
(c)
Threshold Transit Response

Figure 4.2 FLIC

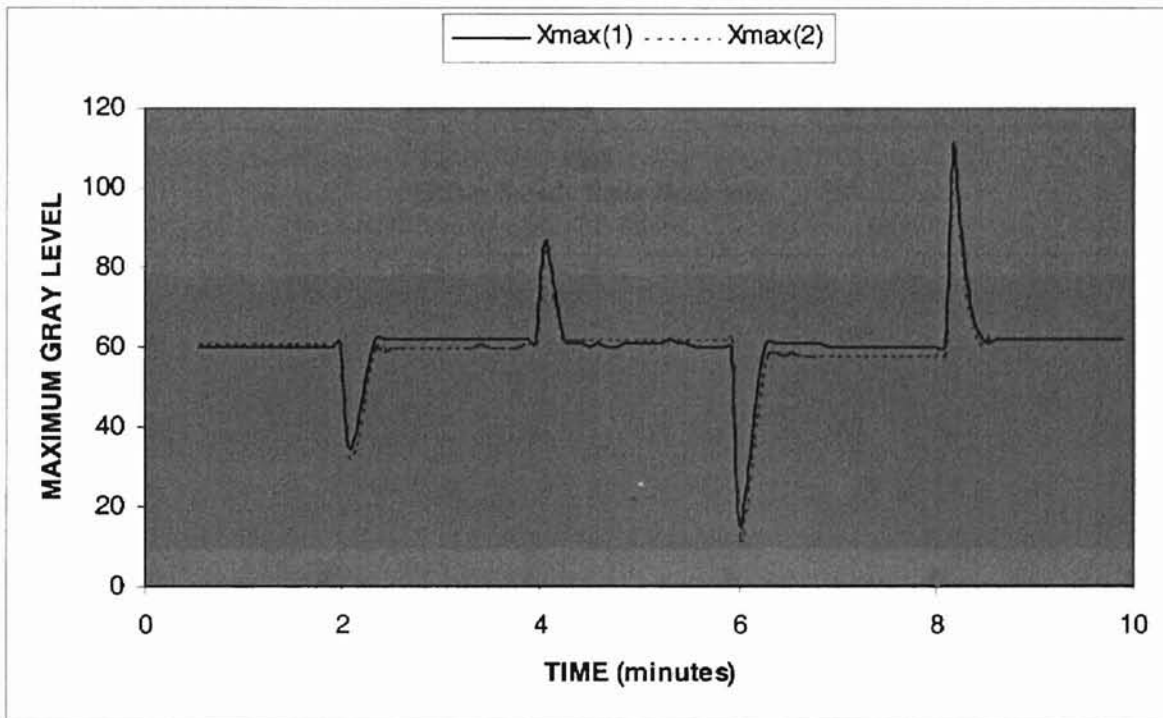
response is likely unacceptable for this application. The sum of integral square errors, I_s , is 23825 and 25227 for Camera 1 and 2, respectively.

Figures 4.3 (a) and (b) show the dynamic responses of minimum and maximum gray levels during the steady state responses tests. In Figure 4.3 (a), when lighting tubes were covered at 2 and 6 minutes, the minimum gray levels of both Camera 1 and 2 fell to 0, and then recovered more slowly to 10. When the lighting tubes were uncovered at 4 and 8 minutes, the minimum gray levels of Camera 1 and 2 reached approximately 35 and 57, respectively, and recovered more slowly to 10. In Figure 4.3 (b), when lighting tubes were covered at 2 and 6 minutes, the maximum gray levels of Camera 1 and 2 fell to 35 and 10, respectively, then recovered to 60 slowly. When the lighting tubes were uncovered at 4 and 8 minutes, the maximum gray levels of both cameras reached about 88 and 112, respectively, and then recovered more slowly to 60. Figures 4.4 (a), (b), and (c) show the dynamic responses of the control variables of gain, offset, and threshold, respectively. We note that during the time of ambient lighting change, when the tubes were covered and uncovered at 2, 4, 6, and 8 minutes, the control variables gain and offset reached new steady plateaus, while threshold returned to a nominal value between 30 and 40. When one lighting tube was covered at 2 to 4 minutes, for cameras 1 and 2, respectively, gain decreased by 3 (173 to 170) and 33 (219 to 252), offset increased by 24 (59 to 83) and 11 (49 to 60), while threshold increased by 4 (33 to 37) and 2 (35 to 37). When two lighting tubes was covered at 6 to 8 minutes, for cameras 1 and 2, respectively, gain increased by 12 (173 to 185) and 22 (231 to 253), offset increased by 40 (59 to 99) and 33 (46 to 79), while threshold decreased by 2 (34 to 32) and 0. The summed integral squared error, I_s , was 4331 for Camera 1 and 4003 for Camera 2.

Compare the steady state response results with those in Chapter 2 where no control was added to the vision system. In Figure 2.5 and 2.6, when the lighting tubes were covered during 2 to 4 minutes and 6 to 8 minutes, the minimum gray levels of both cameras fell to 0 and the

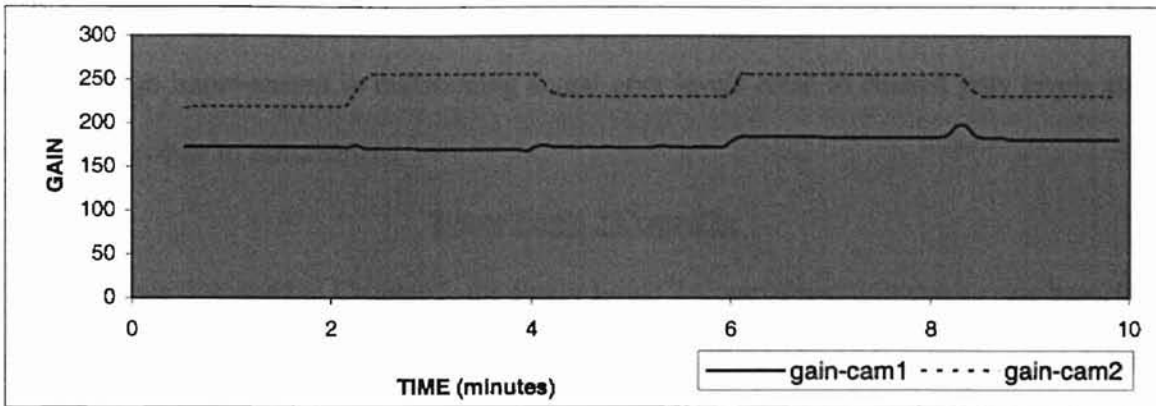


(a)
Minimum Gray Level Steady State Response

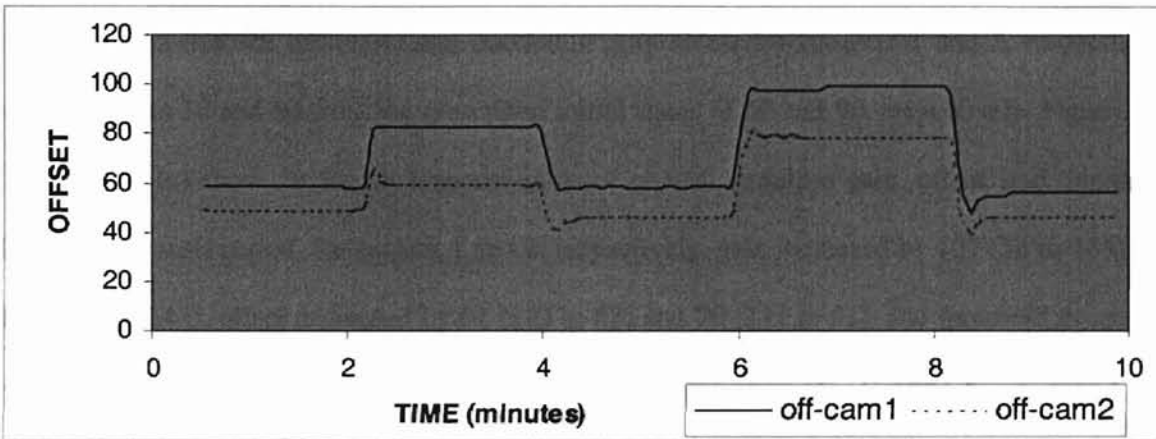


(b)
Maximum Gray Level Steady State Response

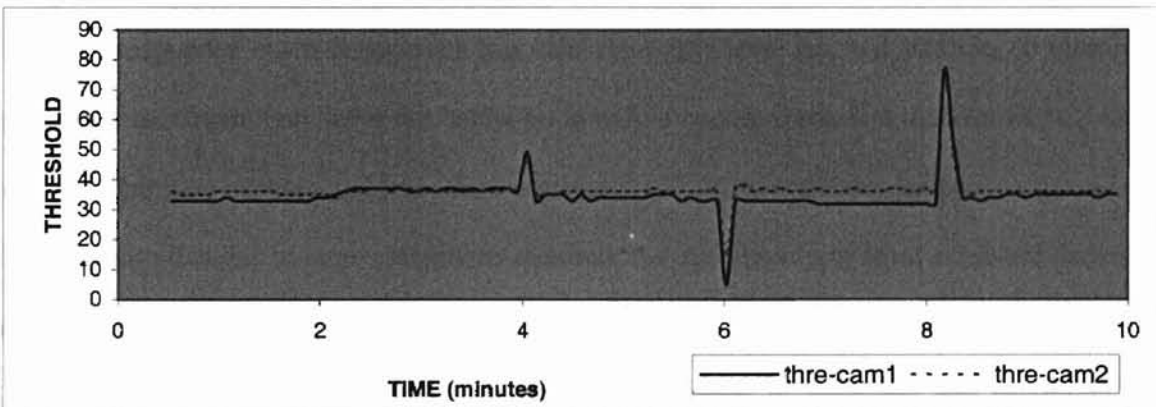
Figure 4.3 FLIC



(a)
Gain Steady State Response



(b)
Offset Steady State Response



(c)
Threshold Steady State Response

Figure 4.4 FL1C

maximum gray levels fell to 45 and lower, and did not recover. It is obviously that the FL1C offers a large improvement in maintaining actual gray levels close to desired gray levels and is reasonably robust to disturbances.

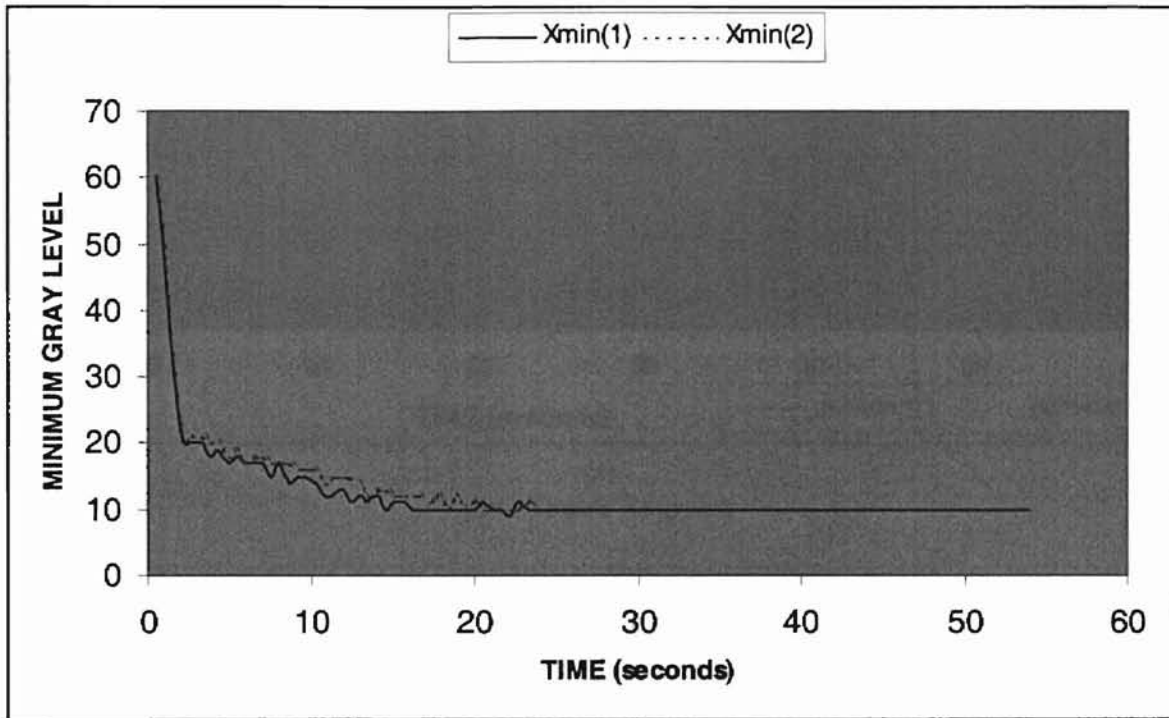
Fuzzy Logic 2 Controller

As we discussed in Chapter 3, the Fuzzy Logic 2 Controller is a linguistic rule base, three layer, double-input, double-output fuzzy logic controller. We designed it to cover a much wider range of disturbances than FL1C, and to give faster response to large disturbances.

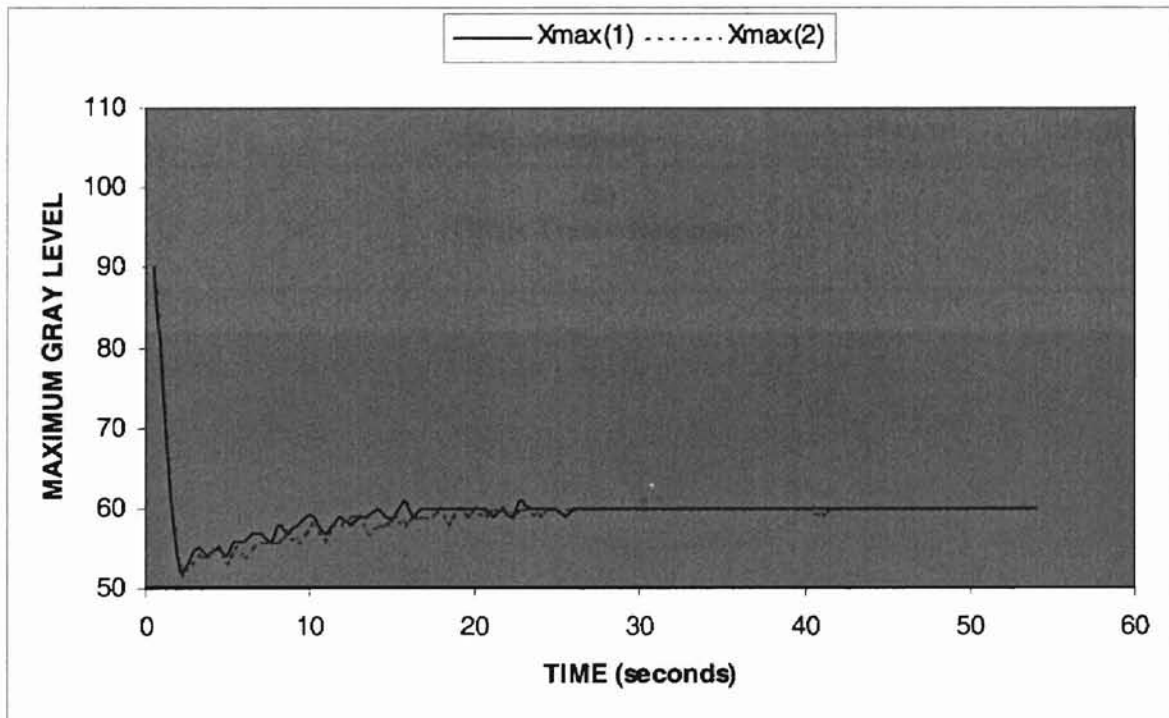
Figure 4.5 (a) and (b) show the transit response of minimum and maximum gray levels. It can be seen that the minimum and maximum gray levels for cameras 1 and 2, respectively, were forced to 10 and 60 from the prescribed initial states of 60 and 90, respectively. Figures 4.6 (a), (b), and (c) show the transit response plots of control variables gain, offset, and threshold. During the transit period, for camera 1 and 2, respectively, gain increased by 107 (58 to 165) and 142 (103 to 245), offset decreased by 61 (123 to 62) and 70 (111 to 41), and threshold decreased by 40 (75-35) and 40 (74-34). The sum of integral square error, I_s , is 6994 for camera 1 and 7639 for camera 2, respectively, a factor of 3 smaller than for the FL1 controller.

Compared to the transit response of the FL1C, the control variables gain, offset, and threshold responded much faster, such that minimum gray level reached 10 from 60 in only 30 cycles and maximum gray level fell below 60 in only 4 cycles, much less than for FL1C. Also it took a larger gain increase and less offset decrease for FL2C to make this happen, compared with FL1C. Notice that due to large corrections required, the maximum gray level exhibited undershoot, and reached 60 in approximately 25 cycles.

Figures 4.7 (a) and (b) show the dynamic responses of minimum and maximum gray levels during the steady state responses tests. In Figure 4.7 (a), when lighting tubes were covered at 2 and 6 minutes, the minimum gray levels of both Cameras 1 and 2 fell to 0 and then recovered to 10. When the lighting tubes were uncovered at 4 and 8 minutes, the minimum gray levels of

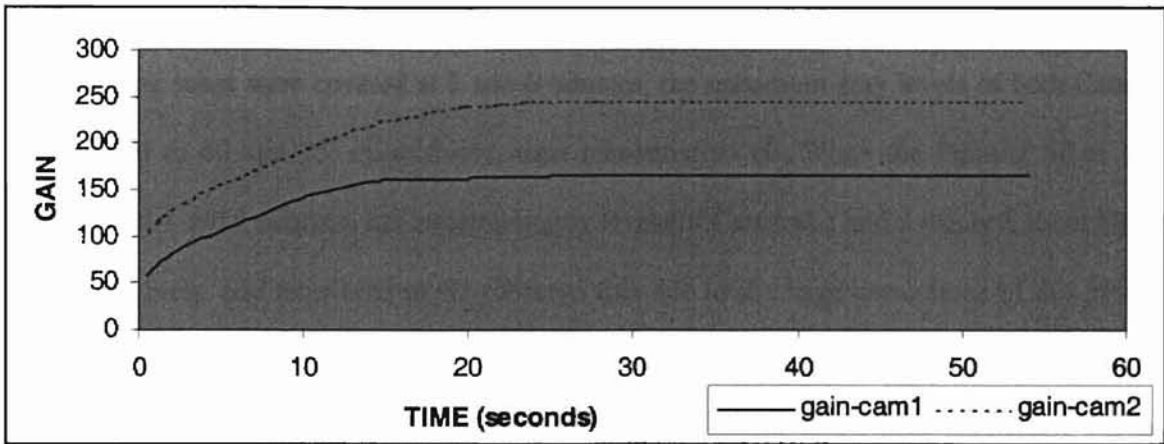


(a)
Minimum Gray Level Transit Response

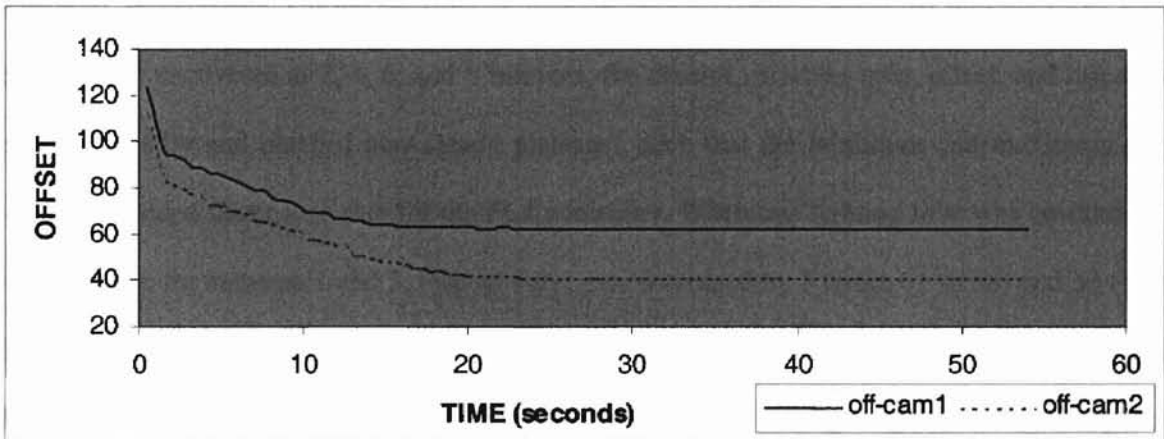


(b)
Maximum Gray Level Transit Response

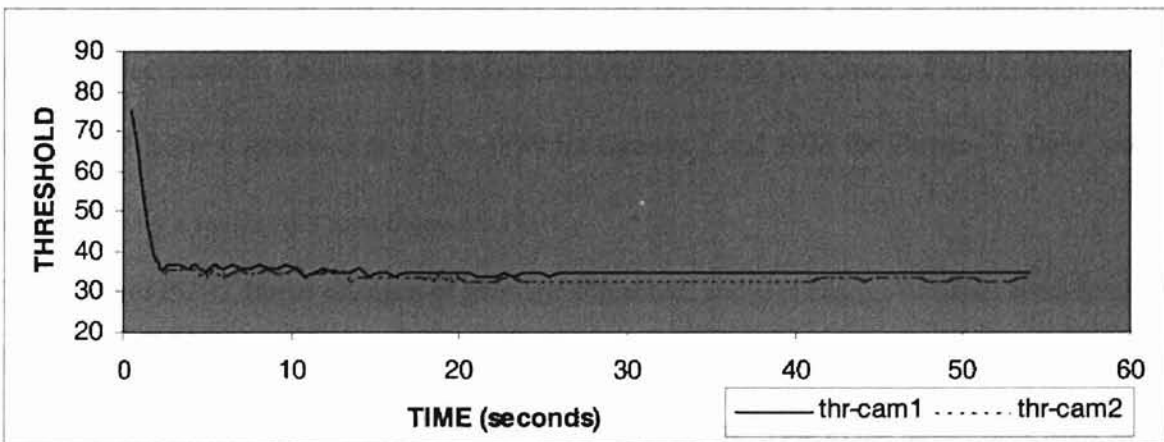
Figure 4.5 FL2C



(a)
Gain Transit Response



(b)
Offset Transit Response

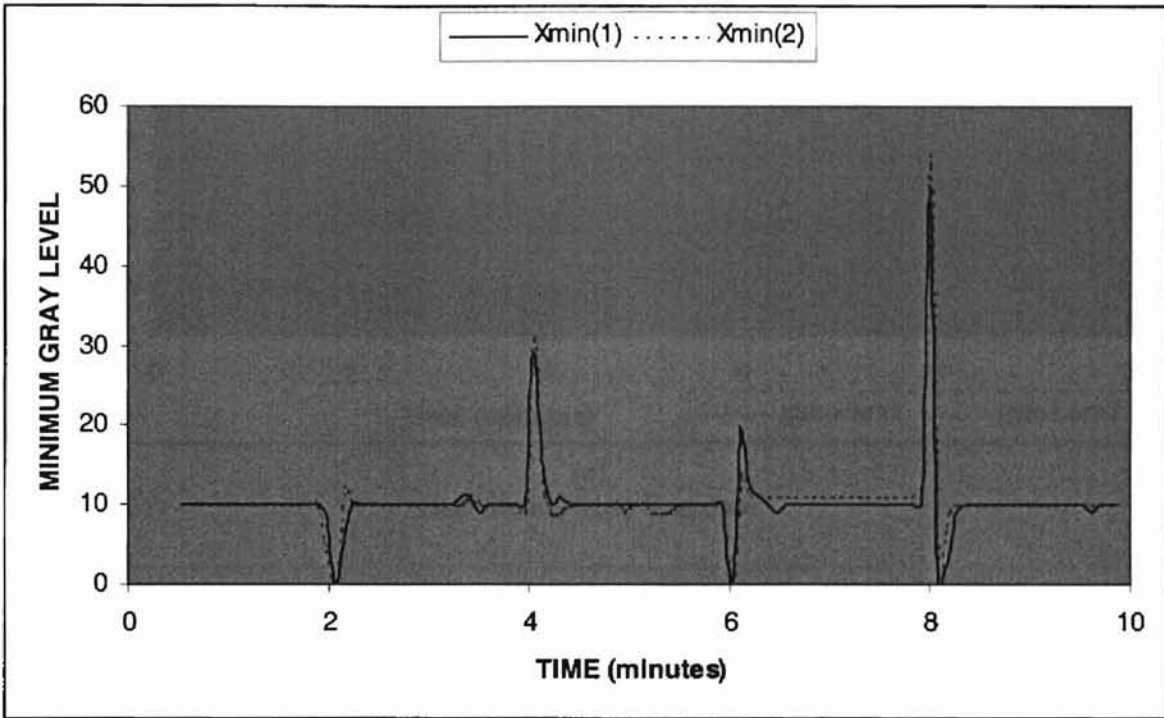


(c)
Threshold Transit Response

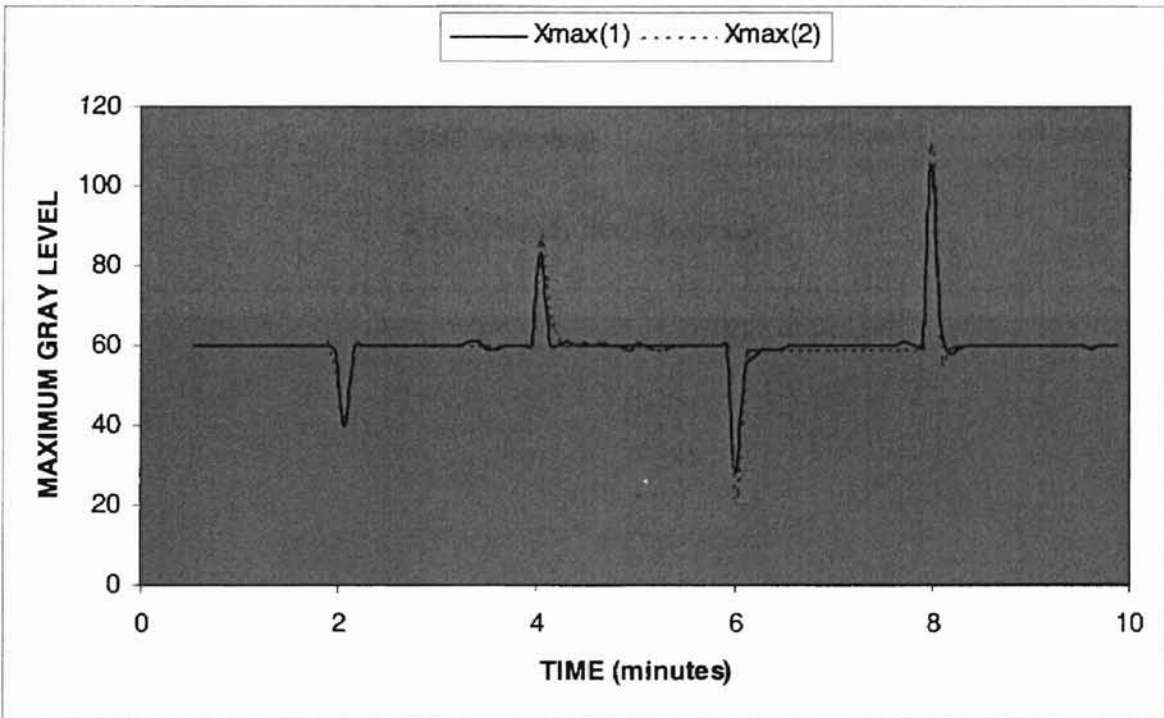
Figure 4.6 FL2C

Camera 1 and 2 reached about 33 and 55, respectively, and recovered to 10. In Figure 4.7 (b), when lighting tubes were covered at 2 and 6 minutes, the maximum gray levels of both Cameras 1 and 2 fell to 40 and 25, respectively, then recovered to 60. When the lighting tubes were uncovered at 4 and 8 minutes, the maximum gray levels of Cameras 1 and 2 reached about 88 and 110, respectively, and recovered to 60. Observe that due to the large corrections of this system, minimum gray levels had an over shot at 6 minutes when two tubes on both sides of the FOV were covered, and an undershoot at 8 minutes when the lighting tubes were uncovered. Figures 4.8 (a), (b), and (c) show the dynamic responses of the control variables of gain, offset, and threshold, respectively. During the time of ambient lighting change, when the lighting tubes were covered and uncovered at 2, 4, 6, and 8 minutes, the control variables gain, offset, and threshold changed rapidly and reached new steady plateaus, such that the minimum and maximum gray levels responded faster than that for the FL1 controller. When one lighting tube was covered at 2 to 4 minutes, for cameras 1 and 2, respectively, gain increased by 7 (from 61 to 68) and 33 (from 212 to 245), offset increased by 14 (from 114 to 128) and 11 (from 49 to 60), while threshold decreased by 10 (from 40 to 30) and 2 (from 36 to 38). When two lighting tubes were covered at 6 to 8 minutes, gain increased by 96 (from 70 to 166) and 41 (from 211 to 252), offset increased by 23 (from 109 to 132) and 29 (from 49 to 78) for Cameras 1 and 2, respectively, while threshold decreased by 18 (from 40 to 22) and 2 (from 36 to 38) for Camera 1 and 2, respectively. The sum of integral square error, I_s , is 1739 for Camera 1 and 1024 for Camera 2. These values are smaller by a factor of 3 than those for FL1C.

Compared to FL1C, larger changes of gain and threshold, and less change of offset were required for FL2C to force the minimum and maximum gray levels to desired values. The response speed of FL2C was faster, with smaller I_s values than that for FL1C. Due to the large corrections in the control variables, the minimum gray level had an overshoot at 6 minutes when two tubes on both

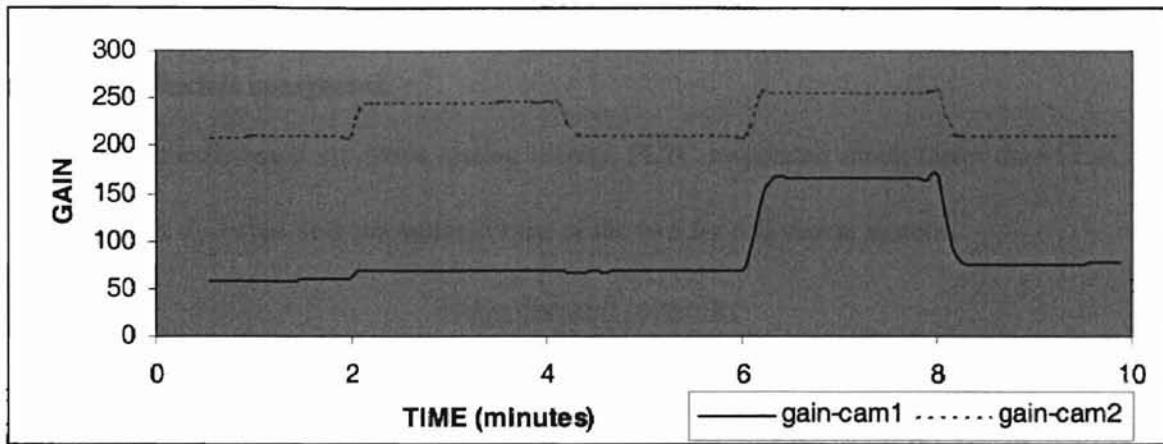


(a)
Minimum Gray Level Steady State Response

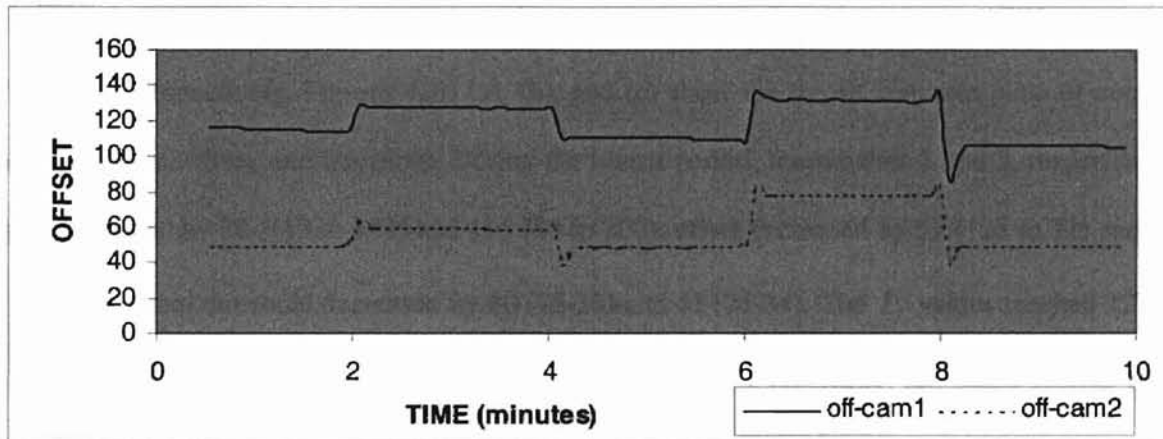


(b)
Maximum Gray Level Steady State Response

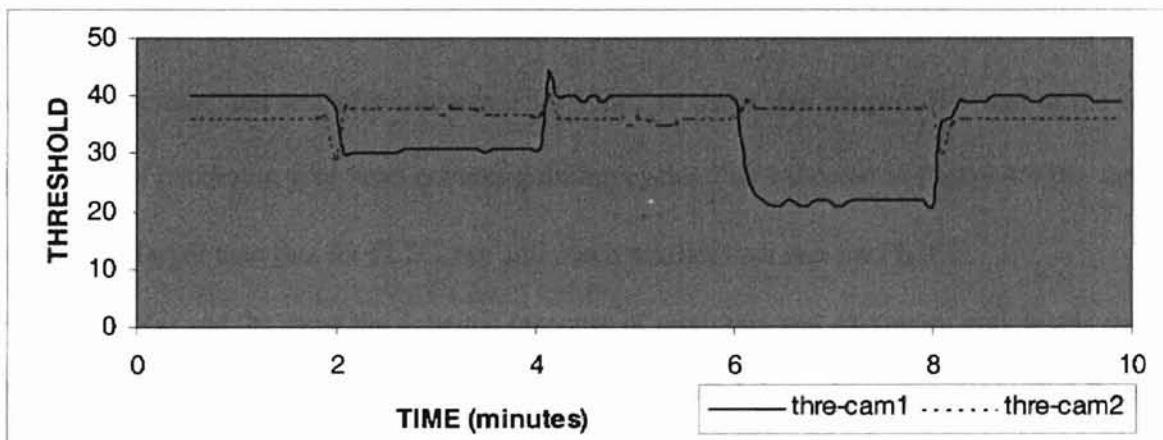
Figure 4.7 FL2C



(a)
Gain Steady State Response



(b)
Offset Steady State Response



(c)
Threshold Steady State Response

Figure 4.8 FL2C

sides of the FOV were covered, and an undershoot at 8 minutes when the lighting tubes were uncovered, which is unexpected.

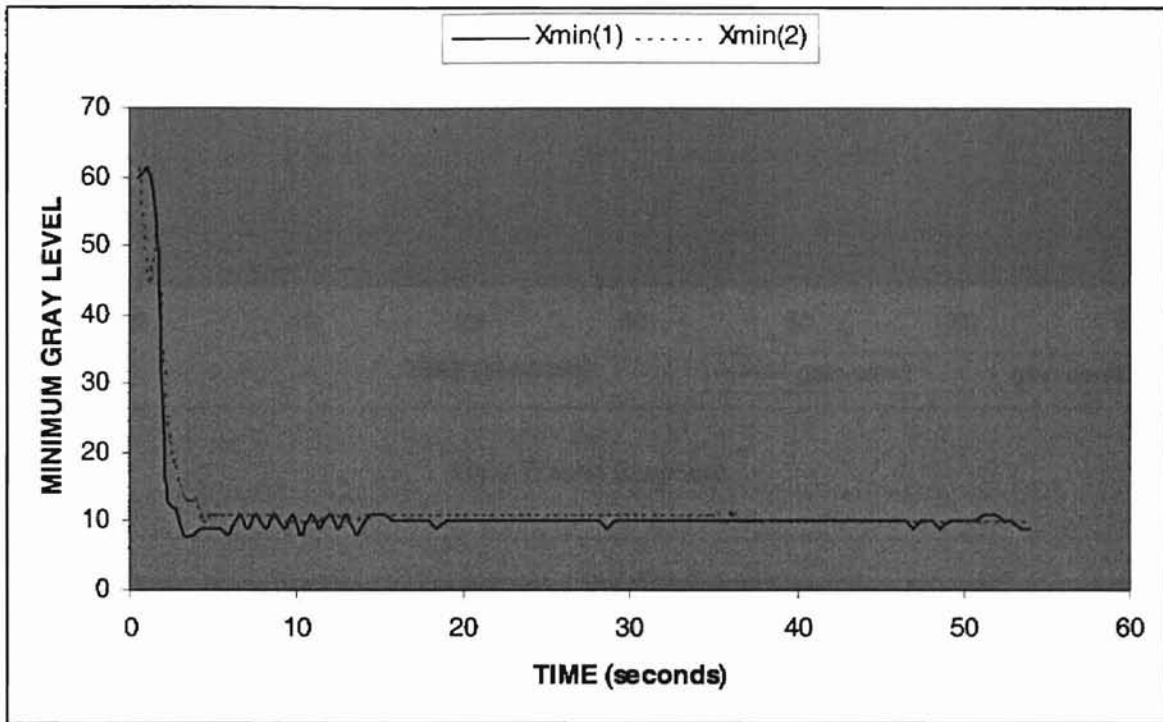
In general, based on above testing results, FL2C responded much faster than FL1C and with a smaller I_s value. It is the better choice of the two for this vision system.

Fuzzy-Integral controller

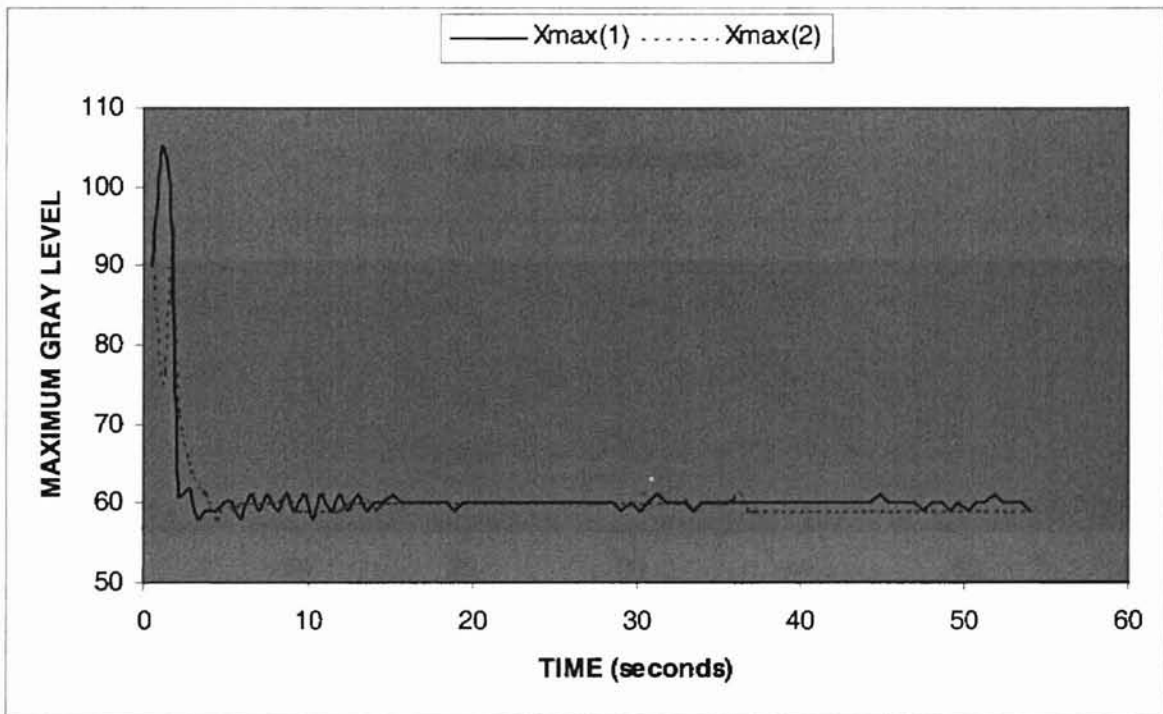
F(E+D)+I is a double-input, double-output controller. It takes advantages of both fuzzy logic and conventional PID control actions. Figures 4.9 (a) and (b) show the transit response of minimum and maximum gray levels. As we can see, for cameras 1 and 2, respectively, the minimum and maximum gray levels are forced to 10 and 60 from the prescribed initial states of 60 and 90, respectively. Figures 4.10 (a), (b), and (c) show the transit response plots of control variables gain, offset, and threshold. During the transit period, for cameras 1 and 2, respectively, gain increased by 28 (113 to 141) and 116 (84 to 200), offset decreased by 57 (128 to 71) and 64 (120 to 56), and threshold decreased by 40 (75-35) and 41 (75-34). The I_s values reached 12780 for Camera 1 and 8834 for Camera 2, larger than for the FL2 controller, but much smaller than for the FL1 controller.

Compared to the FL1C and FL2C, the F(E+D)+I controller responded even faster, such that it only took 12 cycles for both minimum and maximum to reach desired values. Also it took less gain increase and less offset decrease for FL2C to make this happen. But due to the large overshoot of maximum gray level occurring during cycles 1 to 5 showed in Figure 4.9 (b), the I_s values was larger than that for FL2C, but still much smaller than that for FL1C.

Figure 4.11 (a) and (b) show the dynamic response of minimum and maximum gray levels during the steady state responses tests. In Figure 4.11 (a), when lighting tubes were covered at 2 and 6 minutes, the minimum gray levels of both Camera 1 and 2 fell to 0, and then recovered to 10. When the lighting tubes were uncovered at 4 and 8 minutes, the minimum gray levels of both cameras reached about 33 and 52, respectively, and recovered to 10. In Figure 4.11 (b), when

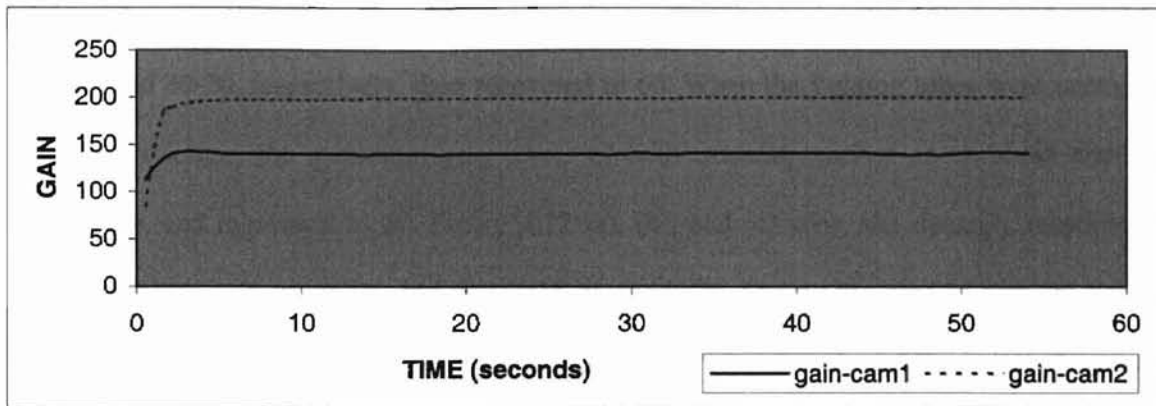


(a)
Minimum Gray Level Transit Response

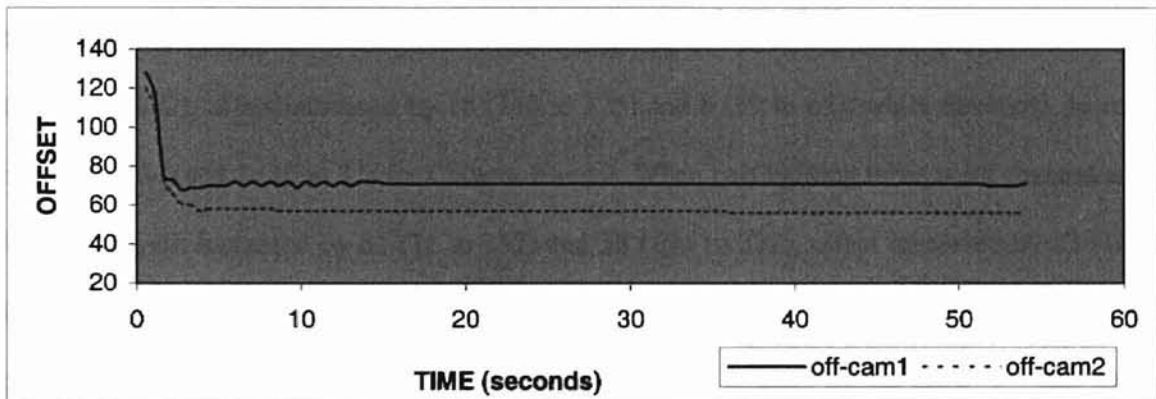


(b)
Maximum Gray Level Transit Response

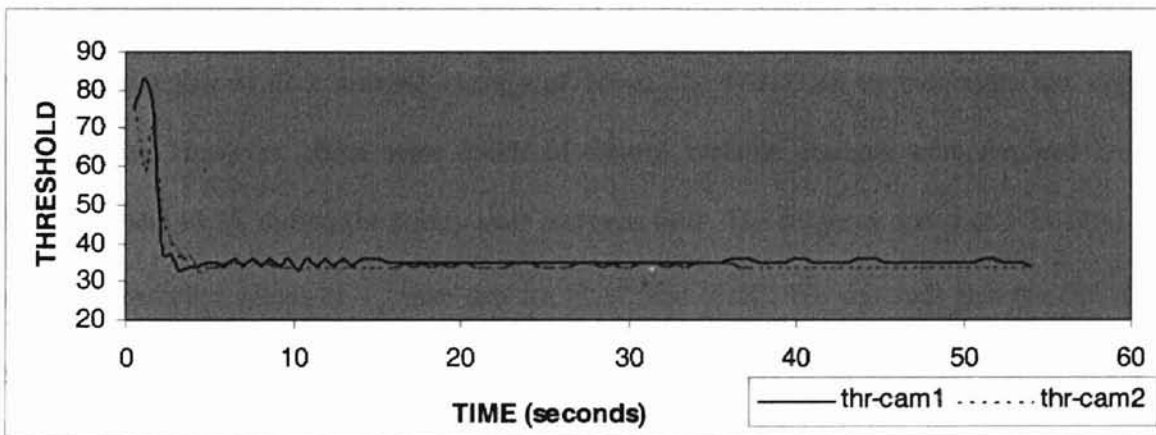
Figure 4.9 F(E+D)+I Controller



(a)
Gain Transit Response



(b)
Offset Transit Response

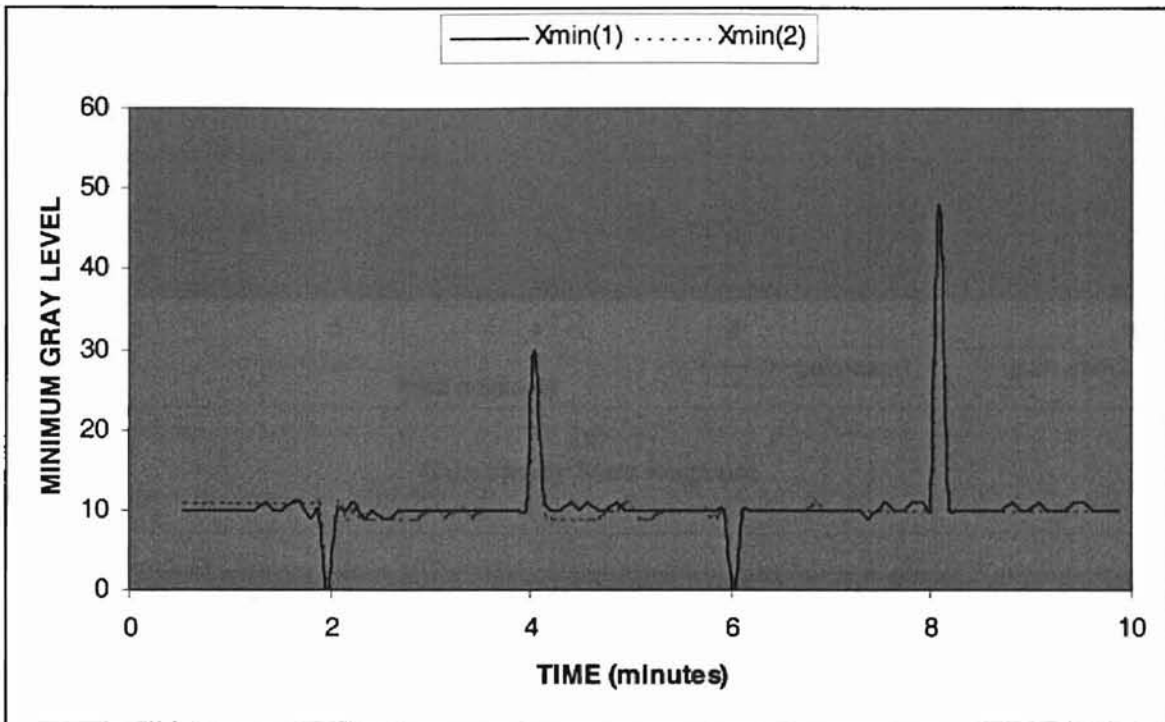


(c)
Threshold Transit Response

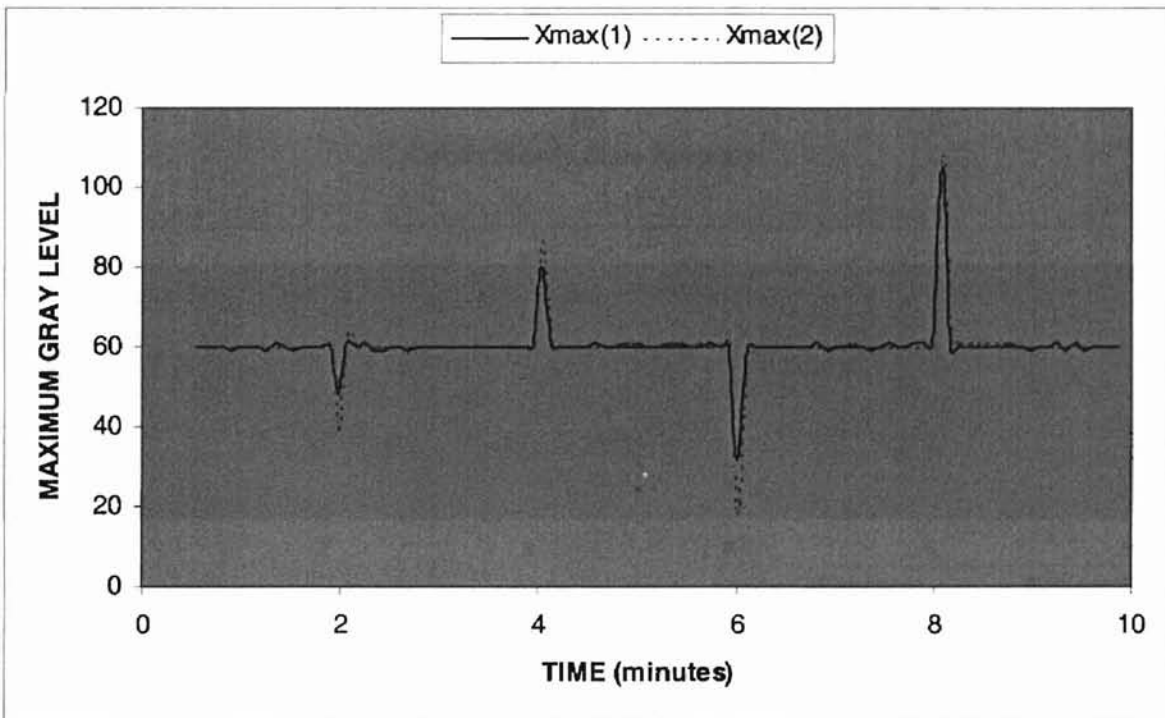
Figure 4.10 F(E+D)+I Controller

lighting tubes were covered at 2 and 6 minutes, the maximum gray levels of Cameras 1 and 2 fell to 50-40 and 30-20, respectively, then recovered to 60. When the lighting tubes were uncovered at 4 and 8 minutes, the maximum gray levels of both cameras reached about 80-85 and 108, respectively, and recovered to 60. Figure 4.12 (a), (b), and (c) show the dynamic responses of control variables of gain, offset, and threshold, respectively. During the time of ambient lighting change, when the lighting tubes were covered and uncovered at 2, 4, 6, and 8 minutes, the control variables gain and offset changed even faster than that for FL2C, such that the minimum and maximum gray levels responded faster than that for FL1C and FL2C. When one lighting tube was covered at 2 to 4 minutes, for Cameras 1 and 2, respectively, gain increased by 41 (66 to 107) and 44 (188 to 232), offset increased by 15 (110 to 125) and 6 (59 to 65), while threshold decreased by 5 (39 to 34) and 1 (36 to 37) for Camera 1 and 2. When two lighting tubes were covered at 6 to 8 minutes, gain increased by 61 (71 to 132) and 58 (194 to 252), offset increased by 23 (107 to 130) and 24 (57 to 81) for Cameras 1 and 2, respectively, while threshold decreased by 15 (40 to 25) and 2 (36 to 38) for Cameras 1 and 2, respectively. The integral square error, I_s , was 887 for camera 1 and 997 for camera 2 in this steady state response test.

Compared to results for the FL1C and FL2C, larger changes of gain and threshold were required, together with a smaller change of offset for F(E+D)+I to overcome the lighting disturbances. However, about same levels of control variable changes were required for the F(E+D)+I and FL2C during the steady state response tests. The response speed of F(E+D)+I was aster, with smaller values of I_s than that for FL1C and FL2C. We conclude that the F(E+D)+I controller works better than the Fuzzy Logic 1 and 2 Controllers as far as steady state response is concerned. It gives faster transit response, but the large overshoot of maximum gray level in transit response testing was unexpected.

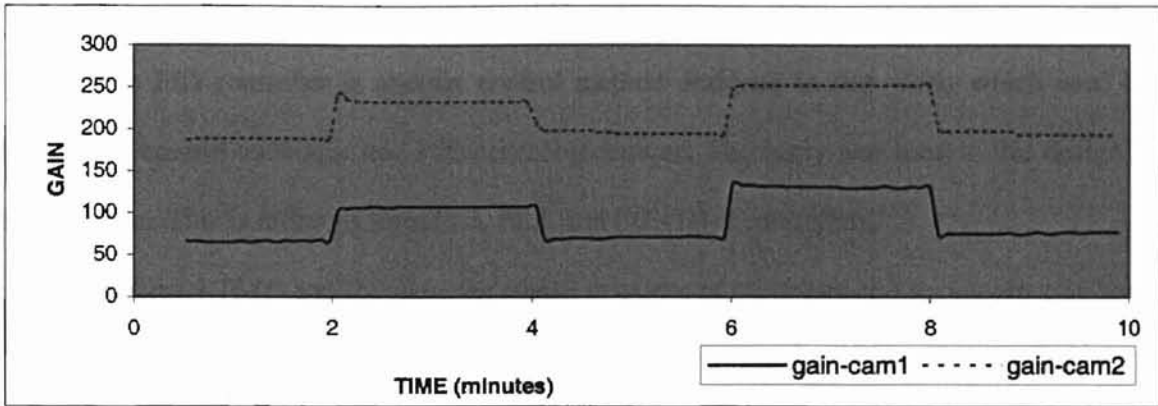


(a)
Minimum Gray Level Steady State Response

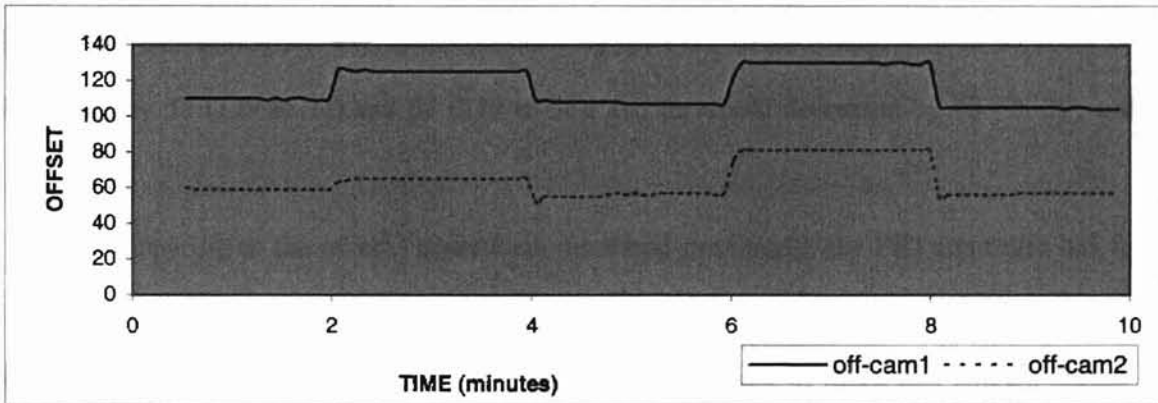


(b)
Maximum Gray Level Steady State Response

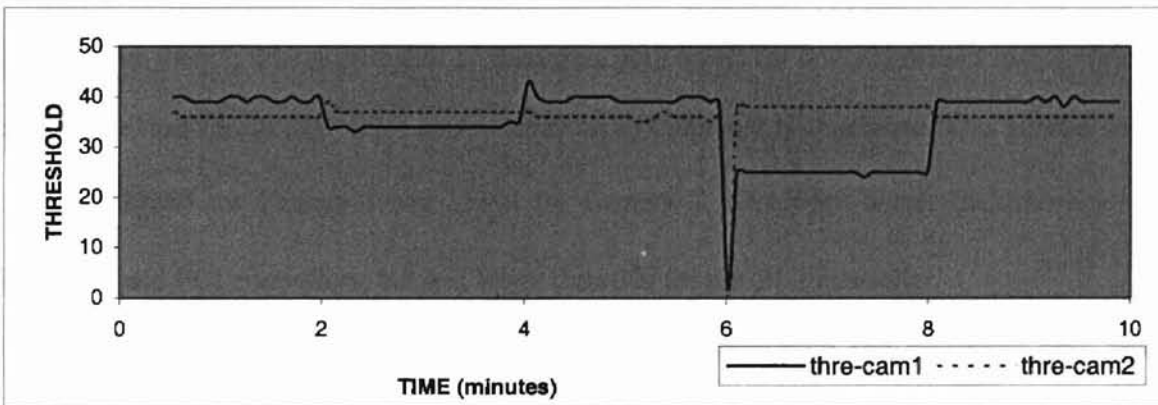
Figure 4.11 F(E+D)+I Controller



(a)
Gain Steady State Response



(b)
Offset Steady State Response



(c)
Threshold Steady State Response

Figure 4.12 F(E+D)+I Controller

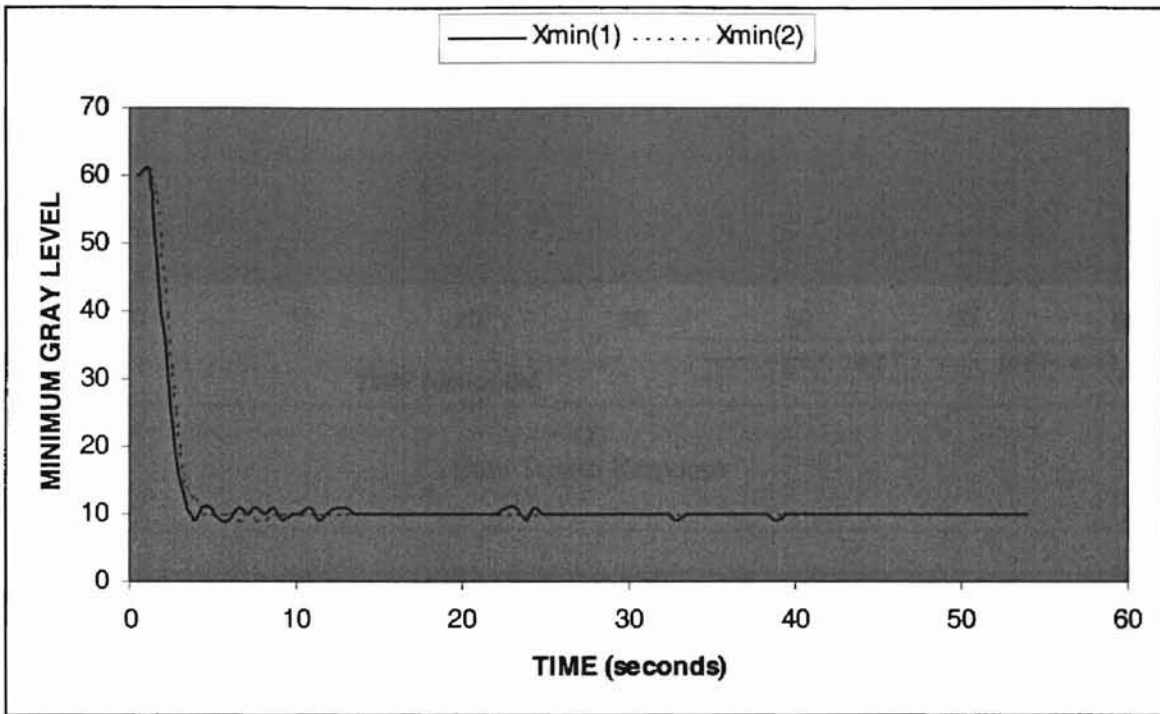
Fuzzy-Integral-Derivative Controller

The FID controller is another control method designed in this work, which used both fuzzy logic control technique and PID control technique. The fuzzy part used in this design is a SISO system. This is different from FL1, FL2, and F(E+D)+I controllers.

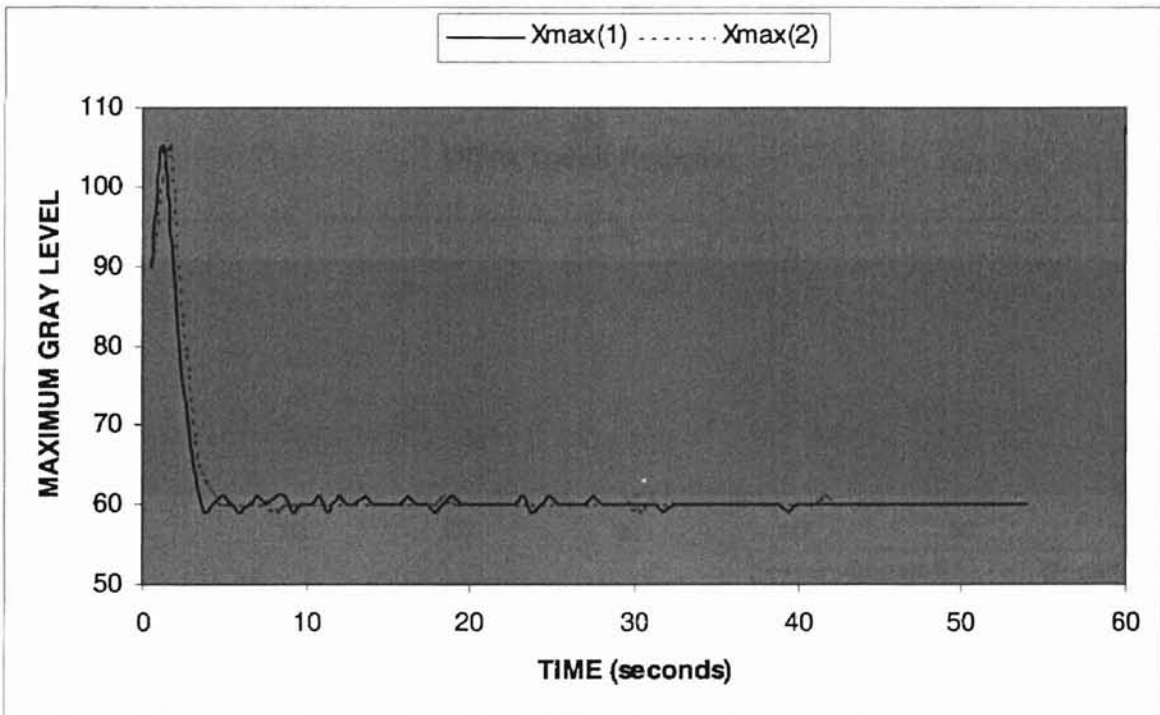
Figures 4.13 (a) and (b) show the transit response of minimum and maximum gray levels. As we can see, the minimum and maximum gray levels are forced to 10 and 60 from the prescribed initial state 60 and 90, respectively. Figures 4.14 (a), (b), and (c) show the transit response plots of control variables gain, offset, and threshold. During the transit period, for Cameras 1 and 2, respectively, gain increased by 28 (112 to 140) and 56 (150 to 206), offset decreased by 57 (129 to 72) and 64 (119 to 55), and threshold decreased by 40 (75-35) and 41 (75-34).

Comparing to the other 3 controllers discussed previously, the FID controller has faster transit response than FL1C and FL2C, and just slightly slower transit response than the F(E+D)+I controller. It took less gain change than the other 3 controllers to make this happen during the transit period. The offset change is smaller than that of FL1C, and about the same as that for the FL2 and F(E+D)+I controllers, while the threshold change is about the same for all 4 controllers. Observe that there is a large overshoot of maximum gray levels for both Cameras 1 and 2, which is even worse than for the F(E+D)+I controller as far as Camera 2 is concerned. This yielded values of I_s of 13285 for Camera 1 and 15604 for Camera 2, which are higher than those for the F(E+D)+I and FL2 controllers, but still lower than that for the FL1 controller.

Figure 4.15 (a) and (b) show the dynamic responses of minimum and maximum gray levels during the steady state responses tests. In Figure 4.15 (a), when lighting tubes were covered at 2 and 6 minutes, the minimum gray levels of both Cameras 1 and 2 fell to 0, and then recovered to 10. When the lighting tubes were uncovered at 4 and 8 minutes, the minimum gray levels of both cameras reached about 33 and 45, respectively, and recovered to 10. In Figure 4.15 (b), when

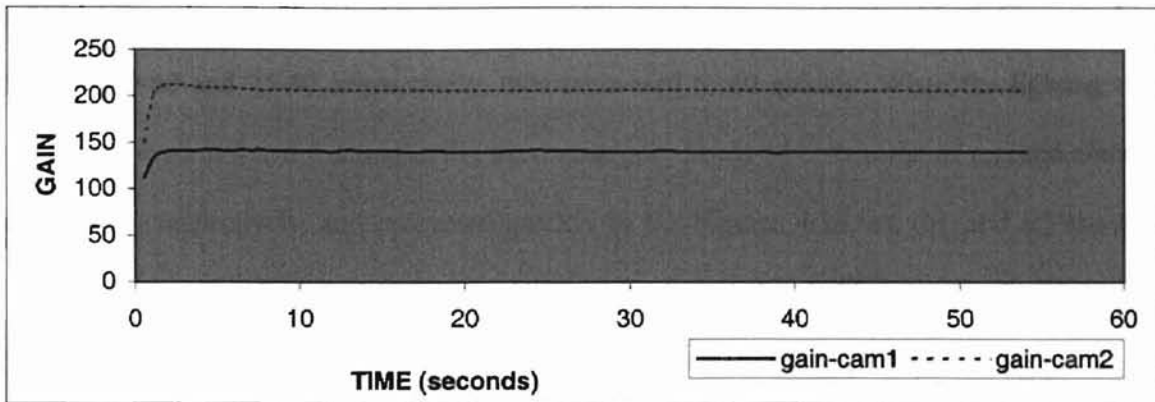


(a)
Minimum Gray Level Transit Response

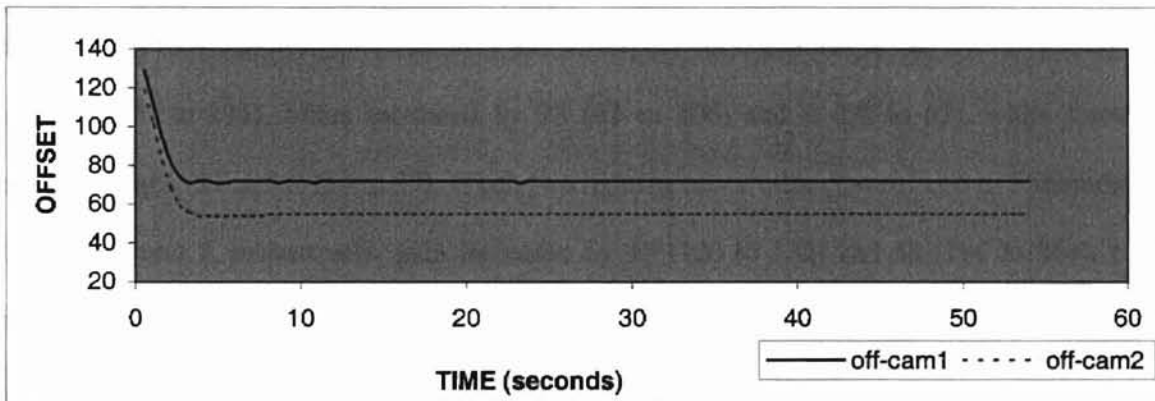


(b)
Maximum Gray Level Transit Response

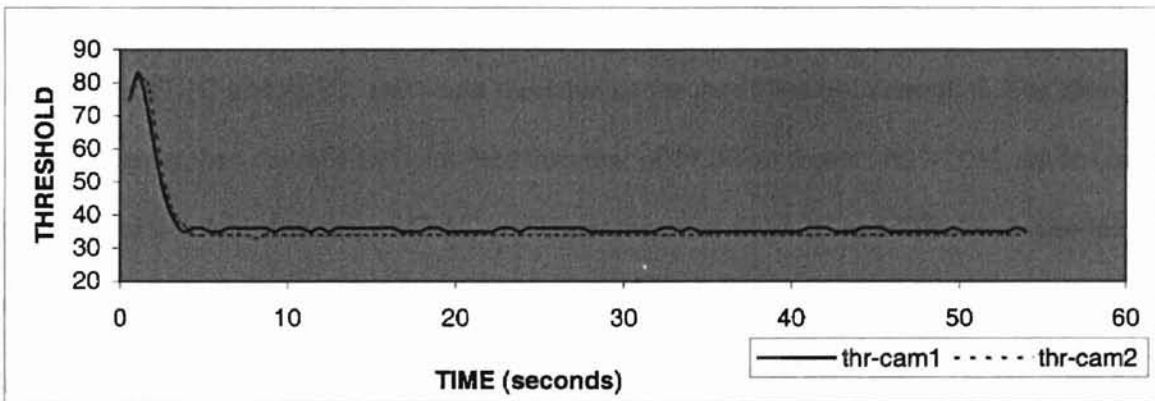
Figure 4.13 FID Controller



(a)
Gain Transit Response



(b)
Offset Transit Response

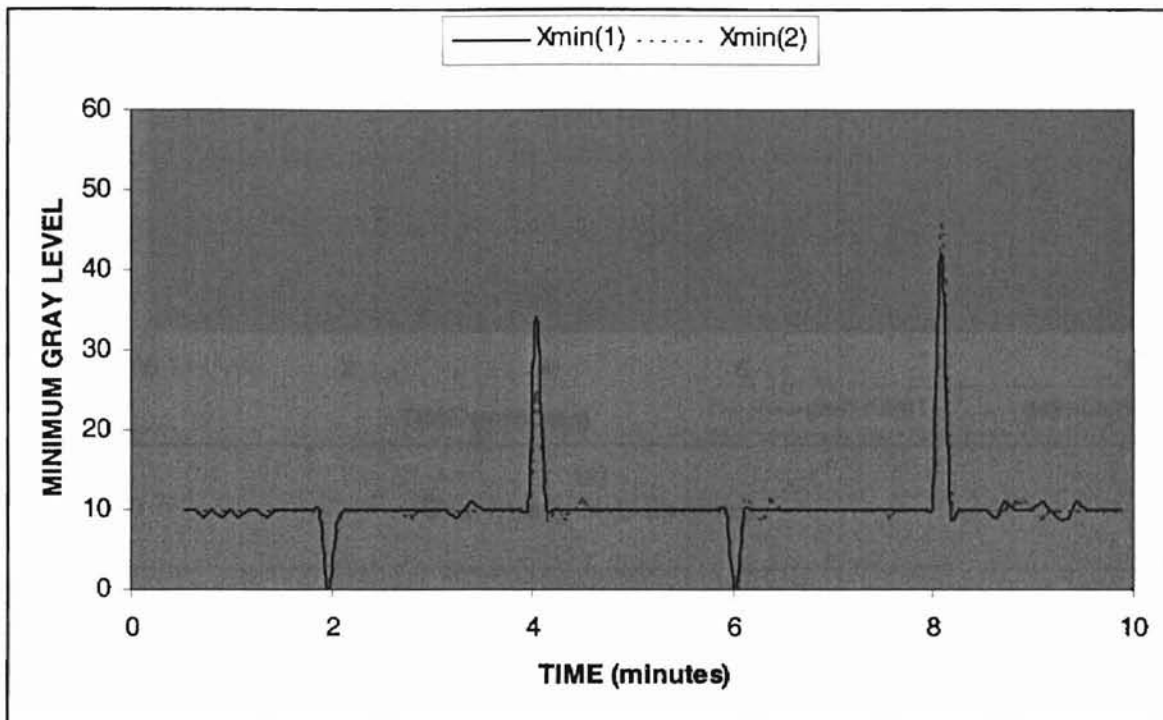


(c)
Threshold Transit Response

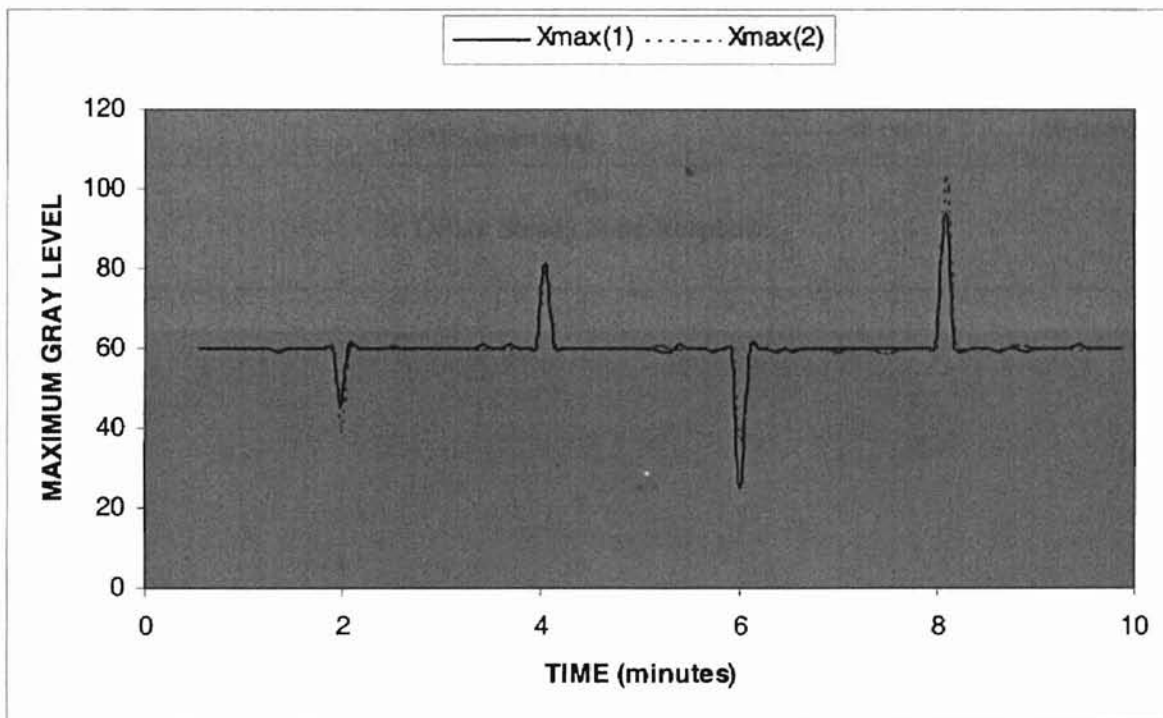
Figure 4.14 FID Controller

lighting tubes were covered at 2 and 6 minutes, the maximum gray levels of both Cameras 1 and 2 fell to 50-40 and 25-40, respectively, then recovered to 60 quickly. When the lighting tubes were uncovered at 4 and 8 minutes, the maximum gray levels of both cameras reached about 80 and 95-105, respectively, and recovered quickly to 60. Figures 4.16 (a), (b), and (c) show the dynamic responses of control variables of gain, offset, and threshold, respectively. During the time of ambient lighting change, when the lighting tubes were covered and uncovered at 2, 4, 6, and 8 minutes, the control variables gain and offset changed very fast, such that the minimum and maximum gray levels responded faster than for FL1C and FL2C. When one lighting tube was covered at 2 to 4 minutes, for Cameras 1 and 2, respectively, gain decreased by 12 (120 to 108) and 35 (196 to 231), offset increased by 23 (83 to 106) and 9 (58 to 67), while threshold decreased by 1 (37 to 38) (36 to 37). When two lighting tubes were covered at 6 to 8 minutes, for Cameras 1 and 2, respectively, gain increased by 15 (120 to 135) and 58 (196 to 254), offset increased by 33 (83 to 116) and 23 (58 to 81), while threshold decreased by 5 (37 to 32) and 2 (36 to 38). The integral square error, I_s , was 759 for Camera 1 and 818 for Camera 2.

Compared to the steady state responses tests of the other 3 controllers discussed previously, the minimum and maximum gray levels in the FID control system responded faster than for the FL1C and FL2C, and about the same as for the F(E+D)+I controller. The change of gain was larger than that of FL1C and less than that of FL2C as well as F(E+D)+I, while change of threshold was less than that of FL1C and larger than that of FL2C and F(E+D)+I. The integral square error, I_s , was the smallest one among the 4 controllers. Based on the steady response test, the FID controller is better than the other 3 controllers discussed previously. On the other hand, for the transit response test it is better than FL1C and FL2C, but not as good as the F(E+D)+I controller. The large overshoot of maximum gray level, in transit response testing, is undesirable.

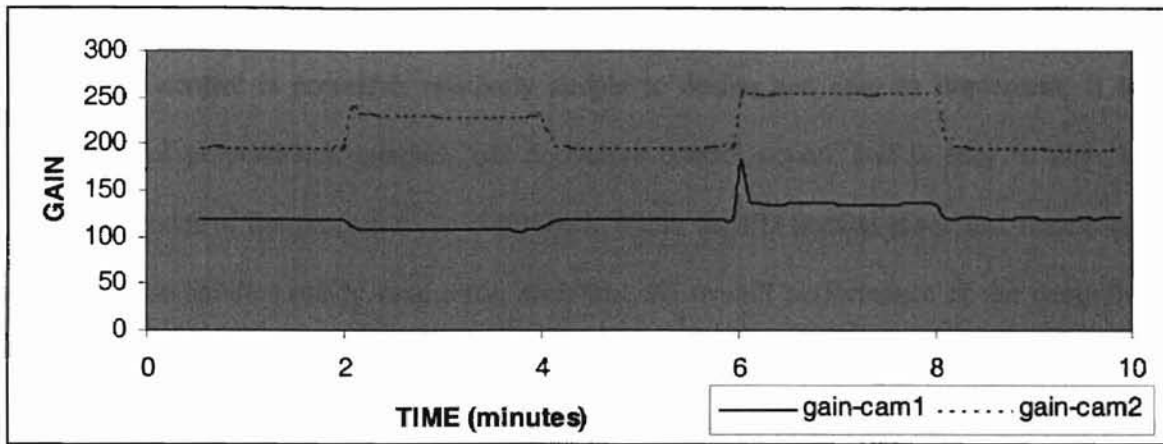


(a)
Minimum Gray Level Steady State Response

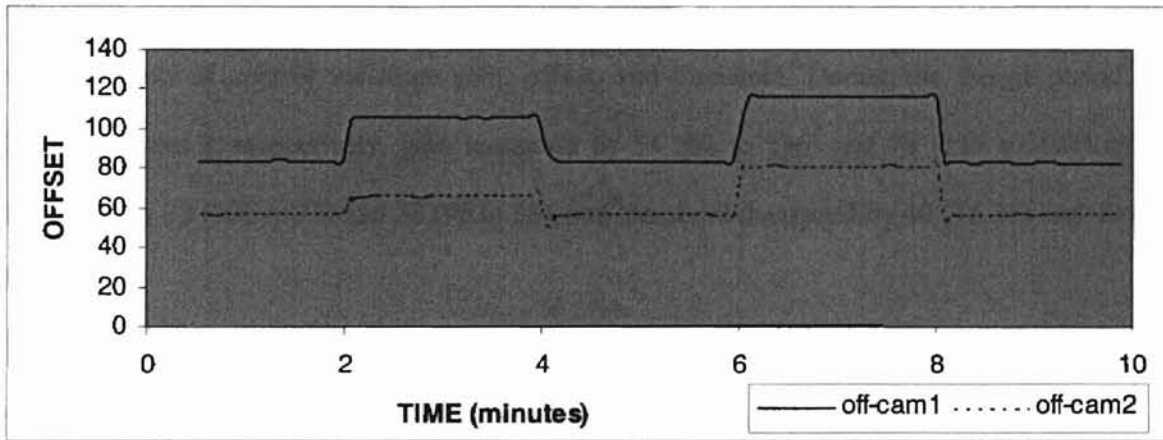


(b)
Maximum Gray Level Steady State Response

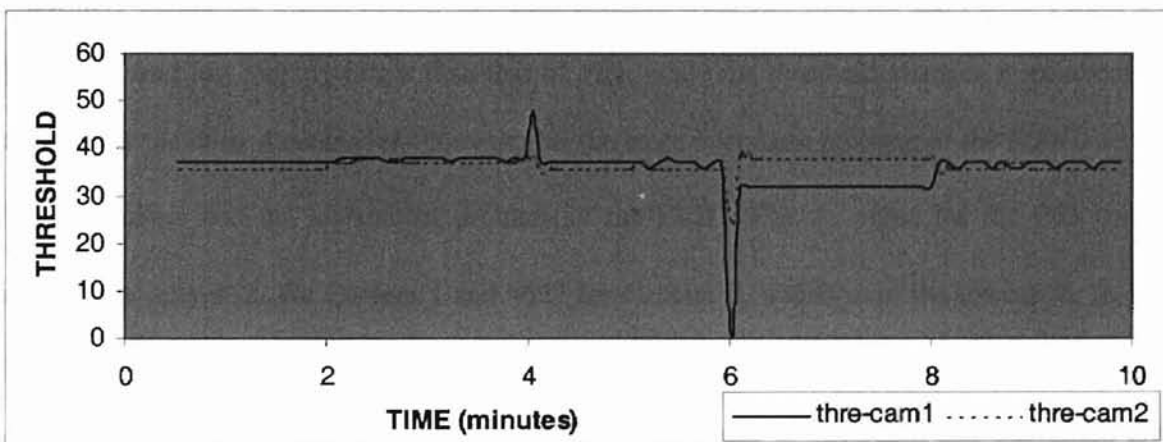
Figure 4.15 FID Controller



(a)
Gain Steady State Response



(b)
Offset Steady State Response



(c)
Threshold Steady State Response

Figure 4.16 FID Controller

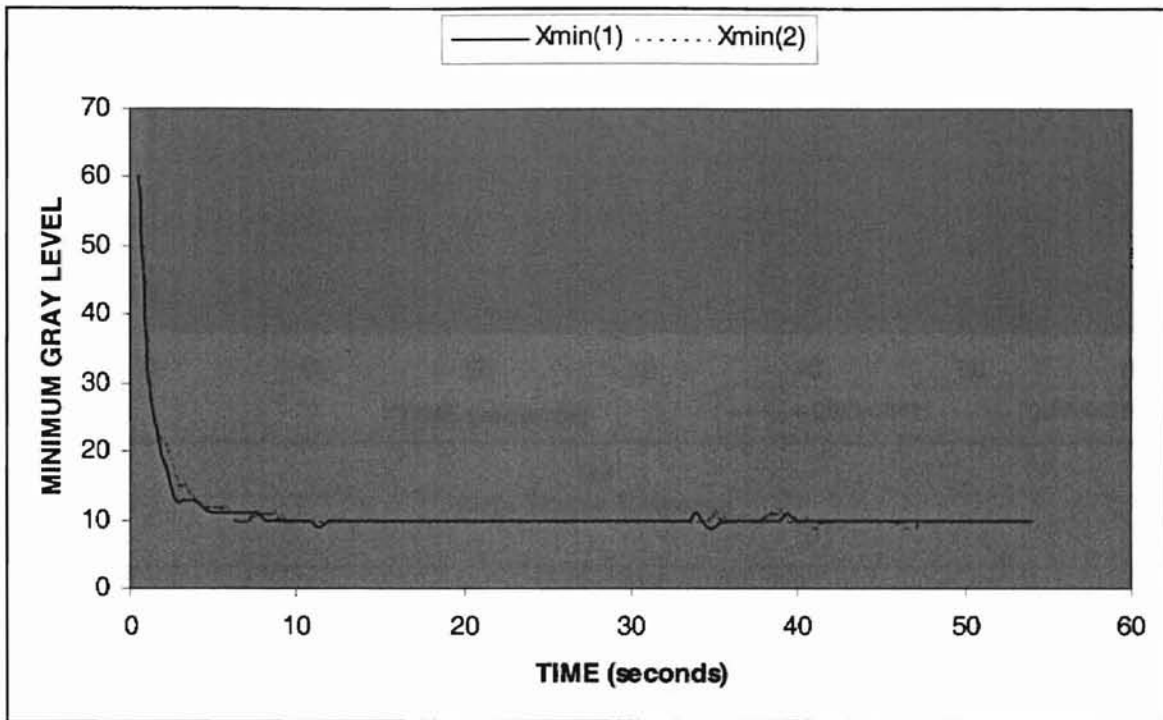
PID Controller

PID control is powerful, relatively simple to design and easy to implement. It takes advantages of proportional, integral, and derivative control action, and is easy to tune, even without a model of the process to be controlled. While the PD portion gives fast response, the integral action handles steady state error, such that the overall performance of the controller is increased significantly.

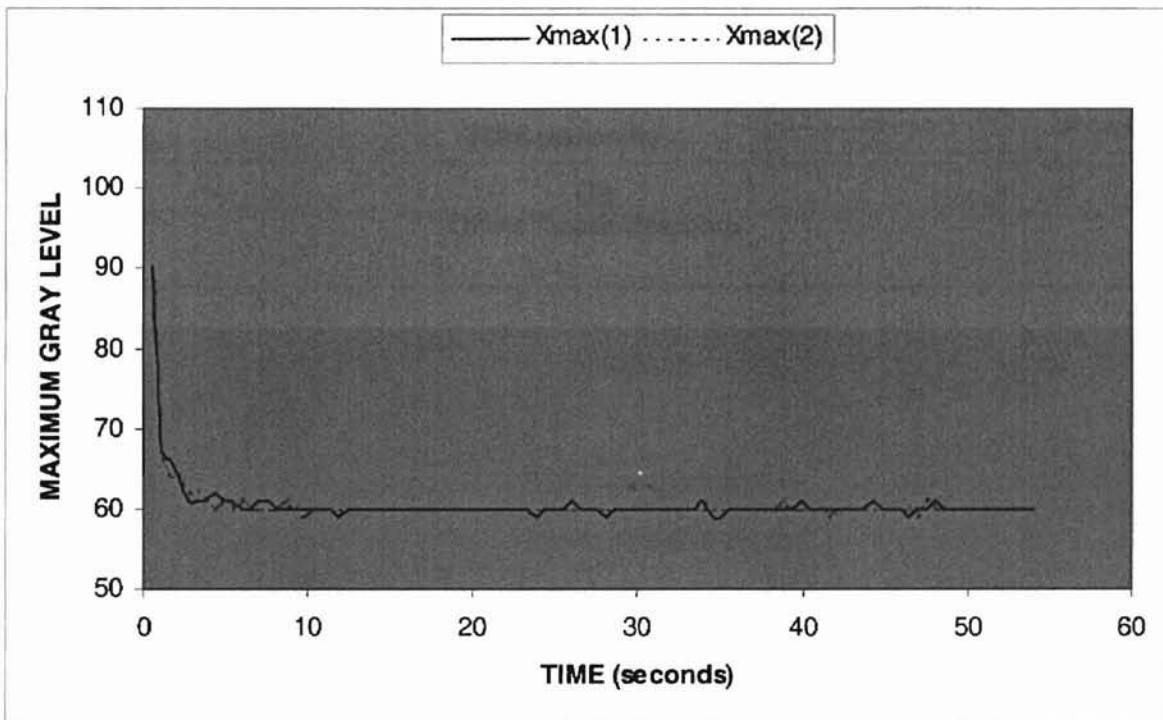
Figures 4.17 (a) and (b) show the transit response of minimum and maximum gray levels. As we can see, the minimum and maximum gray levels are forced to 10 and 60 from the prescribed initial state 60 and 90, respectively. Figure 4.18 (a), (b), and (c) show the transit response plots of control variables gain, offset, and threshold. During the transit period, for Cameras 1 and 2, respectively, gain increased by 54 (82 to 136) and 78 (119 to 197), offset decreased by 33 (106 to 73) and 38 (96 to 58), and threshold decreased by 40 (76-36) and 40 (76-36).

Compared to the other 4 controllers discussed previously, the PID controller had the fastest transit response of these 5 controllers, requiring less than 10 cycles for the system to reach desired minimum and maximum gray levels. It required the smallest offset changes among all 5 controllers during the transit period. The gain changes were less than for the FL1C, FL2C, and F(E+D)+I, and just slightly larger than that of FID, while the threshold changes were about the same as for the other 4 controllers. It overcame the large overshoot problem of the F(E+D)+I and FID controllers and the undershoot problem of the FL2C. The I_s value for the PID control system was only 4622 for Camera 1 and 4627 for Camera 2, which were the lowest for these 5 controllers.

Figures 4.19 (a) and (b) show the dynamic responses of minimum and maximum gray levels during the steady state responses tests. In Figure 4.19 (a), when lighting tubes were covered at 2 and 6 minutes, the minimum gray levels of both Camera 1 and 2 fell to 0, and recovered to 10

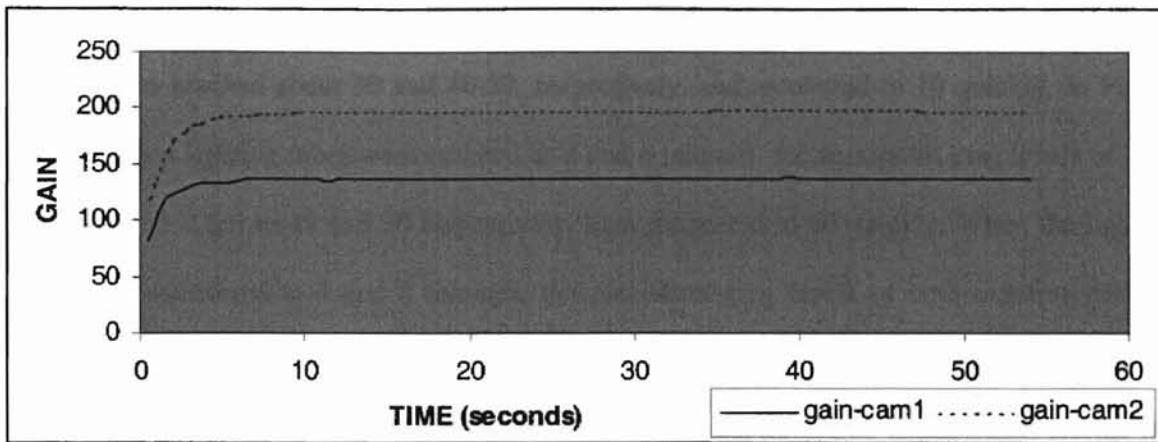


(a)
Minimum Gray Level Transit Response

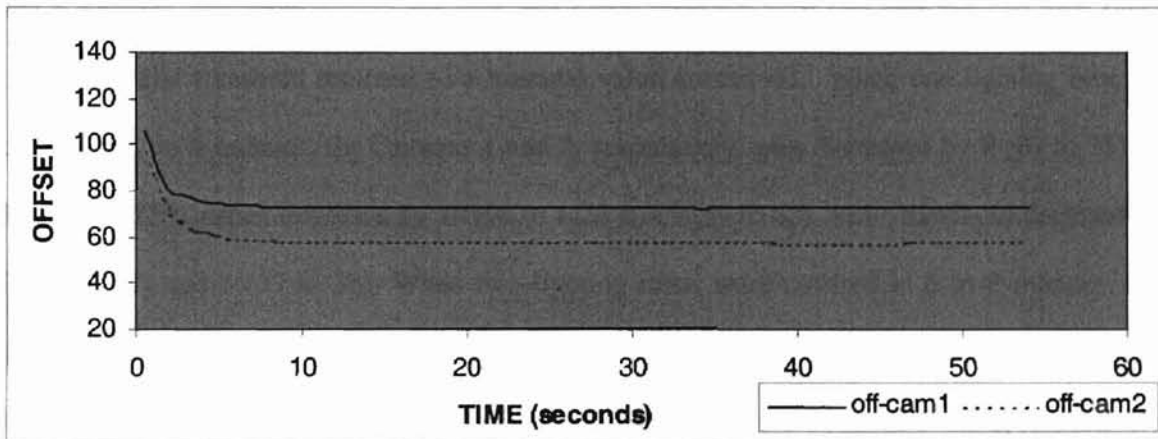


(b)
Maximum Gray Level Transit Response

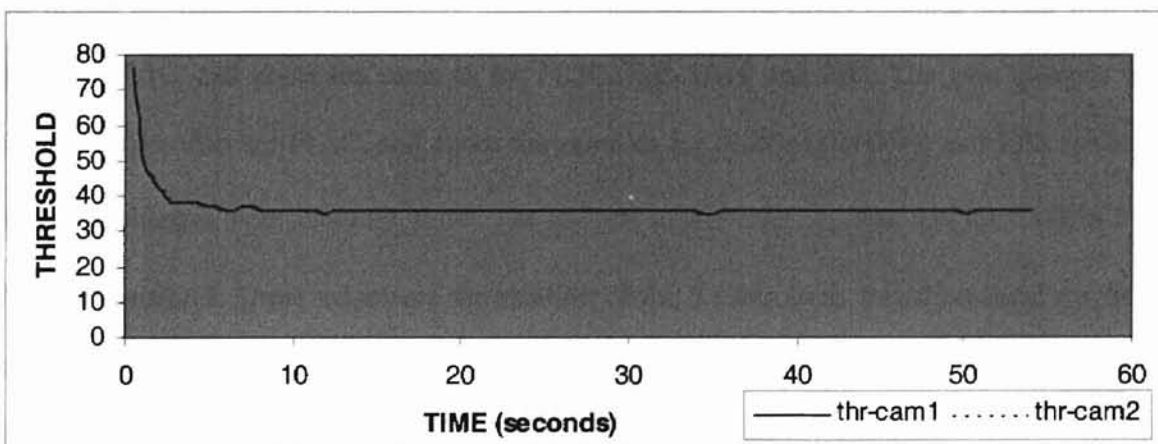
Figure 4.17 PID Controller



(a)
Gain Transit Response



(b)
Offset Transit Response

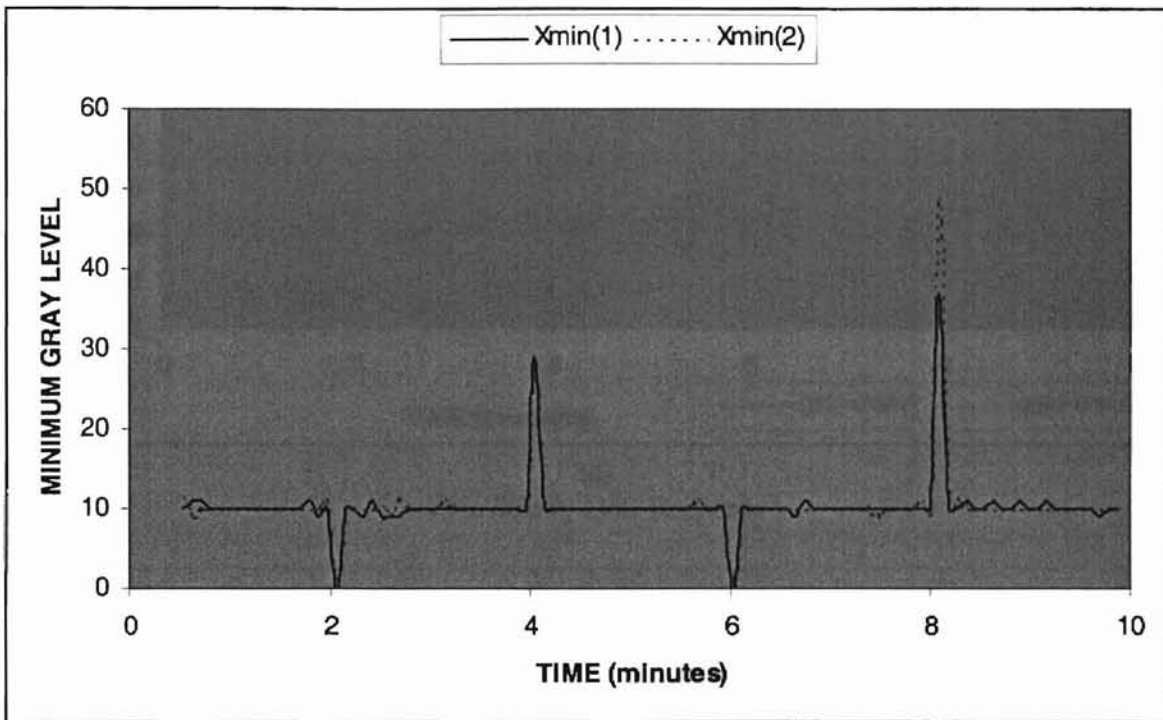


(c)
Threshold Transit Response

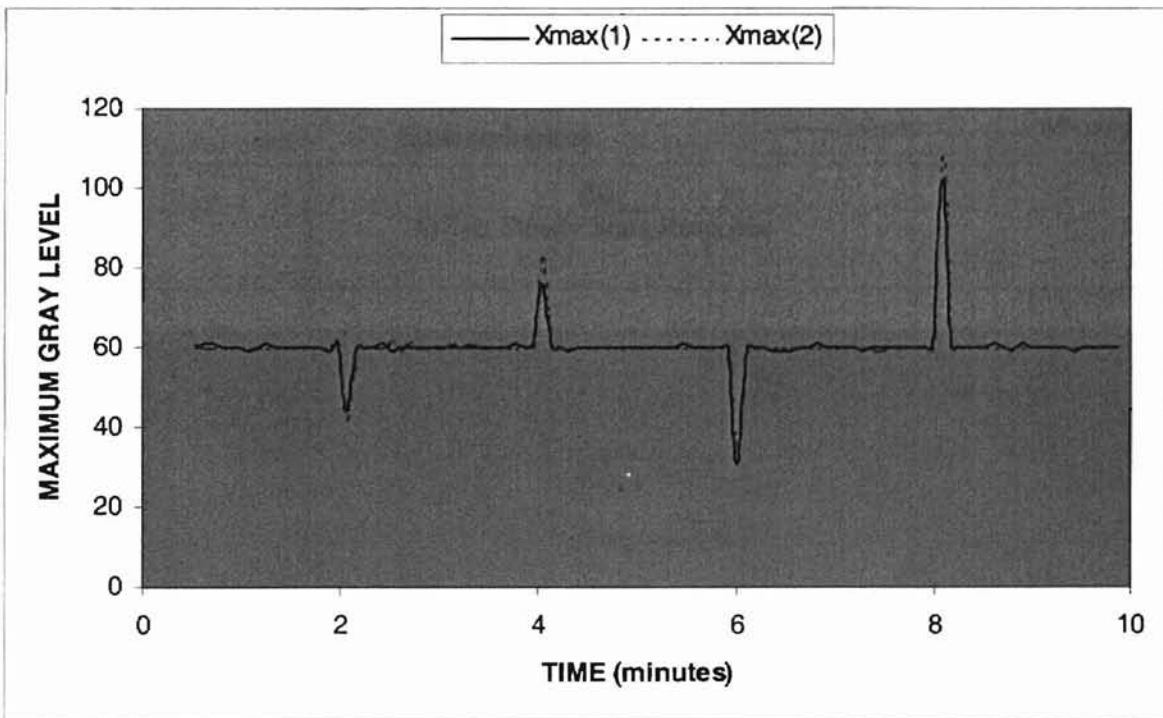
Figure 4.18 PID Controller

rapidly. When the lighting tubes were uncovered at 4 and 8 minutes, the minimum gray levels of both cameras reached about 30 and 40-50, respectively, and recovered to 10 quickly. In Figure 4.19 (b), when lighting tubes were covered at 2 and 6 minutes, the maximum gray levels of both Camera 1 and 2 fell to 45 and 30 respectively, then recovered to 60 quickly. When the lighting tubes were uncovered at 4 and 8 minutes, the maximum gray levels of both cameras reached about 75-80 and 100, respectively, and recovered quickly to 60. Figure 4.20 (a), (b), and (c) show the dynamic responses of control variables of gain, offset, and threshold, respectively. During the time of ambient lighting change, when the lighting tubes were covered and uncovered at 2, 4, 6, and 8 minutes, the control variables gain and offset changed very fast and reached new steady plateaus, while threshold returned to a nominal value around 40. When one lighting tube was covered at 2 to 4 minutes, for Cameras 1 and 2, respectively, gain decreased by 9 (87 to 78) and 31 (189 to 220), offset increased by 19 (99 to 118) and 9 (59 to 68), while threshold decreased by 2 (39 to 37) and 1 (37 to 38). When two lighting tubes were covered at 6 to 8 minutes, gain increased by 29 (89 to 118) and 53 (190 to 243), offset increased by 27 (98 to 125) and 23 (60 to 83), while threshold decreased by 9 (39 to 30) and 2 (37 to 39) for Cameras 1 and 2, respectively.

Compared to the steady state responses of the other 4 controllers, the PID controller had the fastest response speed without overshoot or undershoot. The offset changes needed were less than for FL1C, and about the same as for FL2C, F(E+D)+I, and FID. The gain changes were slightly larger than for FL1C, and about the same as for FL2C, F(E+D)+I, and FID, while the threshold changes were about the same. The integral square error, I_e , was 539 for Camera 1 and 714 for Camera 2. These values are the smallest of the 5 controllers. Based on these results for both transit and steady state response, the PID controller designed in this work is the best one for this application.

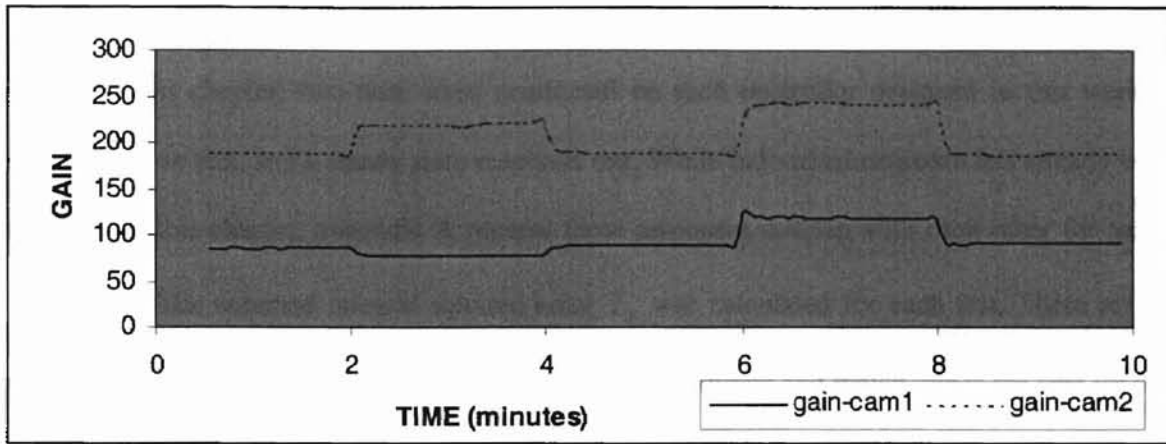


(a)
Minimum Gray Level Steady State Response

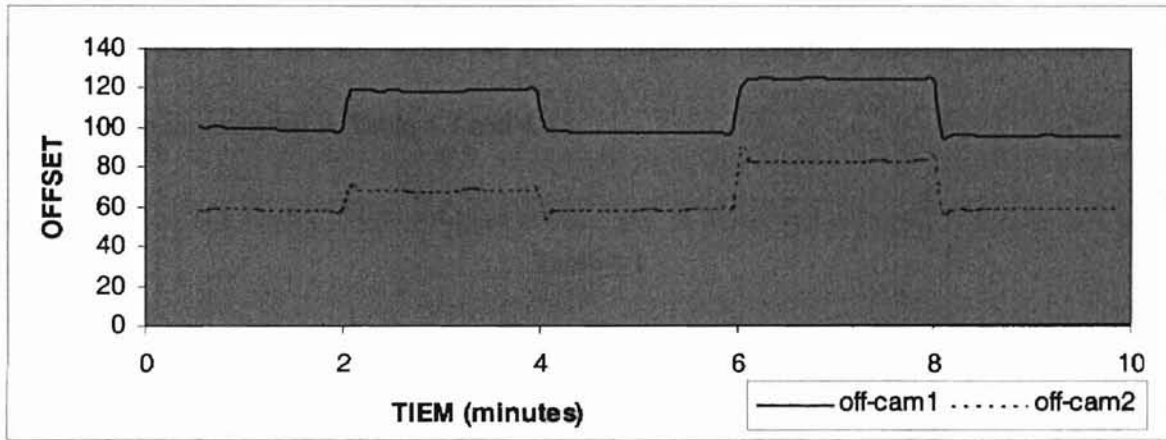


(b)
Maximum Gray Level Steady State Response

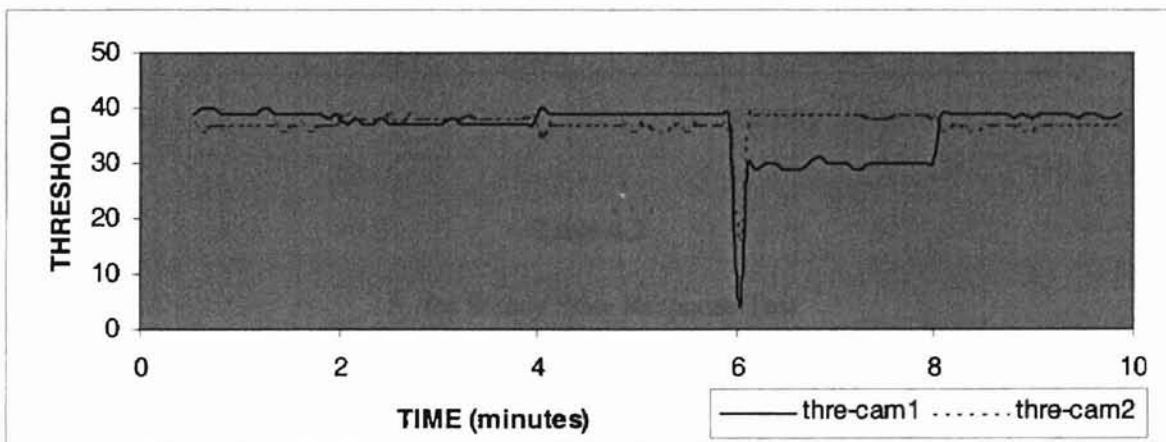
Figure 4.19 PID Controller



(a)
Gain Steady State Response



(b)
Offset Steady State Response



(c)
Threshold Steady State Response

Figure 4.20 PID Controller

Summary

In this chapter, two tests were conducted on each controller designed in this work, a transit response test, and a steady state response test. While individual response has already been presented in this chapter, appendix A present these responses overlap with each other for easier comparison. The summed integral squared error I_s was calculated for each test. These results provided good indication of how well each controller functioned and which one is the best for this application. While the plots of the output variables X_{min} , X_{max} and control variables gain, offset, and threshold in both tests were presented during the discussion, we summarize the values for I_s in Table 4.1 and 4.2, while the peak changes of control variables gain, offset, and threshold are summarized in Table 4.3 and 4.4.

Table 4.1

I_s for Transit Response Test

Controller Camera	FL1C	FL2C	F(E+D)+I	FID	PID
1	23825	6994	12780	13285	4622
2	25227	7639	8834	15604	4627

Table 4.2

I_s for Steady State Response Test

Controller Camera	FL1C	FL2C	F(E+D)+I	FID	PID
1	4331	1739	887	759	539
2	4003	1024	997	818	714

Table 4.3

Peak Control Variable Changes for Transit Response Test

	Controller		FL1C	FL2C	F(E+D)+I	FID	PID
	Camera						
ΔG	1		99(64-163)	107(58-165)	28(113-141)	28(112-140)	54(82-136)
ΔG	2		135(109-244)	142(103-245)	116(84-200)	56(150-206)	78(119-197)
ΔO	1		-72(134-62)	-61(123-62)	-57(128-71)	-57(129-72)	-33(106-73)
ΔO	2		-82(122-40)	-70(111-41)	-64(120-56)	-64(119-55)	-38(96-58)
ΔT	1		-39(74-35)	-40(75-35)	-40(75-35)	-40(75-35)	-40(76-36)
ΔT	2		-41(73-32)	-40(74-34)	-41(75-34)	-41(75-34)	-40(76-36)

where: ΔG = Gain change; ΔO = Offset change; ΔT = Threshold change; "-" means decrease.

Table 4.4

Peak Control Variable Changes for Steady State Response Test

		Controller		FL1C	FL2C	F(E+D)+I	FID	PID
		Camera						
ΔG	One tube	1		-3(173-170)	7(61-68)	41(66-107)	-12(120-108)	-9(87-78)
		2		33(219-252)	33(212-245)	44(188-232)	35(196-231)	31(189-220)
ΔG	Two Tube	1		12(173-185)	96(70-166)	61(71-132)	15(120-135)	29(89-118)
		2		22(231-253)	41(211-252)	58(194-252)	58(196-254)	53(190-243)
ΔO	One tube	1		24(59-83)	14(114-128)	15(110-125)	23(83-106)	19(99-118)
		2		11(49-60)	11(49-60)	6(59-65)	9(58-67)	9(59-68)
ΔO	Two Tube	1		40(59-99)	23(109-132)	23(107-130)	33(83-116)	27(98-125)
		2		33(46-79)	29(49-78)	24(57-81)	23(58-81)	23(60-83)
ΔT	One tube	1		4(33-37)	-10(40-30)	-5(39-34)	1(37-38)	-2(39-37)
		2		2(35-37)	2(36-38)	1(36-37)	1(36-37)	1(37-38)
ΔT	Two Tube	1		-2(34-32)	-18(40-22)	-15(40-25)	-5(37-32)	-9(39-30)
		2		0(36-36)	2(36-38)	2(36-38)	2(36-38)	2(37-39)

where: One Tube = cover one lighting tube each side; Two Tube = cover two lighting tubes each side; and “-” means decrease.

For the transit response tests, the PID controller brought the output variables from the prescribed initial state (60 for minimum and 90 for maximum) to the desired final state (10 for minimum and 60 for maximum) at the fastest rate without overshoot or undershoot and with the smallest value for I_s . It required the smallest offset changes among all these 5 controllers during the transit period. The gain changes were less than for FL1C, FL2C, and F(E+D)+I, and just slightly larger than that of FID, while the threshold changes were about the same as for the other 4 controllers, as listed in Table 4.3. We judge it the best, by far, for this test. Based on the plots of minimum and maximum gray levels, the FID and F(E+D)+I controllers are faster than FL2C from the prescribed initial state to the desired final state. However, due to the large overshoot that occurred in the maximum gray level transit response, the I_s values for the FID and F(E+D)+I controllers in Table 4.1 are much larger than that for the FL2 Controller. Among these 5 controllers, the FL1 Controller had the slowest transit response with highest I_s value, and was judged the worst.

For steady state response tests, when ambient lighting changed at 2, 4, 6, and 8 minutes, the PID controller exhibited the fastest correction with control variables gain, offset, and threshold, such that the minimum and maximum gray level outputs were forced to recover from disturbances to desired values in the least time. And as indicated in Table 4.2, the I_s value for the PID controller is the lowest of the 5 controllers. The offset changes needed were less than for FL1C, and about the same as for FL2C, F(E+D)+I, and FID. The gain changes were slightly larger than for FL1C, and about the same as for FL2C, F(E+D)+I, and FID, while the threshold changes were about the same, as listed in Table 4.4. Again the PID controller was the best as far as speed of response and robustness are concerned. The FID and F(E+D)+I controllers showed better performance than FL2 Controller for the steady state response test. Again, the FL1

Controller exhibited the slowest response to all ambient lighting disturbances and had the highest I_s value as listed in Table 4.2. In summary, the PID controller designed in this work is the best choice for our dishwashing automation system, based on both transit response and steady state response tests.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

The goal of this research was to analyze CCD camera performance dynamics and develop a good controller to implement automatic and continual calibration of the cameras in real time. Two tests were conducted in order to evaluate the camera performance dynamics. One was a 55-minute long term test without ambient lighting disturbances; the other was a 10-minute short term test with ambient lighting disturbances applied intentionally. Five different controllers were designed using conventional PID and fuzzy logic control techniques. Two tests were used to evaluate the performance of each controller, a transit response and a steady state response test. The integral square error (ISE) was calculated on line during each test and gave a quantitative indication of performance. Following are the major contributions of this work together with some recommendations.

Conclusions

1. Without control, the minimum and maximum gray levels for two tested CCD cameras shifted away from the desired initial setting (in this experiment 10 for minimum and 60 for maximum) over time, even without large disturbances in ambient lighting. The variations were unacceptable for high quality, repeatable imaging.
2. With substantial disturbances in lighting intensity introduced, the minimum and maximum gray levels changed dramatically if no control was applied. The larger the lighting disturbance, the larger was the variation in gray level. The variations were unacceptable for high quality, repeatable imaging.

3. Five camera controllers were proposed and implemented to realize automatic and continual adjustment of the camera parameters “gain”, “offset”, and “threshold” to compensate for power fluctuation, changes in ambient lighting, and camera sensitivity drift in our machine vision system. These controllers were:
 - Fuzzy Logic 1 Controller (FL1C)
 - Fuzzy Logic 2 Controller (FL2C)
 - Fuzzy-Integral Controller (F(E+D)+I)
 - Fuzzy-Integral-Derivative Controller (FID)
 - Proportional-Integral-Derivative Controller (PID)
4. Each of these 5 controllers applied to the vision system improved the performance significantly. Among these 5 controllers, the PID controller was the best for this machine vision system, giving the fastest transit response without overshoot and undershoot, and maintaining the highest robustness during the steady state test when intentional ambient lighting disturbances were applied. The ISE value for the PID controller in both tests was the smallest of the 5 controllers.
5. During the transit response test with the PID controller, the minimum and maximum gray levels reached desired values quickly from preset initial states without any oscillation, as shown in Figure 4.17. During the steady state response test, the PID controller reacted to disturbances quickly, forcing the minimum and maximum gray levels back to desired values without any oscillation, as shown in Figure 4.19. Moreover, the ambient lighting disturbances, introduced during the steady state response test, were relatively large in amplitude at a relatively high frequency (once per two minutes), which constituted much worse conditions than that would likely ever occur in industry applications. Based on our extensive test results, not all of which were documented herein, the PID control system demonstrated very stable performance over time. All of these results give a high level of

confidence that the PID camera control system will exhibit good, stable behavior, even though a separate stability analysis was not investigated in this work.

Recommendations

After investigating transit and steady state responses, and calculating the ISE value for each controller, the PID controller appeared the best for this machine vision application. Analyzing the control action of the PID controller during our steady state response test, as shown in Figure 4.19 and 4.20, demonstrated that different levels of control action were needed to overcome different intensity of disturbances, as expected. Larger ambient lighting disturbances require larger control actions. If the disturbances are extremely large, the control action of one of the parameters could reach its limit, such as 256 for gain, 255 for offset, or 127 for threshold. In this case, control saturation would occur, and new control algorithms would need to be investigated. Since the control action is determined by certain combinations of the control parameters gain, offset and threshold, it may be possible to fix the parameter that reached its limit and increase or decrease a preset increment on the other parameters according to an appropriate algorithm. More experiments need to be done to address possible control saturation problems. Moreover, in this research we did not address system stability for each control system, mainly because we sought to find a reasonable controller without modeling camera dynamics. In order to address system stability analytically, a model of the camera dynamics needs to be found, and further investigation would be required.

BIBLIOGRAPHY

1. Adept V/V+ Programming Course Manual. (1991). Adept Technology, Inc. San Jose, CA.
2. AdeptVision AGS-GV Programming Course Manual. (1991). Adept Technology, Inc. San Jose, CA.
3. AdeptVision FGS User's Guide. (1990). Adept Technology, Inc. San Jose, CA.
4. AdeptVision Reference Guide. (1990). Adept Technology, Inc. San Jose, CA.
5. Brainard, D. H., Wandell, B. A., and Cowan, W. B. (1989). Black Light: How Sensors Filter Spectral Variation of The Illuminant, *IEEE Trans. Biomedical Engineering*, **36** (1), 140-149.
6. Byler, E., Chun, W., Hoff, W., and Layne, D. (1995, March). Autonomous Hazardous Waste Drum Inspection Vehicle, *IEEE Robotics and Automation Magazine*, 6-17.
7. Chang, Y. C. and Reid, J. F. (1996). RGB Calibration for Color Image Analysis in Machine Vision, *IEEE Trans. Image Processing*, **5** (10), 1414-1422.
8. Chen, B., and Hoberock, L. L. (1995 August). Fuzzy Logic controller for Automatic Vision Parameter Adjustment in a Robotic dish Handling System. *In Proc. The 10th IEEE International symposium on Intelligent Control*, Monterey, CA.
9. Chen, B. (1996, May). Unsupervised and Supervised Fuzzy Neural Network Architecture, With Applications in Machine Vision fuzzy Object recognition and Inspection, Thesis, Oklahoma State University, Stillwater, OK.
10. Dorf, R. C., and Bishop, R. H. (1995). *Modern Control Systems*, 7th Edition, Addison Wesley, Reading, MA.
11. Feng, K., and Hoberock, L. L. (1996 March). A Modified ARTMAP Network, With Applications to Scheduling of A Robot-Vision-Tracking Systems, *ASME Journal of Dynamic Systems, Measurement, and Control*, **118** (1).
12. Freeman, H. (1989). *Machine Vision for Inspection and Measurement*, Academic press, Inc.
13. Gerald, C. F., and Wheatley, P. O. (1994), *Applied Numerical Analysis*, 5th Edition, Addison Wesley, Reading, MA.

14. Gergenback, V., Nagel H. H., Tonko, M., and Schafer, K.(1996). Automatic Dismantling Integration Optical Flow into a Machine Vision-Controlled Robot System, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, **2**, 1320-1325. Minneapolis, Minnesota.
15. Grewe, L., and Kak, A. (1993). Integration of Geometric and Non-Geometric Attributes for Fast Object Recognition, *Proceedings SPID – The International Society for Optical Engineering. Applications of Artificial Intelligence. 1993: Machine Vision and Robotics*, **1964**, 13-28, Orlando, Florida.
16. He, D., Hujic, D., Mills, J. K., and Benhabib, B. (1996). Moving – Object Recognition Using Premarking and Active Vision, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, **3**, 1980-1985. Minneapolis, Minnesota.
17. Healey, G. and Kondepudy, R.(1994). Radiometric CCD Camera Calibration and Noise Estimation, *IEEE Trans. Pattern Analysis and Machine Intelligence*, **16** (3), 267-276.
18. Hellendoorn, H., Thomas, C. (1993). Defuzzification in Fuzzy Controllers, *Journal of Intelligent and Fuzzy Systems*, **1**, 109-123.
19. Jain, R., Kasturi, R., and Schunck, B. G. (1995). *Machine Vision*, McGraw-Hill, Inc.
20. Johnson, A. K. (1993, July). *Machine Vision sorting and Inspection in Commercial Automatic Dishwashing*, Thesis, Oklahoma State University, Stillwater, OK.
21. Joshi, R., and Sanderson, A. C.(1996). Application of Feature-Based Muti-View Servoing for Lamp Filament Alignment, *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, **2**, 1306-1313. Minneapolis, Minnesota.
22. Kim, I., and Vachtsevanos, G. (1998, September). Overlapping Object Recognition : Aparadiam for Multiple Sensor Fusion, *IEEE Robotics and Automation Magazine*, 37-44.
23. Kuo, B. C.(1995). *Automatic Control System*, 7th Edition, Prentice Hall, Englewood Cliffs, NJ.
24. Li, Y. F., and Lee, M. H.(1996, March). Applying Vision Guidance in Robotic Food Handling, *IEEE Robotics and Automation Magazine*, 4-12.
25. Mandow, A., Gomez-de-gabriel, J. M., Martinez, J. L., Munoz, V. F., Ollero, A., and Garcia-Cerezo, A. (1996, December). The autonomous Mobile Robot Aurora for Greenhouse Operation, *IEEE Robotics and Automation Magazine*, 18-28.

26. Mohtadi, O., Sagar, F., and Sanz, J. L. C. (1992). AGUILA: an automatic tube detection system, *Proceedings of SPIE-The International Society for Optical Engineering. Applications of Artificial Intelligence X: Machine Vision and Robotics*, **1708**, 483-495. Orlando, Florida.
27. Murase, H., and Nayar, S. K. (1993, July). Learning Object Models From Appearance, *Proceedings of AAAI*, Washington, D.C.
28. Murase, H., and Nayar, S. K. (1994, June). Illumination Planning for Object Recognition in Structured Environments, *Proceedings of IEEE, International Conference on Computer Vision and Pattern Recognition, Seattle*, 31-38.
29. Murase, H., and Nayar, S. K. (1995, January). Visual Learning and Recognition of 3D Objects from Appearance. *International Journal of Computer Vision*, **14** (1), 5-24.
30. Nelson, B. J., Papanikolopoulos, N. P., and Khosla, P. K. (1996, June). Robotic Visual Servoing and Robotic Assembling Tasks, *IEEE Robotics and Automation Magazine*, 23-31.
31. Parra-Loera, R. (1992). Partial Shape Recognition of 2-K Shapes Using Normalized Internal Vertex Descriptors, *Proceedings of SPIE – The International Society for Optical Engineering. Machine Vision Applications, Architectures, and Systems Integration*. **1823**, 37-45, Boston, Massachusetts.
32. Pascoal, A., Oliveira, P., Silvestre, C., Bjerrum, A., Lshoy, A., Pignon, J. P., Ayela, G., and Petzelt, C. (1997, December). Marius : An Autonomous Underwater Vehicle for Coastal Oceanography, *IEEE Robotics and Automation Magazine*, 46-59.
33. Ross, T. J. (1995). *Fuzzy Logic With Engineering Applications*, McGraw-Hill, Inc. New York, NY.
34. Tappan, J. H., Wright, M. E., and Sistler, F. E. (1987). Error Sources In A Digital Image Analysis System, *Computers and Electronics in Agriculture*, **2**, 109-118.

APPENDIXES

APPENDIX A
COMPOSITE PLOTS FOR TRANSIT RESPONSE TEST
AND STEADY STATE RESPONSE TEST

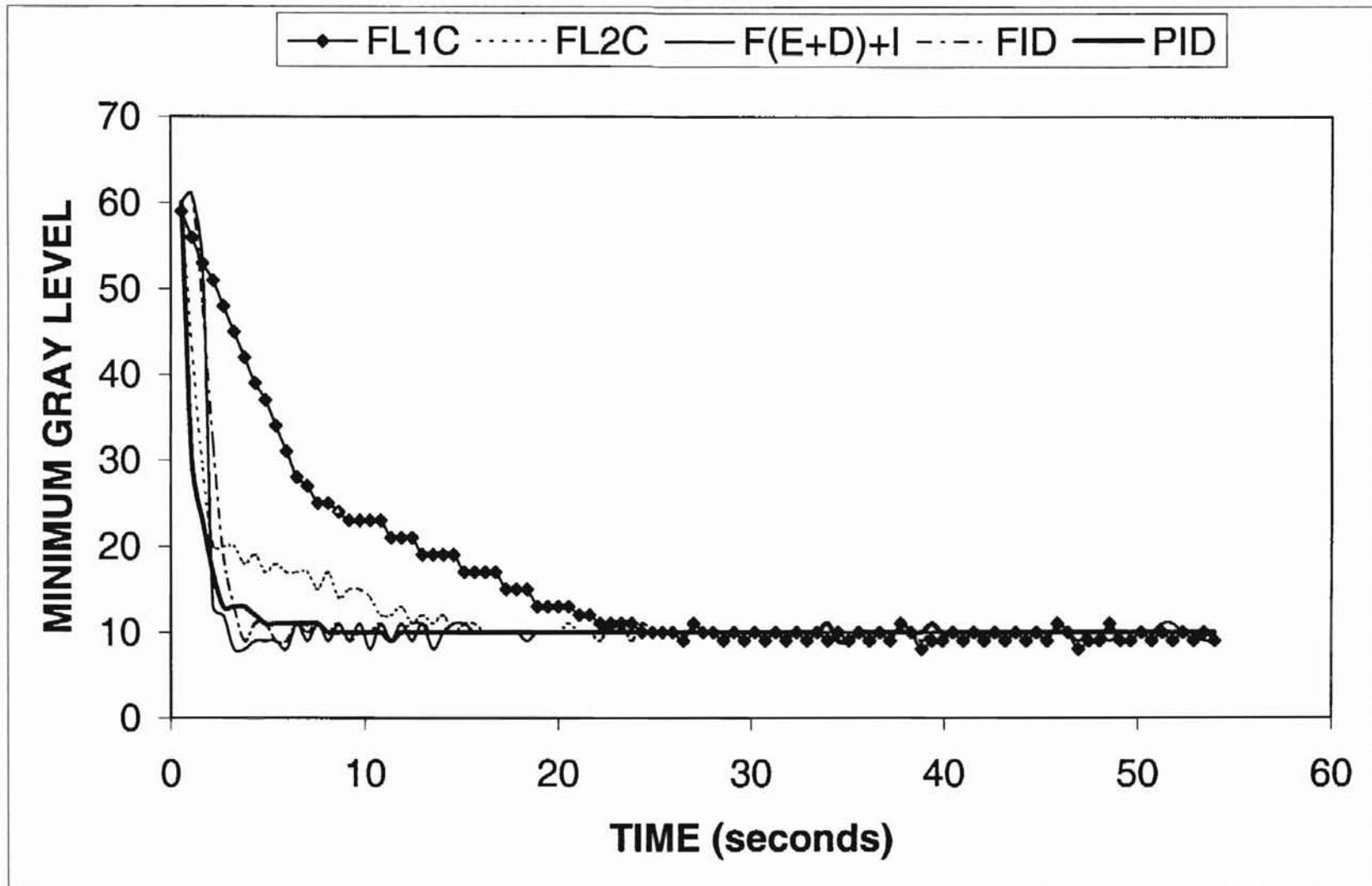


Figure A.1
Transit Response-Minimum Gray level (camera 1)

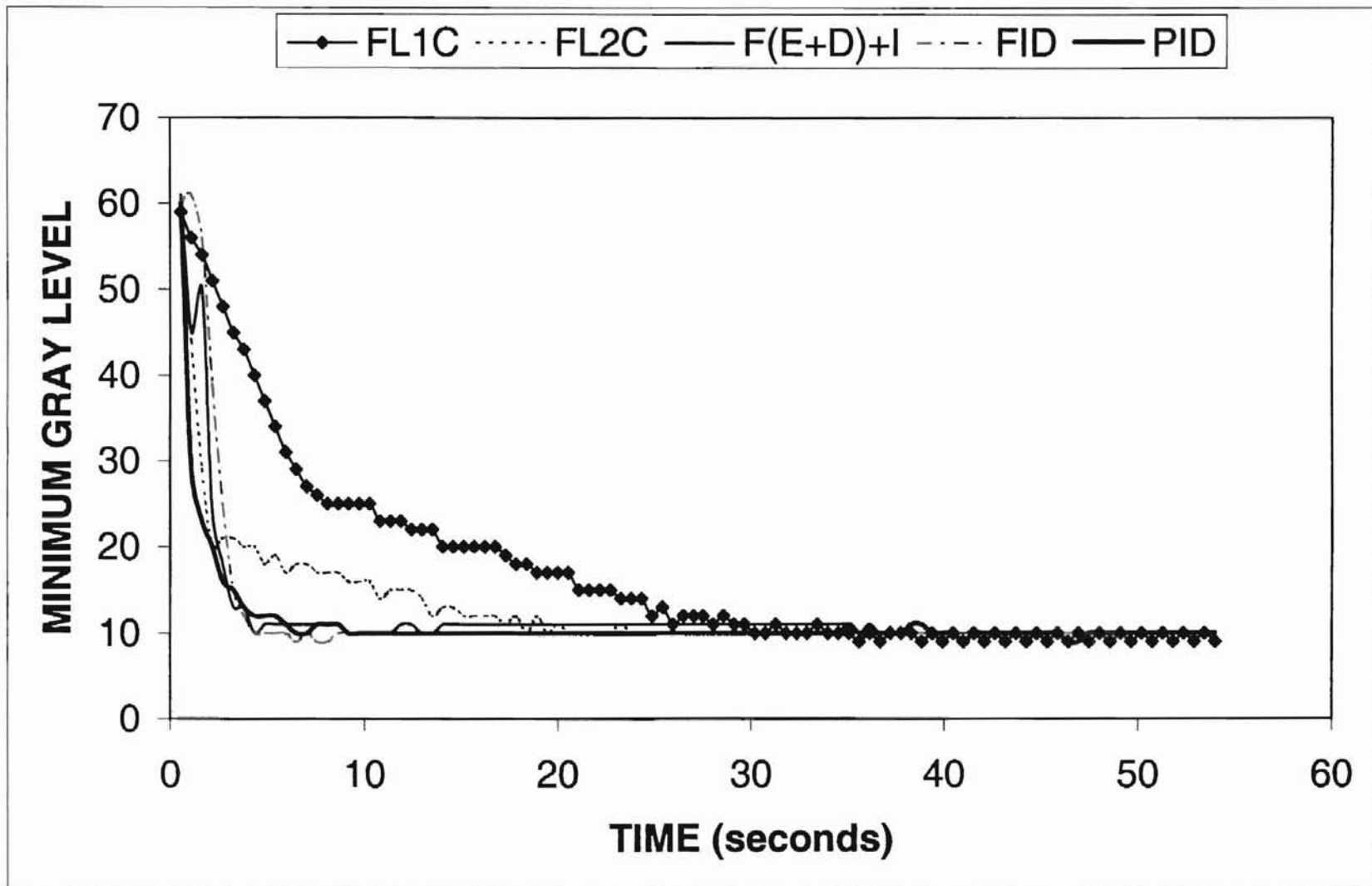


Figure A.2
Transit Response-Minimum Gray Level (camera 2)

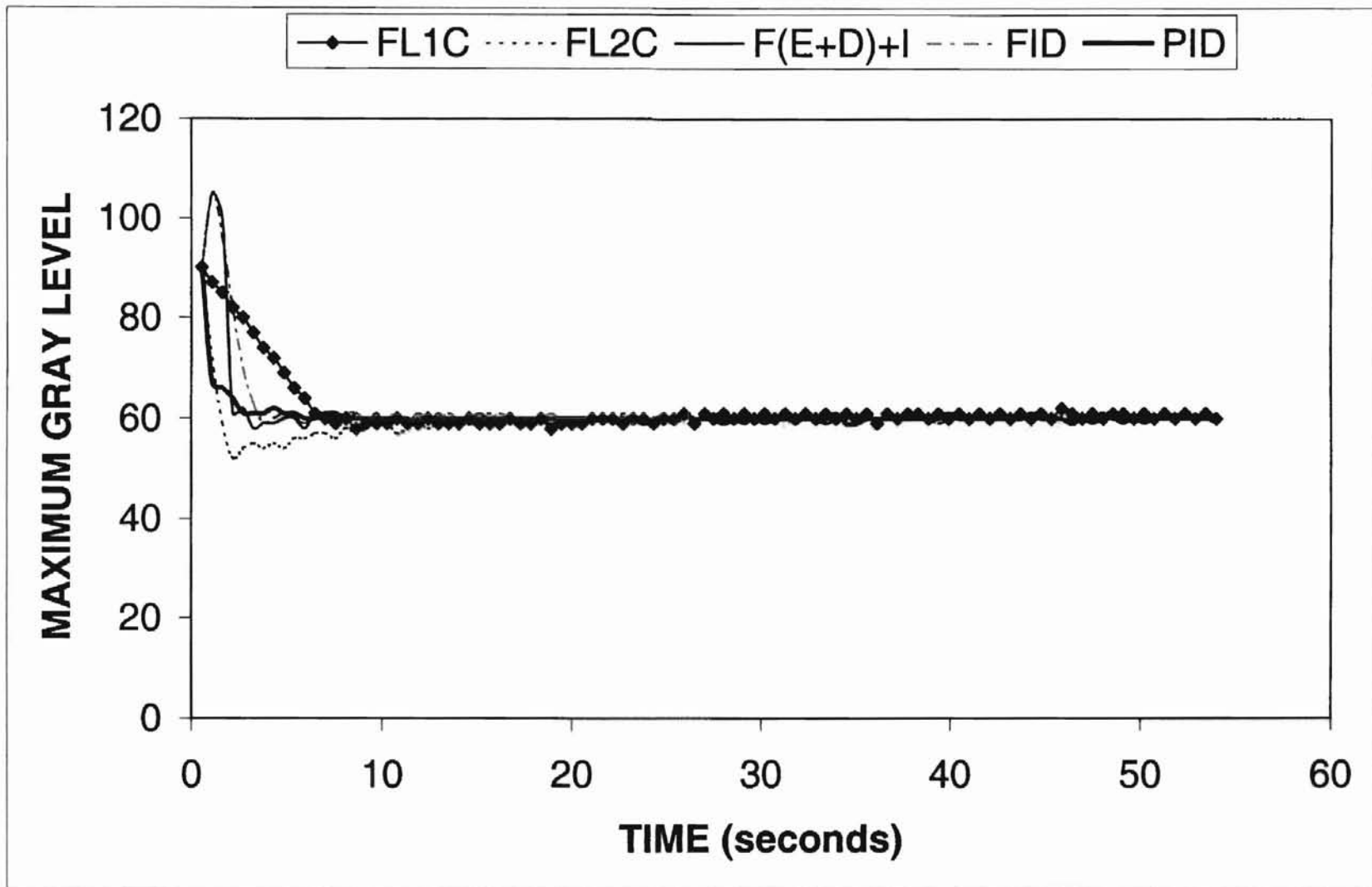


Figure A.3
Transit Response-Maximum Gray Level (camera 1)

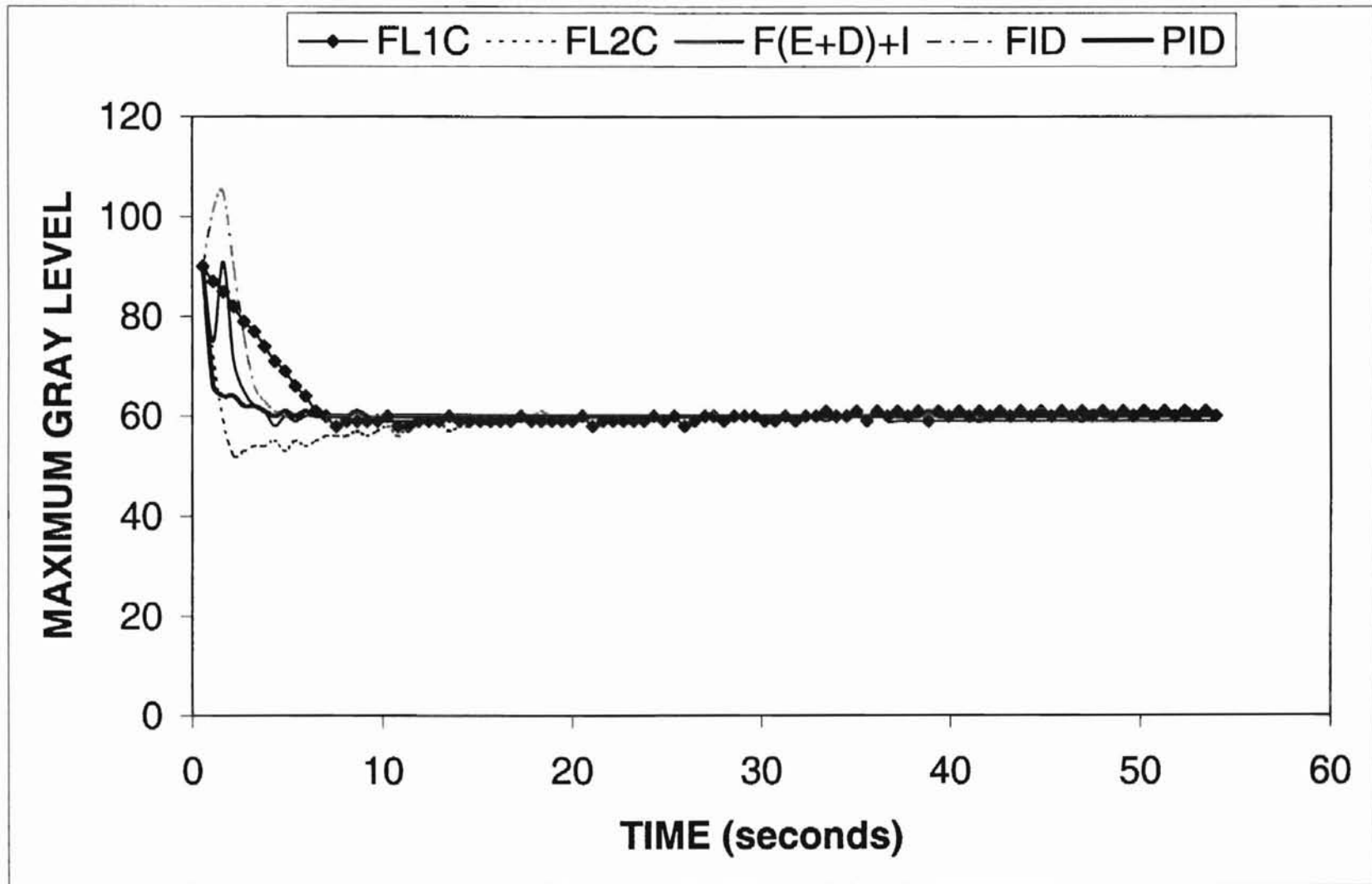


Figure A.4
Transit Response-Maximum Gray Level (camera 2)

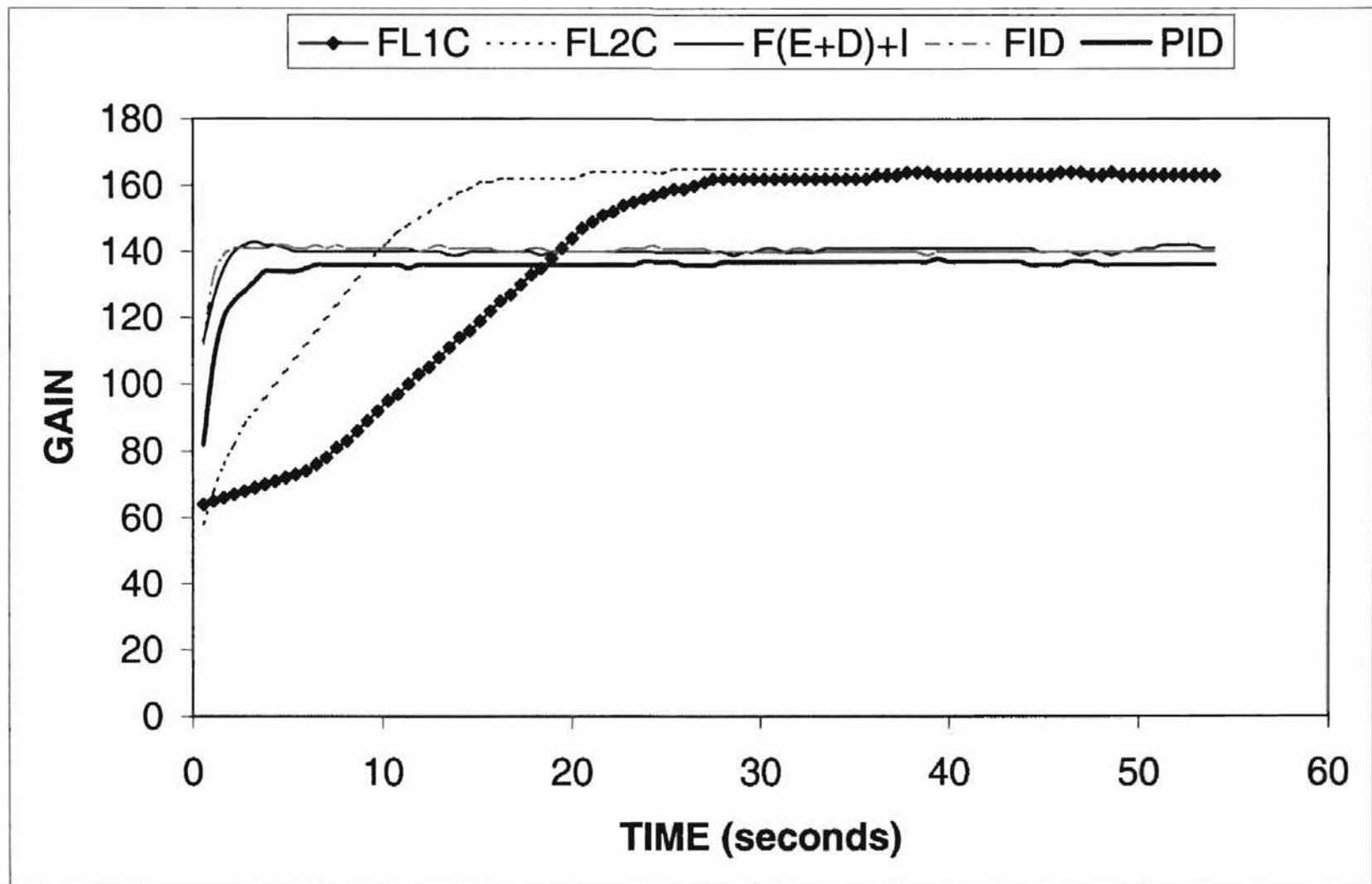


Figure A.5
Transit Response-Gain (camera 1)

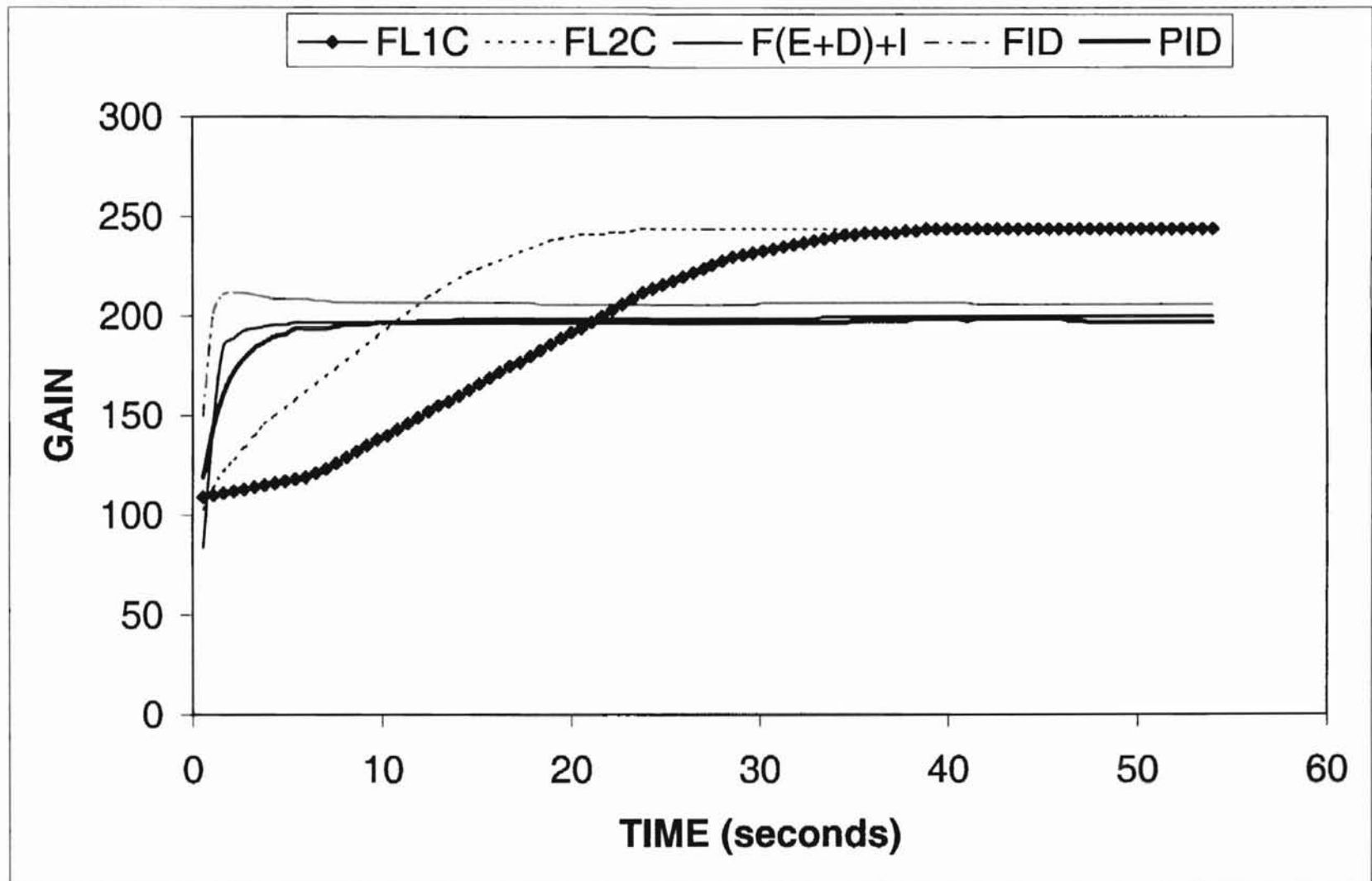


Figure A.6
Transit Response-Gain (camera 2)

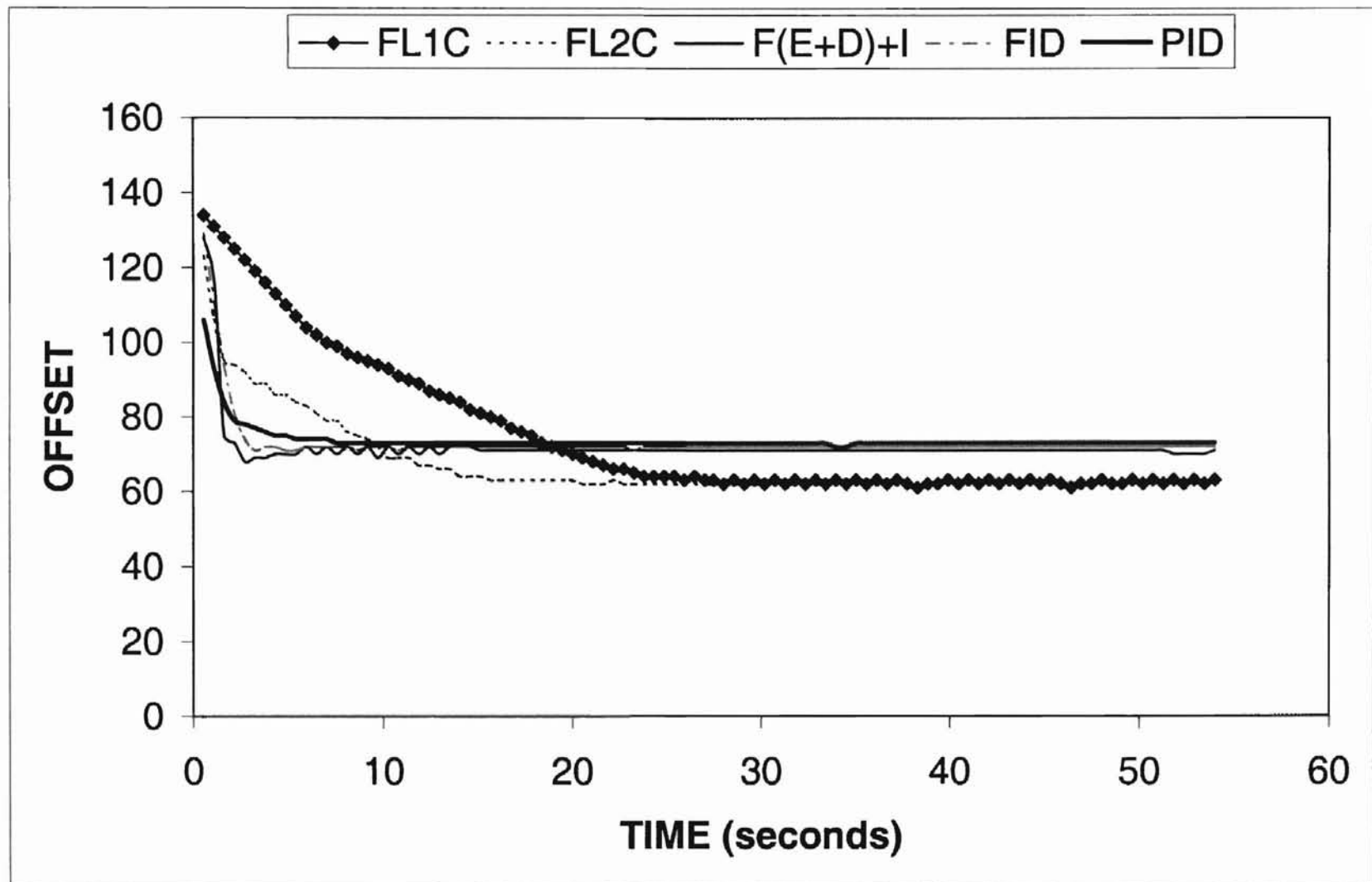


Figure A.7
Transit Response-Offset (camera 1)

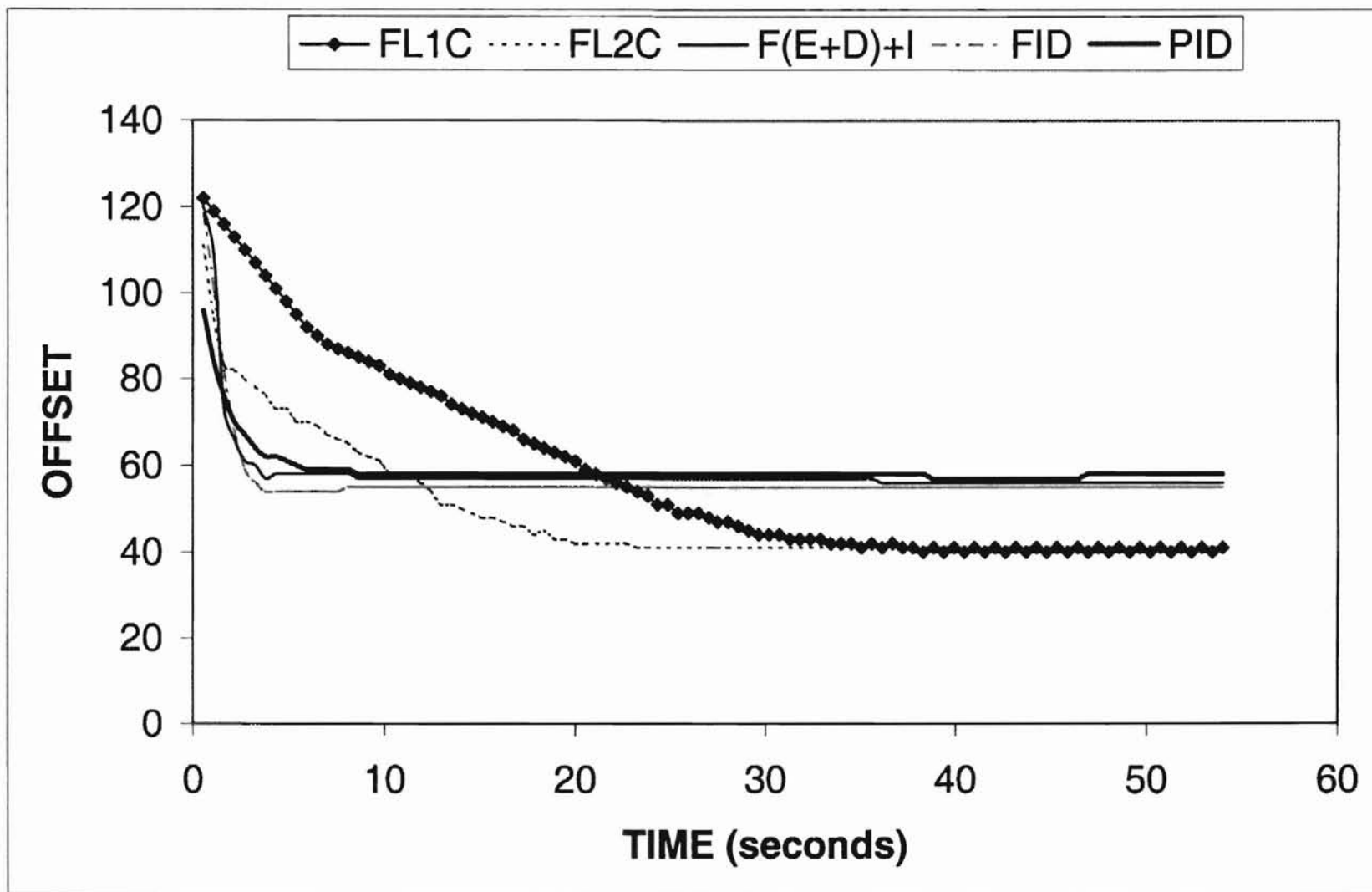


Figure A.8
Transit Response-Offset (camera 2)

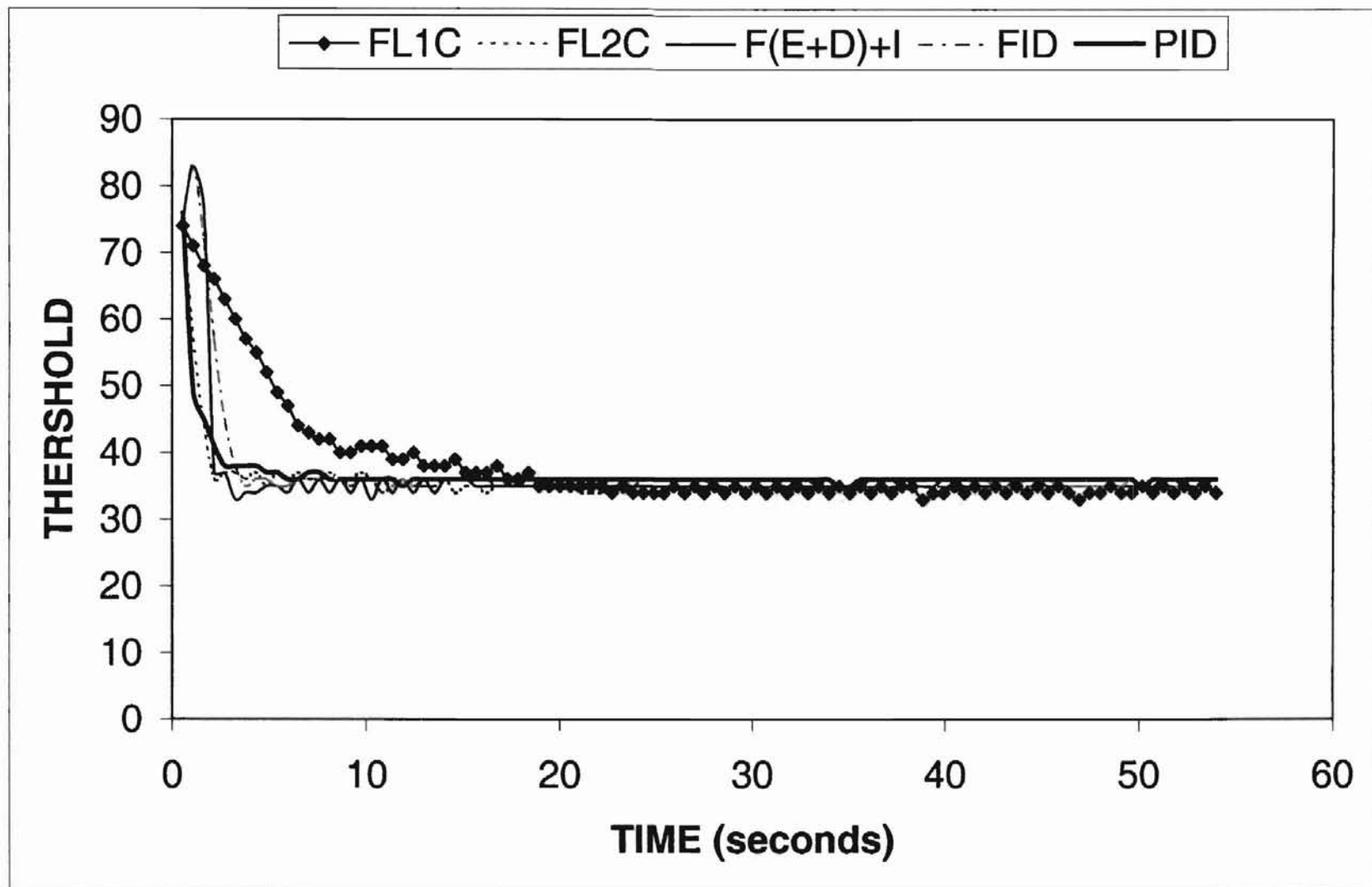


Figure A.9
Transit Response-Threshold (camera 1)

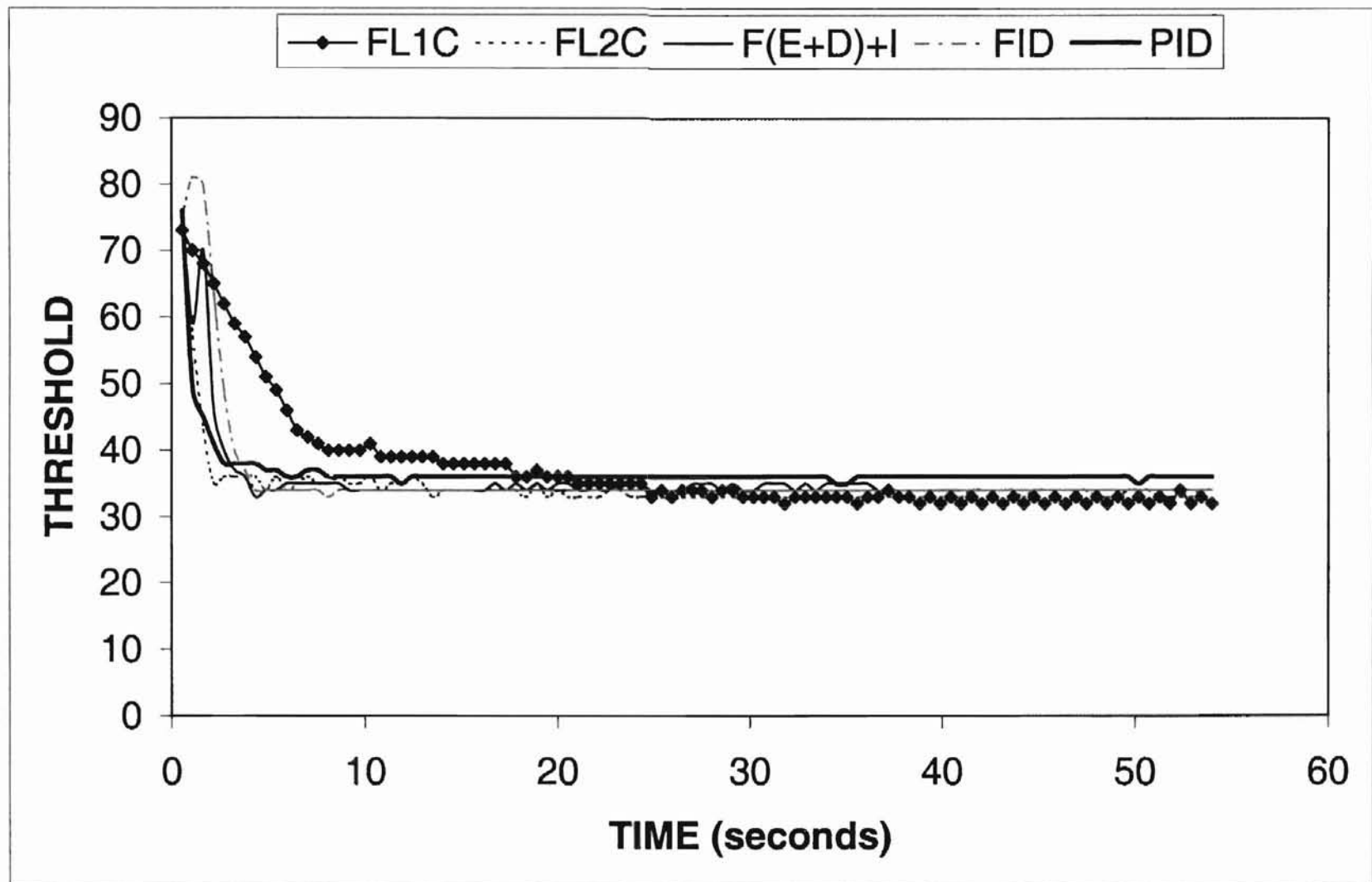


Figure A.10
Transit Response-Threshold (camera 2)

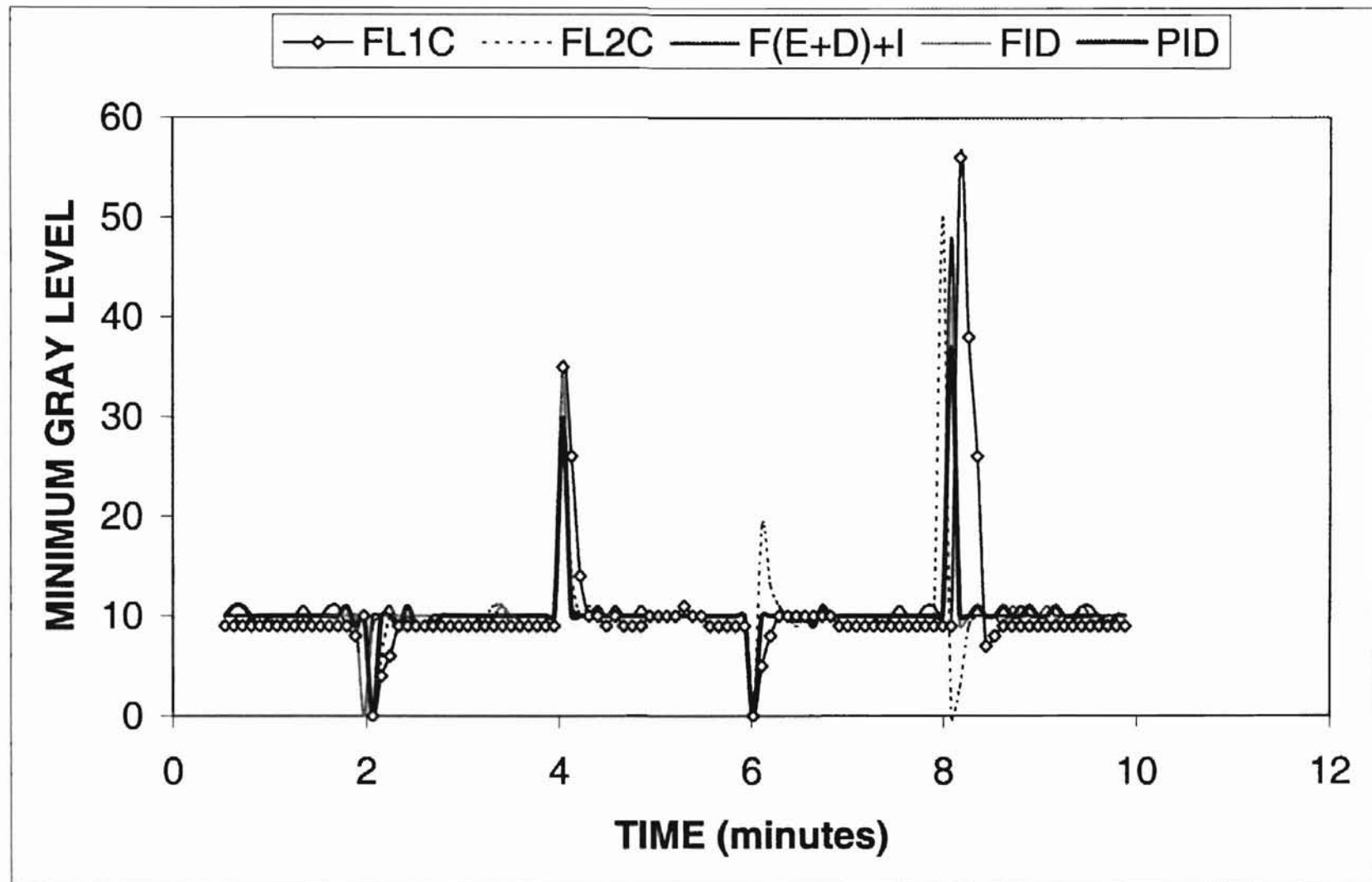


Figure A.11
Steady State Response-Minimum Gray Level (camera 1)

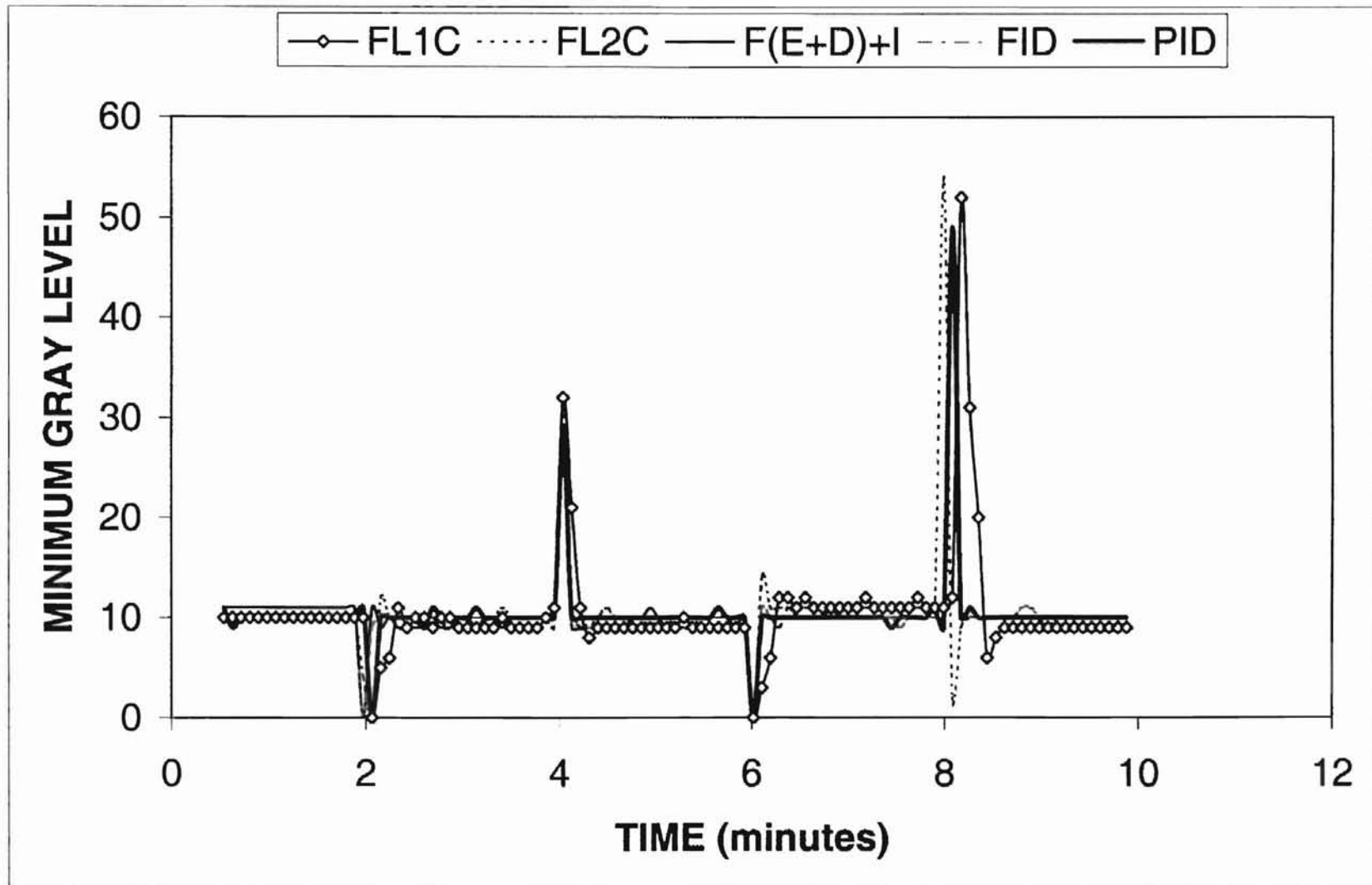


Figure A.12
Steady State Response-Minimum Gray Level (camera 2)

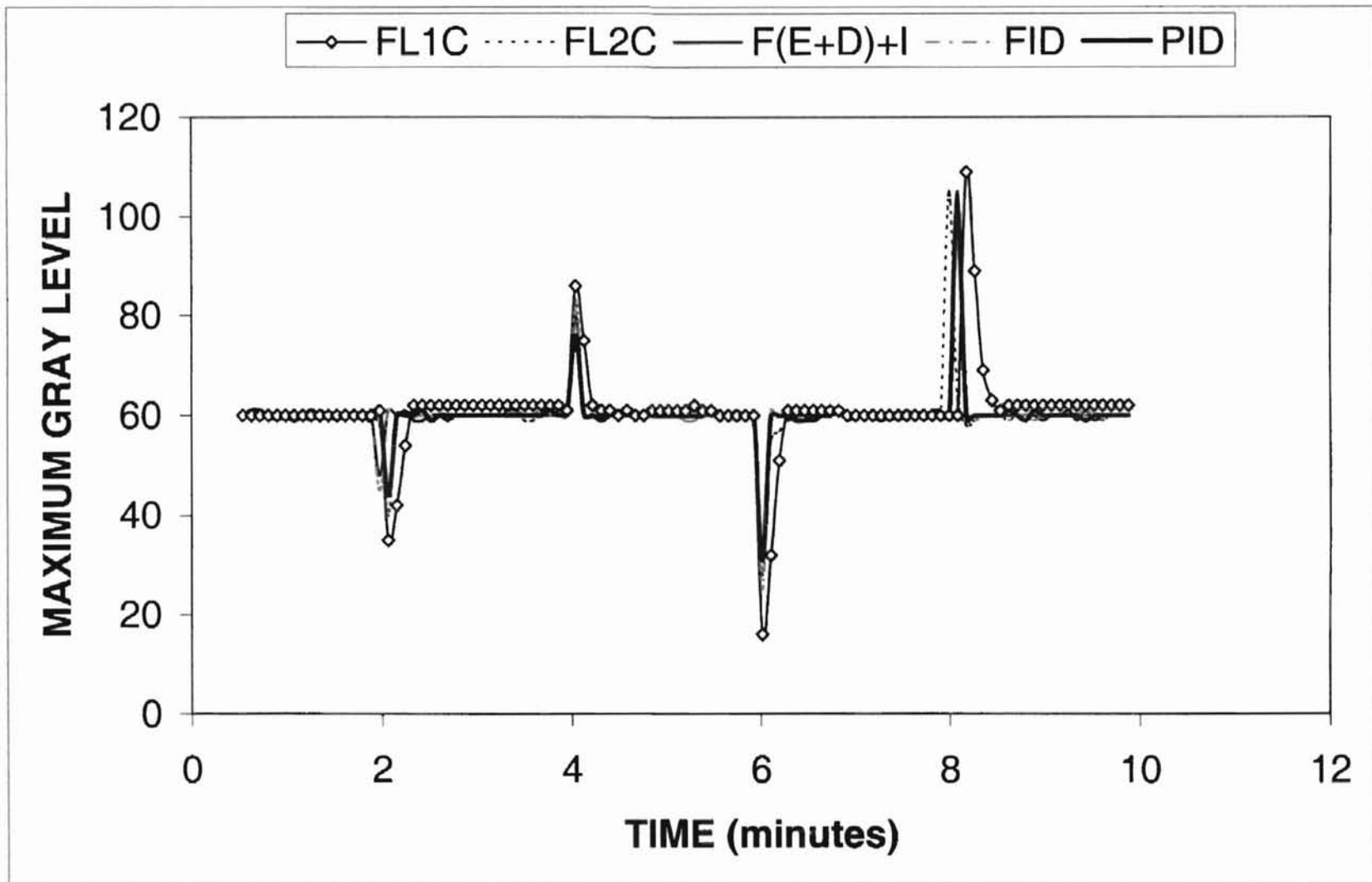


Figure A.13
Steady State Response-Maximum Gray Level (camera 1)

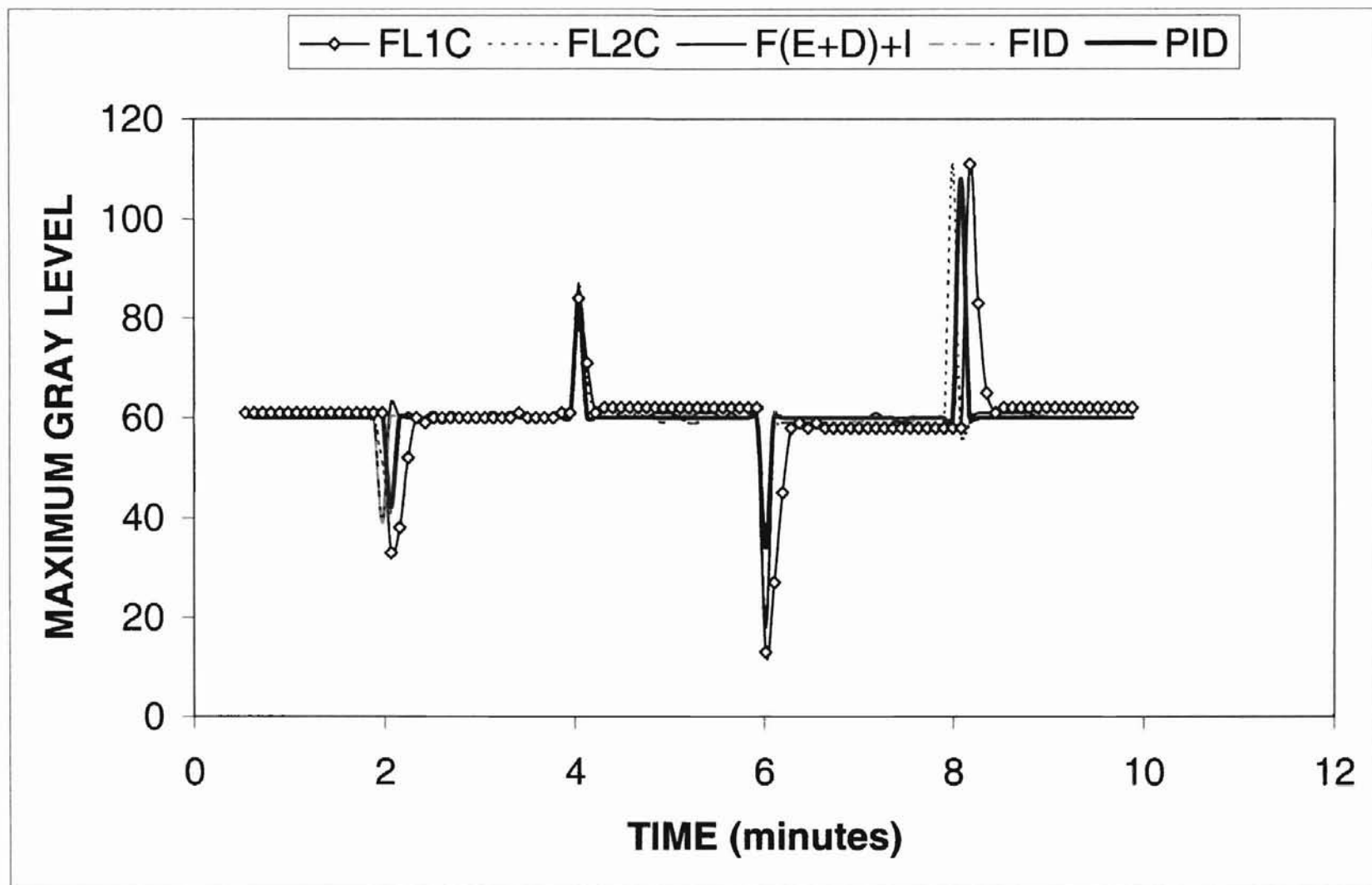


Figure A.14
Steady State Response-Maximum Gray Level (camera 2)

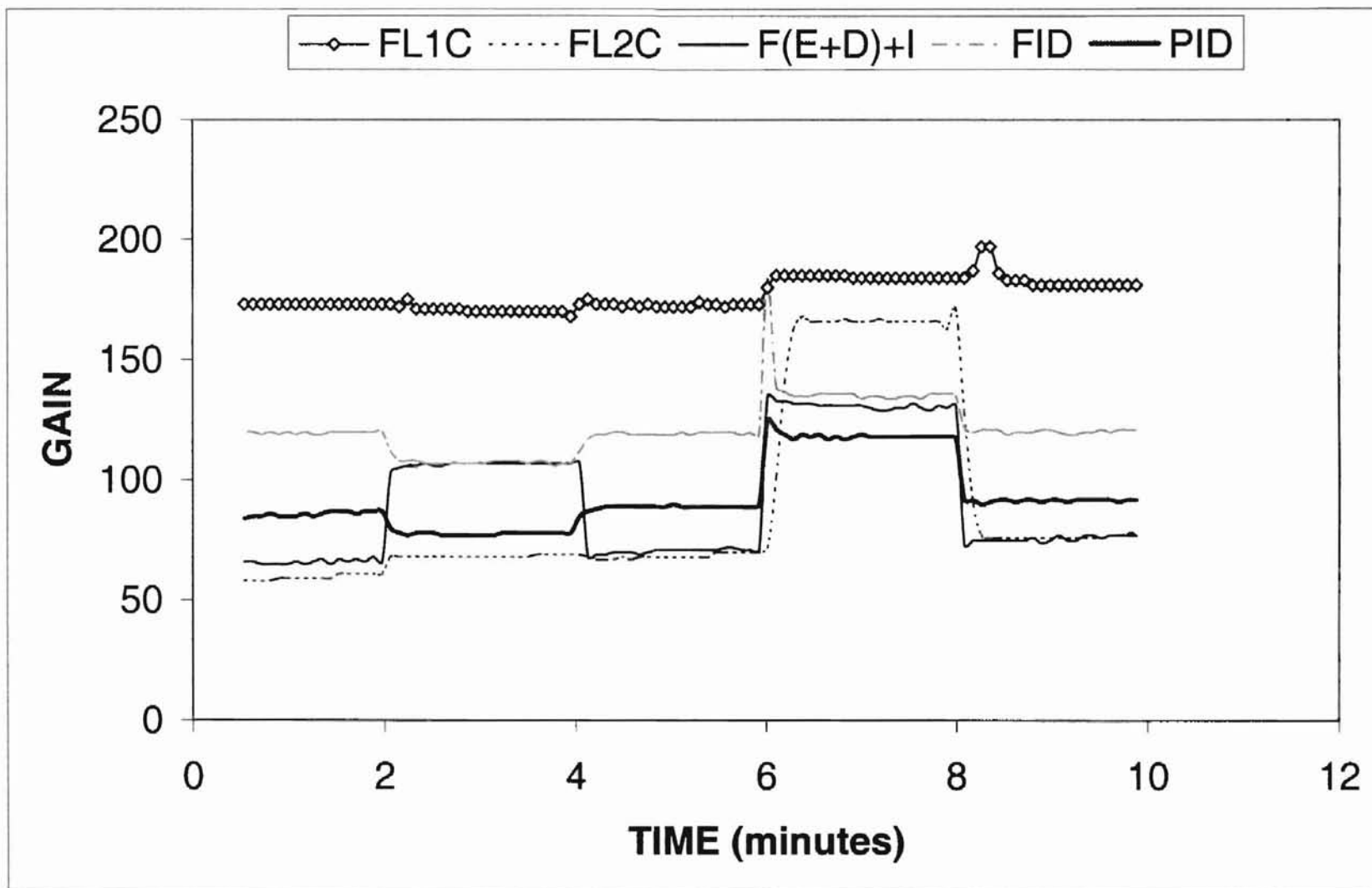


Figure A.15
Steady State Response-Gain (camera 1)

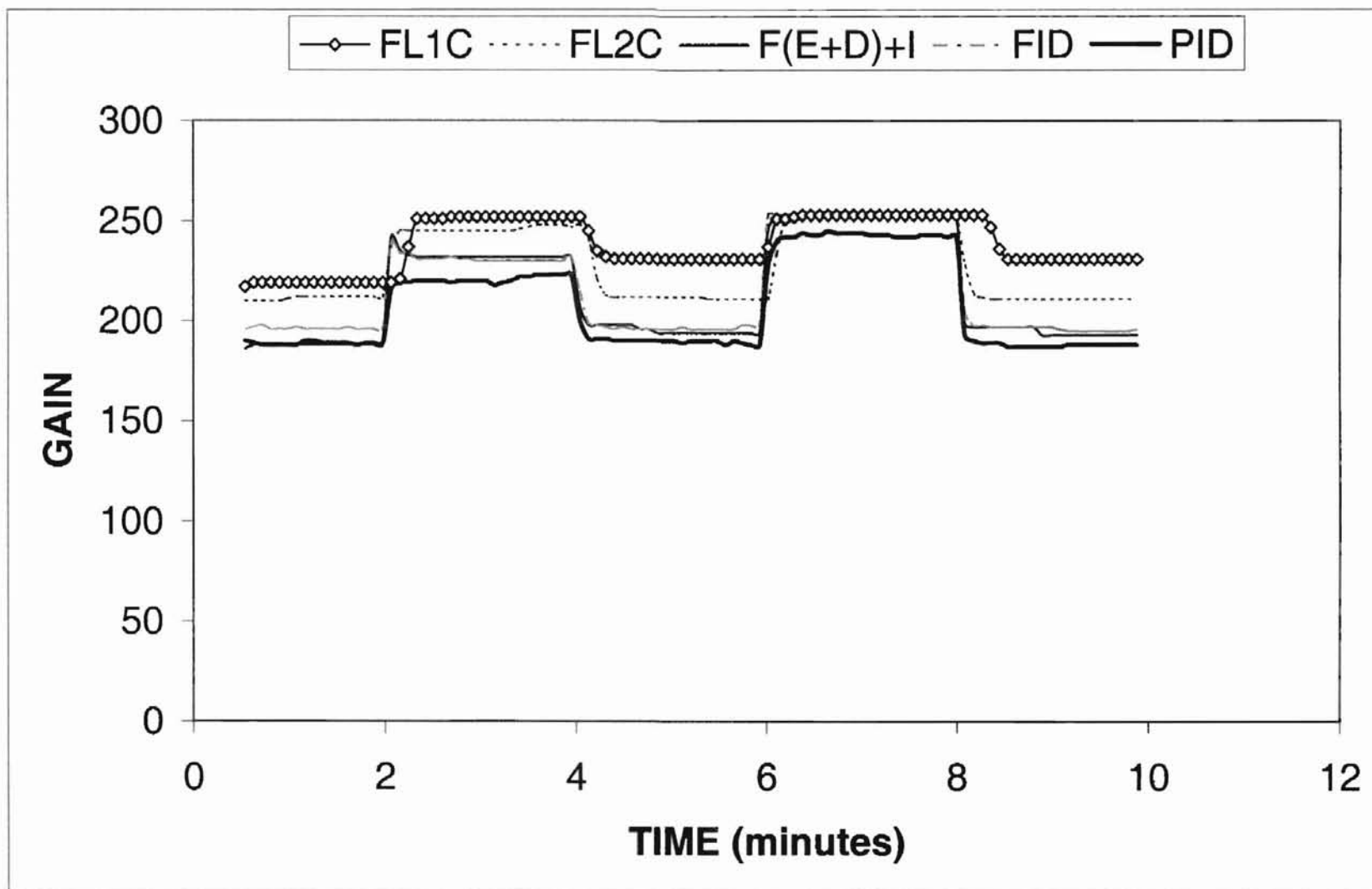


Figure A.16
Steady State Response-Gain (camera 2)

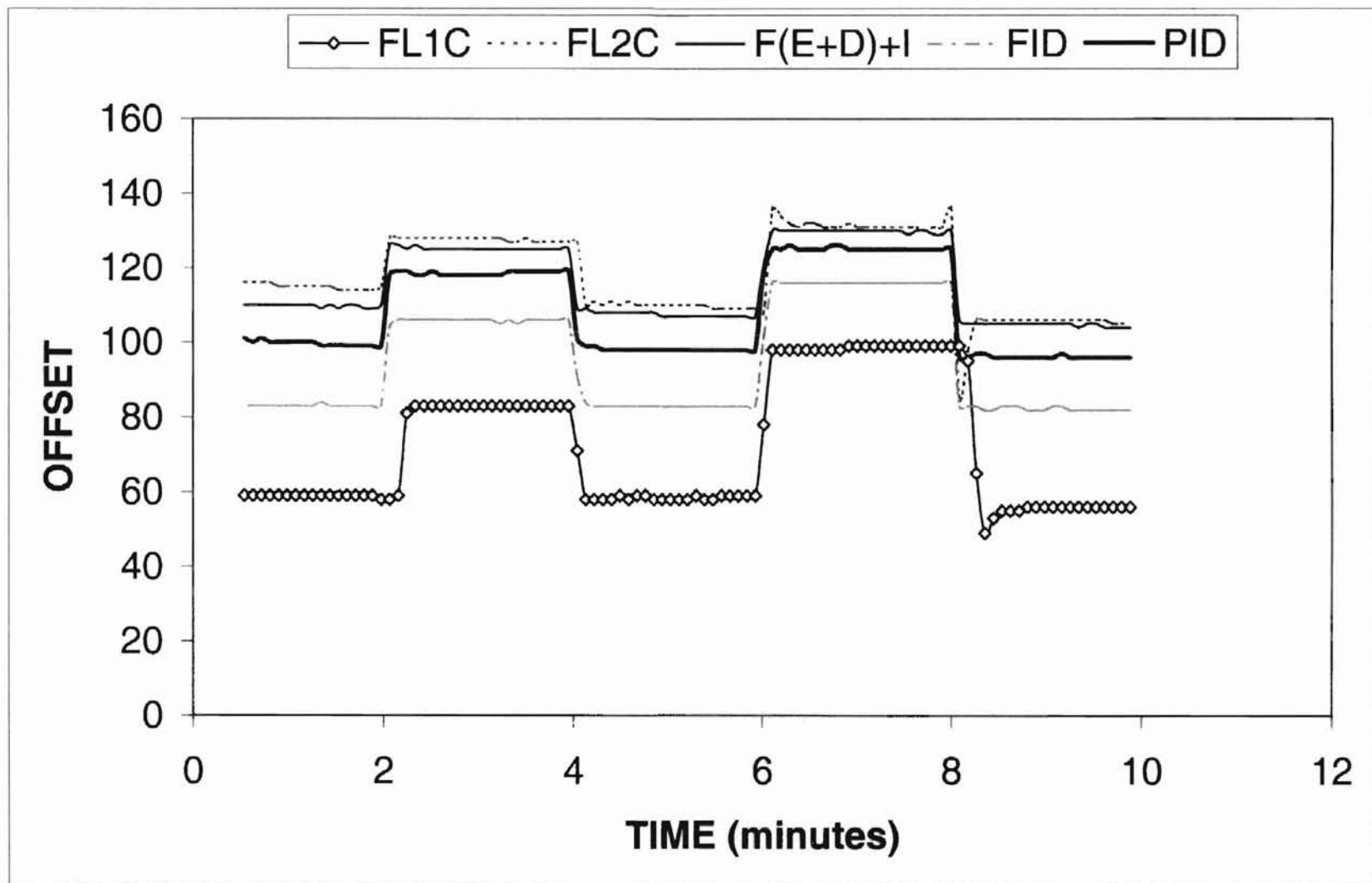


Figure A.17
Steady State Response-Offset (camera 1)

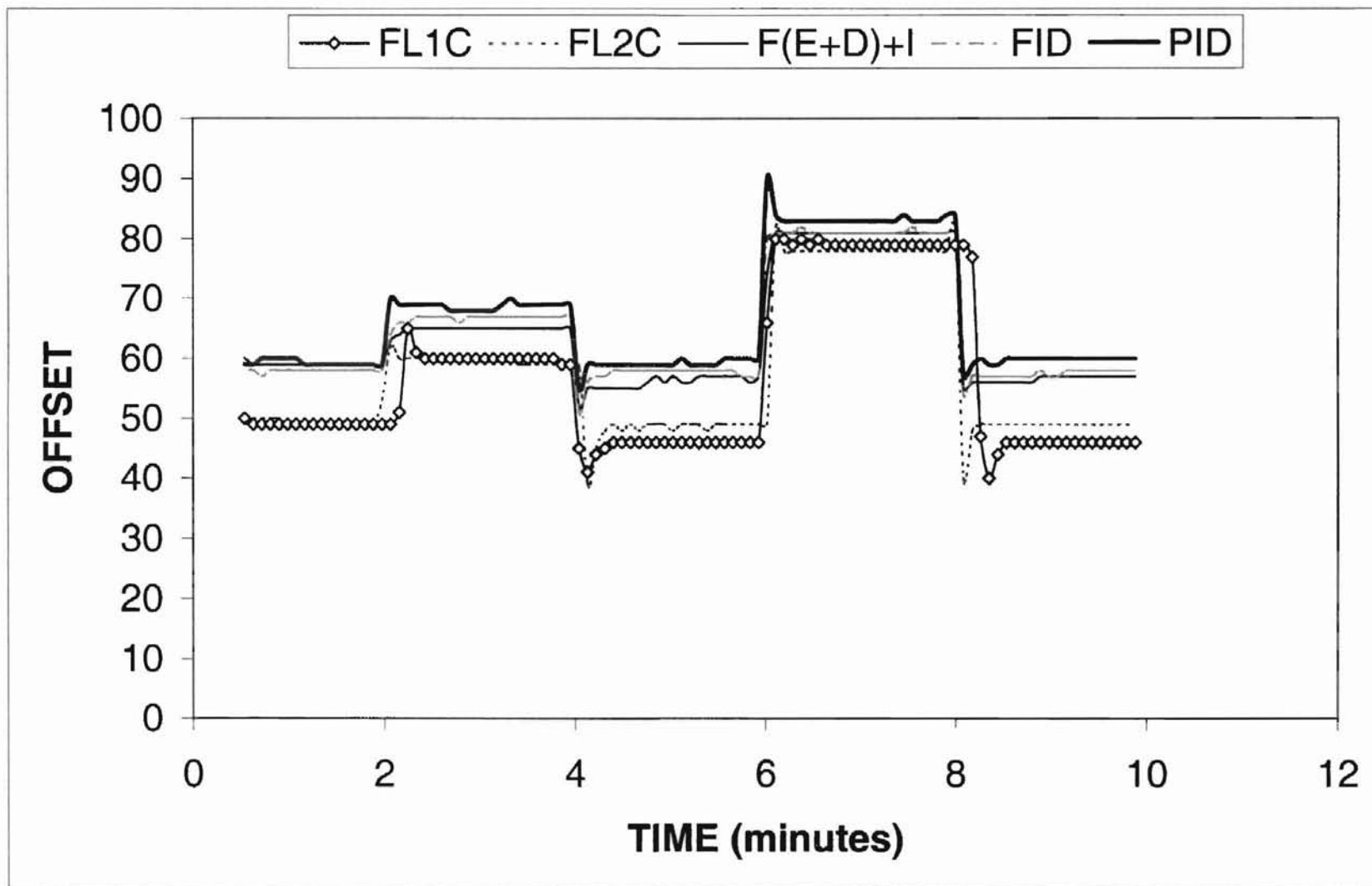


Figure A.18
Steady State Response-Offset (camera 2)

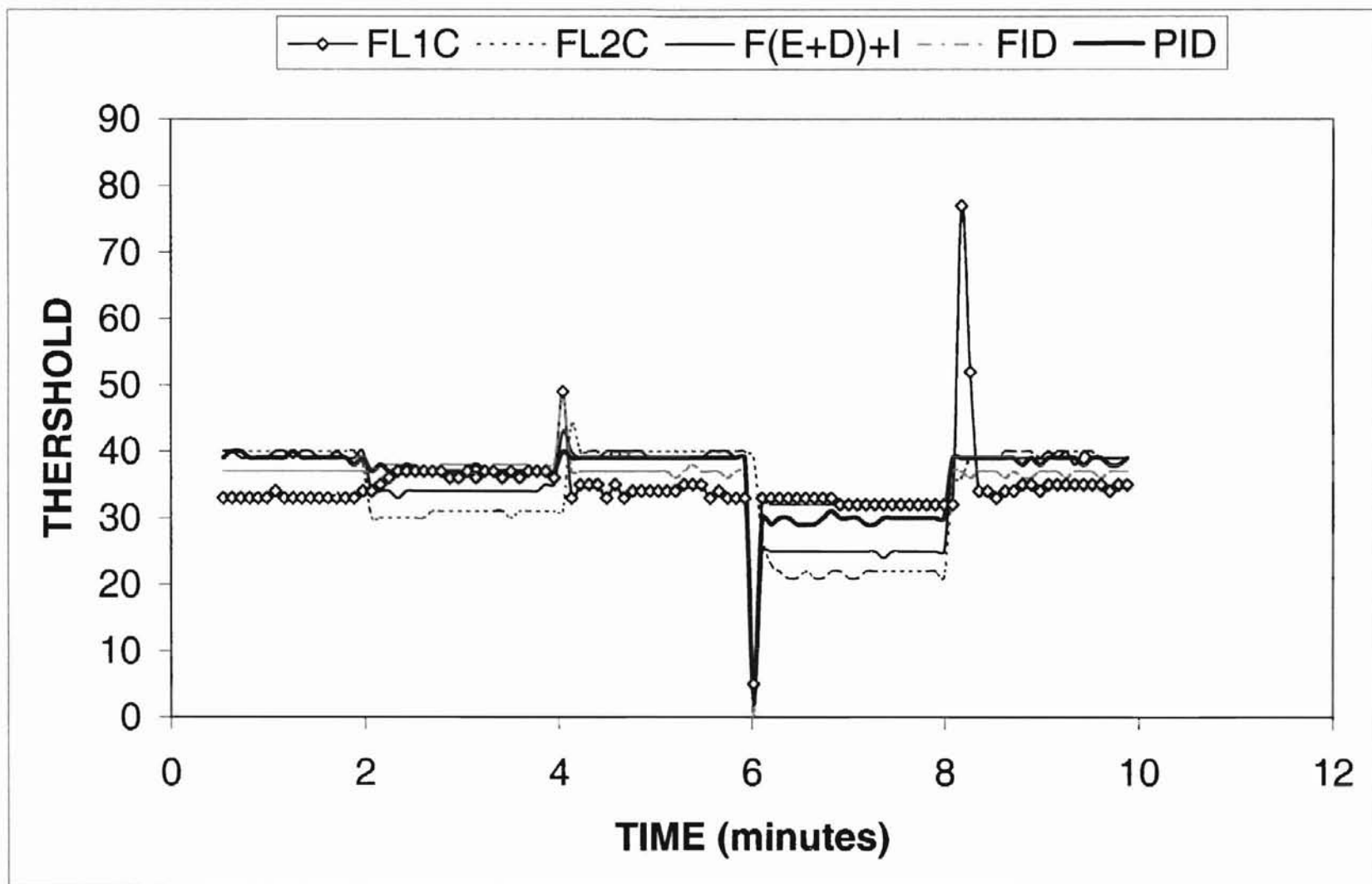


Figure A.19
Steady State Response-Threshold (camera 1)

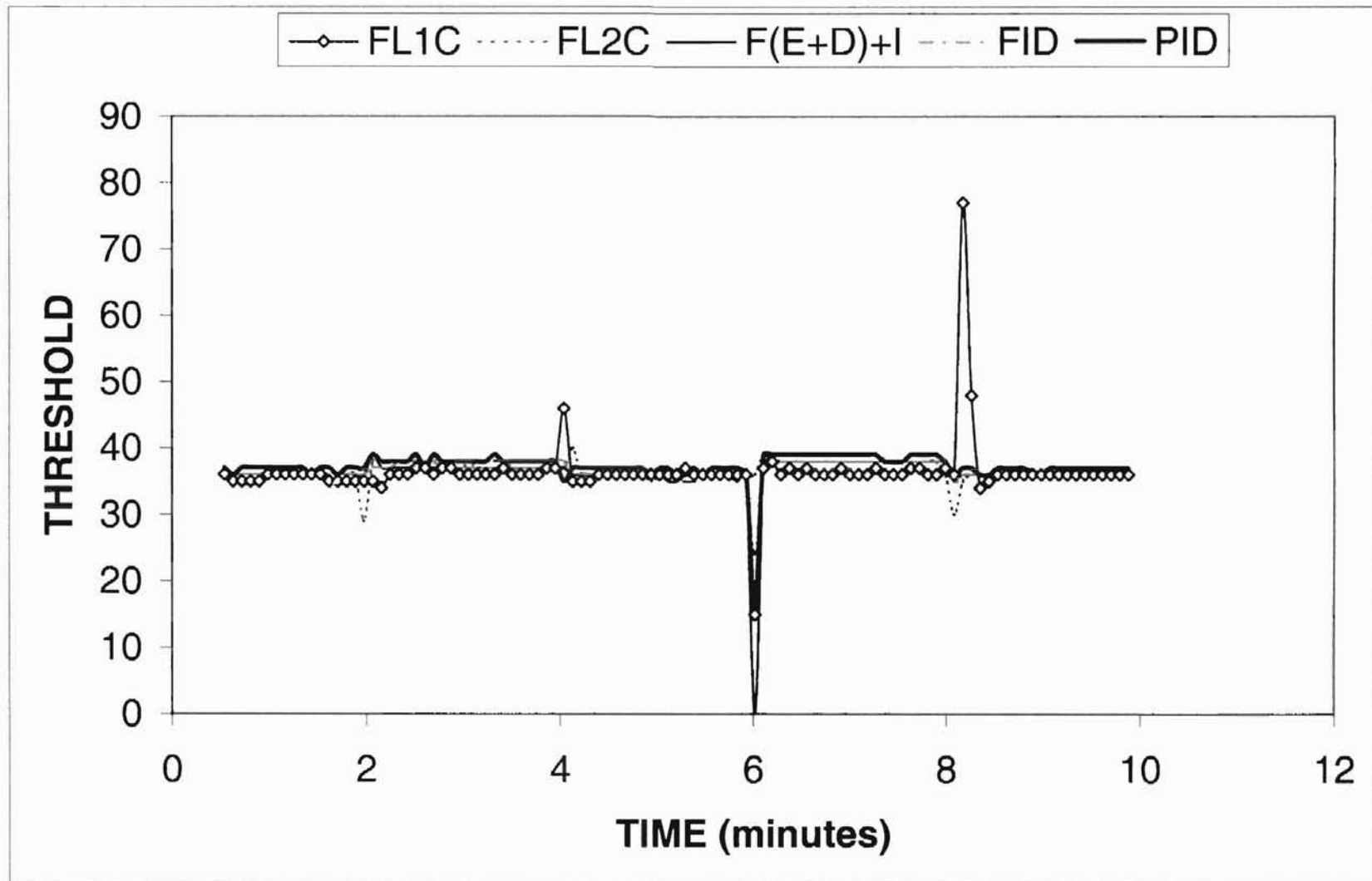


Figure A.20
Steady State Response-Threshold (camera 2)

APPENDIX B
SOURCE CODE IN V/V+

AGS VISION CODE IN V/V+
ADEPT ONE LANGUAGE FOR
FUZZY LOGIC 1
CONTROLLER

```

/*****wb_FL1C.200*****/
.PROGRAM fuzzy.set.gain()
;
; ABSTRACT:set gain and offset for vision system by a fuzzy controller
; INPUT PAR:   none
; OUTPUT PAR:  vision parameters(gain,offset,threshold)
    LOCAL clock[], frame1, frame2
    LOCAL cam, i, camera
    CALL vi.init.cam()
    yscale = 1/0.5397496
    xscale = 1/0.542155
    IF NOT DEFINED(frame1) THEN
        frame1 = 1001
        frame2 = 1002
    END
    FOR camera = 1 TO 3
        PARAMETER V.GAIN[camera] = 1
        PARAMETER V.OFFSET[camera] = 128
    END
    first.time = 0
    IF first.time == 0 THEN
        first.time = 1
        area.ref = PARAMETER(V.LAST.COL[camera]) -
            PARAMETER(V.FIRST.COL[camera]) + 1
        area.ref = area.ref * (PARAMETER(V.LAST.LINE[camera]) -
            PARAMETER(V.FIRST.LINE[camera]) + 1)
        area.ref = area.ref / 1000
        min.gray[1] = 60 ;decided
        max.gray[1] = 90 ;decided
        min.gray[2] = 60 ;decided
        max.gray[2] = 90 ;decided
        min.gray[3] = 30 ;decided
        max.gray[3] = 80 ;decided
    END
    PARAMETER V.THRESHOLD[1] = 65
    PARAMETER V.THRESHOLD[2] = 65
    PARAMETER V.THRESHOLD[3] = 65
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_a_sum[i,0] = 0
        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
        i_s_sum[i,0] = 0
    END
    PARAMETER V.FIRST.LINE[1] = v.first.line1
    PARAMETER V.LAST.LINE[1] = v.last.line1
    PARAMETER V.FIRST.COL[1] = v.first.col1
    PARAMETER V.LAST.COL[1] = v.last.col1
    PARAMETER V.FIRST.LINE[2] = v.first.line2
    PARAMETER V.LAST.LINE[2] = v.last.line2
    PARAMETER V.FIRST.COL[2] = v.first.col2
    PARAMETER V.LAST.COL[2] = v.last.col2

```

```

PARAMETER V.FIRST.LINE[3] = v.first.line3
PARAMETER V.LAST.LINE[3] = v.last.line3
PARAMETER V.FIRST.COL[3] = v.first.col3
PARAMETER V.LAST.COL[3] = v.last.col3
DISABLE V.BOUNDARIES
DISABLE V.CENTROID
count.fuz.end = 200
TIMER (4) = 0
TIMER (5) = 0
FOR count.fuz = 1 TO count.fuz.end
  IF count.fuz == 101 THEN
    TIMER (5) = 0
    min.gray[1] = 10 ;decided
    max.gray[1] = 60 ;decided
    min.gray[2] = 10 ;decided
    max.gray[2] = 60 ;decided
    min.gray[3] = 10 ;decided
    max.gray[3] = 60 ;decided
    FOR i = 1 TO 3
      i_a_min[i,100] = 0
      i_a_min_ave[i,100] = 0
      i_a_max[i,100] = 0
      i_a_max_ave[i,100] = 0
      i_a_sum[i,100] = 0
      i_s_min[i,100] = 0
      i_s_min_ave[i,100] = 0
      i_s_max[i,100] = 0
      i_s_max_ave[i,100] = 0
      i_s_sum[i,100] = 0
    END
    END
    VPICTURE (1, TRUE) 2, 0
    CALL fuzzy.ctrl(1, frame1)
    VPICTURE (2, TRUE) 2, 0
    CALL fuzzy.ctrl(2, frame1)
;
;   VPICTURE (3, TRUE) 2, 0
;   CALL fuzzy.ctrl(3, frame1)
    tran_time[count.fuz] = TIMER(4)
    TYPE count.fuz, " ", tran_time[count.fuz], " ", TIMER(5)
    TYPE
  END
RETURN
.END

```

```

.PROGRAM fuzzy.ctrl(camera, frame)
;ABSTRACT:Automatically sets the system paramters V.GAIN and V.OFFSET
;for the specified camera in an AdeptVision AGS system.Fuzzy controller
;is used to make the target have stable minimun and maximun gray level
;as well as a suitable threshold.
;INPUT PARM:camera and frame,The virtual camera and frame number to use
; OUTPUT PARM: None
    LOCAL gain, offset
    LOCAL var, x0, xn
    LOCAL area.x0, area.xn, pixel[]
    dmode = 0
    VWAIT
    VHISTOGRAM (dmode) hst[] = frame
; set threshold for the cameras
    thresh = 0
    pixel[camera] = hst[0]
    WHILE (pixel[camera] < set.point[camera]) DO
        thresh = thresh+1
        pixel[camera] = pixel[camera]+hst[thresh]
    END
    x0 = 0
    var = hst[0]
    WHILE (var <= area.ref) AND (x0 < 128) DO
        x0 = x0+1
        var = var+hst[x0]
    END
    xn = 127
    var = hst[127]
    WHILE (var <= area.ref) AND (x0 < xn) DO
        xn = xn-1
        var = var+hst[xn]
    END
    xn = xn+1
    area.x0.e = x0-min.gray[camera]
    area.xn.e = max.gray[camera]-xn
    k = 0
    l = 0
    CALL infer(area.x0.e, area.xn.e)
    gain = PARAMETER(V.GAIN[camera])
    offset = PARAMETER(V.OFFSET[camera])
    gain = MIN(256,MAX(1,INT(gain+gain.fuzzy+1)))
    offset = MIN(256,MAX(1,INT(offset-offset.fuzzy+0.5)))
    PARAMETER V.GAIN[camera] = gain
    PARAMETER V.OFFSET[camera] = offset
    PARAMETER V.THRESHOLD[camera] = thresh

;*****KEEP ALL THE PARAMETER FOR ANALYSIS*****
;*****warm up period (transit respond)*****
    count.gen = 1
    x0_warm[camera,count.gen,count.fuz] = x0
    xn_warm[camera,count.gen,count.fuz] = xn
    i_aa0 = ABS(x0-min.gray[camera])
    i_aan = ABS(xn-max.gray[camera])
    i_ss0 = i_aa0*i_aa0
    i_ssn = i_aan*i_aan
    i_a_min[camera,count.fuz] = i_a_min[camera,count.fuz-1]+i_aa0

```

```

        i_a_min_ave[camera,count.fuz] =
i_a_min[camera,count.fuz]/TIMER(5)*60
        i_a_max[camera,count.fuz] = i_a_max[camera,count.fuz-1]+i_aan
        i_a_max_ave[camera,count.fuz] =
i_a_max[camera,count.fuz]/TIMER(5)*60
        i_a_sum[camera,count.fuz] =
            i_a_min_ave[camera,count.fuz]+i_a_max_ave[camera,count.fuz]
        i_s_min[camera,count.fuz] = i_s_min[camera,count.fuz-
1]+i_ss0
        i_s_min_ave[camera,count.fuz] =
i_s_min[camera,count.fuz]/TIMER(5)*60
        i_s_max[camera,count.fuz] = i_s_max[camera,count.fuz-1]+i_ssn
        i_s_max_ave[camera,count.fuz] =
i_s_max[camera,count.fuz]/TIMER(5)*60
        i_s_sum[camera,count.fuz] =
            i_s_min_ave[camera,count.fuz]+i_s_max_ave[camera,count.fuz]
        thresh_warm[camera,count.gen,count.fuz] = thresh
        gain_warm[camera,count.gen,count.fuz] = gain
        offset_warm[camera,count.gen,count.fuz] =
TYPE /S, camera, " ", x0, " ", xn
TYPE /S, " ga= ", gain, " of= ", offset, " th= ", thresh
TYPE /S, " ", i_s_min_ave[camera,count.fuz], " ",
        i_s_max_ave[camera,count.fuz]
TYPE " ", i_s_sum[camera,count.fuz]
RETURN
.END

```

```

.PROGRAM infer(e, ie)
; initialize outputs and parameters
  LOCAL z, oz, sumac, osumac, sumacz, osumacz, i, j, emax, iemax
  LOCAL zmax, ozmax, enm, ens, ezo, eps, epm, epb, banda, ienb,
ienm
  LOCAL iens, iezo, iepm, iepb, bandb, bandc, obandc
  LOCAL znb, znm, zns, zzo, zps, zpm, zpb, oznb, oznm, ozns, ozzo
  LOCAL ozps, ozpm, ozpb
  LOCAL mua[], mub[], area, oarea      ;m,n
  z = 0
  oz = 0
  sumac = 0
  osumac = 0
  sumacz = 0
  osumacz = 0
;calculate fuzzy numbers and band width for each input
  IF NOT DEFINED(enb) THEN
    emax = 3
    iemax = 3
    zmax = 3
    ozmax = 3
;
    enb = -emax
    enm = -2*emax/3
    ens = -emax/3
    ezo = 0
    eps = -ens
    epm = -enm
    epb = emax
    banda = emax/3
;
    ienb = -iemax
    ienm = -2*iemax/3
    iens = -iemax/3
    iezo = 0
    iepm = -iens
    iepb = -ienm
    iepb = iemax
    bandb = iemax/3
;
    znb = -zmax
    znm = -2*zmax/3
    zns = -zmax/3
    zzo = 0
    zps = -zns
    zpm = -znm
    zpb = zmax
    bandc = zmax/3
;
    oznb = -ozmax
    oznm = -2*ozmax/3
    ozns = -ozmax/3
    ozzo = 0
    ozps = -ozns
    ozpm = -oznm
    ozpb = ozmax
    obandc = ozmax/3

```



```

END

; calculate index and mu vector
m = INT((e+4*banda)/banda)
IF m < 0 THEN
  m = 0
  mua[m] = 1
  mua[m+1] = 0
ELSE
  IF m > 7 THEN
    m = 7
    mua[m] = 1
    mua[m+1] = 0
  ELSE
    mua[m] = (banda*(m-3)-e)/banda
    mua[m+1] = 1-mua[m]
  END
END

n = INT((ie+4*bandb)/bandb)
IF n < 0 THEN
  n = 0
  mub[n] = 1
  mub[n+1] = 0
ELSE
  IF n > 7 THEN
    n = 7
    mub[n] = 1
    mub[n+1] = 0
  ELSE
    mub[n] = (bandb*(n-3)-ie)/bandb
    mub[n+1] = 1-mub[n]
  END
END

IF ((m == 3) OR (m == 4)) AND ((n == 3) OR (n == 4)) THEN
  CALL sub.infer(e, ie)
ELSE

; calculate the output using larsens rule and COA for defuzzification
FOR i = m TO m+1
  CASE i OF
    VALUE 0, 1:
      FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
      CASE j OF
        VALUE 0, 1:
          sumacz = sumacz+area*znb
          osumacz = osumacz+oarea*ozzo
        VALUE 2, 3:
          sumacz = sumacz+area*znm
          osumacz = osumacz+oarea*ozns
        VALUE 4, 5:
          sumacz = sumacz+area*zns

```

```

        osumacz = osumacz+oarea*oznm
    ANY
        sumacz = sumacz+area*zzo
        osumacz = osumacz+oarea*oznb
    END
END

VALUE 2:
    FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
            VALUE 0, 1, 2:
                sumacz = sumacz+area*znm
                osumacz = osumacz+oarea*ozzo
            VALUE 3, 4:
                sumacz = sumacz+area*zns
                osumacz = osumacz+oarea*ozns
            ANY
                sumacz = sumacz+area*zzo
                osumacz = osumacz+oarea*oznm
        END
    END
END

VALUE 3:
    FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
            VALUE 0, 1:
                sumacz = sumacz+area*znm
                osumacz = osumacz+oarea*ozps
            VALUE 2, 3:
                sumacz = sumacz+area*zns
                osumacz = osumacz+oarea*ozzo
            VALUE 4, 5, 6:
                sumacz = sumacz+area*zzo
                osumacz = osumacz+oarea*ozns
            ANY
                sumacz = sumacz+area*zps
                osumacz = osumacz+oarea*oznm
        END
    END
END

VALUE 4:
    FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
            VALUE 0, 1, 2:
                sumacz = sumacz+area*zns

```

```

        osumacz = osumacz+oarea*ozps
    VALUE 3, 4, 5:
        sumacz = sumacz+area*zzo
        osumacz = osumacz+oarea*ozzo
    VALUE 6:
        sumacz = sumacz+area*zps
        osumacz = osumacz+oarea*ozns
    ANY
        sumacz = sumacz+area*zpm
        osumacz = osumacz+oarea*oznm
    END
END
VALUE 5:
    FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
            VALUE 0, 1:
                sumacz = sumacz+area*zns
                osumacz = osumacz+oarea*ozpm
            VALUE 2, 3, 4:
                sumacz = sumacz+area*zzo
                osumacz = osumacz+oarea*ozps
            VALUE 5, 6:
                sumacz = sumacz+area*zps
                osumacz = osumacz+oarea*ozzo
            ANY
                sumacz = sumacz+area*zpm
                osumacz = osumacz+oarea*ozns
        END
    END
END
VALUE 6:
    FOR j = n TO n+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
            VALUE 0, 1, 2, 3:
                sumacz = sumacz+area*zzo
                osumacz = osumacz+oarea*ozpm
            VALUE 4:
                sumacz = sumacz+area*zps
                osumacz = osumacz+oarea*ozps
            VALUE 5:
                sumacz = sumacz+area*zps
                osumacz = osumacz+oarea*ozzo
            ANY
                sumacz = sumacz+area*zpm
                osumacz = osumacz+oarea*ozzo
        END
    END
END

```

```

VALUE 7, 8:
  FOR j = n TO n+1
    area = bandc*MIN(mua[i],mub[j])
    oarea = obandc*MIN(mua[i],mub[j])
    sumac = sumac+area
    osumac = osumac+oarea
    CASE j OF
      VALUE 0, 1, 2:
        sumacz = sumacz+area*zzo
        osumacz = osumacz+oarea*ozpb
      VALUE 3, 4:
        sumacz = sumacz+area*zps
        osumacz = osumacz+oarea*ozpm
      VALUE 5, 6:
        sumacz = sumacz+area*zpm
        osumacz = osumacz+oarea*ozps
      ANY
        sumacz = sumacz+area*zpb
        osumacz = osumacz+oarea*ozzo
    END
  END
END
END
IF sumac == 0 THEN
  sumac = 1E-06
END
IF osumac == 0 THEN
  osumac = 1E-06
END
gain.fuzzy = sumacz/sumac
offset.fuzzy = osumacz/osumac
END
RETURN

```

.END

```

.PROGRAM sub.infer(e, ie)
; initialize outputs and parameters
  LOCAL z, oz, sumac, osumac, sumacz, osumacz, i, j, emax, iemax
  LOCAL zmax, ozmax, ez, ep, banda, ien
  LOCAL iez, iep, bandb, bandc, obandc
  LOCAL zn, zz, zp, ozn, ozz, ozp
  LOCAL mua[], mub[], area, oarea
  z = 0
  oz = 0
  sumac = 0
  osumac = 0
  sumacz = 0
  osumacz = 0
; calculate fuzzy numbers and band width for each input
  IF NOT DEFINED(en) THEN
    emax = 3.5
    iemax = 3.5
    zmax = 1.5
    ozmax = 1.5
;
    en = -emax
    ez = 0
    ep = emax
    banda = emax/2
;
    ien = -iemax
    iez = 0
    iep = iemax
    bandb = iemax/2
;
    zn = -zmax
    zz = 0
    zp = zmax
    bandc = zmax/2
;
    ozn = -ozmax
    ozz = 0
    ozp = ozmax
    obandc = ozmax/2
  END
; calculate index and mu vector
  k = MIN(INT((e+3*banda)/banda), 4)
  IF k == 1 THEN
    mua[k] = 1-(banda*(k-2)-e)/banda
    mua[k+1] = 0
  ELSE
    IF k == 4 THEN
      mua[k] = (banda*(k-2)-e)/banda
      mua[k+1] = 0
    ELSE
      mua[k] = (banda*(k-2)-e)/banda
      mua[k+1] = 1-mua[k]
    END
  END
END

```

```

l = MIN(INT((ie+3*bandb)/bandb), 4)
IF l == 1 THEN
  mub[l] = 1-(bandb*(l-2)-ie)/bandb
  mub[l+1] = 0
ELSE
  IF l == 4 THEN
    mub[l] = (bandb*(l-2)-ie)/bandb
    mub[l+1] = 0
  ELSE
    mub[l] = (bandb*(l-2)-ie)/bandb
    mub[l+1] = 1-mub[l]
  END
END
END

```

; calculate the output using larsens rule and COA for defuzzification

```

FOR i = k TO k+1
  CASE i OF
    VALUE 1, 2:
      FOR j = 1 TO l+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
          VALUE 1, 2:
            sumacz = sumacz+area*zn
            osumacz = osumacz+oarea*ozz
          VALUE 3:
            sumacz = sumacz+area*zn
            osumacz = osumacz+oarea*ozn
          ANY
            sumacz = sumacz+area*zz
            osumacz = osumacz+oarea*ozn
        END
      END
    VALUE 3:
      FOR j = 1 TO l+1
        area = bandc*MIN(mua[i],mub[j])
        oarea = obandc*MIN(mua[i],mub[j])
        sumac = sumac+area
        osumac = osumac+oarea
        CASE j OF
          VALUE 1, 2:
            sumacz = sumacz+area*zn
            osumacz = osumacz+oarea*ozp
          VALUE 3:
            sumacz = sumacz+area*zz
            osumacz = osumacz+oarea*ozz
          ANY
            sumacz = sumacz+area*zp
            osumacz = osumacz+oarea*ozn
        END
      END
    ANY
  END
  FOR j = 1 TO l+1

```

```

area = bandc*MIN(mua[i],mub[j])
oarea = obandc*MIN(mua[i],mub[j])
sumac = sumac+area
osumac = osumac+oarea
CASE j OF
  VALUE 1, 2:
    sumacz = sumacz+area*zz
    osumacz = osumacz+oarea*ozp
  VALUE 3:
    sumacz = sumacz+area*zp
    osumacz = osumacz+oarea*ozp
  ANY
    sumacz = sumacz+area*zp
    osumacz = osumacz+oarea*ozz
END
  END
END
END
IF sumac == 0 THEN
  sumac = 1E-06
END
IF osumac == 0 THEN
  osumac = 1E-06
END

gain.fuzzy = sumacz/sumac
offset.fuzzy = osumacz/osumac
RETURN

.END

```

```

.PROGRAM vi.init.cam()
; Virtual camera setup and initialization for 3 colinear units
  LOCAL cam
    frame1 = 1001      ;global value for frame store one
    frame2 = 1002      ;global value for frame store two
; Set Basic Camera switches:
  ENABLE V.BINARY
  DISABLE V.BACKLIGHT
  ENABLE V.BOUNDARIES
  DISABLE V.FIT.ARCS
  DISABLE V.DISJOINT
  DISABLE V.RECOGNITION
  DISABLE V.SUBTRACT.HOLE
  DISABLE V.PERIMETER
  DISABLE V.STROBE
  DISABLE V.2ND.MOMENTS
  ENABLE V.HOLES
  FOR i = 1 TO 3
    ENABLE V.CENTROID[i]
    DISABLE V.2ND.MOMENTS[i]
    DISABLE V.HOLES[i] ;hole count unnessessary for sorting
  END

; Disable display switches:
  DISABLE V.SHOW.EDGES
  DISABLE V.SHOW.BOUNDS
  DISABLE V.SHOW.GRIP
  DISABLE V.SHOW.RECOG
; Set Parameters:

  PARAMETER V.MAX.AREA = 260000
  PARAMETER V.MIN.AREA = 35000
  PARAMETER V.MIN.HOLE.AREA = 50
  PARAMETER D.SCALE.MODE = 3

  PARAMETER V.FIRST.LINE = 1      ;bottom line      (1)
  PARAMETER V.LAST.LINE = 484     ;top line       (484)
  PARAMETER V.FIRST.COL = 1       ;first column   (1)
  PARAMETER V.LAST.COL = 512      ;last column    (512)
  PARAMETER V.FIRST.LINE = 50     ;bottom line    (1)
  PARAMETER V.EDGE.STRENGTH[1] = 7 ;used for food detection
  PARAMETER V.EDGE.STRENGTH[2] = 9 ;used for food detection
  PARAMETER V.EDGE.STRENGTH[3] = 10 ;used for food detection

  set_point = FALSE
  IF set_point == TRUE THEN

    FOR cam = 1 TO 3
      VPICTURE (cam) 2
      t = PARAMETER(V.THRESHOLD[cam])
      tmin = MAX(0,t-70)
      tmax = MIN(127,t+70)
      VWAIT
      VAUTOTHR (0, tmin, tmax) thrs[]
    ; IF thrs[0] > 1 THEN
      PARAMETER V.THRESHOLD[cam] = thrs[1]
    ; END

```



```

END
PARAMETER V.FIRST.LINE[1] = 3
PARAMETER V.LAST.LINE[1] = 33
PARAMETER V.FIRST.COL[1] = 100
PARAMETER V.LAST.COL[1] = 200
PARAMETER V.FIRST.LINE[2] = 3
PARAMETER V.LAST.LINE[2] = 33
PARAMETER V.FIRST.COL[2] = 92
PARAMETER V.LAST.COL[2] = 192
PARAMETER V.FIRST.LINE[3] = 3
PARAMETER V.LAST.LINE[3] = 33
PARAMETER V.FIRST.COL[3] = 102
PARAMETER V.LAST.COL[3] = 202

FOR cam = 1 TO 3
  VPICTURE (cam) 2
  dmode = 0
  VWAIT
  VHISTOGRAM (dmode) hst[]
  t = PARAMETER(V.THRESHOLD[cam])
  thresh = 0
  pixel[cam] = hst[0]
  WHILE (thresh < t) DO
    thresh = thresh+1
    pixel[cam] = pixel[cam]+hst[thresh]
  END
  set.point[cam] = pixel[cam]
  TYPE t, set.point[cam]

  END
ELSE
  set.point[1] = 3900 ;2:10pm
  set.point[2] = 3900 ; 2100
  set.point[3] = 3900
END

PARAMETER V.FIRST.LINE = 1 ;bottom line (1)
PARAMETER V.LAST.LINE = 484 ;top line (484)
PARAMETER V.FIRST.COL = 1 ;first column (1)
PARAMETER V.LAST.COL = 512 ;last column (512)
PARAMETER V.FIRST.LINE = 42 ;bottom line (1)

;set reference value for some subroutines
; 1.) vi.find.spot
  l.length = 110
  l.width = 180
  s.length = 120
  s.width = 90
  angle = 90
  large = 1

;camera II
  thre.pixa[5] = 76
  thre.pixb[5] = 99 ;clean: min=77, max=99
  thre.pixa[8] = 73
  thre.pixb[8] = 95 ;clean: min=72, max=94 for small spacer

```

```

    thre.pixa[11] = 60
    thre.pixb[11] = 77 ;clean: min=72, max=94 for large spacer

;camera I
    thre.pixa[4] = 56
    thre.pixb[4] = 77 ;clean: min=56, max=77 for small spacer
    thre.pixa[7] = 53
    thre.pixb[7] = 75 ;clean: min=51, max=73 for small spacer
    thre.pixa[10] = 42
    thre.pixb[10] = 70 ;clean: min=42, max=70 for large spacer

;camera III
    thre.pixa[6] = 89
    thre.pixb[6] = 123 ;clean: min=88,max=107
    thre.pixa[9] = 81
    thre.pixb[9] = 107 ;clean: min=88,max=107 for small spacer
    thre.pixa[12] = 63
    thre.pixb[12] = 90 ;clean: min=63,max=90 for large spacer

;dirty threshold
    thre.dirty1[4] = 6
    thre.dirty1[5] = 10
    thre.dirty1[6] = 50
    thre.dirty1[7] = 5
    thre.dirty1[8] = 15
    thre.dirty1[9] = 50
    thre.dirty1[10] = 10
    thre.dirty1[11] = 10
    thre.dirty1[12] = 10

    thre.dirty2[4] = 90
    thre.dirty2[5] = 110
    thre.dirty2[6] = 115
    thre.dirty2[7] = 85
    thre.dirty2[8] = 98
    thre.dirty2[9] = 110
    thre.dirty2[10] = 60
    thre.dirty2[11] = 80
    thre.dirty2[12] = 85

; 2.) vi.inspect.sm
    yscale = 1/0.5397496
    xscale = 1/0.542155

    cxbig = 512/xscale/2
    widthbig = 512/xscale

    cy1 = 155
    cy2 = 140
    cy3 = 180
    height = 180

    gr.win.show = -1 ;3
    gr.mode = 5 ;2 ;5
    gr.overlay = 0 ;1; 0

; 3.)vi.sort

```

```

area1.ref[1] = 56500
area2.ref[1] = 53555
area1.ref[2] = 54800
area2.ref[2] = 52000
area1.ref[3] = 53055
area2.ref[3] = 51500
gray.ref[1] = 67
gray.ref[2] = 86
gray.ref[3] = 98
count.ref0[1] = 55
count.ref1[1] = 85
count.ref0[2] = 65
count.ref1[2] = 100
count.ref0[3] = 80 ;68
count.ref1[3] = 120 ;98

; 4.)fuzzy.set.gain and real.time.ctrl
v.first.line1 = INT(5*yscale)
v.last.line1 = INT(25*yscale)
v.first.col1 = INT(115*xscale)
v.last.col1 = INT(195*xscale)
v.first.line2 = INT(5*yscale)
v.last.line2 = INT(25*yscale)
v.first.col2 = INT(102*xscale)
v.last.col2 = INT(192*xscale)
v.first.line3 = INT(2*yscale)
v.last.line3 = INT(22*yscale)
v.first.col3 = INT(112*xscale)
v.last.col3 = INT(192*xscale)

FOR cam = 1 TO 12
    VDISPLAY (cam) 2, 1
END
RETURN
.END

```

AGS VISION CODE IN V/V+
ADEPT ONE LANGUAGE FOR
FUZZY LOGIC 2
CONTROLLER

```

/*****wb_FL2C.200*****/
.PROGRAM fuzzy.set.gain()
; ABSTRACT:set gain and offset for vision system by a fuzzy controller
; INPUT PAR:   none
; OUTPUT PAR:  vision parameters(gain,offset,threshold)
    LOCAL clock[], frame1, frame2
    LOCAL cam, i, camera
    CALL vi.init.cam()
    yscale = 1/0.5397496
    xscale = 1/0.542155
    IF NOT DEFINED(frame1) THEN
        frame1 = 1001
        frame2 = 1002
    END
    FOR camera = 1 TO 3
        PARAMETER V.GAIN[camera] = 1
        PARAMETER V.OFFSET[camera] = 128
    END
    area.ref = PARAMETER(V.LAST.COL[camera]) -
PARAMETER(V.FIRST.COL[camera])+1
    area.ref = area.ref*(PARAMETER(V.LAST.LINE[camera]) -
PARAMETER(V.FIRST.LINE[camera])+1)
    area.ref = area.ref/1000          ;Significant amount of
overlap,original 5000
    min.gray[1] = 60 ;decided
    max.gray[1] = 90 ;decided
    min.gray[2] = 60 ;decided
    max.gray[2] = 90 ;decided
    min.gray[3] = 60 ;decided
    max.gray[3] = 90 ;decided

    PARAMETER V.THRESHOLD[1] = 65
    PARAMETER V.THRESHOLD[2] = 65
    PARAMETER V.THRESHOLD[3] = 65
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_a_sum[i,0] = 0
        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
        i_s_sum[i,0] = 0
    END
    DISABLE V.BOUNDARIES
    DISABLE V.CENTROID
    count.fuz.end = 200
    TIMER (4) = 0
    TIMER (5) = 0
    FOR count.fuz = 1 TO count.fuz.end
        IF count.fuz == 101 THEN
            TIMER (5) = 0
            min.gray[1] = 10 ;decided
            max.gray[1] = 60 ;decided
            min.gray[2] = 10 ;decided

```

```

max.gray[2] = 60 ;decided
min.gray[3] = 10 ;decided
max.gray[3] = 60 ;decided
FOR i = 1 TO 3
  i_a_min[i,100] = 0
  i_a_min_ave[i,100] = 0
  i_a_max[i,100] = 0
  i_a_max_ave[i,100] = 0
  i_a_sum[i,100] = 0

  i_s_min[i,100] = 0
  i_s_min_ave[i,100] = 0
  i_s_max[i,100] = 0
  i_s_max_ave[i,100] = 0
  i_s_sum[i,100] = 0
END
END
VPICTURE (1, TRUE) 2, 0
CALL fuzzy.ctrl(1, frame1)

VPICTURE (2, TRUE) 2, 0
CALL fuzzy.ctrl(2, frame1)

tran_time[count.fuz] = TIMER(4)
TYPE count.fuz, " ", tran_time[count.fuz], " ", TIMER(5)
TYPE
END
RETURN

```

.END

```

.PROGRAM fuzzy.ctrl(camera, frame)
; ABSTRACT: Automatically sets paramters V.GAIN and V.OFFSET
; for the specified camera in an AGS system. Fuzzy controller
; is used to make the target have stable minimum and maximum gray level
; as well as a suitable threshold.
; INPUT PARM: camera and frame The camera and frame number to use
; OUTPUT PARM: None
    LOCAL gain, offset
    LOCAL var, x0, xn
    LOCAL area.x0, area.xn, pixel[]
    dmode = 0
    VWAIT
    VHISTOGRAM (dmode) hst[] = frame
; set threshold for the cameras
    thresh = 0
    pixel[camera] = hst[0]
    WHILE (pixel[camera] < set.point[camera]) DO
        thresh = thresh+1
        pixel[camera] = pixel[camera]+hst[thresh]
    END
    x0 = 0
    var = hst[0]
    WHILE (var <= area.ref) AND (x0 < 128) DO
        x0 = x0+1
        var = var+hst[x0]
    END
    xn = 127
    var = hst[127]
    WHILE (var <= area.ref) AND (x0 < xn) DO
        xn = xn-1
        var = var+hst[xn]
    END
    area.x0.e = x0-min.gray[camera]
    area.xn.e = xn-max.gray[camera]
    k = 0
    l = 0
    CALL infer(area.x0.e, area.xn.e)
    gain = PARAMETER(V.GAIN[camera])
    offset = PARAMETER(V.OFFSET[camera])
    gain = MIN(256,MAX(1,INT(gain+gain.fuzzy+0.5)))
    offset = MIN(255,MAX(1,INT(offset+offset.fuzzy+0.5)))
    PARAMETER V.GAIN[camera] = gain ;Make the changes ;L03-
    PARAMETER V.OFFSET[camera] = offset
    PARAMETER V.THRESHOLD[camera] = thresh

;*****KEEP ALL THE PARAMETER FOR ANALYSIS*****
;*****warm up period (transit respond)*****
    count.gen = 1
;    IF (count.gen <= 1) THEN
        x0_warm[camera,count.gen,count.fuz] = x0
        xn_warm[camera,count.gen,count.fuz] = xn
        i_aa0 = ABS(x0-min.gray[camera])
        i_aan = ABS(xn-max.gray[camera])
        i_ss0 = i_aa0*i_aa0
        i_ssn = i_aan*i_aan
        i_a_min[camera,count.fuz] = i_a_min[camera,count.fuz-1]+i_aa0

```

```

        i_a_min_ave[camera,count.fuz] =
i_a_min[camera,count.fuz]/TIMER(5)*60
        i_a_max[camera,count.fuz] = i_a_max[camera,count.fuz-1]+i_aan
        i_a_max_ave[camera,count.fuz] =
i_a_max[camera,count.fuz]/TIMER(5)*60
        i_a_sum[camera,count.fuz] =
i_a_min_ave[camera,count.fuz]+i_a_max_ave[camera,count.fuz]

        i_s_min[camera,count.fuz] = i_s_min[camera,count.fuz-1]+i_ss0
        i_s_min_ave[camera,count.fuz] =
i_s_min[camera,count.fuz]/TIMER(5)*60
        i_s_max[camera,count.fuz] = i_s_max[camera,count.fuz-1]+i_ssn
        i_s_max_ave[camera,count.fuz] =
i_s_max[camera,count.fuz]/TIMER(5)*60
        i_s_sum[camera,count.fuz] =
i_s_min_ave[camera,count.fuz]+i_s_max_ave[camera,count.fuz]

        thresh_warm[camera,count.gen,count.fuz] = thresh
        gain_warm[camera,count.gen,count.fuz] = gain
        offset_warm[camera,count.gen,count.fuz] = offset
        TYPE /S, camera, " ", x0, " ", xn
        TYPE /S, " ga=", gain, " of=", offset, " th=", thresh
        TYPE /S, " ", i_s_min[camera,count.fuz], " ",
i_s_max[camera,count.fuz]
        TYPE " ", i_s_sum[camera,count.fuz]
        RETURN
.END

```



```

.PROGRAM infer(e, ie)
;***** program variable*****
    LOCAL emin, emax, gnum, gden, onum, oden, emin.member,
emax.member, layer
    LOCAL emin.spread, emax.spread, g.spread, o.spread, emin.n,
emax.n, g.n, o.n
    LOCAL emin.nl, emin.nm, emin.ns, emin.nz, emin.ze, emin.pz,
emin.ps, emin.pm, emin.pl
    LOCAL emax.nl, emax.nm, emax.ns, emax.nz, emax.ze, emax.pz,
emax.ps, emax.pm, emax.pl
    LOCAL gnl, gnm, gns, gnz, gze, gpz, gps, gpm, gpl
    LOCAL onl, onm, ons, onz, oze, opz, ops, opm, opl
    LOCAL mu.emin[], mu.emax[]
    emin = e
    emax = ie
    gden = 0
    oden = 0
    gnum = 0
    onum = 0

;*****assign initial value and calculate fuzzy numbers*****
    range1 = 4
    range2 = 12
    IF ((emin <= range1) AND (emin >= -range1)) AND ((emax <=
range1) AND (emax >= -range1)) THEN
        Layer = 1
        emin.support = range1/2
        emax.support = range1/2
        g.support = 2
        o.support = 2
    ELSE
        IF ((emin <= range2) AND (emin >= -range2)) AND ((emax <=
range2) AND (emax >= -range2)) THEN
            emin.support = range2/2
            emax.support = range2/2
            g.support = 4
            o.support = 4
        ELSE
            emin.support = range2
            emax.support = range2
            g.support = 8
            o.support = 8
        END
    END

    emin.n = 2
    emax.n = 2
    g.n = 2
    o.n = 2

    emin.spread = emin.support*emin.n
    emax.spread = emax.support*emax.n
    g.spread = g.support*g.n
    o.spread = o.support*o.n
;*****partition the universe of discourse for input fuzzy variable***
    emin.pl = emin.spread
    emin.pm = emin.spread*3/4

```

```

emin.ps = emin.support          ; (emin.spread/2)
emin.pz = emin.support/2        ; (emin.spread/4)
emin.ze = 0
emin.nz = -emin.pz
emin.ns = -emin.ps
emin.nm = -emin.pm
emin.nl = -emin.pl

emax.pl = emax.spread
emax.pm = emax.spread*3/4
emax.ps = emax.support          ; (emax.spread/2)
emax.pz = emax.support/2        ; (emax.spread/4)
emax.ze = 0
emax.nz = -emax.pz
emax.ns = -emax.ps
emax.nm = -emax.pm
emax.nl = -emax.pl
;*****partition the universe of discourse for output fuzzy variable****
gpl = g.spread
gpm = g.spread*3/4
gps = g.support
gpz = g.support/2
gze = 0
gnz = -gpz
gns = -gps
gnm = -gpm
gnl = -gpl

opl = o.spread
opm = o.spread*3/4
ops = o.support
opz = o.support/2
oze = 0
onz = -opz
ons = -ops
onm = -opm
onl = -opl
;*calculate input range and mu vector for tow consequentive partitions*
IF (emin >= 0) THEN
    emin.num = INT(emin/emin.support*2)
ELSE
    emin.num = INT(emin/emin.support*2-1)
END
IF (emin.num < -4) THEN
    emin.num = -4
ELSE
    IF (emin.num >= 4) THEN
        emin.num = 4
    END
END

IF (emax >= 0) THEN
    emax.num = INT(emax/emax.support*2)
ELSE
    emax.num = INT(emax/emax.support*2-1)
END
IF (emax.num < -4) THEN

```

```

    emax.num = -4
ELSE
    IF (emax.num >= 4) THEN
        emax.num = 4
    END
END

mu.emin[1] = ((emin.num+1)*emin.support/2-emin)/emin.support/2
mu.emin[2] = 1-mu.emin[1]
mu.emax[1] = ((emax.num+1)*emax.support/2-emax)/emax.support/2
mu.emax[2] = 1-mu.emax[1]

;*****calculate fuzzy output gain.fuzzy and offset.fuzzy using*****
;*****weighted average method for defuzzification*****
FOR i = 1 TO 2
    emin.member = emin.num+i-1
    CASE emin.member OF
        VALUE -4:
            FOR j = 1 TO 2
                emax.member = emax.num+j-1
                gden = gden+MIN(mu.emin[i],mu.emax[j])
                oden = gden
                CASE emax.member OF
                    VALUE -4:
                        gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+opl*MIN(mu.emin[i],mu.emax[j])
                    VALUE -3:
                        gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
                    VALUE -2:
                        gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
                    VALUE -1, 0, 1:
                        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
                    VALUE 2:
                        gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
                    VALUE 3:
                        gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
                    VALUE 4:
                        gnum = gnum+gnl*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
                    VALUE 5:
                        gnum = gnum+(gnl-
g.support/2)*MIN(mu.emin[i],mu.emax[j])
                        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])

                END
            END
        VALUE -3:
            FOR j = 1 TO 2
                emax.member = emax.num+j-1
                gden = gden+MIN(mu.emin[i],mu.emax[j])
                oden = gden
                CASE emax.member OF

```

```

VALUE -4:
gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
onum = onum+opl*MIN(mu.emin[i],mu.emax[j])
VALUE -3:
gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
VALUE -2:
gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
VALUE -1, 0:
gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
VALUE 1, 2:
gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
VALUE 3:
gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
VALUE 4:
gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
VALUE 5:
gnum = gnum+gnl*MIN(mu.emin[i],mu.emax[j])
onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
END
END
VALUE -2:
FOR j = 1 TO 2
emax.member = emax.num+j-1
gden = gden+MIN(mu.emin[i],mu.emax[j])
oden = gden
CASE emax.member OF
VALUE -4:
gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
VALUE -3:
gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
VALUE -2:
gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
VALUE -1:
gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
VALUE 0, 1:
gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
VALUE 2:
gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
VALUE 3:
gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
VALUE 4:
gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
VALUE 5:

```

```

        gnum = gnum+gnl*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
    END
END
VALUE -1:
FOR j = 1 TO 2
    emax.member = emax.num+j-1
    gden = gden+MIN(mu.emin[i],mu.emax[j])
    oden = gden
    IF (layer = 1) THEN
        CASE emax.member OF
            VALUE -4:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
            VALUE -3:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
            VALUE -2:
                gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
            VALUE -1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE 0:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE 1:
                gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
                onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
            VALUE 2:
                gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 3:
                gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 4:
                gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
            VALUE 5:
                gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        END
    ELSE
        CASE emax.member OF
            VALUE -4:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
            VALUE -3:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
            VALUE -2:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE -1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE 0:

```

```

        gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 1:
        gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 2:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 3:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
        VALUE 4:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
        VALUE 5:
        gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
    END
END
    END
        VALUE 0:
    FOR j = 1 TO 2
        emax.member = emax.num+j-1
        gden = gden+MIN(mu.emin[i],mu.emax[j])
        oden = gden
    IF (layer = 1) THEN
        CASE emax.member OF
            VALUE -4:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opm*MIN(mu.emin[i],mu.emax[j])
            VALUE -3, -2:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
            VALUE -1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE 0:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
            VALUE 1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 2:
                gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 3, 4:
                gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
            VALUE 5:
                gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        END
    ELSE
        CASE emax.member OF
            VALUE -4:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opm*MIN(mu.emin[i],mu.emax[j])

```

```

        VALUE -3, -2:
        gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
        onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
        VALUE -1:
        gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 0:
        gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 1:
        gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE 2:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
        VALUE 3, 4:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
        VALUE 5:
        gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
    END
END
END
VALUE 1:
FOR j = 1 TO 2
    emax.member = emax.num+j-1
    gden = gden+MIN(mu.emin[i],mu.emax[j])
    oden = gden
    IF (layer = 1) THEN
        CASE emax.member OF
            VALUE -4, -3:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
            VALUE -2:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+opz*MIN(mu.emin[i],mu.emax[j])
            VALUE -1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
            VALUE 0:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 1:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 2:
                gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE 3, 4:
                gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
            VALUE 5:
                gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        END
    END
ELSE

```

```

CASE emax.member OF
  VALUE -4, -3:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
  VALUE -2:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE -1:
    gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE 0:
    gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE 1:
    gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
    onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
  VALUE 2:
    gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
    onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
  VALUE 3, 4:
    gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
    onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
  VALUE 5:
    gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
    onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
END
END
END
VALUE 2:
FOR j = 1 TO 2
  emax.member = emax.num+j-1
  gden = gden+MIN(mu.emin[i],mu.emax[j])
  oden = gden
CASE emax.member OF
  VALUE -4:
    gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
    onum = onum+ops*MIN(mu.emin[i],mu.emax[j])
  VALUE -3:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE -2:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE -1:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
  VALUE 0:
    gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
    onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
  VALUE 1:
    gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
    onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
  VALUE 2:
    gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
    onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
  VALUE 3:
    gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])

```



```

        onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
    VALUE 4:
        gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
    VALUE 5:
        gnum = gnum+gnm*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
    END
END
VALUE 3:
FOR j = 1 TO 2
    emax.member = emax.num+j-1
    gden = gden+MIN(mu.emin[i],mu.emax[j])
    oden = gden
    CASE emax.member OF
        VALUE -4:
            gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
            onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE -3:
            gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
            onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE -2:
            gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
            onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE -1:
            gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
            onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
        VALUE 0, 1:
            gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
            onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
        VALUE 2:
            gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
            onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
        VALUE 3:
            gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
            onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        VALUE 4:
            gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
            onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
        VALUE 5:
            gnum = gnum+gns*MIN(mu.emin[i],mu.emax[j])
            onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
    END
END
VALUE 4:
FOR j = 1 TO 2
    emax.member = emax.num+j-1
    gden = gden+MIN(mu.emin[i],mu.emax[j])
    oden = gden
    CASE emax.member OF
        VALUE -4:
            gnum = gnum+gpl*MIN(mu.emin[i],mu.emax[j])
            onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE -3:
            gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
            onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
        VALUE -2:

```

```

        gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
        VALUE -1:
        gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
        onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
        VALUE 0, 1:
        gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        VALUE 2:
        gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
        VALUE 3:
        gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
        VALUE 4:
        gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])
        onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
        VALUE 5:
        gnum = gnum+gnz*MIN(mu.emin[i],mu.emax[j])
        onum = onum+(onl-
o.support/2)*MIN(mu.emin[i],mu.emax[j])
        END
    END
    VALUE 5:
    FOR j = 1 TO 2
        emax.member = emax.num+j-1
        gden = gden+MIN(mu.emin[i],mu.emax[j])
        oden = gden
        CASE emax.member OF
            VALUE -4:
                gnum =
gnum+(gpl+g.support/2)*MIN(mu.emin[i],mu.emax[j])
                onum = onum+oze*MIN(mu.emin[i],mu.emax[j])
            VALUE -3:
                gnum = gnum+gpl*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onz*MIN(mu.emin[i],mu.emax[j])
            VALUE -2:
                gnum = gnum+gpl*MIN(mu.emin[i],mu.emax[j])
                onum = onum+ons*MIN(mu.emin[i],mu.emax[j])
            VALUE -1:
                gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onm*MIN(mu.emin[i],mu.emax[j])
            VALUE 0, 1:
                gnum = gnum+gpm*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
            VALUE 2:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
            VALUE 3:
                gnum = gnum+gps*MIN(mu.emin[i],mu.emax[j])
                onum = onum+onl*MIN(mu.emin[i],mu.emax[j])
            VALUE 4:
                gnum = gnum+gpz*MIN(mu.emin[i],mu.emax[j])
                onum = onum+(onl-
o.support/2)*MIN(mu.emin[i],mu.emax[j])
            VALUE 5:
                gnum = gnum+gze*MIN(mu.emin[i],mu.emax[j])

```

```
        onum = onum+(onl-
o.support/2)*MIN(mu.emin[i],mu.emax[j])
        END
    END
END
gain.fuzzy = gnum/gden
offset.fuzzy = onum/oden
RETURN
```

```
.END
```

```

.PROGRAM wb.fuz.test()
;ABSTRACT: This program is used to test the FUZ controller
;INPUT PAR:   None
;OUTPUT PAR:  Gain offset and threshold, min gray and max gray level
;INIT DATE:   Aug 28, 1998
;LAST UPDATE: Aug 28, 1998
      LOCAL camera.num
      CALL vi.init.cam()

      FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_a_sum[i,0] = 0

        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
        i_s_sum[i,0] = 0
      END
      TIMER (2) = 0
      FOR count.gen = 1 TO 120
        CALL fuzzy.set.gain()
        TYPE count.gen, " ", TIMER(2)
        time.gen[count.gen] = TIMER(2)
        FOR camera.num = 1 TO 2
          TYPE /S, "cam", camera.num, " x0=",
x0_arr[camera.num,count.gen], " xn=", xn_arr[camera.num,count.gen]
          TYPE /S, " pixel=", pixel_arr[camera.num,count.gen]
          TYPE /S, " thresh=", thresh_arr[camera.num,count.gen], "
gain=", gain_arr[camera.num,count.gen]
          TYPE " offset=", offset_arr[camera.num,count.gen]
;          TYPE gain.fuzzy, offset.fuzzy
          TYPE /S, "      iaa0=", i_a_min_ave[camera.num,count.gen]
          TYPE /S, "      iaan=", i_a_max_ave[camera.num,count.gen]
          TYPE /S, "      iss0=", i_s_min_ave[camera.num,count.gen]
          TYPE "      issn=", i_s_max_ave[camera.num,count.gen]
          TYPE /S, "      iaa_sum=", i_a_sum[camera.num,count.gen]
          TYPE "      iss_sum=", i_s_sum[camera.num,count.gen]
        END
        TYPE
        TIMER (3) = 0
        WAIT TIMER(3) > 15
      END
.END

```

AGS VISION CODE IN V/V+
ADEPT ONE LANGUAGE FOR
FUZZY INTEGRAL
CONTROLLER

```

/*****F(E+D)+I.200*****/
.PROGRAM wb.fd.i.test()
;ABSTRACT: This program is used to test the (fuz+d)+i controller
;INPUT PAR: None
;OUTPUT PAR: Gain offset and threshold, min gray and max gray level
;INIT DATE: Oct 30, 1998
;LAST UPDATE: Aug 30, 1998
    LOCAL camera.num
    first.time = 0
    TIMER (2) = 0
    CALL wb.vi.init.cam()
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0

        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0

        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0

        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
    END
    FOR count.gen = 1 TO 120
        CALL wb.pid.cal()
        TYPE count.gen, " ", TIMER(2)
        time.gen[count.gen] = TIMER(2)
        FOR camera.num = 1 TO 2
            TYPE /S, "cam", camera.num, " x0=",
x0_arr[camera.num,count.gen], " xn=", xn_arr[camera.num,count.gen]
            TYPE /S, " pixel=", pixel_arr[camera.num,count.gen]
            TYPE /S, " thresh=", thresh_arr[camera.num,count.gen], "
gain=", gain_arr[camera.num,count.gen]
            TYPE " offset=", offset_arr[camera.num,count.gen]
            TYPE /S, " aa0=", i_a_min_ave[camera.num,count.gen]
            TYPE /S, " aan=", i_a_max_ave[camera.num,count.gen]
            TYPE /S, " ss0=", i_s_min_ave[camera.num,count.gen]
            TYPE " ssn=", i_s_max_ave[camera.num,count.gen]
            TYPE /S, " aa_sum=", i_a_sum[camera.num,count.gen]
            TYPE " ss_sum=", i_s_sum[camera.num,count.gen]
;                TYPE
        END
        TYPE
        TIMER (3) = 0
        WAIT TIMER(3) > 15
    END
.END

```

```

.PROGRAM wb.pid.cal()
;ABSTRACT: set gain, offset and threshold using a PID controller
;INPUT PARAMETER: none
;OUTPUT PARAMETER: vision parameter such as: gain, offset, threshold
;INITIAL DATE: Aug 16, 1998
;LAST UPDATE: Aug 22, 1998
;****variables declaration****
    LOCAL camera1, camera2, camera3, camera.number
    camera1 = 1
    camera2 = 2
    camera3 = 3
    count.pid.end = 200
;****initialize cameras****
    first.time = 0 ;flag for initialization at only the first time
;    CALL wb.vi.init.cam()
;****following is for testing the program defuz_fd_i****
;    first.time = 0
;    TIMER (2) = 0
    CALL wb.vi.init.cam()
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0

        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0

        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0

        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
    END
;****end testing
;****calibrate vision parameters using PID controller****
    FOR camera.number = camera1 TO camera3
        gain.inc.sum[camera.number] = 0
        gain.inc.der[camera.number] = 0
        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
    TIMER (1) = 0
    TIMER (5) = 0
    FOR count.pid = 1 TO count.pid.end
        IF count.pid == 101 THEN
            TIMER (5) = 0
            min.gray[1] = 10 ;decided
            max.gray[1] = 60 ;decided
            min.gray[2] = 10 ;decided
            max.gray[2] = 60 ;decided
            min.gray[3] = 10 ;decided
            max.gray[3] = 60 ;decided
            FOR i = 1 TO 3
                i_a_min[i,100] = 0
                i_a_min_ave[i,100] = 0

                i_a_max[i,100] = 0
                i_a_max_ave[i,100] = 0
            END
        END IF
    END FOR

```

```

        i_s_min[i,100] = 0
        i_s_min_ave[i,100] = 0

        i_s_max[i,100] = 0
        i_s_max_ave[i,100] = 0
    END
    FOR camera.number = camera1 TO camera3
        gain.inc.sum[camera.number] = 0
        gain.inc.der[camera.number] = 0
        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
    END
    END

    VPICTURE (camera1, TRUE) 2, 0
    CALL wb.pid.control(camera1, frame1)

    VPICTURE (camera2, TRUE) 2, 0
    CALL wb.pid.control(camera2, frame1)
;
    VPICTURE (camera3, TRUE) 2, 0
;
    CALL wb.pid.control(camera3, frame1)
    tran_time[count.pid] = TIMER(1)
    TYPE count.pid, " ", tran_time[count.pid], " ", TIMER(5)
    TYPE
;
    TYPE /B, TIMER(1), " sec"
    END

    RETURN

.END

```



```

.PROGRAM defuz_fd_i(e1, e2, e1_der, e2_der)
;***** program variable*****
;*****Oct 30 1998*****
LOCAL eg, eo, gnum, gden, onum, oden, eg.member, eo.member
LOCAL eg.spread, eo.spread, g.spread, o.spread, eg.n, eo.n, g.n,
o.n
LOCAL eg.support, eo.support, g.support, o.support
LOCAL eg.nl, eg.nm, eg.ns, eg.nz, egze, eg.pz, eg.ps, eg.pm,
eg.pl
LOCAL eo.nl, eo.nm, eo.ns, eo.nz, eo.ze, eo.pz, eo.ps, eo.pm,
eo.pl
LOCAL gnl, gnm, gns, gnz, gze, gpz, gps, gpm, gpl
LOCAL onl, onm, ons, onz, oze, opz, ops, opm, opl
LOCAL mu.eg[], mu.eo[]
LOCAL eg.range1, eg.range2, eo.range1, eo.range2
LOCAL eg.num, eo.num
eg = e1
eo = e2
egd = e1_der
eod = e2_der
gden = 0
oden = 0
gnum = 0
onum = 0
;*****assign initial value and calculate fuzzy numbers*****
eg.range1 = 2
eg.range2 = 4
eo.range1 = 4
eo.range2 = 12
IF ((eg <= eg.range1) AND (eg >= -eg.range1)) THEN
eg.support = eg.range1/2
g.support = 2
ELSE
IF ((eg <= eg.range2) AND (eg >= -eg.range2)) THEN
eg.support = eg.range2/2
g.support = 8
ELSE
eg.support = eg.range2
g.support = 32
END
END

IF ((eo <= eo.range1) AND (eo >= -eo.range1)) THEN
eo.support = eo.range1/2
o.support = 2
ELSE
IF ((eo <= eo.range2) AND (eo >= -eo.range2)) THEN
eo.support = eo.range2/2
o.support = 4
ELSE
eo.support = eo.range2
o.support = 8
END
END

END
egd.support = 4
eod.support = 4

```

```

eg.n = 2
eo.n = 2
egd.n = 2
eod.n = 2
g.n = 2
o.n = 2

eg.spread = eg.support*eg.n
eo.spread = eo.support*eo.n
egd.spread = egd.support*egd.n
eod.spread = eod.support*eod.n
g.spread = g.support*g.n
o.spread = o.support*o.n

;*****partition the universe of discourse for input fuzzy variable
eg.pl = eg.spread
eg.pm = eg.spread*3/4
eg.ps = eg.support                ; (emin.spread/2)
eg.pz = eg.support/2            ; (emin.spread/4)
eg.ze = 0
eg.nz = -eg.pz
eg.ns = -eg.ps
eg.nm = -eg.pm
eg.nl = -eg.pl

egd.pl = egd.spread
egd.pm = egd.spread*3/4
egd.ps = egd.support            ; (emin.spread/2)
egd.pz = egd.support/2        ; (emin.spread/4)
egd.ze = 0
egd.nz = -egd.pz
egd.ns = -egd.ps
egd.nm = -egd.pm
egd.nl = -egd.pl

eo.pl = eo.spread
eo.pm = eo.spread*3/4
eo.ps = eo.support                ; (emin.spread/2)
eo.pz = eo.support/2            ; (emin.spread/4)
eo.ze = 0
eo.nz = -eo.pz
eo.ns = -eo.ps
eo.nm = -eo.pm
eo.nl = -eo.pl

eod.pl = eod.spread
eod.pm = eod.spread*3/4
eod.ps = eod.support            ; (emin.spread/2)
eod.pz = eod.support/2        ; (emin.spread/4)
eod.ze = 0
eod.nz = -eod.pz
eod.ns = -eod.ps
eod.nm = -eod.pm
eod.nl = -eod.pl

;*****partition the universe of discourse for output fuzzy variable*****

```

```

gpl = g.spread
gpm = g.spread*3/4
gps = g.support
gpz = g.support/2
gze = 0
gnz = -gpz
gns = -gps
gnm = -gpm
gnl = -gpl

opl = o.spread
opm = o.spread*3/4
ops = o.support
opz = o.support/2
oze = 0
onz = -opz
ons = -ops
onm = -opm
onl = -opl
; **calculate input range and mu vector for tow consequentive partitions
IF (eg >= 0) THEN
    eg.num = INT(eg/eg.support*2)
ELSE
    eg.num = INT(eg/eg.support*2-1)
END
IF (eg.num < -4) THEN
    eg.num = -4
ELSE
    IF (eg.num >= 4) THEN
        eg.num = 4 ;*****mu.emin[1]=0 and mu.emin[2]=1***
    END
END

IF (egd >= 0) THEN
    egd.num = INT(egd/egd.support*2)
ELSE
    egd.num = INT(egd/egd.support*2-1)
END
IF (egd.num < -4) THEN
    egd.num = -4
ELSE
    IF (egd.num >= 4) THEN
        egd.num = 4 ;*****mu.emin[1]=0 and mu.emin[2]=1***
    END
END

IF (eo >= 0) THEN
    eo.num = INT(eo/eo.support*2)
ELSE
    eo.num = INT(eo/eo.support*2-1)
END
IF (eo.num < -4) THEN
    eo.num = -4
ELSE
    IF (eo.num >= 4) THEN
        eo.num = 4 ;*****mu.emax[1]=0 and mu.emax[2]=1***
    END
END

```

```

END

IF (eod >= 0) THEN
    eod.num = INT(eod/eod.support*2)
ELSE
    eod.num = INT(eod/eod.support*2-1)
END
IF (eod.num < -4) THEN
    eod.num = -4
ELSE
    IF (eod.num >= 4) THEN
        eod.num = 4 ;*****mu.emax[1]=0 and mu.emax[2]=1*****
    END
END

mu.eg[1] = ((eg.num+1)*eg.support/2-eg)/eg.support/2
mu.eg[2] = 1-mu.eg[1]
mu.egd[1] = ((egd.num+1)*egd.support/2-egd)/egd.support/2
mu.egd[2] = 1-mu.egd[1]
mu.eo[1] = ((eo.num+1)*eo.support/2-eo)/eo.support/2
mu.eo[2] = 1-mu.eo[1]
mu.eod[1] = ((eod.num+1)*eod.support/2-eod)/eod.support/2
mu.eod[2] = 1-mu.eod[1]

;*****calculate fuzzy output gain.fuzzy and offset.fuzzy using*****
;*****weighted average method for defuzzification*****
FOR i = 1 TO 2
    eg.member = eg.num+i-1
    CASE eg.member OF
        VALUE -4:
            FOR j = 1 TO 2
                egd.member = egd.num+j-1
                gden = gden+MIN(mu.eg[i],mu.egd[j])
                CASE egd.member OF
                    VALUE -4:
                        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
                    VALUE -3:
                        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
                    VALUE -2:
                        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
                    VALUE -1, 0, 1:
                        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
                    VALUE 2:
                        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
                    VALUE 3:
                        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
                    VALUE 4:
                        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
                    VALUE 5:
                        gnum = gnum+(gnl-g.support/2)*MIN(mu.eg[i],mu.egd[j])
                END
            END
        VALUE -3:
            FOR j = 1 TO 2
                egd.member = egd.num+j-1
                gden = gden+MIN(mu.eg[i],mu.egd[j])
                CASE egd.member OF

```

```

        VALUE -4:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
        VALUE -1, 0:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE -2:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE 0, 1:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
        gnum = gnum+gnl*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE -1:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:

```

```

        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
    VALUE 0:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
    VALUE 1:
        gnum = gnum+gnz*MIN(mu.eg[i],mu.egd[j])
    VALUE 2:
        gnum = gnum+gns*MIN(mu.eg[i],mu.egd[j])
    VALUE 3:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
    VALUE 4:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
    VALUE 5:
        gnum = gnum+gnm*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 0:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
            gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 1:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
            gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
            gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
            gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
            gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
            gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])

```

```

        VALUE 1:
        gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
        gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
        gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
        gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
        gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 2:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
        gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
        gnum = gnum+gze*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 3:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
        gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
        gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
        gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:

```

```

        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
    VALUE 3:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
    VALUE 4:
        gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
    VALUE 5:
        gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 4:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
            gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
            gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 4:
            gnum = gnum+gps*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
            gnum = gnum+gpz*MIN(mu.eg[i],mu.egd[j])
    END
END
VALUE 5:
FOR j = 1 TO 2
    egd.member = egd.num+j-1
    gden = gden+MIN(mu.eg[i],mu.egd[j])
    CASE egd.member OF
        VALUE -4:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -3:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -2:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE -1:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 0:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 1:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 2:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])
        VALUE 3:
            gnum = gnum+gpl*MIN(mu.eg[i],mu.egd[j])

```



```

        VALUE 4:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
        VALUE 5:
        gnum = gnum+gpm*MIN(mu.eg[i],mu.egd[j])
    END
END
END
END
;***calculate fuzzy number for offset*****
FOR i = 1 TO 2
    eo.member = eo.num+i-1
    CASE eo.member OF
        VALUE -4:
        FOR j = 1 TO 2
            eod.member = eod.num+j-1
            oden = oden+MIN(mu.eo[i],mu.eod[j])
            CASE eod.member OF
                VALUE -4:
                onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
                VALUE -3:
                onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
                VALUE -2:
                onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
                VALUE -1, 0, 1:
                onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
                VALUE 2:
                onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
                VALUE 3:
                onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
                VALUE 4:
                onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
                VALUE 5:
                onum = onum+(onl-o.support/2)*MIN(mu.eo[i],mu.eod[j])
            END
        END
    END
    VALUE -3:
    FOR j = 1 TO 2
        eod.member = eod.num+j-1
        oden = oden+MIN(mu.eo[i],mu.eod[j])
        CASE eod.member OF
            VALUE -4:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
            VALUE -3:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
            VALUE -2:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
            VALUE -1, 0:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
            VALUE 1:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
            VALUE 2:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
            VALUE 3:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
            VALUE 4:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
            VALUE 5:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
        END
    END

```

```

        onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
    END
END
VALUE -2:
FOR j = 1 TO 2
    eod.member = eod.num+j-1
    oden = oden+MIN(mu.eo[i],mu.eod[j])
    CASE eod.member OF
        VALUE -4:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
        VALUE -3:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE -2:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
        VALUE -1:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
        VALUE 0, 1:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
        VALUE 2:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
        VALUE 3:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
        VALUE 4:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
        VALUE 5:
            onum = onum+onl*MIN(mu.eo[i],mu.eod[j])
    END
END
VALUE -1:
FOR j = 1 TO 2
    eod.member = eod.num+j-1
    oden = oden+MIN(mu.eo[i],mu.eod[j])
    CASE eod.member OF
        VALUE -4:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE -3:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE -2:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE -1:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE 0:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE 1:
            onum = onum+onz*MIN(mu.eo[i],mu.eod[j])
        VALUE 2:
            onum = onum+ons*MIN(mu.eo[i],mu.eod[j])
        VALUE 3:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
        VALUE 4:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
        VALUE 5:
            onum = onum+onm*MIN(mu.eo[i],mu.eod[j])
    END
END
VALUE 0:
FOR j = 1 TO 2

```

```

eod.member = eod.num+j-1
oden = oden+MIN(mu.eo[i],mu.eod[j])
CASE eod.member OF
  VALUE -4:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE -3:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE -2:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE -1:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 0:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 1:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 2:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 3:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 4:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  VALUE 5:
    onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
END
END
VALUE 1:
FOR j = 1 TO 2
  eod.member = eod.num+j-1
  oden = oden+MIN(mu.eo[i],mu.eod[j])
  CASE eod.member OF
    VALUE -4:
      onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
    VALUE -3:
      onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
    VALUE -2:
      onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
    VALUE -1:
      onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    VALUE 0:
      onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    VALUE 1:
      onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    VALUE 2:
      onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    VALUE 3:
      onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    VALUE 4:
      onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
    VALUE 5:
      onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
  END
END
END
VALUE 2:
FOR j = 1 TO 2
  eod.member = eod.num+j-1
  oden = oden+MIN(mu.eo[i],mu.eod[j])
  CASE eod.member OF

```

```

VALUE -4:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE -3:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE -2:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE -1:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 0:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 1:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 2:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 3:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 4:
onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
VALUE 5:
onum = onum+oze*MIN(mu.eo[i],mu.eod[j])
END
END
VALUE 3:
FOR j = 1 TO 2
eod.member = eod.num+j-1
oden = oden+MIN(mu.eo[i],mu.eod[j])
CASE eod.member OF
VALUE -4:
onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
VALUE -3:
onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
VALUE -2:
onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
VALUE -1:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE 0:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE 1:
onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
VALUE 2:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 3:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 4:
onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
VALUE 5:
onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
END
END
VALUE 4:
FOR j = 1 TO 2
eod.member = eod.num+j-1
oden = oden+MIN(mu.eo[i],mu.eod[j])
CASE eod.member OF
VALUE -4:
onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
VALUE -3:

```

```

        onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
    VALUE -2:
        onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
    VALUE -1:
        onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
    VALUE 0:
        onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
    VALUE 1:
        onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
    VALUE 2:
        onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
    VALUE 3:
        onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
    VALUE 4:
        onum = onum+ops*MIN(mu.eo[i],mu.eod[j])
    VALUE 5:
        onum = onum+opz*MIN(mu.eo[i],mu.eod[j])
    END
END
VALUE 5:
FOR j = 1 TO 2
    eod.member = eod.num+j-1
    oden = oden+MIN(mu.eo[i],mu.eod[j])
    CASE eod.member OF
        VALUE -4:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE -3:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE -2:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE -1:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE 0:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE 1:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE 2:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE 3:
            onum = onum+opl*MIN(mu.eo[i],mu.eod[j])
        VALUE 4:
            onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
        VALUE 5:
            onum = onum+opm*MIN(mu.eo[i],mu.eod[j])
    END
END
END
END
END
; calculate the output using weighted average sum for defuzzification
END
fid.g.fuzzy = gnum/gden
fid.o.fuzzy = onum/oden
RETURN
.END

```

```

.PROGRAM wb.pid.control(cam.number, fra.number)
;ABSTRACT: Using PID controller to set gain, offset and threshold
;INPUT PAR: Camera number(cam.number) and frame number(fra.number)
;OUTPUT PAR: Gain offset and threshold
;INIT DATE: Aug 16, 1998
;LAST UPDATE: Aug 22, 1998
    LOCAL x0, xn, hst[], h
    gain.i.gain = 0.1
    IF cam.number == 1 THEN
        offset.i.gain = 0.1
    ELSE
        offset.i.gain = 0.1
    END

    h = 0.3
    gain.inc[0] = 0
    offset.change[0] = 0

    dmode = 0
    VWAIT
    VHISTOGRAM (dmode) hst[] = fra.number

;*****THRESHOLD CONTROL*****
    thresh = 0
    pixel[cam.number] = hst[0]
    WHILE ((pixel[cam.number] < set.point[cam.number]) AND (thresh <
127)) DO
        thresh = thresh+1
        pixel[cam.number] = pixel[cam.number]+hst[thresh]
    END
    x0 = 0
    var = hst[0]
    WHILE (var <= area.ref[cam.number]) AND (x0 < 128) DO
        x0 = x0+1
        var = var+hst[x0]
    END
    xn = 127
    var = hst[127]
    WHILE (var <= area.ref[cam.number]) AND (x0 < xn) DO
        xn = xn-1
        var = var+hst[xn]
    END

;*****GAIN CONTROL*****
    var[cam.number,count.pid] = ((x0-min.gray[cam.number])+
(max.gray[cam.number]-xn))/2
    var.offset[cam.number,count.pid] = ((max.gray[cam.number]+
min.gray[cam.number])-(xn+x0))/2
;*****calculate the fuzzy change for the gain and offset*
    IF cam.number == 1 THEN
        CALL defuz_fd_i(var[cam.number,count.pid],
var.offset[cam.number,count.pid], gain.inc.der[cam.number],
offset.chg.der[cam.number])
    ELSE
        CALL defuz_fd_i_2(var[cam.number,count.pid],
var.offset[cam.number,count.pid], gain.inc.der[cam.number],
offset.chg.der[cam.number])

```

```

END
gain = PARAMETER(V.GAIN[cam.number])
gain.inc[count.pid] = INT(fid.g.fuzzy+gain.inc.sum[cam.number]*
gain.i.gain-0.5)
IF gain+gain.inc[count.pid] > 256 THEN
    gain.inc[count.pid] = 256-gain
END
IF gain+gain.inc[count.pid] < 1 THEN
    gain.inc[count.pid] = 1-gain
END
;****simpson 1/3 rule to do 2 interval integration****
IF count.pid < 3 THEN
    IF count.pid < 2 THEN
        gain.inc.sum[cam.number] = var[cam.number,count.pid]
    ELSE
        gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
var[cam.number,count.pid-1])/2
    END
ELSE
    gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-2]+
4* var[cam.number,count.pid-1]+ var[cam.number,count.pid])
END
IF count.pid > 100 THEN
    IF count.pid < 103 THEN
        IF count.pid < 102 THEN
            gain.inc.sum[cam.number] = var[cam.number,count.pid]
        ELSE
            gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
var[cam.number,count.pid-1])/2
        END
    ELSE
        gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-
2]+
4* var[cam.number,count.pid-1]+
var[cam.number,count.pid])
    END
END
;****backward approximation of first derivative****
gain.inc.der[cam.number] = (var[cam.number,count.pid]-
var[cam.number,count.pid-1])/h
;****calculate new gain value, but not change real parameter****
gain = gain+gain.inc[count.pid]

;*****OFFSET CONTROL*****
offset = PARAMETER(V.OFFSET[cam.number])
IF (var.offset[cam.number,count.pid] <> 0) THEN
    offset.change[count.pid] =
INT(fid.o.fuzzy+offset.chg.sum[cam.number]*offset.i.gain)
    IF offset+offset.change[count.pid] < 0 THEN
        offset.change[count.pid] = -offset
    END
    IF offset+offset.change[count.pid] > 255 THEN
        offset.change[count.pid] = 255-offset
    END
ELSE
    offset.change[count.pid] = 0
END
END

```

```

;****simpson 1/3 rule to calculate 2 interval integral****
  IF count.pid < 3 THEN
    IF count.pid < 2 THEN
      offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]
    ELSE
      offset.chg.sum[cam.number] =
h*(var.offset[cam.number,count.pid]+
var.offset[cam.number,count.pid-1])/2
    END
  ELSE
    offset.chg.sum[cam.number] = h/3*(
var.offset[cam.number,count.pid-
2]+4* var.offset[cam.number,count.pid-1]+
var.offset[cam.number,count.pid])
  END
  IF count.pid > 100 THEN
    IF count.pid < 3 THEN
      IF count.pid < 2 THEN
        offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]
      ELSE
        offset.chg.sum[cam.number] =
*(var.offset[cam.number,count.pid]+
var.offset[cam.number,count.pid-1])/2
      END
    ELSE
      offset.chg.sum[cam.number] = h/3*(
var.offset[cam.number,count.pid-
2]+4* var.offset[cam.number,count.pid-1]+
var.offset[cam.number,count.pid])
    END
  END
;****backward approximation of first derivative****
  offset.chg.der[cam.number] = (var.offset[cam.number,count.pid]-
var.offset[cam.number,count.pid-1])/h
;****calculate new offset value before change real parameter***
  offset = offset+offset.change[count.pid]
;****make change to the gain offset and threshold****
  PARAMETER V.THRESHOLD[cam.number] = thresh
  PARAMETER V.GAIN[cam.number] = gain
  PARAMETER V.OFFSET[cam.number] = offset
;*****KEEP ALL THE PARAMETER FOP ANALYSIS*****
;****warm up period (transit respond)****
  count.gen = 1 ; for transient response test,when run main
pro,takeoff
  x0_warm[cam.number,count.gen,count.pid] = x0
  xn_warm[cam.number,count.gen,count.pid] = xn
  i_aa0 = ABS(x0-min.gray[cam.number])
  i_aan = ABS(xn-max.gray[cam.number])
  i_ss0 = i_aa0*i_aa0
  i_ssn = i_aan*i_aan

  i_a_min[cam.number,count.pid] = i_a_min[cam.number,count.pid-
1]+i_aa0
  i_a_min_ave[cam.number,count.pid] =
i_a_min[cam.number,count.pid]/TIMER(5)*60

```



```

        i_a_max[cam.number,count.pid] = i_a_max[cam.number,count.pid-
1]+i_aan
        i_a_max_ave[cam.number,count.pid] =
i_a_max[cam.number,count.pid]/TIMER(5)*60
        i_a_sum[cam.number,count.pid] =
i_a_min_ave[cam.number,count.pid]+i_a_max_ave[cam.number,count.pid]

        i_s_min[cam.number,count.pid] = i_s_min[cam.number,count.pid-
1]+i_ss0
        i_s_min_ave[cam.number,count.pid] =
i_s_min[cam.number,count.pid]/TIMER(5)*60
        i_s_max[cam.number,count.pid] = i_s_max[cam.number,count.pid-
1]+i_ssn
        i_s_max_ave[cam.number,count.pid] =
i_s_max[cam.number,count.pid]/TIMER(5)*60
        i_s_sum[cam.number,count.pid] =
i_s_min_ave[cam.number,count.pid]+i_s_max_ave[cam.number,count.pid]
        pixel_warm[cam.number,count.gen,count.pid] = pixel[cam.number]
        thresh_warm[cam.number,count.gen,count.pid] = thresh
        gain_warm[cam.number,count.gen,count.pid] = gain
        offset_warm[cam.number,count.gen,count.pid] = offset

        TYPE /S, cam.number, " th=", thresh, " ga=",
PARAMETER(V.GAIN[cam.number])
        TYPE /S, " of=", PARAMETER(V.OFFSET[cam.number]), " ", x0, " ",
xn
        TYPE /S, " g.i=", gain.inc[count.pid], " o.i=",
offset.change[count.pid]
        TYPE /S, " ", i_s_min_ave[cam.number,count.pid], " ",
i_s_max_ave[cam.number,count.pid]
        TYPE " ", i_s_sum[cam.number,count.pid]
        RETURN

.END

```

AGS VISION CODE IN V/V+
ADEPT ONE LANGUAGE FOR
FUZZY-INTEGRAL-DERIVATIVE
CONTROLLER

```

/*****FID.200*****/
.PROGRAM wb.pid.cal()
;ABSTRACT:set gain, offset and threshold using a PID controller
;INPUT PARAMETER: none
;OUTPUT PARAMETER: vision parameter such as: gain, offset, threshold
;INITIAL DATE: Aug 16, 1998
;LAST UPDATE: Aug 22, 1998
;****variables declaration****
    LOCAL camera1, camera2, camera3, camera.number
    camera1 = 1
    camera2 = 2
    camera3 = 3
    count.pid.end = 200
;****initialize cameras****
    first.time = 0
    CALL wb.vi.init.cam()
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
    END
;****calibrate vision parameters using PID controller****
    FOR camera.number = camera1 TO camera3
        gain.inc.sum[camera.number] = 0
        gain.inc.der[camera.number] = 0
        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
    TIMER (1) = 0
    TIMER (5) = 0
    FOR count.pid = 1 TO count.pid.end
        IF count.pid == 101 THEN
            TIMER (5) = 0
            min.gray[1] = 10 ;decided
            max.gray[1] = 60 ;decided
            min.gray[2] = 10 ;decided
            max.gray[2] = 60 ;decided
            min.gray[3] = 10 ;decided
            max.gray[3] = 60 ;decided
            FOR i = 1 TO 3
                i_a_min[i,100] = 0
                i_a_min_ave[i,100] = 0
                i_a_max[i,100] = 0
                i_a_max_ave[i,100] = 0
                i_s_min[i,100] = 0
                i_s_min_ave[i,100] = 0
                i_s_max[i,100] = 0
                i_s_max_ave[i,100] = 0
            END
            FOR camera.number = camera1 TO camera3
                gain.inc.sum[camera.number] = 0
                gain.inc.der[camera.number] = 0
            END
        END
    END

```

```

        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
END
VPICTURE (camera1, TRUE) 2, 0
CALL wb.pid.control(camera1, frame1)

VPICTURE (camera2, TRUE) 2, 0
CALL wb.pid.control(camera2, frame1)
;
    VPICTURE (camera3, TRUE) 2, 0
;
    CALL wb.pid.control(camera3, frame1)
    tran_time[count.pid] = TIMER(1)
    TYPE count.pid, " ", tran_time[count.pid], " ", TIMER(5)
    TYPE
;
    TYPE /B, TIMER(1), " sec"
END
RETURN
.END

```

```

.PROGRAM fid_defuz(e1, e2)
    LOCAL eg, eo, gnum, gden, onum, oden, eg.member, eo.member
    LOCAL eg.spread, eo.spread, g.spread, o.spread, eg.n, eo.n, g.n,
o.n
    LOCAL eg.nl, eg.nm, eg.ns, eg.nz, egze, eg.pz, eg.ps, eg.pm,
eg.pl
    LOCAL eo.nl, eo.nm, eo.ns, eo.nz, eo.ze, eo.pz, eo.ps, eo.pm,
eo.pl
    LOCAL gnl, gnm, gns, gnz, gze, gpz, gps, gpm, gpl
    LOCAL onl, onm, ons, onz, oze, opz, ops, opm, opl
    LOCAL mu.eg[], mu.eo[]
    LOCAL eg.range1, eg.range2, eo.range1, eo.range2
    eg = e1
    eo = e2
    gden = 0
    oden = 0
    gnum = 0
    onum = 0

;*****assign initial value and calculate fuzzy numbers*****
    eg.range1 = 2
    eg.range2 = 4
    eo.range1 = 4
    eo.range2 = 12
    IF ((eg <= eg.range1) AND (eg >= -eg.range1)) THEN
        g.layer = 1
        eg.support = eg.range1/2
        g.support = 2
    ELSE
        IF ((eg <= eg.range2) AND (eg >= -eg.range2)) THEN
            eg.support = eg.range2/2
            g.support = 8
        ELSE
            eg.support = eg.range2
            g.support = 32
        END
    END
END

    IF ((eo <= eo.range1) AND (eo >= -eo.range1)) THEN
        o.layer = 1
        eo.support = eo.range1/2
        o.support = 2
    ELSE
        IF ((eo <= eo.range2) AND (eo >= -eo.range2)) THEN
            eo.support = eo.range2/2
            o.support = 4
        ELSE
            eo.support = eo.range2
            o.support = 8
        END
    END
END

    eg.n = 2
    eo.n = 2
    g.n = 2
    o.n = 2
    eg.spread = eg.support*eg.n

```

```

eo.spread = eo.support*eo.n

g.spread = g.support*g.n
o.spread = o.support*o.n
;*****partition the universe of discourse for input fuzzy variable**
eg.pl = eg.spread
eg.pm = eg.spread*3/4
eg.ps = eg.support          ;(emin.spread/2)
eg.pz = eg.support/2      ;(emin.spread/4)
eg.ze = 0
eg.nz = -eg.pz
eg.ns = -eg.ps
eg.nm = -eg.pm
eg.nl = -eg.pl

eo.pl = eo.spread
eo.pm = eo.spread*3/4
eo.ps = eo.support          ;(emin.spread/2)
eo.pz = eo.support/2      ;(emin.spread/4)
eo.ze = 0
eo.nz = -eo.pz
eo.ns = -eo.ps
eo.nm = -eo.pm
eo.nl = -eo.pl

;*****partition the universe of discourse for output fuzzy variable****
gpl = g.spread
gpm = g.spread*3/4
gps = g.support
gpz = g.support/2
gze = 0
gnz = -gpz
gns = -gps
gnm = -gpm
gnl = -gpl

opl = o.spread
opm = o.spread*3/4
ops = o.support
opz = o.support/2
oze = 0
onz = -opz
ons = -ops
onm = -opm
onl = -opl
;calculate input range and mu vector for tow consequentive partitions**
IF (eg >= 0) THEN
    eg.num = INT(eg/eg.support*2)
ELSE
    eg.num = INT(eg/eg.support*2-1)
END
IF (eg.num < -4) THEN
    eg.num = -4
ELSE
    IF (eg.num >= 4) THEN
        eg.num = 4          ;*****mu.emin[1]=0 and mu.emin[2]=1***
    END

```

```

END

IF (eo >= 0) THEN
    eo.num = INT(eo/eo.support*2)
ELSE
    eo.num = INT(eo/eo.support*2-1)
END
IF (eo.num < -4) THEN
    eo.num = -4
ELSE
    IF (eo.num >= 4) THEN
        eo.num = 4 ;*****mu.emax[1]=0 and mu.emax[2]=1****
    END
END

mu.eg[1] = ((eg.num+1)*eg.support/2-eg)/eg.support/2
mu.eg[2] = 1-mu.eg[1]
mu.eo[1] = ((eo.num+1)*eo.support/2-eo)/eo.support/2
mu.eo[2] = 1-mu.eo[1]

;*****calculate fuzzy output gain.fuzzy and offset.fuzzy using*****
;*****weighted average method for defuzzification*****
FOR i = 1 TO 2
    eg.member = eg.num+i-1
    gden = gden+mu.eg[i]
    IF g.layer < 2 THEN
        CASE eg.member OF
            VALUE -4:
                gnum = gnum+gnl*mu.eg[i]
            VALUE -3:
                gnum = gnum+gnm*mu.eg[i]
            VALUE -2:
                gnum = gnum+gns*mu.eg[i]
            VALUE -1:
                gnum = gnum+gze*mu.eg[i]
            VALUE 0:
                gnum = gnum+gze*mu.eg[i]
            VALUE 1:
                gnum = gnum+gze*mu.eg[i]
            VALUE 2:
                gnum = gnum+gps*mu.eg[i]
            VALUE 3:
                gnum = gnum+gpm*mu.eg[i]
            VALUE 4:
                gnum = gnum+gpl*mu.eg[i]
            VALUE 5:
                gnum = gnum+gpl*mu.eg[i]
        END
    ELSE
        CASE eg.member OF
            VALUE -4:
                gnum = gnum+gnl*mu.eg[i]
            VALUE -3:
                gnum = gnum+gnm*mu.eg[i]
            VALUE -2:
                gnum = gnum+gns*mu.eg[i]
            VALUE -1:

```

```

    gnum = gnum+gnz*mu.eg[i]
      VALUE 0:
    gnum = gnum+gze*mu.eg[i]
      VALUE 1:
    gnum = gnum+gpz*mu.eg[i]
      VALUE 2:
    gnum = gnum+gps*mu.eg[i]
      VALUE 3:
    gnum = gnum+gpm*mu.eg[i]
      VALUE 4:
    gnum = gnum+gpl*mu.eg[i]
      VALUE 5:
    gnum = gnum+gpl*mu.eg[i]
  END
END
  eo.member = eo.num+i-1
  oden = oden+mu.eo[i]
IF o.layer < 2 THEN
CASE eo.member OF
  VALUE -4:
    onum = onum+onl*mu.eo[i]
  VALUE -3:
    onum = onum+onm*mu.eo[i]
  VALUE -2:
    onum = onum+ons*mu.eo[i]
  VALUE -1:
    onum = onum+onz*mu.eo[i]
  VALUE 0:
    onum = onum+oze*mu.eo[i]
  VALUE 1:
    onum = onum+opz*mu.eo[i]
  VALUE 2:
    onum = onum+ops*mu.eo[i]
  VALUE 3:
    onum = onum+opm*mu.eo[i]
  VALUE 4:
    onum = onum+opl*mu.eo[i]
  VALUE 5:
    onum = onum+opl*mu.eo[i]
END
ELSE
CASE eo.member OF
  VALUE -4:
    onum = onum+onl*mu.eo[i]
  VALUE -3:
    onum = onum+onm*mu.eo[i]
  VALUE -2:
    onum = onum+ons*mu.eo[i]
  VALUE -1:
    onum = onum+onz*mu.eo[i]
  VALUE 0:
    onum = onum+oze*mu.eo[i]
  VALUE 1:
    onum = onum+opz*mu.eo[i]

    VALUE 2:
    onum = onum+ops*mu.eo[i]

```



```
        VALUE 3:
onum = onum+opm*mu.eo[i]
        VALUE 4:
onum = onum+opl*mu.eo[i]
        VALUE 5:
onum = onum+opl*mu.eo[i]
    END
END
END
fid.g.fuzzy = gnum/gden
fid.o.fuzzy = onum/oden
RETURN
.END
```

```

.PROGRAM wb.pid.control(cam.number, fra.number)
;ABSTRACT: Using PID controller to set gain, offset and threshold
;INPUT PAR: Camera number(cam.number) and frame number(fra.number)
;OUTPUT PAR: Gain offset and threshold
;INIT DATE: Aug 16, 1998
;LAST UPDATE: Aug 22, 1998
LOCAL x0, xn, hst[], h
gain.gain = 3.84
gain.i.gain = 0.1
gain.p.gain = 0.01
offset.gain = 1
offset.i.gain = 0.1
offset.p.gain = 0.01
h = 0.3
gain.inc[0] = 0
offset.change[0] = 0
dmode = 0
VWAIT
VHISTOGRAM (dmode) hst[] = fra.number
;*****THRESHOLD CONTROL*****
thresh = 0
pixel[cam.number] = hst[0]
WHILE ((pixel[cam.number] < set.point[cam.number]) AND (thresh <
127)) DO
    thresh = thresh+1
    pixel[cam.number] = pixel[cam.number]+hst[thresh]
END
x0 = 0
var = hst[0]
WHILE (var <= area.ref[cam.number]) AND (x0 < 128) DO
    x0 = x0+1
    var = var+hst[x0]
END
xn = 127
var = hst[127]
WHILE (var <= area.ref[cam.number]) AND (x0 < xn) DO
    xn = xn-1
    var = var+hst[xn]
END

;*****GAIN CONTROL*****
var[cam.number,count.pid] = ((x0-
min.gray[cam.number])+(max.gray[cam.number]-xn))/2
var.offset[cam.number,count.pid] =
((max.gray[cam.number]+min.gray[cam.number])-(xn+x0))/2
;*****calculate the fuzzy change for the gain and offset*****
CALL fid_defuz(var[cam.number,count.pid],
var.offset[cam.number,count.pid])
gain = PARAMETER(V.GAIN[cam.number])
gain.inc[count.pid] =
INT(fid.g.fuzzy+gain.inc.sum[cam.number]*gain.i.gain+gain.i
nc.der[cam.number]*gain.p.gain-0.5)
"
IF gain+gain.inc[count.pid] > 256 THEN
    gain.inc[count.pid] = 256-gain
END
IF gain+gain.inc[count.pid] < 1 THEN

```

```

        gain.inc[count.pid] = 1-gain
    END
;****simpson 1/3 rule to do 2 interval integration****
    IF count.pid < 3 THEN
        IF count.pid < 2 THEN
            gain.inc.sum[cam.number] = var[cam.number,count.pid]
        ELSE
            gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
            var[cam.number,count.pid-1])/2
        END
    ELSE
        gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-2]+
        4* var[cam.number,count.pid-1]+ var[cam.number,count.pid])
    END
    IF count.pid > 100 THEN
        IF count.pid < 103 THEN
            IF count.pid < 102 THEN
                gain.inc.sum[cam.number] = var[cam.number,count.pid]
            ELSE
                gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
                var[cam.number,count.pid-1])/2
            END
        ELSE
            gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-
2]+
            4* var[cam.number,count.pid-1]+
var[cam.number,count.pid])
        END
    END
;****backward approximation of first derivative****
    gain.inc.der[cam.number] = (var[cam.number,count.pid]-
    var[cam.number,count.pid-1])/h
;****calculate new gain value, but not change real parameter****
    gain = gain+gain.inc[count.pid]

;*****OFFSET CONTROL*****
    offset = PARAMETER(V.OFFSET[cam.number])
    IF (var.offset[cam.number,count.pid] <> 0) THEN
        offset.change[count.pid] =
INT(fid.o.fuzzy+offset.chg.sum[cam.number]*offset.i.gain+offset.chg.der
[cam.number])
        IF offset+offset.change[count.pid] < 0 THEN
            offset.change[count.pid] = -offset
        END
        IF offset+offset.change[count.pid] > 255 THEN
            offset.change[count.pid] = 255-offset
        END
    ELSE
        offset.change[count.pid] = 0
    END
;****simpson 1/3 rule to calculate 2 interval integral****
    IF count.pid < 3 THEN
        IF count.pid < 2 THEN
            offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]
        ELSE

```

```

        offset.chg.sum[cam.number] =
h*(var.offset[cam.number,count.pid]+
var.offset[cam.number,count.pid-1])/2
    END
ELSE
offset.chg.sum[cam.number] = h/3*( var.offset[cam.number,count.pid-
2]+4* var.offset[cam.number,count.pid-1]+
var.offset[cam.number,count.pid])
END
IF count.pid > 100 THEN
    IF count.pid < 3 THEN
        IF count.pid < 2 THEN
            offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]
        ELSE
            offset.chg.sum[cam.number] =
h*(var.offset[cam.number,count.pid]+
var.offset[cam.number,count.pid-1])/2
        END
    ELSE
        offset.chg.sum[cam.number] = h/3*(
var.offset[cam.number,count.pid-
2]+4* var.offset[cam.number,count.pid-1]+
var.offset[cam.number,count.pid])
    END
END
;****backward approximation of first derivative****
offset.chg.der[cam.number] = (var.offset[cam.number,count.pid]-
var.offset[cam.number,count.pid-1])/h
offset.chg.der[cam.number] =
offset.chg.der[cam.number]*offset.p.gain
;****calculate new offset value before change real parameter***
offset = offset+offset.change[count.pid]

;*****make change to the gain offset and threshold****
PARAMETER V.THRESHOLD[cam.number] = thresh
PARAMETER V.GAIN[cam.number] = gain
PARAMETER V.OFFSET[cam.number] = offset

;*****KEEP ALL THE PARAMETER FOR ANALYSIS*****
;****warm up period (transit respond)****
count.gen = 1
;
    IF (count.gen <= 1) THEN
        x0_warm[cam.number,count.gen,count.pid] = x0
        xn_warm[cam.number,count.gen,count.pid] = xn
        i_aa0 = ABS(x0-min.gray[cam.number])
        i_aan = ABS(xn-max.gray[cam.number])
        i_ss0 = i_aa0*i_aa0
        i_ssn = i_aan*i_aan
        i_a_min[cam.number,count.pid] = i_a_min[cam.number,count.pid-
1]+i_aa0
        i_a_min_ave[cam.number,count.pid] =
i_a_min[cam.number,count.pid]/TIMER(5)*60
        i_a_max[cam.number,count.pid] = i_a_max[cam.number,count.pid-
1]+i_aan
        i_a_max_ave[cam.number,count.pid] =
i_a_max[cam.number,count.pid]/TIMER(5)*60

```

```

        i_a_sum[cam.number,count.pid] =
i_a_min_ave[cam.number,count.pid]+i_a_max_ave[cam.number,count.pid]

        i_s_min[cam.number,count.pid] = i_s_min[cam.number,count.pid-
1]+i_ss0
        i_s_min_ave[cam.number,count.pid] =
i_s_min[cam.number,count.pid]/TIMER(5)*60
        i_s_max[cam.number,count.pid] = i_s_max[cam.number,count.pid-
1]+i_ssn
        i_s_max_ave[cam.number,count.pid] =
i_s_max[cam.number,count.pid]/TIMER(5)*60
        i_s_sum[cam.number,count.pid] =
i_s_min_ave[cam.number,count.pid]+i_s_max_ave[cam.number,count.pid]

        pixel_warm[cam.number,count.gen,count.pid] = pixel[cam.number]
        thresh_warm[cam.number,count.gen,count.pid] = thresh
        gain_warm[cam.number,count.gen,count.pid] = gain
        offset_warm[cam.number,count.gen,count.pid] = offset
        TYPE /S, cam.number, " th=", thresh, " ga=",
PARAMETER(V.GAIN[cam.number])
        TYPE /S, " of=", PARAMETER(V.OFFSET[cam.number]), " ", x0, " ",
xn
        TYPE /S, " g.i=", gain.inc[count.pid], " o.i=",
offset.change[count.pid]
        TYPE /S, " ", i_s_min[cam.number,count.pid], " ",
i_s_max[cam.number,count.pid]
        TYPE " ", i_s_sum[cam.number,count.pid]
        RETURN
    .END

```

AGS VISION CODE IN V/V+
ADEPT ONE LANGUAGE FOR
PROPORTIONAL-INTEGRAL-DERIVATIVE
CONTROLLER

```

/*****PID.200*****/
.PROGRAM wb.pid.cal()
;ABSTRACT:      set gain, offset and threshold using a PID
controller
;INPUT PARAMETER:  none
;OUTPUT PARAMETER: vision parameter such as: gain, offset, threshold
;INITIAL DATE:    Aug 16, 1998
;LAST UPDATE:    Aug 22, 1998
;****variables declaration****
    LOCAL camera1, camera2, camera3, camera.number
    camera1 = 1
    camera2 = 2
    camera3 = 3
    count.pid.end = 200
    count.pid.cutof = 101
;****initialize cameras****
    first.time = 0          ;flag for initialization at only the first
time
    CALL wb.vi.init.cam()
    FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_a_sum[i,0] = 0

        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
        i_s_sum[i,0] = 0
    END

;****calibrate vision parameters using PID controller****
    FOR camera.number = camera1 TO camera3
        gain.inc.sum[camera.number] = 0
        gain.inc.der[camera.number] = 0
        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
    TIMER (1) = 0
    TIMER (5) = 0
    FOR count.pid = 1 TO count.pid.end
        IF count.pid == count.pid.cutof THEN
            TIMER (5) = 0
            min.gray[1] = 10 ;decided
            max.gray[1] = 60 ;decided
            min.gray[2] = 10 ;decided
            max.gray[2] = 60 ;decided
            min.gray[3] = 10 ;decided
            max.gray[3] = 60 ;decided
            FOR i = 1 TO 3
                i_a_min[i,count.pid-1] = 0
                i_a_min_ave[i,count.pid-1] = 0
                i_a_max[i,count.pid-1] = 0
                i_a_max_ave[i,count.pid-1] = 0
                i_a_sum[i,count.pid-1] = 0
            END
        ENDIF
    END

```

```

        i_s_min[i,count.pid-1] = 0
        i_s_min_ave[i,count.pid-1] = 0
        i_s_max[i,count.pid-1] = 0
        i_s_max_ave[i,count.pid-1] = 0
        i_s_sum[i,count.pid-1] = 0
    END
    FOR camera.number = camera1 TO camera3
        gain.inc.sum[camera.number] = 0
        gain.inc.der[camera.number] = 0
        offset.chg.sum[camera.number] = 0
        offset.chg.der[camera.number] = 0
    END
END
END

VPICTURE (camera1, TRUE) 2, 0
CALL wb.pid.control(camera1, frame1)
VPICTURE (camera2, TRUE) 2, 0
CALL wb.pid.control(camera2, frame1)
;
;
VPICTURE (camera3, TRUE) 2, 0
CALL wb.pid.control(camera3, frame1)
tran_time[count.pid] = TIMER(1)
TYPE count.pid, " ", tran_time[count.pid], " ", TIMER(5)
TYPE
;
TYPE /B, TIMER(1), " sec"
END
RETURN
.END

```



```

PROGRAM wb.pid.control(cam.number, fra.number)
;ABSTRACT: Using PID controller to set gain, offset and threshold
;INPUT PAR: Camera number(cam.number) and frame number(fra.number)
;OUTPUT PAR: Gain offset and threshold
;INIT DATE: Aug 16, 1998
;LAST UPDATE: Aug 22, 1998
LOCAL x0, xn, hst[], h, acc_time
gain.gain = 3.84
gain.i.gain = 0.1
gain.p.gain = 0.05
offset.gain = 1
offset.i.gain = 0.1
offset.p.gain = 0.05
h = 0.3
gain.inc[0] = 0
offset.change[0] = 0

dmode = 0
VWAIT
VHISTOGRAM (dmode) hst[] = fra.number

;*****THRESHOLD CONTROL*****
thresh = 0
pixel[cam.number] = hst[0]
WHILE ((pixel[cam.number] < set.point[cam.number]) AND (thresh <
127)) DO
    thresh = thresh+1
    pixel[cam.number] = pixel[cam.number]+hst[thresh]
END
x0 = 0
var = hst[0]
WHILE (var <= area.ref[cam.number]) AND (x0 < 128) DO
    x0 = x0+1
    var = var+hst[x0]
END
IF x0 > 0 THEN
    x0 = x0-1
END
xn = 127
var = hst[127]
WHILE (var <= area.ref[cam.number]) AND (x0 < xn) DO
    xn = xn-1
    var = var+hst[xn]
END

;*****GAIN CONTROL*****
var[cam.number,count.pid] = ((x0-min.gray[cam.number])+
(max.gray[cam.number]-xn))/2
gain = PARAMETER(V.GAIN[cam.number])
gain.inc[count.pid] = INT(var[cam.number,count.pid]*
gain.gain+gain.inc.sum[cam.number]*gain.i.gain+gain.inc.der[cam.n
umber])*
gain.p.gain)
IF gain+gain.inc[count.pid] > 256 THEN
    gain.inc[count.pid] = 256-gain
END
IF gain+gain.inc[count.pid] < 1 THEN

```

```

        gain.inc[count.pid] = 1-gain
    END
;****simpson 1/3 rule to do 2 interval integration****
    IF count.pid < 3 THEN
        IF count.pid < 2 THEN
            gain.inc.sum[cam.number] = var[cam.number,count.pid]
        ELSE
            gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
            var[cam.number,count.pid-1])/2
        END
    ELSE
        gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-2]+
        4* var[cam.number,count.pid-1]+ var[cam.number,count.pid])
    END
    IF count.pid > 100 THEN
        IF count.pid < 103 THEN
            IF count.pid < 102 THEN
                gain.inc.sum[cam.number] = var[cam.number,count.pid]
            ELSE
                gain.inc.sum[cam.number] = h*(var[cam.number,count.pid]+
                var[cam.number,count.pid-1])/2
            END
        ELSE
            gain.inc.sum[cam.number] = h/3*(var[cam.number,count.pid-
2]+
            4* var[cam.number,count.pid-1]+
var[cam.number,count.pid])
        END
    END
;****backward approximation of first derivative****
    gain.inc.der[cam.number] = (var[cam.number,count.pid]-
    var[cam.number,count.pid-1])/h
;****calculate new gain value, but not change real parameter****
    gain = gain+gain.inc[count.pid]
;*****OFFSET CONTROL*****
    var.offset[cam.number,count.pid] = ((max.gray[cam.number]+
    min.gray[cam.number])-(xn+x0))/2
    offset = PARAMETER(V.OFFSET[cam.number])
    IF (var.offset[cam.number,count.pid] <> 0) THEN
        offset.change[count.pid] =
INT(var.offset[cam.number,count.pid]*
        offset.gain+offset.chg.sum[cam.number]*
        offset.i.gain+offset.chg.der[cam.number])
        IF offset+offset.change[count.pid] < 0 THEN
            offset.change[count.pid] = -offset
        END
        IF offset+offset.change[count.pid] > 255 THEN
            offset.change[count.pid] = 255-offset
        END
    ELSE
        offset.change[count.pid] = 0
    END
;****simpson 1/3 rule to calculate 2 interval integral****
    IF count.pid < 3 THEN
        IF count.pid < 2 THEN
            offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]

```

```

        ELSE
            offset.chg.sum[cam.number] =
h*(var.offset[cam.number,count.pid]+
            var.offset[cam.number,count.pid-1])/2
        END
    ELSE
        offset.chg.sum[cam.number] = h/3*(
var.offset[cam.number,count.pid-
            2]+4* var.offset[cam.number,count.pid-1]+
            var.offset[cam.number,count.pid])
        END
    IF count.pid > 100 THEN
        IF count.pid < 3 THEN
            IF count.pid < 2 THEN
                offset.chg.sum[cam.number] =
var.offset[cam.number,count.pid]
            ELSE
                offset.chg.sum[cam.number] =
h*(var.offset[cam.number,count.pid]+
                var.offset[cam.number,count.pid-1])/2
            END
        ELSE
            offset.chg.sum[cam.number] = h/3*(
var.offset[cam.number,count.pid-
                2]+4* var.offset[cam.number,count.pid-1]+
                var.offset[cam.number,count.pid])
            END
        END
;****backward approximation of first derivative****
offset.chg.der[cam.number] = (var.offset[cam.number,count.pid]-
var.offset[cam.number,count.pid-1])/h
offset.chg.der[cam.number] =
offset.chg.der[cam.number]*offset.p.gain
;****calculate new offset value before change real parameter***
offset = offset+offset.change[count.pid]

;****make change to the gain offset and threshold****
PARAMETER V.THRESHOLD[cam.number] = thresh
PARAMETER V.GAIN[cam.number] = gain
PARAMETER V.OFFSET[cam.number] = offset

;*****KEEP ALL THE PARAMETER FOR ANALYSIS*****
;****warm up period (transit respond)****
count.gen = 1
x0_warm[cam.number,count.gen,count.pid] = x0
xn_warm[cam.number,count.gen,count.pid] = xn
pixel_warm[cam.number,count.gen,count.pid] = pixel[cam.number]
thresh_warm[cam.number,count.gen,count.pid] = thresh
gain_warm[cam.number,count.gen,count.pid] = gain
offset_warm[cam.number,count.gen,count.pid] = offset
IF count.pid < count.pid.cutof THEN
    acc_time = TIMER(1)
ELSE
    acc_time = TIMER(5)
END
i_aa0 = ABS(x0-min.gray[cam.number])
i_aan = ABS(xn-max.gray[cam.number])

```

```

        i_ss0 = i_aa0*i_aa0
        i_ssn = i_aan*i_aan
        i_a_min[cam.number,count.pid] = i_a_min[cam.number,count.pid-
1]+i_aa0
        i_a_min_ave[cam.number,count.pid] =
i_a_min[cam.number,count.pid]/acc_time*60
        i_a_max[cam.number,count.pid] = i_a_max[cam.number,count.pid-
1]+i_aan
        i_a_max_ave[cam.number,count.pid] =
i_a_max[cam.number,count.pid]/acc_time*60
        i_a_sum[cam.number,count.pid] =
i_a_min_ave[cam.number,count.pid]+i_a_max_ave[cam.number,count.pid]

        i_s_min[cam.number,count.pid] = i_s_min[cam.number,count.pid-
1]+i_ss0
        i_s_min_ave[cam.number,count.pid] =
i_s_min[cam.number,count.pid]/acc_time*60
        i_s_max[cam.number,count.pid] = i_s_max[cam.number,count.pid-
1]+i_ssn
        i_s_max_ave[cam.number,count.pid] =
i_s_max[cam.number,count.pid]/acc_time*60
        i_s_sum[cam.number,count.pid] =
i_s_min_ave[cam.number,count.pid]+i_s_max_ave[cam.number,count.pid]

        TYPE /S, "cam ", cam.number, " th=", thresh, " ga=",
PARAMETER(V.GAIN[cam.number])
        TYPE /S, PARAMETER(V.OFFSET[cam.number]), " ", x0, " ", xn
        TYPE /S, " g.i=", gain.inc[count.pid], " o.i=",
offset.change[count.pid]
        TYPE /S, " ", i_s_min_ave[cam.number,count.pid], " ",
i_s_max_ave[cam.number,count.pid]
        TYPE " ", i_s_sum[cam.number,count.pid]
        RETURN
.END

```

```

.PROGRAM wb.pid.test()
;ABSTRACT: This program is used to test the PID controller
;INPUT PAR:   None
;OUTPUT PAR:  Gain offset and threshold, min gray and max gray level
;INIT DATE:   Aug 22, 1998
;LAST UPDATE: Aug 22, 1998
      LOCAL camera.num
      first.time = 0           ;flag for initialization at only the
first time
      CALL wb.vi.init.cam()

      TIMER (2) = 0
      FOR i = 1 TO 3
        i_a_min[i,0] = 0
        i_a_min_ave[i,0] = 0
        i_a_max[i,0] = 0
        i_a_max_ave[i,0] = 0
        i_a_sum[i,0] = 0

        i_s_min[i,0] = 0
        i_s_min_ave[i,0] = 0
        i_s_max[i,0] = 0
        i_s_max_ave[i,0] = 0
        i_s_sum[i,0] = 0
      END

      FOR count.gen = 1 TO 120
        CALL wb.pid.cal()
        TYPE count.gen, " ", TIMER(2)
        time.gen[count.gen] = TIMER(2)
        FOR camera.num = 1 TO 2
          TYPE /S, "cam", camera.num, " x0=",
x0_arr[camera.num,count.gen], " xn=", xn_arr[camera.num,count.gen]
          TYPE /S, " pixel=", pixel_arr[camera.num,count.gen]
          TYPE /S, " thresh=", thresh_arr[camera.num,count.gen], "
gain=", gain_arr[camera.num,count.gen]
          TYPE " offset=", offset_arr[camera.num,count.gen]
          TYPE /S, "   iaa0=", i_a_min_ave[camera.num,count.gen]
          TYPE /S, " iaan=", i_a_max_ave[camera.num,count.gen]
          TYPE /S, " iss0=", i_s_min_ave[camera.num,count.gen]
          TYPE " issn=", i_s_max_ave[camera.num,count.gen]
          TYPE /S, "   iaa_sum=", i_a_sum[camera.num,count.gen]
          TYPE " iss_sum=", i_s_sum[camera.num,count.gen]
        END
        TYPE
        TIMER (3) = 0
        WAIT TIMER(3) > 15
      END

.END

```

```

.PROGRAM wb.vi.init.cam()
;ABSTRACT: initialize all the parameter for 3 cameras before using
PID control
;INPUT PAR: none
;OUTPUT PAR: initial vision parameters
;INIT DATE: Aug 16, 1998
;LAST UPDATE:Aug 16, 1998
    frame1 = 1001      ;global value for frame store one
    frame2 = 1002      ;global value for frame store two
; Set Basic Camera switches:
    ENABLE V.BINARY
    DISABLE V.BACKLIGHT
    ENABLE V.BOUNDARIES
    DISABLE V.FIT.ARCS
    DISABLE V.DISJOINT
    DISABLE V.RECOGNITION
    DISABLE V.SUBTRACT.HOLE
    DISABLE V.PERIMETER
    DISABLE V.STROBE
    DISABLE V.2ND.MOMENTS
    ENABLE V.HOLES
    FOR i = 1 TO 3
        ENABLE V.CENTROID[i]
        DISABLE V.2ND.MOMENTS[i]
        DISABLE V.HOLES[i] ;hole count unessessary for sorting
    END

; Disable display switches:
    DISABLE V.SHOW.EDGES
    DISABLE V.SHOW.BOUNDS
    DISABLE V.SHOW.GRIP
    DISABLE V.SHOW.RECOG

;Set up parameters for 3 cameras
    yscale = 1/0.5397496
    xscale = 1/0.542155
    v.first.line1 = INT(5*yscale)
    v.last.line1 = INT(25*yscale)
    v.first.col1 = INT(115*xscale)
    v.last.col1 = INT(195*xscale)
    v.first.line2 = INT(5*yscale)
    v.last.line2 = INT(25*yscale)
    v.first.col2 = INT(102*xscale)
    v.last.col2 = INT(192*xscale)
    v.first.line3 = INT(2*yscale)
    v.last.line3 = INT(22*yscale)
    v.first.col3 = INT(112*xscale)
    v.last.col3 = INT(192*xscale)
    PARAMETER V.FIRST.LINE[1] = v.first.line1
    PARAMETER V.LAST.LINE[1] = v.last.line1
    PARAMETER V.FIRST.COL[1] = v.first.col1
    PARAMETER V.LAST.COL[1] = v.last.col1
    PARAMETER V.FIRST.LINE[2] = v.first.line2
    PARAMETER V.LAST.LINE[2] = v.last.line2
    PARAMETER V.FIRST.COL[2] = v.first.col2
    PARAMETER V.LAST.COL[2] = v.last.col2
    PARAMETER V.FIRST.LINE[3] = v.first.line3

```

```

PARAMETER V.LAST.LINE[3] = v.last.line3
PARAMETER V.FIRST.COL[3] = v.first.col3
PARAMETER V.LAST.COL[3] = v.last.col3

IF first.time == 0 THEN
  first.time = 1
  FOR j = 1 TO 3
    area.ref[j] = PARAMETER(V.LAST.COL[j]) -
PARAMETER(V.FIRST.COL[j])+1
    area.ref[j] = area.ref[j]*(PARAMETER(V.LAST.LINE[j]) -
PARAMETER(V.FIRST.LINE[j])+1)
    area.ref[j] = area.ref[j]/1000          ;Significant amount of
overlap
  END
  min.gray[1] = 60      ;decided
  max.gray[1] = 90      ;decided
  min.gray[2] = 60      ;decided
  max.gray[2] = 90      ;decided
  min.gray[3] = 60      ;decided
  max.gray[3] = 90      ;decided

;****set gain to low and offset to neutral values****
PARAMETER V.GAIN[1] = 1
PARAMETER V.GAIN[2] = 1
PARAMETER V.GAIN[3] = 1
PARAMETER V.OFFSET[1] = 128
PARAMETER V.OFFSET[2] = 128
PARAMETER V.OFFSET[3] = 128

;****set initial threshold****
PARAMETER V.THRESHOLD[1] = 65      ;decided
PARAMETER V.THRESHOLD[2] = 65      ;decided
PARAMETER V.THRESHOLD[3] = 65      ;decided

END

DISABLE V.BOUNDARIES
DISABLE V.CENTROID

FOR cam = 1 TO 12
  VDISPLAY (cam) 2, 1
END

;*****set pixel numbers FOR THRESHOLD CONTROL*****
set.point[1] = 3900
set.point[2] = 3900
set.point[3] = 3900

RETURN

.END

```

APPENDIX C
SAMPLE OUTPUT DATA

OUTPUT DATA
TRANSIT RESPONSE TEST FOR
PID CONTROLLER
(FIGURE 4.17 AND 4.18)

COLUMN

1. Accumulated time in seconds
2. Transit response testing time in seconds
3. Minimum gray level for camera 1
4. Minimum gray level for camera 2
5. Maximum gray level for camera 1
6. Maximum gray level for camera 2
7. Gain for camera 1
8. Gain for camera 2
9. Offset for camera 1
10. Offset for camera 2
11. Threshold for camera 1
12. Threshold for camera 2

Time(sec.)	Time(sec.)	Xmin(1)	Xmin(2)	Xmax(1)	Xmax(2)	gain-cam1	gain-cam2	off-cam1	off-cam2	thr-cam1	thr-cam2
52.648	0.543998	60	60	90	90	82	119	106	96	76	76
53.188	1.083999	30	29	67	66	107	144	93	84	49	49
53.73	1.625999	23	23	66	64	120	161	84	76	45	45
54.268	2.164001	17	20	64	64	125	172	79	70	41	41
54.808	2.703998	13	16	61	62	128	179	78	67	38	38
55.35	3.245998	13	15	61	62	131	184	77	64	38	38
55.89	3.785999	13	13	61	61	134	187	76	62	38	38
56.43	4.326	12	12	62	60	134	190	75	62	38	38
56.97	4.866001	11	12	61	61	134	191	75	61	37	37
57.51	5.405998	11	12	61	60	134	194	74	60	37	37
58.052	5.947998	11	11	60	61	135	194	74	59	36	36
58.592	6.487999	11	10	60	60	136	194	74	59	36	36
59.13	7.026001	11	10	61	60	136	194	74	59	37	37
59.672	7.568	11	11	61	60	136	195	73	59	37	37
60.212	8.108001	10	11	60	60	136	196	73	59	36	36
60.754	8.650001	10	11	60	61	136	196	73	58	36	36
61.292	9.187999	10	10	60	60	136	196	73	58	36	36
61.832	9.728	10	10	60	59	136	197	73	58	36	36
62.374	10.27	10	10	60	60	136	197	73	58	36	36
62.914	10.81	10	10	60	60	136	197	73	58	36	36
63.456	11.352	9	10	60	60	135	197	73	58	36	36
63.994	11.89	10	10	59	60	136	197	73	58	35	35
64.536	12.432	10	10	60	60	136	197	73	58	36	36
65.076	12.972	10	10	60	60	136	197	73	58	36	36
65.618	13.514	10	10	60	60	136	197	73	58	36	36
66.158	14.054	10	10	60	60	136	197	73	58	36	36
66.696	14.592	10	10	60	60	136	197	73	58	36	36
67.238	15.134	10	10	60	60	136	197	73	58	36	36
67.778	15.674	10	10	60	60	136	197	73	58	36	36
68.318	16.214	10	10	60	60	136	197	73	58	36	36
68.858	16.754	10	10	60	60	136	197	73	58	36	36
69.4	17.296	10	10	60	60	136	197	73	58	36	36
69.94	17.836	10	10	60	60	136	197	73	58	36	36

70.482	18.378	10	10	60	60	136	197	73	58	36	36
71.022	18.918	10	10	60	60	136	197	73	58	36	36
71.562	19.458	10	10	60	60	136	197	73	58	36	36
72.104	20	10	10	60	60	136	197	73	58	36	36
72.644	20.54	10	10	60	60	136	197	73	58	36	36
73.186	21.082	10	10	60	60	136	197	73	58	36	36
73.724	21.62	10	10	60	60	136	197	73	58	36	36
74.264	22.16	10	10	60	60	136	197	73	58	36	36
74.804	22.7	10	10	60	60	136	197	73	58	36	36
75.346	23.242	10	10	60	60	136	197	73	58	36	36
75.886	23.782	10	10	59	60	137	197	73	58	36	36
76.424	24.32	10	10	60	60	137	197	73	58	36	36
76.966	24.862	10	10	60	60	137	197	73	58	36	36
77.506	25.402	10	10	60	60	137	197	73	58	36	36
78.046	25.942	10	10	61	60	136	197	73	58	36	36
78.586	26.482	10	10	60	60	136	197	73	58	36	36
79.126	27.022	10	10	60	60	136	197	73	58	36	36
79.594	27.49	10	10	60	60	136	197	73	58	36	36
80.136	28.032	10	10	59	60	137	197	73	58	36	36
80.676	28.572	10	10	60	60	137	197	73	58	36	36
81.218	29.114	10	10	60	60	137	197	73	58	36	36
81.758	29.654	10	10	60	60	137	197	73	58	36	36
82.298	30.194	10	10	60	60	137	197	73	58	36	36
82.838	30.734	10	10	60	60	137	197	73	58	36	36
83.38	31.276	10	10	60	60	137	197	73	58	36	36
83.922	31.818	10	10	60	60	137	197	73	58	36	36
84.46	32.356	10	10	60	60	137	197	73	58	36	36
85	32.896	10	10	60	60	137	197	73	58	36	36
85.54	33.436	10	10	60	60	137	197	73	58	36	36
86.082	33.978	11	10	61	60	137	197	72	58	36	36
86.622	34.518	9	10	59	60	137	197	72	58	35	35
87.162	35.058	9	11	59	60	137	198	73	58	35	35
87.704	35.6	10	10	60	60	137	198	73	58	36	36
88.244	36.14	10	10	60	60	137	198	73	58	36	36

88.786	36.682	10	10	60	60	137	198	73	58	36	36
89.324	37.22	10	10	60	60	137	198	73	58	36	36
89.864	37.76	10	10	60	60	137	198	73	58	36	36
90.406	38.302	10	11	60	60	137	199	73	58	36	36
90.946	38.842	10	11	60	61	137	199	73	57	36	36
91.486	39.382	11	10	60	60	138	199	73	57	36	36
92.026	39.922	10	10	61	60	137	199	73	57	36	36
92.566	40.462	10	10	60	60	137	199	73	57	36	36
93.108	41.004	10	9	60	60	137	198	73	57	36	36
93.648	41.544	10	10	60	59	137	199	73	57	36	36
94.188	42.084	10	10	60	60	137	199	73	57	36	36
94.728	42.624	10	10	60	60	137	199	73	57	36	36
95.27	43.166	10	10	60	60	137	199	73	57	36	36
95.812	43.708	10	10	60	60	137	199	73	57	36	36
96.35	44.246	10	10	61	60	136	199	73	57	36	36
96.89	44.786	10	10	60	60	136	199	73	57	36	36
97.43	45.326	10	10	60	60	136	199	73	57	36	36
97.972	45.868	10	10	60	60	136	199	73	57	36	36
98.514	46.41	10	9	59	60	137	198	73	57	36	36
99.052	46.948	10	9	60	59	137	198	73	58	36	36
99.592	47.488	10	10	60	61	137	197	73	58	36	36
100.134	48.03	10	10	61	60	136	197	73	58	36	36
100.674	48.57	10	10	60	60	136	197	73	58	36	36
101.214	49.10999	10	10	60	60	136	197	73	58	36	36
101.754	49.64999	10	10	60	60	136	197	73	58	36	36
102.296	50.19199	10	10	60	60	136	197	73	58	35	35
102.836	50.73199	10	10	60	60	136	197	73	58	36	36
103.378	51.27399	10	10	60	60	136	197	73	58	36	36
103.918	51.81399	10	10	60	60	136	197	73	58	36	36
104.458	52.354	10	10	60	60	136	197	73	58	36	36
104.998	52.894	10	10	60	60	136	197	73	58	36	36
105.54	53.436	10	10	60	60	136	197	73	58	36	36
106.082	53.978	10	10	60	60	136	197	73	58	36	36

OUTPUT DATA
STEADY STATE RESPONSE TEST FOR
PID CONTROLLER
(FIGURE 4.19 AND 4.20)

COLUMN

1. Testing time in seconds
2. Testing time in minutes
3. Minimum gray level for camera 1
4. Minimum gray level for camera 2
5. Maximum gray level for camera 1
6. Maximum gray level for camera 2
7. Gain for camera 1
8. Gain for camera 2
9. Offset for camera 1
10. Offset for camera 2
11. Threshold for camera 1
12. Threshold for camera 2

Time(sec.)	Time(min)	Xmin(1)	Xmin(2)	Xmax(1)	Xmax(2)	gain-cam1	gain-cam2	off-cam1	off-cam2	thre-cam1	thre-cam2
32.212	0.536867	10	11	60	61	84	190	101	59	39	37
37.576	0.626267	11	9	61	60	85	189	100	59	40	36
42.966	0.7161	11	10	61	60	85	188	101	60	40	37
48.358	0.805967	10	10	60	60	86	188	100	60	39	37
53.748	0.8958	10	10	60	60	85	188	100	60	39	37
59.14	0.985667	10	10	60	60	85	188	100	60	39	37
64.532	1.075533	10	10	59	60	85	188	100	60	39	37
69.924	1.1654	10	10	60	60	86	190	100	59	39	37
75.318	1.2553	10	10	61	60	85	190	100	59	40	37
80.708	1.345133	10	10	60	60	86	189	99	59	39	37
86.098	1.434967	10	10	60	60	87	189	99	59	39	36
91.49	1.524833	10	10	60	60	87	189	99	59	39	37
96.882	1.6147	10	10	60	60	87	189	99	59	39	37
102.274	1.704567	10	10	60	60	86	188	99	59	39	36
107.666	1.794433	11	10	60	60	87	189	99	59	39	37
113.06	1.884333	9	10	59	60	87	189	99	59	38	37
118.452	1.9742	10	11	61	61	87	189	99	59	39	37
123.844	2.064067	0	0	44	42	80	216	118	70	37	39
129.236	2.153933	10	10	60	60	78	219	119	69	38	38
134.628	2.2438	10	10	60	60	77	219	119	69	37	38
140.024	2.333733	9	10	60	60	78	220	118	69	37	38
145.416	2.4236	11	10	61	60	78	220	118	69	38	38
150.81	2.5135	9	10	59	61	78	220	119	69	37	39
156.21	2.6035	9	9	60	59	77	220	118	69	37	37
161.532	2.6922	9	11	60	61	77	219	118	68	37	39
166.924	2.782067	10	10	60	60	77	220	118	68	37	38
172.316	2.871933	10	10	60	60	77	220	118	68	37	38
177.708	2.9618	10	10	60	60	77	220	118	68	37	38
183.102	3.0517	10	10	60	60	77	220	118	68	37	38
188.498	3.141633	10	11	60	61	77	218	118	68	38	38
193.892	3.231533	10	10	60	60	78	219	118	69	37	38
199.284	3.3214	10	10	60	61	78	220	119	70	37	39
204.68	3.411333	10	10	60	60	78	222	119	69	37	38

210.072	3.5012	10	10	60	60	78	222	119	69	37	38
215.464	3.591067	10	10	60	60	78	223	119	69	37	38
220.86	3.681	10	10	60	60	78	223	119	69	37	38
226.252	3.770867	10	10	61	60	78	223	119	69	37	38
231.648	3.8608	10	10	60	60	78	223	119	69	37	38
237.042	3.9507	10	10	61	60	78	223	119	69	37	38
242.436	4.0406	29	28	76	82	85	200	101	55	40	35
247.756	4.129267	10	10	60	60	87	191	99	59	39	37
253.148	4.219133	10	10	60	60	88	191	99	59	39	37
258.548	4.309133	10	10	59	60	89	191	98	59	39	37
263.942	4.399033	10	10	60	60	89	190	98	59	39	37
269.334	4.4889	10	10	60	60	89	190	98	59	39	37
274.726	4.578767	10	10	60	60	89	190	98	59	39	37
280.12	4.668667	10	10	60	60	89	190	98	59	39	37
285.516	4.7586	10	10	60	60	89	190	98	59	39	37
290.908	4.848466	10	10	60	60	89	190	98	59	39	37
296.3	4.938333	10	10	60	60	89	190	98	59	39	36
301.698	5.0283	10	10	60	60	90	190	98	59	39	37
307.092	5.1182	10	10	60	60	89	189	98	60	39	37
312.484	5.208067	10	10	60	60	89	190	98	59	39	36
317.878	5.297967	10	10	60	60	89	190	98	59	39	36
323.272	5.387867	10	10	60	60	89	190	98	59	39	37
328.668	5.4778	10	10	61	60	89	190	98	59	39	36
334.062	5.5677	10	10	60	60	89	188	98	60	39	37
339.454	5.657567	10	11	60	60	89	190	98	60	39	37
344.85	5.7475	10	10	60	60	89	189	98	60	39	37
350.242	5.837367	10	10	60	60	89	188	98	60	39	37
355.636	5.927266	10	10	60	60	90	188	98	60	39	36
361.028	6.017134	0	0	31	34	125	228	119	90	4	16
366.424	6.107067	10	10	60	60	121	240	125	84	30	39
371.816	6.196933	10	10	60	60	119	242	125	83	29	39
377.212	6.286867	10	10	60	60	117	242	126	83	30	39
382.608	6.3768	10	10	59	60	119	244	125	83	30	39
388.004	6.466733	10	10	59	60	118	243	125	83	29	39

393.398	6.556634	10	10	59	60	119	243	125	83	29	39
398.79	6.6465	9	10	60	60	117	245	125	83	29	39
404.184	6.7364	11	10	60	60	118	244	126	83	30	39
409.508	6.825133	10	10	61	60	117	244	126	83	31	39
414.902	6.915033	10	10	60	60	118	244	125	83	30	39
420.3	7.005	10	10	60	60	119	244	125	83	30	39
425.694	7.0949	10	10	60	60	118	243	125	83	30	39
431.092	7.184867	10	10	60	60	118	243	125	83	29	39
436.412	7.273533	10	10	59	60	118	243	125	83	29	39
441.804	7.3634	10	10	60	60	118	242	125	83	30	38
447.198	7.4533	10	9	60	59	118	242	125	84	30	38
452.594	7.543233	10	10	60	60	118	242	125	83	30	38
457.99	7.633167	10	10	60	60	118	243	125	83	30	39
463.382	7.723033	10	10	60	60	118	243	125	83	30	39
468.776	7.812933	10	10	60	60	118	243	125	83	30	39
474.168	7.9028	10	11	60	60	118	242	125	84	30	39
479.564	7.992733	10	9	60	59	117	242	125	84	30	38
484.962	8.0827	37	49	102	108	92	193	96	57	39	36
490.356	8.1726	10	10	60	60	92	190	96	59	39	37
495.75	8.2625	10	11	60	60	90	189	97	60	39	37
501.144	8.3524	11	10	60	60	91	189	97	59	39	36
506.54	8.442333	10	10	60	60	92	189	96	59	39	36
511.932	8.5322	10	10	60	60	92	187	96	60	39	37
517.324	8.622066	11	10	61	60	91	187	96	60	39	37
522.72	8.712	10	10	60	60	92	187	96	60	39	37
528.114	8.8019	10	10	59	60	92	187	96	60	38	37
533.504	8.891734	11	10	61	60	91	187	96	60	39	36
538.9	8.981667	10	10	60	60	92	187	96	60	38	36
544.292	9.071533	10	10	60	60	92	187	96	60	39	37
549.688	9.161466	11	10	60	60	91	188	97	60	39	37
555.084	9.2514	10	10	60	60	92	188	96	60	39	37
560.48	9.341333	10	10	60	60	92	188	96	60	39	37
565.876	9.431266	10	10	59	60	92	188	96	60	38	37
571.27	9.521167	10	10	60	60	92	188	96	60	39	37

576.662	9.611033	10	10	60	60	92	188	96	60	39	37
582.056	9.700934	9	10	60	60	91	188	96	60	38	37
587.452	9.790867	10	10	60	60	92	188	96	60	38	37
592.848	9.8808	10	10	60	60	92	188	96	60	39	37

VITA ²

BIN WANG

Candidate for the Degree of
Master of Science

Thesis: AUTOMATIC CAMERA CALIBRATION USING PID AND FUZZY LOGIC
CONTROL

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Tianjin, China, March 20, 1966, the son of Liuxiang Li and Zhiyi Wang.

Education: Graduated from Tianjin No. 7 High School, Tianjin, China, in July, 1984; Received Bachelor of Science Degree in Mechanical Engineering from Harbin Institute of Technology in July, 1988; Completed requirements for the Master of Science Degree at Oklahoma State University in May, 1999.

Professional Experience: Graduate Research and Teaching Assistant, School of Mechanical and Aerospace Engineering, Oklahoma State University, from August, 1997, to December, 1998; Manager of Technology Development Department, Tianjin Wix Filter Corp. Ltd. from March, 1993, to August, 1997; Mechanical Engineer, Tianjin Wix Filter Corp. Ltd. from April, 1991, to March, 1993; Mechanical Engineer, Tianjin Automotive Industry Corp. from July, 1988, to April, 1991.