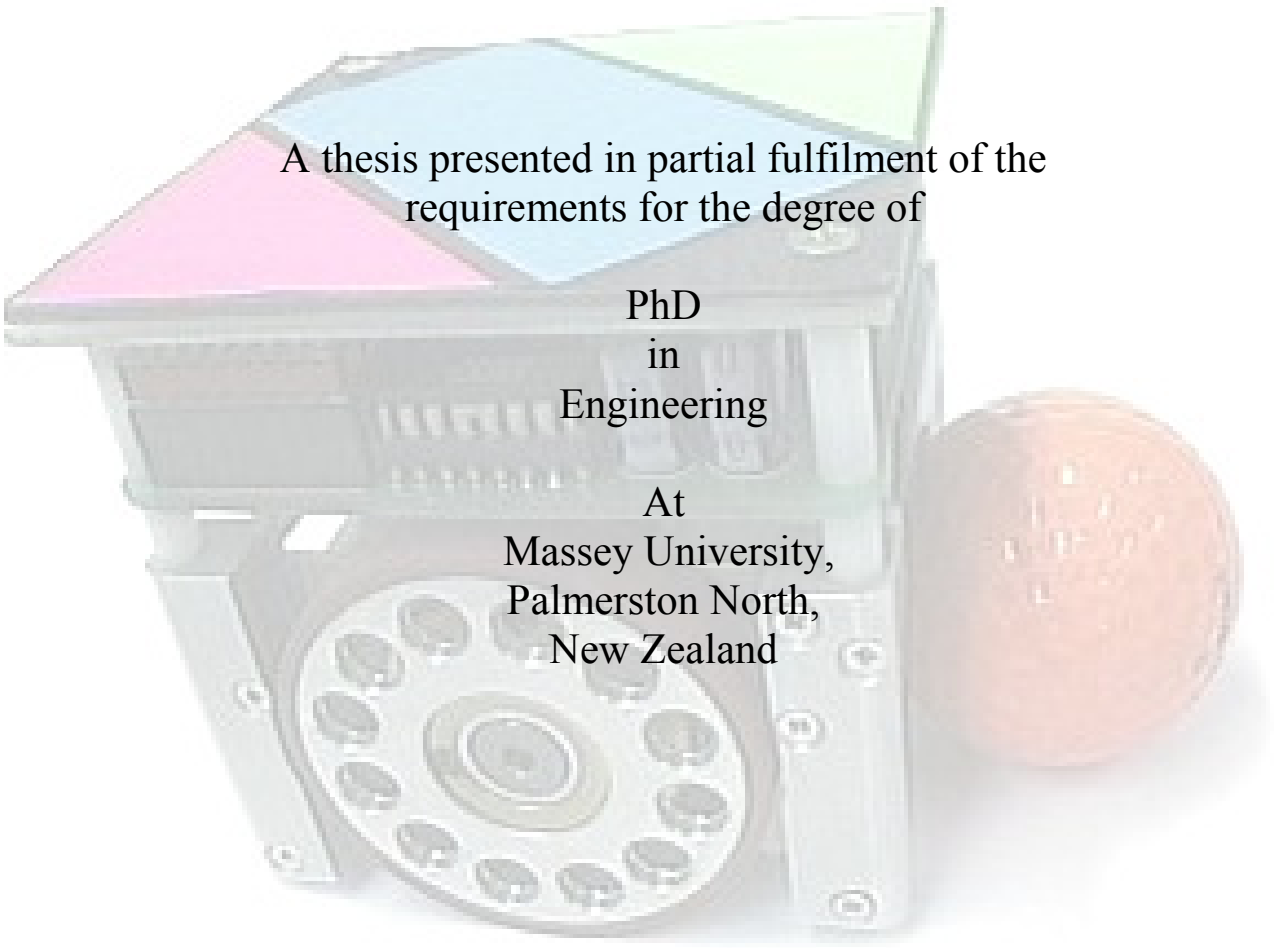
Autonomous Agents in a Dynamic Collaborative Environment



A thesis presented in partial fulfilment of the
requirements for the degree of

PhD
in
Engineering

At
Massey University,
Palmerston North,
New Zealand

Gourab Sen Gupta
2008

Abstract

The proliferation of robots in industry and every day human life is gaining momentum. After the initial few decades of employment of robots in the industry, especially the automotive assembly plants, robots are now entering the home and offices. From being pick-and-place manipulators, robots are slowly being transformed in shape and form to be more anthropomorphic. The wheeled robots are however here to stay for the foreseeable future until such time as artificial muscles, and efficient means to control them, are well developed.

The next phase of development of robots will be for the service industry. Robots will cooperate with each other to accomplish collaborative tasks to aid human life. They will also collaborate with human beings to assist them in doing tasks such as lifting loads and moving objects. At the same time, with the advancement of hardware, robots are becoming very fast and are capable of being programmed with more intelligence. Coupled with this is the availability of sophisticated sensors with which the robots can perceive the real world around them. Combinations of these factors have created many challenging areas of research.

Several factors affect the performance of robots in a dynamic collaborative environment. The research presented in this thesis has identified the major contributing factors, namely *fast vision processing*, *behaviour programming, predictive movement and interception control,* and *precise motion control*, that collectively have influence on the performance of robots which are engaged in a collaborative effort to accomplish a task. Several novel techniques have been proposed in this thesis to enhance the collective performance of collaborating robots.

In many systems, vision is used as one of the sensory inputs for the robot's perception of the environment. This thesis describes a new colour space and the use of discrete look-up-tables (LUT) for very fast and robust colour segmentation and real-time identification of objects in the robot's work space. A distributed camera system and a stereo vision using a single camera are reported. Advanced filtering has been applied to the vision data for predictive identification of the position and orientation of moving robots and targets, and for anticipatory interception control.

Collaborative tasks are generally complex and robots need to be capable of exhibiting sophisticated behaviours. This thesis has detailed the use of State Transition Based Control (STBC) methodology to build a hierarchy of complex behaviour. Behaviour of robots in a robot soccer game and features such as role selection and obstacle avoidance have been built using STBC.

A novel methodology for advanced control of fast robots is detailed. The algorithm uses a combination of Triangular Targeting Algorithm (TTA) and Proximity Positioning Algorithm (PPA) to position a robot behind an object aligned with a target. Various forms of velocity profiling have been proposed and validated with substantial test results.

The thesis ends by looking at future scenarios where robots and human beings will coexist and work together to do many collaborative tasks. Anthropomorphic robots will be more prevalent in future and teleoperation will gain momentum.

Throughout the thesis, the engineering applicability of proposed algorithms and architectures have been emphasised by testing on real robots.

Acknowledgements

In my quest for knowledge, I embarked on this journey of doing a PhD. The journey was exciting but, as expected, fraught with challenges. I would not have been able to overcome these challenges if not for the able guidance and tremendous support I received from numerous people.

I would like to sincerely thank my supervisors Prof. Serge Demidenko and Dr Chris Messom, who have given me immense help and guidance all along. They never ceased to encourage me and their support was awesome. The many hours that they have spent in directing my research and refining my papers for various journals and conferences are much appreciated. My apprenticeship as a researcher was all the more enjoyable because Prof. Demidenko and Dr Messom were not only competent supervisors, they were good friends too.

I have spent many hours with A/Prof Donald Bailey discussing various problems and solutions. I would like to thank him for his invaluable inputs and ideas for my research, especially the vision processing work. Many thanks to A/Prof. Subhas Mukhopadhyay for the company he has given me over innumerable cups of coffee; always goading me to accelerate my work and at times providing the welcome distraction from work and studies. Many thanks to Colin Plaw and Ken Mercer for their help in building the robot soccer platform complete with light fixtures and camera mounting unit, and for repairing battered robots and batteries many times.

Sincere thanks to Singapore Polytechnic for the financial support given to me to pursue this PhD, especially to Dr Dave Chong for his unflinching support for research.

I would also like to thank all my students who have worked with me on various projects which have enhanced my research, especially Kay Kidd, Robert Paddison, Jonathan Seal and Tim Brooks.

Last, but not the least, thanks to my family, especially my wife Mamta, for their kind support and for putting up with my absence from home for long periods. The sacrifices that they have made have allowed me to achieve my goal for which I am indebted to them.

Table of Contents

Abstract i
Acknowledgements iii
List of Figures ix
List of Tables xiv

1. Introduction 1

2. Cooperative Robotics 9

2.1 Cooperative Behaviour
2.2 Benefits of Cooperative Robotics
2.3 Cooperative Robotics Today
2.3.1 Cooperative Robotics for Entertainment– Static and Dynamic Systems
2.3.2 Multiple-Robot Path Planning for Collaboration
2.3.3 Centralized/Decentralized Control Architecture of Multi-Agent Systems
2.3.4 Communication Structures in Collaborative Tasks
2.3.5 Local and Global Vision Based Agents
2.4 Robot Soccer System: A Test-bed for Collaborative Robotics

3. Fast Image Capture and Vision Processing 23

3.1 Global Vision – Sources of Error
3.1.1 Separate Processing of Odd and Even Scan Fields of an Interlaced Bit-Mapped Image
3.1.2 Variation of Light Intensity
3.1.3 Inherent Sensor Noise
3.2 Experimental Hardware Setup
3.3 Colour Segmentation, Area Thresholding and Blob Merging
3.4 Interrupt Based Multi-Buffered Image Capture
3.5 Full Tracking vs. Incremental Tracking
3.6 Fast Access Colour Look-Up-Table
3.6.1 Limitations of Using RGB Colour Space
3.6.2 Defining YUV Thresholds
3.6.3 Membership Testing
3.6.4 One-Dimensional Colour Look-Up-Table
3.6.5 Posting the Look-Up-Table
3.6.6 Inspecting the Look-Up-Table

3.7 Discrete YUV Look-Up-Table

3.7.1 Populating the Discrete YUV Look-Up-Table

3.7.2 Testing Colour Class Membership

3.7.3 Colour Space Transformation

3.7.4 A New Colour Space

3.7.5 Experimental Results and Discussion

3.8 Distributed Vision Processing

3.8.1 Distributed System Architecture

3.8.2 Integrating the Distributed System

3.8.3 Role of the Strategy Server

3.8.4 Test Results and Performance Tuning

3.9 Catadioptric Stereo Vision for Robotic Application

3.9.1 Review of Stereo Imaging

3.9.2 System Design

3.9.3 Implementation and Calibration

3.10 Discussion

4. Collaborative Operation Strategy: State Transition Based Control (STBC) 65

4.1 States: The Fundamental Component of STBC

4.2 State Transition Diagram

4.3 Maintaining State with Hysteresis

4.4 Hierarchy of States

4.5 Role Selection in Robot Soccer

4.5.1 Defensive Behaviour

4.5.2 Offensive Behaviour

4.5.3 State Transition Based Shoot Action

4.6 Auto Placement of Robots

4.7 The Formation and Marching Problem - Implementation

4.7.1 State Transition Diagram to Turn Robots

4.7.2 Moving the Robot Contingent from One Place to Another

4.7.4 Synchronised About Turn – Problems and Proposed Solution

4.8 Obstacle Avoidance Using STBC Technique

4.9 Discussion

5. Filtering and its Impact on Predictive Movement and Interception Control ……..…… 87

5.1 Object Interception

5.2 Control Algorithm for Object Targeting

5.3 Application of *g-h* Tracking Filter

5.4 Prediction of an Object's Position

5.5 Experimental Results

5.5.1 Stationary Object

5.5.2 Robot Motion Trajectory

5.5.3 Intercepting or Blocking a Moving Object

5.6 Discussion

6. Advanced Motion Control Algorithm for Fast Mobile Robots …... 129

6.1 Classical PID Control for Robot Positioning and Orientation

6.1.1 Robot Orientation Control

6.1.2 Gain Scheduled Proportional Control

6.1.3 Effect of Scaling Ka_p on the System Response

6.1.4 Fine-Tuning the Proportional Gain, Ka_p

6.1.5 The Effect of Derivative Gain, Ka_d

6.1.6 Combining *g-h* Filter and PD Control

6.2 Classical PD Control for Robot Position

6.2.1 Robot Position Control

6.2.2 Multi-tiered Kd_p for Robot Positioning

6.3 Advanced Robot Positioning Algorithm

6.3.1 Pre-processing Inputs

6.3.2 Re-targeting System

6.4 Triangular Targeting Algorithm

6.5 Proximity Positioning Algorithm

6.6 Velocity Profiling

6.6.1 Angular Velocity Profiling

6.6.2 Linear Velocity Profiling

6.6.3 Cross-Effects Profiling

6.7 Experimental Results

6.7.1 Angular Motion Function

6.7.2 Linear Motion Function

6.7.3 Cross-Effects Optimization

6.7.4 Testing Triangular Targeting Algorithm

6.8 Discussion

7. Summary and Future Research ……..… 165

8. References ……..…… 171

9. List of Selected Relevant Publications by the Author..…….. 179

Appendix A: Program listing of OOP implementation of STBC-based robot behaviour (goalkeeper)

Appendix B: Program listing of OOP implementation of *g-h* filter

Appendix C: System for Real-Time Measurement of Robot Movements and Analysis of Collaborative Tasks

List of Figures

Figure 1.1	Industrial robots collaborating on production floor ……..	1
Figure 1.2	A battle-field robot carrying a soldier.....	2
Figure 1.3	*enon*, Fujitsu's service robot.....	2
Figure 1.4	A sentry robot with local camera vision system.....	4
Figure 2.1	Cooperative task execution with multiple robots.....	10
Figure 2.2	Multiple robots cooperating to lift and move a load.....	11
Figure 2.3	TALON II robots used by US army for detection of landmines.....	11
Figure 2.4	A bomb disposal robot.....	12
Figure 2.5	A material handling robot on the shop floor……..	13
Figure 2.6	An autonomous robot hauling items for packing…….....	13
Figure 2.7	Robotic Band (Photo courtesy of SEEE, Singapore Polytechnic).....	14
Figure 2.8	Robot with a local vision - the camera can tilt and pan to detect the target.....	17
Figure 2.9	Robot with a local omni-directional vision using parabolic mirror cone.....	18
Figure 2.10	Multi-agent collaborative system (robot soccer) using global vision.....	18
Figure 2.11	Robot Soccer System setup using global vision……..	20
Figure 2.12	Bi-wheeled robot used in experiments.....	21
Figure 3.1	Software structure of a multi-agent robotic controller using global vision.....	24
Figure 3.2	Bit-map of an interlaced image.....	25
Figure 3.3	Image capture card in *off-screen capture mode*.....	27
Figure 3.4	Colour jacket for identification of robot.....	28
Figure 3.5	Colour jacket for improved accuracy of orientation.....	28
Figure 3.6	The binary image.....	29
Figure 3.7	Image after the first pass.....	30
Figure 3.8	Image after the second pass.....	30
Figure 3.9	Multi-buffered image capture.....	31
Figure 3.10	Tracking window centred on last known position of object.....	32
Figure 3.11	Tracking window centred on predicted position of object.....	32
Figure 3.12	Robot path and the incremental tracking window.....	34
Figure 3.13	Convex RGB colour subspace.....	35
Figure 3.14	Relative positioning of YUV and RGB colour space.....	36
Figure 3.15	Posting the LUT with colour ID.....	37
Figure 3.16	Inspecting the LUT.....	38
Figure 3.17	YUV thresholds for Colour 3.....	39
Figure 3.18	Colour representations in the LUT element.....	39
Figure 3.19	YUV LUT populated for Colour 3 (not to exact scale).....	40
Figure 3.20	Posting the LUT with colour ID.....	41
Figure 3.21	Testing colour class membership.....	41
Figure 3.22	Comparison of incremental tracking time.....	44
Figure 3.23	Comparison of full tracking time.....	44
Figure 3.24	Chart showing the performance of the four systems for incremental tracking.....	45
Figure 3.25	Chart showing the performance of the four systems for full tracking.....	45
Figure 3.26	Summary of improvements in processing speed (in %) achieved by Y'U'V' for incremental tracking.....	45
Figure 3.27	Summary of improvements in processing speed (in %) achieved by Y'U'V' for full tracking.....	46
Figure 3.28	Overall system architecture of distributed vision processing.....	48

Figure 3.29	Overlap regions (region for full tracking).....	48
Figure 3.30	Image of the left half of the field.....	50
Figure 3.31	Image of the right half of the field.....	50
Figure 3.32	Two camera stereo system.....	52
Figure 3.33	Close and far objects seen through a parallel stereo system with two cameras.....	52
Figure 3.34	Panoramic camera system.....	53
Figure 3.35	Single camera moving through known space system.....	54
Figure 3.36	Map of the images on the sensor.....	54
Figure 3.37	Catadioptric stereo vision system.....	55
Figure 3.38	The catadioptric system showing the position of mirrors, pinhole and the CCD.....	56
Figure 3.39	Moving an object point by *d* changes the disparity between the images by λ.....	57
Figure 3.40	Effect of the lens focal length on the field of view for 6.4 mm x 4.4mm sensor with a working depth of 2 meters.....	59
Figure 3.41	Effect of the distance of the pinhole from the vertex of the inside mirrors, on the system size. The shaded region is where the system is self-occluding.....	59
Figure 3.42	The lens and mirror arrangement to show how the size of the lens limits the distance between the pinhole and the vertex of the inside mirrors.....	60
Figure 3.43	The completed catadioptric system.....	61
Figure 3.44	The calibration grid seen through the vision system.....	61
Figure 3.45	Measuring average tracking time……	63
Figure 3.46	Dialog box to adjust the thresholds for colour classes……	64
Figure 3.47	Dialog box to set minimum patch size for colour classes……	64
Figure 4.1	Goal area, defensive and offensive regions of a robot soccer field.....	66
Figure 4.2	State transition diagram of goalkeeper action.....	67
Figure 4.3	State transition diagram of the *Clear Ball* operation.....	70
Figure 4.4	Boundaries for robot soccer role selection.....	71
Figure 4.5	State transition diagram for role selection.....	71
Figure 4.6	Boundaries for defensive behaviour.....	73
Figure 4.7	State transition diagram of the defensive behaviour.....	73
Figure 4.8	Boundaries for offensive behaviour.....	74
Figure 4.9	State transition diagram of the offensive behaviour.....	74
Figure 4.10	State transition diagram of the *CShoot* action.....	75
Figure 4.11	State transition diagram for auto placement of robots.....	76
Figure 4.12	State transition diagram to turn robots on the spot.....	78
Figure 4.13	Initial and final position/orientation of the robot contingent.....	79
Figure 4.14	State transition diagram to position the robot contingent.....	79
Figure 4.15	Two robots turning in opposite directions during an about turn operation.....	81
Figure 4.16	State transition diagram for a 2-step turn.....	81
Figure 4.17	State transition diagram for obstacle avoidance.....	82
Figure 4.18	OOP implementation of a '*state transition method'*.....	84
Figure 4.19	OOP implementation of an *'action method'*.....	85
Figure 4.20	Graphical User Interface (GUI) for tuning robot behaviour……	85
Figure 5.1	Calculating the ball interception position.....	89
Figure 5.2	Flow chart for predicting the point of interception.....	90
Figure 5.3	Calculating wheel velocities for stationary object targeting.....	91
Figure 5.4	Object position in odd and even scan fields.....	93
Figure 5.5	Simulated path using original and modified control algorithms.....	95
Figure 5.6	Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.8, h=0.2*).....	97

Figure 5.7	Plot of a stationary ball's X-position (*g=0.8, h=0.2*)…….....	97
Figure 5.8	Plot of the difference between actual and filtered X-position (*g=0.8, h=0.2*).....	98
Figure 5.9	Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.6, h=0.4*).....	98
Figure 5.10	Plot of a stationary ball's X-position (*g=0.6, h=0.4*).....	99
Figure 5.11	Plot of the difference between actual and filtered X-position (*g=0.6, h=0.4*).....	99
Figure 5.12	Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.1, h=0.9*).....	100
Figure 5.13	Plot of a stationary ball's X-position (*g=0.1, h=0.9*).....	100
Figure 5.14	Plot of the difference between actual and filtered X-position (*g=0.1, h=0.9*).....	101
Figure 5.15	Filter evaluation based on measurement of position (X coordinate).....	102
Figure 5.16	Filter evaluation based on measurement of position (Y coordinate).....	103
Figure 5.17	Penalty shot taken by the opponent striker.....	104
Figure 5.18	Plot of variation of calculated goal keeper position (dx) for a penalty shot.....	104
Figure 5.19	Plot of calculated goalkeeper position (dy) for a penalty shot.....	105
Figure 5.20	Frame-to-frame angle variation of a stationary robot (*g=0.8, h=0.2*).....	106
Figure 5.21	Plot of a stationary robot's angle (*g=0.8, h=0.2*).....	107
Figure 5.22	Plot of the difference between actual and filtered angles (*g=0.8, h=0.2*).....	107
Figure 5.23	Frame-to-frame angle variation of a stationary robot (*g=0.7, h=0.3*).....	108
Figure 5.24	Plot of a stationary robot's angle (*g=0.7, h=0.3*).....	108
Figure 5.25	Plot of the difference between actual and filtered angles (*g=0.7, h=0.3*).....	109
Figure 5.26	Frame-to-frame angle variation of a stationary robot (*g=0.6, h=0.4*).....	109
Figure 5.27	Plot of a stationary robot's angle (*g=0.6, h=0.4*).....	110
Figure 5.28	Plot of the difference between actual and filtered angles (*g=0.6, h=0.4*).....	110
Figure 5.29	Frame-to-frame angle variation of a stationary robot (*g=0.1, h=0.9*).....	111
Figure 5.30	Plot of a stationary robot's angle (*g=0.1, h=0.9*).....	111
Figure 5.31	Plot of the difference between actual and filtered angles (*g=0.1, h=0.9*).....	112
Figure 5.32	Filter evaluation based on measurement of angle.....	113
Figure 5.33	Filter evaluation based on measurement of angle (*g=0.6, h=0.0 to 1.0*)……	113
Figure 5.34	Robot motion trajectories with and without filtering (*scaleFactor* = 0.8).....	115
Figure 5.35	Robot motion trajectories with and without filtering (*scaleFactor* = 1.0).....	115
Figure 5.36	Robot motion trajectories with and without filtering.....	117
Figure 5.37	Robot motion trajectories with and without filtering at higher speeds.....	119
Figure 5.38	Goalkeeper intercepting a fast moving target.....	120
Figure 5.39	Goalkeeper intercepting a slow moving target.....	120
Figure 5.40	Goalkeeper intercepting a target on rebound.....	121
Figure 5.41	Robot intercepting a moving target (target moving away from the robot).....	122
Figure 5.42	Robot intercepting a moving target (target moving towards the robot).....	122
Figure 5.43	Failure to intercept a moving target without filtering/prediction.....	123
Figure 5.44	Variation of ball position (*dy*) for *g=0.8, h=0.2* in separate odd-even scans……	125
Figure 5.45	Variation of ball position (*dy*) for *g=0.8, h=0.2* in full scan mode.....	126
Figure 5.46	Variation of ball position (*dy*) for *g=0.6, h=0.4* in separate odd-even scans.....	126
Figure 5.47	Variation of ball position (*dy*) for *g=0.6, h=0.4* in full scan mode.....	127
Figure 6.1	Block diagram of a PID controller for motor position control.....	130
Figure 6.2	Orientation control - robot's current and final direction.....	131
Figure 6.3	Multi-tiered proportional gains, Ka_p.....	133
Figure 6.4	Effect of scaling proportional gains, Ka_p.....	135
Figure 6.5	Fine-tuning proportional gain, Ka_p.....	135
Figure 6.6	Tweaking proportional gains for faster system response.....	136
Figure 6.7	Gain scheduled derivative gain, ka_d.....	136

Figure 6.8	Filtering with derivative control	137
Figure 6.9	Filtering without derivative control	137
Figure 6.10	Movement control - robot's current and final position/orientation	138
Figure 6.11	Determining 'direction factor' *d* for robot movement	139
Figure 6.12	Multi-tiered proportional gains, Kd_p	140
Figure 6.13	Effect of increasing proportional gain, Kd_p	141
Figure 6.14	Robot positioning process	142
Figure 6.15	Pre-processing inputs	142
Figure 6.16	*Turn-move-turn* approach to reach a target	143
Figure 6.17	Selecting the most appropriate re-targeting algorithm	144
Figure 6.18	Triangular Targeting Algorithm. Robot's path towards the target	145
Figure 6.19	Various angles and distances for TTA	146
Figure 6.20	Relationship between the maximum triangle height and the distance error	147
Figure 6.21	Exponential profiling of triangle height	148
Figure 6.22	Effects of curvature on robot movement	148
Figure 6.23	Triangle heights for different robot positions	148
Figure 6.24	PPA algorithm at work	149
Figure 6.25	Exponential profiling of angular velocity	150
Figure 6.26	Exponential profiling of linear velocity	151
Figure 6.27	Exponential cross-effect curve	152
Figure 6.28	Evaluation of angular movement	153
Figure 6.29	Evaluation of linear movement	154
Figure 6.30	Evaluation of cross-effect curve	155
Figure 6.31	Optimized 'fuzzy' cross-effect curves	156
Figure 6.32	Relating linear velocity to angular velocity (velocity clamping curve)	156
Figure 6.33	Robot starting position (20, 25, 90^o), end posture (115, 65, 0^o)	156
Figure 6.34	Robot starting position (75, 25, 90^o), end posture (115, 65, 0^o)	157
Figure 6.35	Robot starting posture (115, 25, 90^o), end posture (110, 65, 0^o)	157
Figure 6.36	Robot starting posture (135, 25, 90^o), end posture (110, 65, 0^o)	158
Figure 6.37	Robot starting posture (20, 105, 90^o), end posture (115, 65, 0^o)	158
Figure 6.38	Robot starting posture (75, 105, 90^o), end posture (115, 65,0^o)	159
Figure 6.39	Robot starting posture (115, 105, 90^o), end posture (110, 65, 0^o)	159
Figure 6.40	Robot starting posture (135, 105, 90^o), end posture (110, 65, 0^o)	160
Figure 6.41	Robot starting posture (20, 25, 90^o), end posture (130, 110, 0^o)	160
Figure 6.42	Robot starting posture (115, 25, 90^o), end posture (130, 110, 0^o)	161
Figure 6.43	Robot starting posture (20, 110, 0^o), end posture (130, 110, 0^o)	161
Figure 6.44	GUI for tuning TTA parameters	163
Figure 6.45	Performance evaluation of long and short range TTA	164
Figure 7.1	Factors significantly affecting collaborative behaviour in a vision based domain	166
Figure C.1	System hardware architecture for analysis and control of collaborative tasks	193
Figure C.2	A screenshot of the raw data viewer	195
Figure C.3	A screenshot of the game replay simulator	196
Figure C.4	A screenshot of the game statistics screen	197
Figure C.5	Work done by the robots, shown using 3D line graph	198
Figure C.6	Total work done, as a percentage, by the robots in a game, shown in a pie chart	199
Figure C.7	Distance travelled by the robots, shown in a stacked bar chart	199

List of Tables

Table 3.1	Summary of vision test results for different look-up-tables.....	43
Table 4.1	State transition conditions for goalkeeper behaviour…….....	68
Table 4.2	Modified state transition conditions for goalkeeper behaviour incorporating sensor hysteresis factor……....	69
Table 4.3	Modified state transition conditions for goalkeeper behaviour incorporating sensor hysteresis factor and behavioural hysteresis factor……....	69
Table 4.4	State transition conditions of Clear Ball operation of goalkeeper…….....	70
Table 4.5	State transition conditions for role selection…….....	72
Table 4.6	State transition conditions for *CShoot* action…….....	76
Table 4.7	State transition conditions for auto placement of robots…….....	77
Table 4.8	State transition conditions to turn robots on the spot…….....	78
Table 4.9	State transition conditions to position the robot contingent…….....	80
Table 4.10	State transition conditions for a 2-step turn…….....	82
Table 5.1	Filter evaluation and comparison data for ball position (X coordinate).....	101
Table 5.2	Filter evaluation and comparison data for ball position (Y coordinate).....	102
Table 5.3	Filter evaluation and comparison data for angle variation.....	112
Table 5.4	Filter evaluation and comparison data for angle variation (*g=0.6, h=0.0 to 1.0*).....	114
Table 5.5	Actual distance travelled by the robot (*scaleFactor* = 1).....	116
Table 5.6	Analysis of actual distance travelled by the robot.....	117
Table 5.7	Analysis of positional error of the robot at the destination.....	118
Table 5.8	Actual distance travelled by the robot (*scaleFactor* = 1.2).....	119
Table 5.9	Evaluation matrix for two filters for different scan modes.....	127
Table 6.1	Multi-tiered proportional gains used for experimentation.....	134
Table 6.2	Performance matrix for TTA: start posture (135, 105, 90^o), end posture (110, 65, 0^o)	162
Table 6.3	Performance matrix for TTA: start posture (20, 25, 90^o), end posture (130, 110, 0^o)..	163

1 Introduction

Autonomous agents working in groups and operating collaboratively are gaining wide applications in industry, defence, personal assistance and entertainment. There are numerous examples of collaborative tasks being performed by robots on the *production floor* [1], reconnaissance in the *battle field* [2] and other areas. A major thrust in future research will be in the development of *service robots* for personal care and to assist human co-workers at the workplace [3]. The Japanese government invests over US$35 million a year in robotic technology so that Granny can have a hot meal and get down the stairs! Another highly visible field of the application of collaborative multi-agent systems is the *entertainment industry* [4]. Groups of robots can entertain by producing music, dancing, playing games etc.

Figure 1.1 shows two KUKA industrial robots collaborating to lift and place a metal sheet (www.euclidlabs.it).

Figure 1.1 Industrial robots collaborating on production floor

The Sydney Morning Herald newspaper reported on February 18, 2005 that the Pentagon is spending US$161 billion on a program to build heavily-armed robots for the battlefield in the hope that future wars will be fought without the loss of its soldiers' lives. The scheme, known as Future Combat Systems, is the largest military contract in American history. The ultimate aim is to take members of the armed forces out of harm's way. They would be replaced by robots capable of hunting in groups and killing

enemies. Figure 1.2 shows a BEAR robot, meant for use in battle field, carrying a soldier (www.defensetech.org/images/battlefield-bear.jpg)

Figure 1.2 A battle-field robot carrying a soldier

Fujitsu Laboratories started limited sales of the service robot "enon" in November 2005 for task support. *Enon* is an advanced practical-use service robot that can assist in tasks such as providing guidance, escorting guests, transporting objects and security patrolling, while being linked to a network. It is designed to support individuals in offices or commercial establishments. Figure 1.3 shows the *enon* robots (www.fujitsu.com).

Figure 1.3 *enon*, Fujitsu's service robot

If a robot is intended to execute even very common tasks, such as grasping and lifting an object, a multitude of sensors and actuators will be needed, resulting in a large sized robot. However, trying to reduce the size of a robot by enclosing many components together in a small unit will result in a significant problem at the time of maintenance. A mechanically simple robot will obviously have lesser ability. Instead of using a single complex robot in such a difficult situation, it is considered more useful and easier to employ multiple simple robots and realize the desired task with their cooperation [5].

Controlling dynamic, multi-agent collaborative systems is a challenging problem by itself. However, the complexity of the task is further increased in a dynamically changing environment and can be dramatically aggravated when the agents' operation is carried out in hostile conditions with competing opponents.

The field of multi-agent robotics has recently reached a level of maturity in that systems are beginning to transition from proof-of-concept laboratory environments to deployed real-world systems. This trend is mainly performance-driven since multi-robot systems offer a number of advantages and additional capabilities over their single-robot counterparts, including redundancy, increased spatial coverage and throughput, flexible reconfigurability, spatially diverse functionality, and the fusing of physically distributed sensors and actuators. Applications in which these capabilities constitute enabling technologies range from remote and in situ sensing to the physical manipulation of objects, and the domains for such applications include land, sea, air and space. Despite remarkable research developments in the area, numerous technical challenges remain that must be overcome in order to field cost-effective multi-robot systems. These challenges include inter-robot communications, relative and absolute position sensing and actuation, control paradigms appropriate to real-time multi-agent systems, the fusion of distributed sensors and actuators, man-machine interfaces allowing efficient human direction and supervision of these systems, effective reconfiguration of the system's functionality, and design approaches supporting the economical production of such systems.

In the last decade robot soccer has been actively employed as a test bed for intensive research and development of multi-agent control systems [6]. The system uses visual (or other sensor) information about the positions of the agents under control (mobile robots of a team), the fixed and mobile targets (a goal and a ball), and the competing agents (opposition collaborative robotic team). It employs special adaptive algorithms maximising the effect of team cooperation as well as optimising the actions of each individual agent in reaching overall superiority over the opponents.

In order to achieve superiority over the actively competing opponents, a complex (system) approach has to be taken in enhancing and advancing the system. This entails

further improvements in those key components of the system, where the effect of the innovation would have the most beneficial outcome on its overall performance. The enhancement of the performance of the multi agent system in a dynamically changing and competitive environment has been the ultimate goal of the research presented in this thesis.

The research is initiated with the analysis of the entire multi-agent collaborative system. This is done in order to determine the key elements of the system whose advancement, as mentioned earlier, would be of paramount importance in achieving the overall improved performance. Among such factors are fast image capture and accurate processing, predictive movement and interception control, and collaborative operation strategy. High speed and reliable communication with the agents is another factor that influences the system performance.

Because of the dynamics and high complexity of the multi-agent collaborative systems, as well as manoeuvrability and speed of its agents, accurate path planning and prediction of moving targets has gained special importance. Several techniques have been proposed for path planning including the use of genetic algorithms [7] and fuzzy logic [8]. Coupled to the path-planning problem is the task of obstacle avoidance in a changing environment. The path plans must be dynamically updated to reflect the changes in the environment. This means that the paths must be planned and updated in real time. Obviously, this puts additional pressure on the system's data processing resources.

Figure 1.4 A sentry robot with local camera vision system

In collaborative systems, vision sensors are often mounted on the robots and are the robots' main sensor system to perceive the real world around them [9]. A machine vision system can make a robot much more versatile by allowing it to deal with variation of object position and orientation. The purpose of a machine vision system is to produce a symbolic description of what is being imaged. This description may then be used to direct the interaction of a robotic system with its environment. The sentry robot shown in Figure 1.4 shows the use of camera and vision system to navigate with obstacle avoidance.

The vision system captures and processes the image at a number of frames per second. Real-time image processing techniques such as Run Length Encoding (RLE) are needed to satisfy the required sample time [10]. Improving the image capture and processing is of very high importance, especially for very fast moving robots.

Robotic interception of moving objects in industrial settings has gained much attention in the past decade. Among the most efficient approaches is the APPE (Active Prediction, Planning and Execution) system [11]. The key feature of it is the ability of a robot to perform a task autonomously without complete *a priori* information. However, there is room for further improvement there. One of the possible directions is the use of Kalman predictive filtering [12, 13] that has proved very useful in autonomous and assisted navigation as well as guidance systems, radar tracking of moving objects, etc. The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown. Kalman filtering is also a computationally efficient algorithm, which generates an optimal estimate from a sequence of noisy observations.

Another area of development is the collaborative strategy for jointly operating agents. In the case of robot soccer the robot players have to exhibit basic actions like positioning at designated locations, moving to the ball or blocking opponents, turning to desired angles, circling around the ball and shooting the ball into opponent's goal [14].

The research presented in this thesis concentrates mainly on the above mentioned high-importance components of the system:

- real-time image processing,
- filtering of vision data
- development of strategy for cooperative behaviour
- advanced motion control algorithms,

- Teleoperated anthropomorphic robotic arm to assist collaborative tasks
- real-time analysis of collaborative tasks.

The research contribution for enhancing the real-time image processing includes several novel techniques, including extension of the YUV colour space for very fast and efficient colour classification. The proposed method will pave the way for the future implementation of adaptive colour thresholds for detecting fast moving objects in varying light conditions. This is of great importance in vision based cooperative robotics.

The application of a tracking filter to mobile robot control, particularly the predictive tracking of targets and optimal trajectory control of robots for interception, is a signification achievement of this research. The system successfully predicts the trajectory of the target through the robot's workspace. The robot motion to reach the target in a desired orientation is optimally planned and executed. The system is robust even in a noisy environment. A novel Triangular Targeting Algorithm (TTA) for accurate positioning of an agent is reported in this thesis.

An agent's behaviour programming and the development of a strategy for collaborative actions are rather complex tasks. State Transition Based Control (STBC) methodology has been successfully used to build a hierarchy of complex behaviours. It has also been employed for obstacle avoidance. The effectiveness of the STBC technique has been tested using the robot soccer system and has been reported in the thesis.

In this age of distributed processing, agents are often remote controlled and monitored using local area and wide area networks, including the world-wide web (WWW). Remote monitoring and control of agents via Wide Area Network (WAN) protocols provide a challenging environment due to the uncertain latency of the communication channel and its bandwidth. A system has been developed and tested which allows collaborating robots to be 'seen' on a remote terminal which is physically far away from the location where the task is in progress. Using the remote terminal, the parameters of the STBC controller for the task may also be changed. This is essential in an environment where robots not only cooperate with each other to accomplish a task but also collaborate with the human beings.

The efficiency of various algorithms which have been proposed for the collaborative tasks has been tested by real-time data gathering and analysis. A complete system has been developed for this.

As robots get increasingly sophisticated, we'll expect them to provide help with our daily lives in increasingly complicated ways while working closely with humans and the objects in their environment. To do so, the robots will need to perceive the world

and understand them well enough to perform the tasks desired. Achieving this high level of sophistication presents many challenging research problems for robot scientists.

The rest of the thesis is organised in the following manner-

Chapter 2 gives a general overview of the current state of cooperative robotics. It details the importance of multi-agent systems used to accomplish tasks by collaboration and cooperation amongst robots. The review of centralised/decentralised control architecture, communication architectures for collaborative behaviour, and systems based on local/global vision systems is presented.

Chapter 3 commences by explaining the various sources of errors in a global vision system, which have adverse effects on the performance of a robot. The colour thresholding technique used for the identification of objects, such as robots and obstacles, and how a look-up table (LUT) is efficiently used for speeding up the identification process is enumerated. A novel technique, which uses discreet and much smaller look-up tables for a new YUV-alike colour space is presented in detail. The efficiency and usefulness of this technique have been extensively tested for incremental and full scanning and the results are detailed in this chapter. An architecture for distributed vision processing, using multiple cameras, is explained. The use of stereo imaging is essential in collaborative robotics for depth perception and obstacle avoidance. The design and implementation of a catadioptric stereo vision system, using a single camera, is also presented.

Chapter 4 is devoted to explaining the state transition based control (STBC) methodology for building robot behaviour and strategy for collaboration. The ease with which complex behaviours can be built, using a hierarchy of states, is illustrated using the robots of a soccer system as test objects. In the realm of entertainment robotics, this technique has been successfully employed to solve the problem of pattern formation and synchronised movement of a contingent of robots. The same technique can also be used for simple obstacle avoidance. These are detailed in this chapter.

Chapter 5 deals with the predictive movement and interception control of robots. The control algorithm for object targeting is explained first. The details of the tracking filter, which has been used to improve the accuracy of prediction of a moving target, have been presented. Extensive test results have been included.

Chapter 6 details a novel Triangular Targeting Algorithm (TTA) which has been proposed and implemented to control fast mobile robots. In conjunction with the Proximity Positioning Algorithm (PPA), the robot is able to approach the target from behind and align itself with the target. Various profiles for angular velocity, linear velocity and cross-effects have been presented in detail. This chapter also includes the

experimental results of the angular and linear motion functions, triangular profiling and cross-effect optimisation.

Chapter 7 summarises the research work done and the results achieved. Future direction of research in multi-agent collaborative robotics is briefly discussed.

Appendix A lists the class outline of the STBC based goalkeeper behaviour. These supplements the material presented in chapter 4.

Appendix B outlines the class structure of a *g-h* filter to supplement the material presented in chapter 5.

Appendix C explains in detail the software system that has been developed for real time data collection and analysis of collaborative tasks. The software incorporates a raw-data viewer, task replay simulator and a task statistics calculator. The various graphical outputs of the system are also presented. The software has proven to be very useful in recording and analysing data to test the efficacy of the various algorithms reported in this thesis.

2 Cooperative Robotics

Systems composed of multiple autonomous mobile robots, exhibiting cooperative behaviour, are being constructed, with an aim to study such issues as group architecture, path planning, obstacle avoidance, communication architecture, cooperation methodologies, artificial learning and intelligent control [15]. Cooperative mobile robotics is a highly interdisciplinary field and the challenges of dealing with multiple robots are very significant indeed.

Multi-agent centralized systems have disadvantages that make them unsuitable for large-scale integration, including high reliance on centralized communication, high complexity, lack of scalability, and high cost of integration. The use of distributed intelligence system technologies avoids these weaknesses. Distributed intelligence systems are based on the use of cooperative agents, organized in hardware or software components, that independently handle specialized tasks and cooperate to achieve system-level goals and achieve a high degree of flexibility. By distributing the logistic and strategic requirements of a system, it is possible to achieve greatly improved robustness, reliability, scalability, and security. Key to achieving these benefits is the use of holonic system technologies that establish a peer-to-peer environment to enable coordination, collaboration, and cooperation within the network. Such systems require both hardware and software components.

2.1 Cooperative Behaviour

Collective behaviour generically denotes any behaviour of agents in a system having more than one agent. *Cooperative behaviour* has been defined in many different ways.

1. "*Given some task specified by a designer, a multiple-robot system displays cooperative behaviour if, due to some underlying mechanism (i.e., the "mechanism of cooperation"), there is an increase in the total utility of the system"* [15]
2. "*joint collaborative behaviour that is directed towards some goal in which there is a common interest or reward*" [16]
3. "*a form of interaction, usually based on communication*" [17]
4. "*joining together for doing something that creates a progressive result such as increasing performance or saving time*" [18].

While simple *collective behaviour* may not always necessarily be productive, *cooperation* must be with a purpose and there must be some gain like increase in efficiency, reduction of complexity, adaptability etc.

2.2 Benefits of Cooperative Robotics

The use of multiple robots offers several advantages over single robots. Tasks, which are too difficult for a single robot to accomplish, may be successfully done by task-decomposition while using multiple robots in many cases. Building and using several simple robots may be faster, more cost effective and offer more flexibility over the single robot option. It can also add a dimension of fault-tolerance – if one or more robot fails, the task may still proceed with the help of others. Multiple-robots may be easily reconfigured and adapted for different operating environments and tasks.

Figure 2.1 shows an object being lifted and manipulated by the cooperative effort of a few robots.

Figure 2.1 Cooperative task execution with multiple robots

Cooperative robotics offers scope to research in many areas, including but not limited to

- Design principles (hardware and software) of autonomous agents with a view to effective cooperation
- Multi-agent collaboration strategy, including real-time reasoning and decision-making, to accomplish a common task (*mechanism of cooperation*)
- Sensors and sensor-fusion including vision sensors and image processing
- Reactive behaviour
- Intelligent robot and motion control (for wheeled robots)
- Communication architecture and constraints in group behaviour
- Resource allocation and conflict management

and a number of others.

2.3 Cooperative Robotics Today

The last two decades have witnessed a rapid rise in interest in collaborative robotics. While software simulation has been an important (and often the preferred) way of studying collaborative behaviour, hardware implementations are increasingly being realised in recent times. Simulation platforms have helped to investigate underlying theories of cooperation albeit with simplified assumptions, at times to establish a successful system. Such simplistic assumptions often ignore the difficulties of the corresponding hardware implementation. As such, there are few real world examples of collaborative robotic applications involving a substantially large number of robots. Technological constraints and problems encountered in the sphere of collaborative robotics are several times more than that seen in single-robot systems.

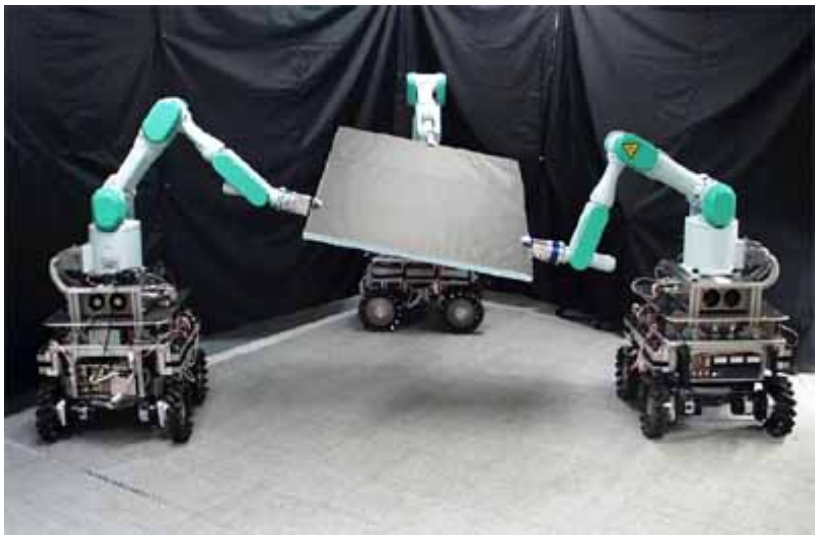


Figure 2.2 Multiple robots cooperating to lift and move a load

Figure 2.3 TALON II robots used by US army for detection of landmines

Figure 2.2 shows a group of robots cooperating to lift and move a load. Figure 2.3 shows the TALON robot (http://www.defenseindustrydaily.com) which has been used, often in groups, by the U. S. army combat engineers handling mines and clearing dangerous ammunition.

Some researchers use simulations as prototypes for large scale studies while to prove the concept they use a small number of real robots. For example, groups of robots have been used to realise multi-robot security systems [19], IED (Improvised Explosive Device) landmine detection and clearance [20, 21], and map-making [22].

Other examples of real-world applications where multi-agent collaborative systems may be beneficially employed are-

- Toxic waste cleanup
- Harvesting
- Bomb defusing and disposal
- Search and rescue
- Conveyance and material order handling on shop floor
- Material order fulfilment system in packing and distribution warehouse

Foraging, in which a group of robots pick up objects scattered in the environment is a useful test bed for cooperative robotics. Figure 2.4 shows a bomb disposal robot. Often more than one such robot is required to coordinate and defuse a bomb. Figure 2.5 (http://www.mmh.com/article/CA6591101.html) shows a material handling robot employed on the shop floor.

Figure 2.4 A bomb disposal robot

Figure 2.5 A material handling robot on the shop floor

Groups of robots, numbering hundreds, are now being used for order fulfilment in distribution warehouses. A very good and practical example of a multi-agent collaborative system for order fulfilment is the Kiva System (www.kivasystems.com) which relies on autonomous mobile robots to enable pick rates in the same range as carousels and automatic storage and retrieval systems. Instead of operators going to the shelves to fetch items, the robots carry the stack of items to the operator at the packaging station. This reduces operator fatigue and increases productivity several folds. Figure 2.6 shows a robot hauling items for packing.

Figure 2.6 An autonomous robot hauling items for packing

A software architecture that facilitates the fault tolerant cooperative control of teams of heterogeneous mobile robots performing missions composed of loosely coupled subtasks has been detailed in [23]. The feasibility of this architecture is demonstrated in an implementation on a team of mobile robots performing a laboratory version of hazardous waste cleanup. A weed removing robot is described as an example of an agricultural robot in [24]. Field tests of harvesting robots have already been conducted and the days are not far away when groups of robots will be picking fruits and

vegetables. An algorithm for a distributed team of autonomous mobile robots to search for an object is presented in [25]. When one robot finds it, they all gather around it, and then manipulate ("rescue") it.

Other grappling issues of the present day multi-agent systems, such as *system dynamism*, *path planning, control architecture*, *communication structures*, and *vision systems* are discussed in the subsequent sub-sections.

2.3.1 Cooperative Robotics for Entertainment– Static and Dynamic Systems

Multi-agent collaborative systems find immense applications in the world of entertainment robotics. For example, groups of robots are built and programmed to play music. One such system which has been developed and deployed by Singapore Polytechnic is shown in Figure 2.7. While there is no need for a corresponding simulation platform for a *Robotic Band* and there is no real physical movement of robots, nonetheless there are elements of cooperation in the common task. Such systems, however, do not encounter problems of resource constraints like space or communication bandwidth. Moreover, there are only a small number of agents to deal with, usually 4 or 5. An extension of this system is the one in which the robots playing music also undertake physical movements like swinging and twisting. The overall system would still be classified as "semi-static".

Figure 2.7 Robotic Band (Photo courtesy of SEEE, Singapore Polytechnic)

A good classical example of a very "dynamic" cooperative robotic system is the *Robot Soccer System*. It has been used as a test platform for cooperative robotics since the mid-nineties. It addresses several major issues of multiple-robotics such as path-planning, obstacle avoidance, real-time strategy and behaviour programming, task allocation, real-time image capture and processing, communication, intelligent motion control, fault-tolerance and operating with resource constraints. While simulation has been used to create a team of 11 robots to play the game of soccer, until recently using 3 to 5 robots for physical implementation has been the norm. Extending the physical implementation to 11 robots is a major step forward and a few research groups around the world have realised it in the past couple of years.

Another example of a "dynamic" multi-agent collaborative system is the *Marching Robots*. Sometimes this is referred to as the problem of *herding* or *flocking*. A cluster of robots move around to form patterns and formations. In its simplest form, robots are required to follow each other, move in a group or follow a 'leader'. Solutions to formation and marching are useful primitives for larger tasks like moving an object by a colony of robots. Positional constraints solved using such systems can be effectively employed in real-world situations. Collective behaviours and formation organization of multi-robot system with simple dynamics and interactions has been presented and addressed in [26]. Coupled with the problem of marching is the issue of avoiding obstacles and together they make a compelling case for further research.

2.3.2 Multiple-Robot Path Planning for Collaboration

Multiple robots working in unison within a limited space typically need to avoid collisions. Collision could be with each other or with other obstacles. This is essentially a problem of resource conflict and implementing collision avoidance is one of the most basic behaviours in a group activity. Collision avoidance has been implemented using several methods like using traffic rules, setting priorities and structured communication protocol to establish the '*right of path*'. However, in situations where the operating environment is very dynamic i.e. fast and is constantly changing (like in a robot soccer game where the opponent robots are constantly moving and changing position), applying fixed rules will often not give good results. And this is where a real-time path planner may become necessary.

2.3.3 Centralized/Decentralized Control Architecture of Multi-Agent Systems

The group architecture of a cooperative multi-agent system provides the core of the infrastructure upon which collective behaviours are implemented. In a centralised control structure a particular single robot acts as the Central Commander/Controller and has all the decision making capabilities. It may gather information from individual

agents (about the environment and agent positions) before making decisions. The main drawback of such a centralised approach to command and control is that it does not offer sufficient degree of fault-tolerance. If the agent fails, the whole system comes to a standstill.

In a decentralized architecture there isn't such a central commander. Instead the control is distributed and all agents have some share with respect to control. A decentralized architecture has several advantages over the centralized one - it offers fault tolerance and improved reliability, scalability and better flexibility. However, the command structure is distributed and it is generally more complex to implement. Often a hierarchical approach in implementing the decentralized architecture has been used to solve the complexity.

The control architecture selected greatly influences the capabilities and limitations of the system. In practice, however, many systems do not conform to a strict centralized/decentralized approach [15]. In some hybrid centralized/decentralized architectures there is a '*central planner*' that exerts high-level control over mostly autonomous agents [27].

2.3.4 Communication Structures in Collaborative Tasks

Inter-agent interaction, determined by the communication structure, is an important aspect of collective behaviour. Three major types of interactions have been categorised in [15]. They are-

Interaction via environment. In this method there is no explicit communication or interaction between agents. The environment itself acts as the communication medium. The agents are said to "cooperate without communication".

Interaction via sensing. In this method agents sense one another and interact locally, but there is no explicit communication. The agents distinguish between other agents in the group and other objects in the environment. The marching robot team's pattern formation can be considered as a good task where this method of communication can be implemented efficiently.

Interaction via communications. This method is characterised by explicit communication between agents. The communication may be directed to a particular agent, to a group of selected agents or broadcasted to the entire team of robots.

2.3.5 Local and Global Vision Based Agents

Vision systems are increasingly being employed in multi-agent collaborative systems. There are two types of vision implementations- local vision systems and global vision systems.

In the systems using local vision, each agent has its own complete vision capture and image processing capability. Image analysis is done locally and the agent may communicate raw or processed data to other robots (in a distributed control architecture) or to a 'leader' (in centralized or hybrid control architectures). In local vision systems, there are two types of camera control mechanisms – one in which the camera is movable and another in which it is fixed. In the movable type, the control mechanism allows the camera to be tilted or panned to search for an object. The tilt angle and panning area are usually restricted. In the fixed type, the camera is immovable. It generally relies on a parabolic cone with silvered surface to capture a full 360 degree view of the workspace [28]. Examples of robots with local vision are shown in Figures 2.8 and 2.9.

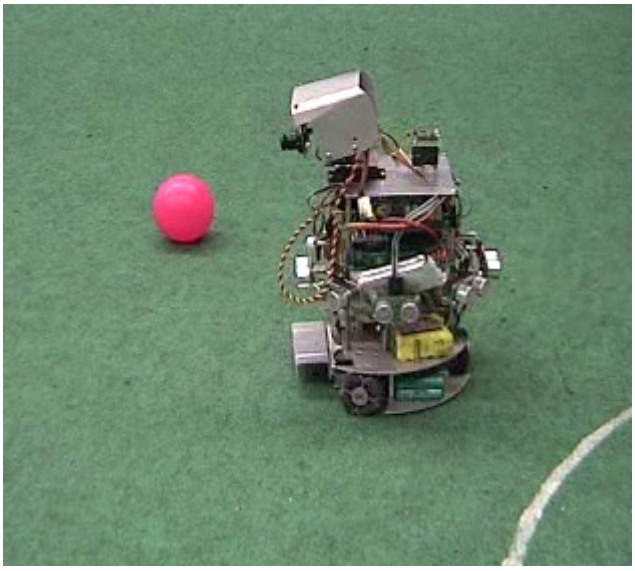


Figure 2.8 Robot with a local vision - the camera can tilt and pan to detect the target

In a system using global vision, central vision facilities are employed to oversee the entire area of operation with the task of detecting the position and orientation of the agents and other objects. This is typical of a centralised command and control approach. While the global vision system may be cheaper to implement, it is not practical if the area of operation is very large. Also, such an approach makes the agents totally dependant on the central controller. It does not offer redundancy and fault tolerance.

Figure 2.9 Robot with a local omni-directional vision using parabolic mirror cone

Figure 2.10 Multi-agent collaborative system (robot soccer) using global vision

In Figure 2.10, two teams of robots are playing a game of soccer in a RoboCup world championship match. The system uses global vision. Often more than one camera is used to view the entire field. Images from multiple cameras are then combined in a Vision Server and the robots' coordinates and orientation are passed to the game controller. Implementing multiple camera systems is a research area in itself.

It is expected that a hybrid vision system will become more prevalent in future. For example it may amalgamate local vision with some form of global sensing (e.g. GPS). In such a system, the local vision will be used to map the local terrain while by means of GPS the central commander will be able to locate the agent positions.

2.4 Robot Soccer System: A Test-bed for Collaborative Robotics

In this section, the hardware of the robot soccer platform is described. This platform has been used to test the proposed algorithms in this thesis for improvements in vision processing, techniques for implementing cooperative behaviour including predictive target interception, advanced Triangular Targeting Algorithm (TTA) for motion control and real-time analysis of collaborative tasks.

Micro Robots are used in education and entertainment in many ways. There are robots which navigate a maze, climb a wall, play a game of soccer, wrestle with other robots, run round a track, balance a pole, mow the lawn and vacuum clean the house. Robot Soccer has become increasingly popular over the last decade not only as a platform for education and entertainment, but also as a very useful platform to test algorithms and techniques for dynamic control systems in a multi-agent collaborative environment. It is a very good replica of the human version of soccer, though often scaled down to between 3 and 11 players in a team (to limit the complexity). It is a powerful vehicle for dissemination of scientific knowledge in a fun and exciting manner. It encompasses several technologies – embedded micro-controller based hardware, wireless radio-frequency data transmission, dynamics and kinematics of motion, motion control algorithms, real-time image capture and processing, multi-agent collaboration etc. Robot soccer involves a wide range of physical movements as well as cognitive skills required to make strategic decisions and play on a team. Because of the dynamics and high complexity of the robot soccer system as well as manoeuvrability and high speed of its robots, the accurate and real-time detection of position and orientation of objects have gained special importance as it greatly affects path planning, prediction of moving targets and obstacle avoidance. It is thus also a good test bed for developing and testing vision systems for real-time tracking of targets. Figure 2.11 shows the typical robot soccer system setup which uses a global vision system.

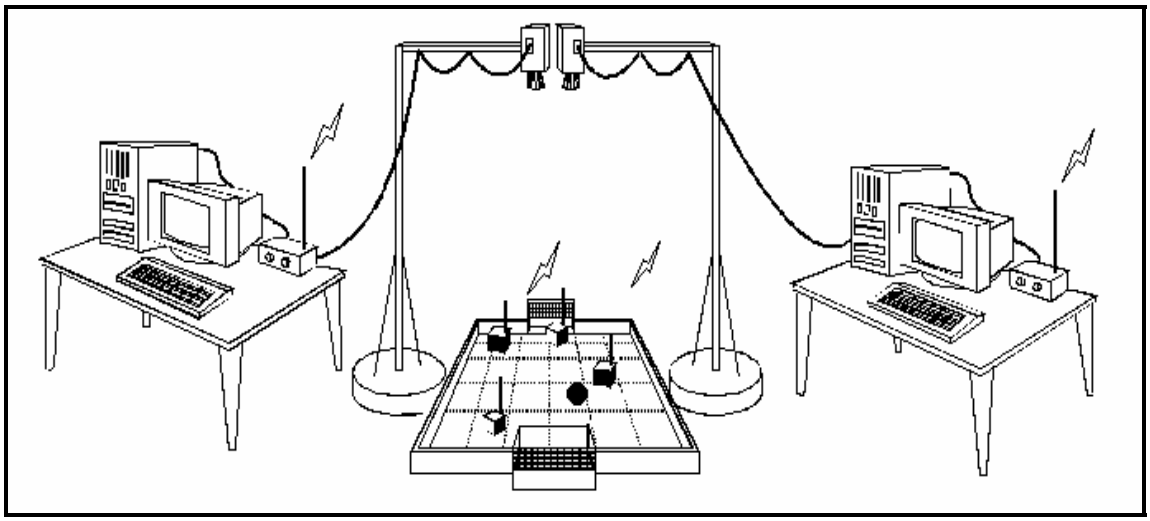


Figure 2.11 Robot Soccer System setup using global vision
(http://fira.net/soccer/mirosot/MiroSot.pdf)

Two cameras are mounted at the top of the soccer field, one for each competing team. In the test setup the cameras used are analog cameras with composite or separate RGB (Red-Green-Blue) interlaced video output. The analog video signal from the camera is fed to the vision capture card, often called the frame grabber card, which has a PCI interface and is plugged into the PC motherboard. The frame grabber card digitizes the analog video signal to digital values. Some robot soccer teams have started using digital cameras with firewire (IEEE 1394) interface, but these are still relatively expensive.

Each robot from the home and opponent team is identified by a colour jacket mounted on top of the robot. The colour jacket has one 'team colour' and one or more 'robot colour' patches to identify not only the team but also the individual within a team. The colour jacket is also used to detect the position and orientation of the robots.

The PC is the *central commander*, also called the *strategy server*. However, it does much more than just the processing of strategy. It runs the software program to capture and process the image, decide the strategy for the collaborative behaviour and assign roles to the individual agents, execute the low level control functions to calculate the wheel velocities and transmit the data packet to the agents using radio frequency (RF) wireless transmission. The test system uses 418 MHz and 433 MHz modulation frequency for the RF transmission. The two teams must use two different frequencies to avoid interference.

The soccer field used has the dimensions of 150 cm x 170 cm. The camera is mounted at a height of 2 m from the field.

One of the robots used in the various experiments is shown in Figure 2.12. Each robot is a complete embedded autonomous system by itself with its own micro-controller,

memory, RF receiver module, motors with magnetic encoders and switches to set the robot identity. The robots are powered by NiMH or Lithium Ion rechargeable batteries.

Figure 2.12 Bi-wheeled robot used in experiments

3 Fast Image Capture and Vision Processing

Vision systems are widely used in the industry for object tracking, intrusion detection, vehicle and mobile robot guidance, inspection automation, etc. [29]. The majority of the commodity vision systems use video signals, most often from a CCD camera, as input to the image capture and analysis subsystems. Typically, such vision systems provide frame rates of 30 Hz or field rates of 60 Hz for interlaced images. Processing these images with useful resolution of 320 x 240 and above in the 33.3 ms and 16.67 ms sample times respectively can pose a significant challenge especially when other processing tasks such as strategy and task allocation, low-level control and communication with the robots, are also to be completed. A lot of research efforts have been spent on improving the speed of image processing for robotic applications [30] and it continues to attract serious attention of researchers. Faster vision processing algorithms result in better motion control and hence better coordination between agents to accomplish a collaborative task.

A computationally inexpensive vision processing algorithm using Run Length Encoding (RLE) has been discussed in papers [10] and [31]. RLE is an image compression technique that preserves the topological features of an image, allowing it to be used for object identification and location [32]. Though the RLE algorithm can be implemented on commodity hardware for multi-agent collaborative systems such as the robot soccer vision system [33, 34], its significant processing speed advantage is when there are a large number of objects to track. For smaller systems with a limited number of agents, say two to five, the commonly used blob identification techniques together with the incremental tracking algorithm is adequate and equally efficient.

Interlaced images introduce the ‘image scattering’ problem for a moving object because of the time delay between the two fields of the image. To overcome this problem, the odd and even scan fields have to be processed separately. However, because of quantization errors in each field, a stationary object may appear to be in two different locations. Filtering techniques are required to minimize such a negative effect. The other sources of errors in the vision system are due to variation of light intensity and inherent sensor noise.

The cumulative effect of the errors in the vision system is very significant, especially in a highly dynamic collaborative system where the agents are moving very fast. Figure 3.1 shows the software structure and data processing flow of a multi-agent robotic controller using global vision. The image processing software identifies the position and

orientation of the robots and other objects of interest in the robots' workspace. The vision data are filtered to reduce the effect of noise and passed on to the strategy/task allocation layer. At this layer of the software architecture, the behaviour required of an agent is determined. The errors in the vision system percolate down the hierarchy of the robotic controller and have profound effect on the robot behaviour and hence the overall performance of the collaborative system. It is thus imperative that careful considerations are given to eliminate or at least minimise the vision processing errors.

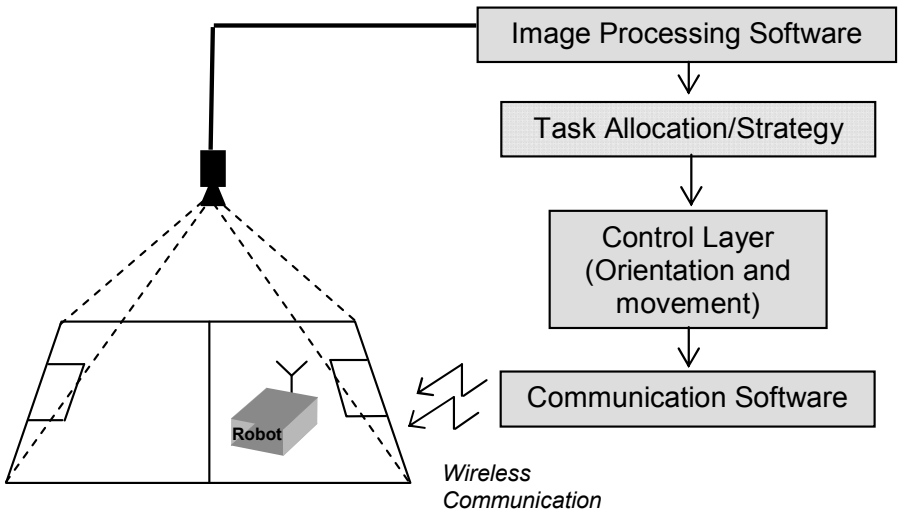


Figure 3.1 Software structure of a multi-agent robotic controller using global vision

3.1 Global Vision – Sources of Error

A global vision system uses a single or multiple cameras to detect and track several objects. The main sources of errors in the vision system are described in the following sub-sections.

3.1.1 Separate Processing of Odd and Even Scan Fields of an Interlaced Bit-Mapped Image

A stationary object may be reported at different locations in each frame due to different quantization errors. This is explained using a bit mapped image of 16x16 pixel resolution as shown in Figure 3.2. The object of interest, the centre position coordinates of which are required to be calculated, is of a square shape.

To calculate the position, the well known zero-order moment and centre-of-gravity equations (3.1) are used. For an *m* x *n* binary image, the coordinates are given by:

$$\text{Area, } A = \sum_{i=1}^{n} \sum_{j=1}^{m} B[i, j]$$

$$\text{COG, } \bar{x} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} jB[i,j]}{A} \qquad \bar{y} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} iB[i,j]}{A} \tag{3.1}$$

$B[i, j]$ is the value of the bit (0 or 1) in the image at location $[i, j]$.

i is the row number (i.e. Y-coordinate)

j is the column number (i.e. X-coordinate)

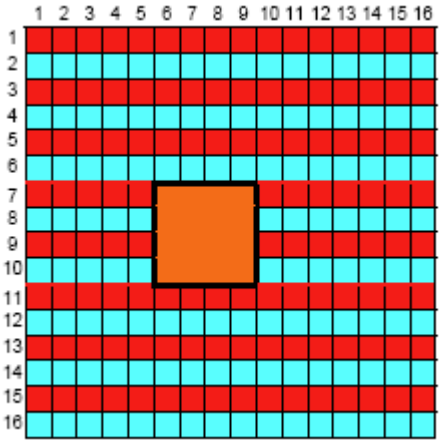


Figure 3.2 Bit-map of an interlaced image

For the illustrated example, the coordinates of the object in the Odd scan is (7.5, 8.0) as shown in the calculations below:

$$A = 8$$

$$\bar{x} = \frac{6\times 2 + 7\times 2 + 8\times 2 + 9\times 2}{8} = 7.5$$

$$\bar{y} = \frac{7\times 4 + 9\times 4}{8} = 8$$

The coordinates of the object in the Even scan is (7.5, 9.0) as shown in the calculations below:

$$\bar{x} = \frac{6\times 2 + 7\times 2 + 8\times 2 + 9\times 2}{8} = 7.5$$

$$\bar{y} = \frac{8\times 4 + 10\times 4}{8} = 9$$

The separate processing of odd and even scan fields do not affect the X Coordinate. Only the Y Coordinate has a shift of 1 pixel. Looking at a physical area of 170 cm x 150 cm and working with an image resolution of 320x240 pixels, 1 pixel translates into a shift of ~0.62 cm in the Y Coordinate of the object. Moreover, this offset will not be

constant for a moving object since the shift can occur both in a positive or negative direction. Filtering techniques are required to minimize the quantization error which is discussed in Chapter 5.

3.1.2 Variation Of Light Intensity

While errors due to separate processing of odd and even scans could be pre-dominant under controlled light conditions, nonetheless, these errors are further compounded by variation in light intensity from one frame to another. In the real-world applications of collaborative robotics, it is often not possible to create ideal (or at least stable) light conditions. To partially overcome this problem, YUV colour thresholding can be employed with the Y (intensity) range extended to the maximum limits. However, extending the colour boundaries too much result in the threshold values of two or more colours overlapping each other. Also extending the threshold values wider will make the vision system error prone as stray pixels from the background will be picked up too. This limits the colour tolerance.

3.1.3 Inherent Sensor Noise

Certain errors are inherent in the system. These originate from the CCD camera, the electronic hardware of the frame grabber card, long connecting cables, etc. Such errors in the vision-generated data have a significant impact on targeting accuracy even when intercepting or striking a stationary object. Interception accuracy suffers when the target is moving. This is due to the fact that actions are initiated based on *predicted* future positions which are different from the *current* position and are calculated based on velocity measures which are noisy.

3.2 Experimental Hardware Setup

The experimental hardware setup consists of a Pulnix 7EX NTSC camera (www.pulnix.com) with analog composite video output and a FlashBus MV Pro frame grabber (www.integraltech.com) card with PCI interface. The image is captured at a resolution of 320x480 at a sampling rate of 30 Hz. The odd and even fields are processed separately; hence the effective image resolution is 320x240 delivered at a sampling rate of 60 Hz. The captured image is processed on a 1.8 GHz Pentium 4 PC with 512 MB RAM. The image capture card was configured for off-screen capture, as shown in Figure 3.3. Off-screen capture mode facilitates fast processing of the image from the system RAM. The image is transferred to the VGA RAM only when a live image is required to be seen on the screen, such as during the setting and testing of colour thresholds. Once the colour tuning is done, the transfer of image to the VGA RAM is switched off.

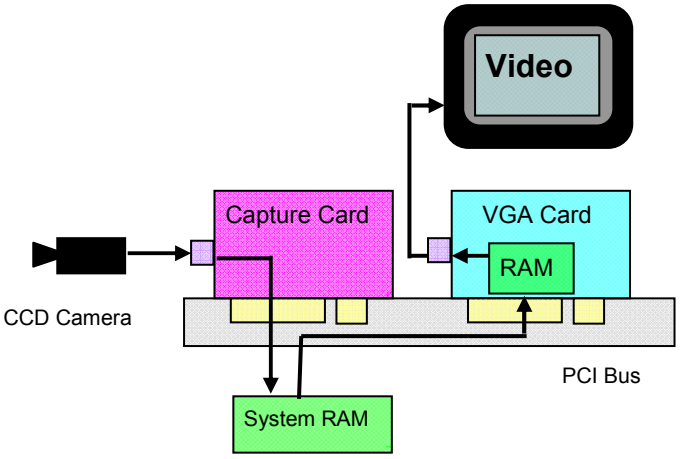


Figure 3.3 Image capture card in *off-screen capture mode*

An important hardware feature of the FlashBus MV Pro frame grabber card is that it generates a Vertical Sync signal for every odd and even field. The vertical sync signal can be captured in the software as an interrupt. This facilitates implementation of an interrupt based system and fixes the sample time of the controller.

3.3 Colour Segmentation, Area Thresholding and Blob Merging

The colour image sequence is processed at three levels: pixel level, blob level, and object level. To facilitate identification and separation of individual objects, a colour jacket comprising two colour patches can be used on each robot, as shown in Figure 3.4. In applications where several groups or teams of robots are involved, one of the colour patches is used to identify the group (team colour patch) and the other is used to identify which robot it is within the group (robot colour patch). The centres of the two colour patches, *Cr* and *Ct*, are first calculated from the image. The inclination of the line joining the two centres gives the orientation of the robot while the coordinates of the centre of the line give its position. For some target objects such as a ball in a robot soccer system, only the centre of the colour patch is calculated as it does not have an orientation. The velocity of the target is used to add a direction vector to its position.

The accuracy of the angle calculation depends on the accuracy with which the centres of the colour patches are detected and how far apart these centres are. If the centres are closer to each other, any small variation in the calculation of *Cr* and *Ct* will result in a large variation in angle. In order to improve the accuracy of angle calculation, experiments with different patterns of colour jackets were performed. The centres of the colour patches in Figure 3.5 are further apart than those in Figure 3.4, thus improving the accuracy of the angle calculation.

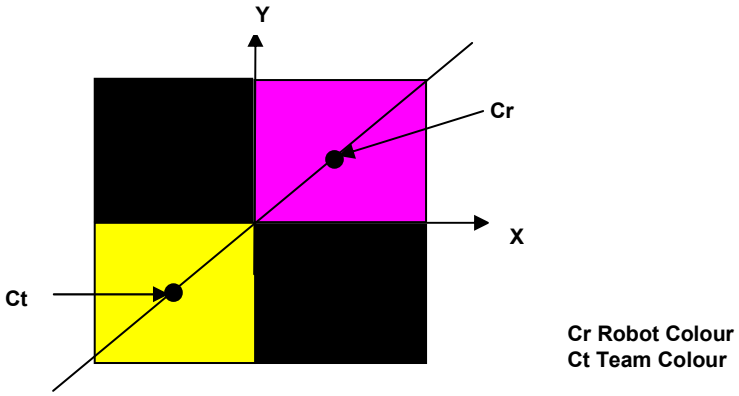


Figure 3.4 Colour jacket for identification of Robot

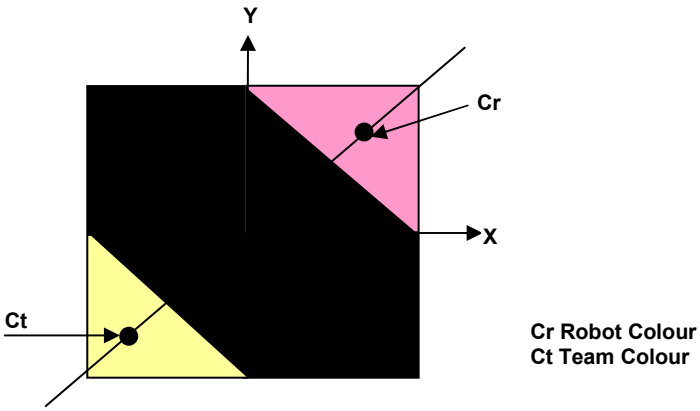


Figure 3.5 Colour jacket for improved accuracy of orientation

The object detection algorithm starts with colour segmentation. It searches the image to determine if the pixels belong to one of the calibrated colour classes. The pixels are then grouped to create colour patches using a '*sequential component labelling algorithm*'. This algorithm uses a two-pass labelling technique [35] with identifiers (labels) that increment from the value of 1. Ideally, the number of labels is equal to the number of desired colour patches on the objects in the entire image.

The procedure for checking the membership and grouping of pixels consists of 5 steps split into two passes.

A. FIRST PASS (Steps 1 to 4)

1. Process the image in the tracking window from left to right, top to bottom, analyzing each pixel.
2. If the pixel in the image is within the YUV threshold values of the colour of interest, then
   (a) If only one of its upper and left neighbours has a label, copy the label.
   (b) If both upper and left neighbours have the same label, copy that label.

(c) If both upper and left neighbours have different labels, copy the upper pixel's label and enter the labels in an equivalence table as equivalent labels.

(d) If not (a), (b) or (c) assign a new label to this pixel and enter it in the equivalence table.

3. If there are more pixels to consider, repeat step 2 for additional pixels, otherwise proceed to step 4.
4. Find the lowest label for each equivalent set in the equivalence table and add to the equivalence table.

B. SECOND PASS (Step 5)

5. Process the picture by replacing each label with the lowest label in its equivalent set.

To illustrate the algorithm, a binary image (rather than a YUV image) is used in the following example. Figure 3.6 shows a representation of the binary image before the first pass of the algorithm. The 0's represent the background of the image and the 1's represent the objects of interest. It can be seen that there are two objects of interest in the image, one in the left and the other in the right.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	1	0
0	1	1	1	1	0	0	0	0	1	1	0	1	0
0	1	1	1	1	1	1	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.6 The binary image

Figure 3.7 shows the image after the first pass of the algorithm. The objects now have multiple labels as the first pass of the algorithm was not able to correctly label all shapes (i.e. the object on the left has labels 3 and 1 and the object on the right has labels 2 and 4). Figure 3.8 shows the image after the second pass of the algorithm, which resolves the problem of multiple labels for single objects.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	2	0
0	3	1	1	1	0	0	0	0	4	4	0	2	0
0	3	1	1	1	1	1	0	0	4	4	4	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.7 Image after the first pass

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	0	0	0	2	0
0	1	1	1	1	0	0	0	0	2	2	0	2	0
0	1	1	1	1	1	1	0	0	2	2	2	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.8 Image after the second pass

The algorithm described above is often called the '2-neighbour' algorithm in which the upper and left neighbouring pixels of the pixel under test, are considered to evaluate the membership (label) of a pixel. Another well know and very similar algorithm is called the '4-neighbour' algorithm in which the four neighbouring pixels – upper, left, bottom and right – are considered for evaluating the label of a pixel. This algorithm is more time consuming and gives only marginal improvement in the accuracy of colour segmentation. Thus the '2-neighbour' algorithm is often preferred over the '4-neighbour' algorithm, especially in real-time applications where the colour patches are relatively large in pixel area.

Once the colours have been segmented, two separate processing steps follow – filtering based on area threshold and blob merging. In order to reduce noise, very small colour blobs are discarded if they fall below a certain area threshold. For example, it is usually safe to discard patches which are only 2 or 3 pixels in area as these would generally be noise from the background. If patches of the same colour are very close to each other, then these are merged to form one patch. This blob merging technique, with a distance tolerance, is widely used in practice and reported in literature [36, 37].

Having identified the separate objects, the centre of each object is calculated using the centre of gravity calculation described earlier (equation 3.1).

3.4 Interrupt Based Multi-Buffered Image Capture

In order to fix the sample time and delay of the vision control loop to a constant value, an interrupt driven approach is adopted. In this method the vision processing and strategy functions are placed in an interrupt service routine. The routine is serviced on each occurrence of the Vertical Sync signal on the frame grabber. The grabber generates a Vertical Sync signal for every odd and even field. For an image resolution of up to 640x480, the card can capture the image at 30 frames per second. Hence the interrupt service routine of the interlaced image is executed every 16.67 ms. This poses a challenge - all the vision processing, strategy related calculations, computation of motion control data and transmission of RF packets to all the robots must be completed within 16.67 ms. The completion of processing within 16.67 ms is achieved by segregating the process of capturing the image and subsequently processing it. A four-stage ring buffer is employed to capture and process the image. If the image processing is guaranteed to complete in the specified sample time only two buffers are required. Since this is not the case for the colour segmentation and blob-identification algorithm presented in section 3.3, a four buffer system is required. The buffer organisation is shown in Figure 3.9. When an image is captured in a buffer, the image from the previous buffer is processed. This helps to avoid buffer contention during image capture and processing [38].

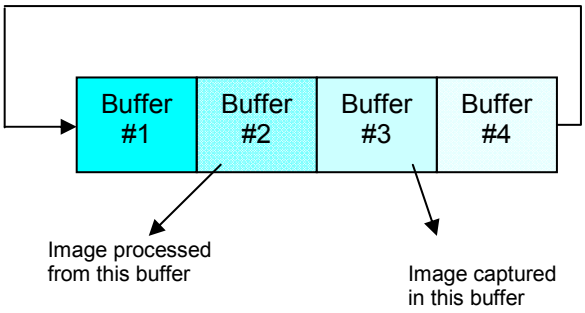


Figure 3.9 Multi-buffered image capture

3.5 Full Tracking vs. Incremental Tracking

The full tracking algorithm searches through the whole image, testing each pixel whether it is a member of one of the calibrated object colour classes. The full tracking process is very inefficient when the objects are small and only represent a small percentage of the whole image. For increased efficiency of image processing, incremental tracking is generally employed [39]. Incremental tracking is the approach whereby a small region around the last known position of the object (Figure 3.10) or its predicted position (Figure 3.11) is processed rather than the whole image, thereby

decreasing the processing time or computational resources required. By limiting the number of objects to be tracked, it is possible to increase the incremental tracking window size and yet keep the vision processing time within the sample period of 16.67 ms. Using larger incremental tracking window sizes, the object being tracked is nearly always identified. In the event that the object is "lost" (i.e. it is not inside the predicted tracking window), the fault tolerant software reverts to the full-tracking mode, whereby the whole image is analyzed to 'recover' the object's position.

To locate the object positions the first time, the image must initially be scanned fully for one frame. This will identify the starting position of all the objects within the field of view. Then the incremental tracking algorithm can continue.

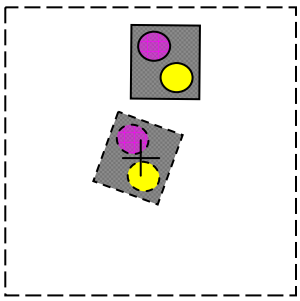


Figure 3.10 Tracking window centred on last known position of object

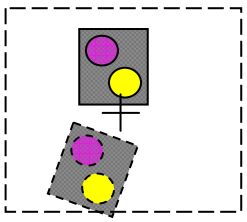


Figure 3.11 Tracking window centred on predicted position of object

The number of pixels that must be processed is related to the size of the tracking window and the number of objects being tracked. This technique is significantly more efficient than full tracking if the sizes of the tracking windows are much smaller than the size of the image divided by the number of objects (equation 3.6).

If the image resolution is H pixels horizontally and W pixels vertically, then the size of the image, I, in number of pixels, is given by equation (3.2).

$$I = H * W \tag{3.2}$$

If the resolution of the incremental tracking window is h pixels horizontally and w pixels vertically, then the size of the incremental tracking window, σ, in number of pixels, is given by equation (3.3).

$$\sigma = h * w \tag{3.3}$$

If t_p is the time to process 1 pixel, time for full tracking, T_F, is given by equation (3.4).

$$T_F = I * t_p \tag{3.4}$$

If there are n objects on the field, the time for processing all the incremental windows, T_I, is given by equation (3.5).

$$T_I = n * \sigma * t_p \tag{3.5}$$

For efficiency, T_I should be less than T_F. Hence,

$$n * \sigma * t_p < I * t_p$$

$$\sigma < \frac{I}{n} \tag{3.6}$$

Reducing the sample period of the system can drastically reduce the required size of the tracking window, since the objects would have moved a shorter distance in the shorter sample time. In a sample case, halving the sample period would have the objects move half the distance, so the length of the tracking window can be halved. This gives an area reduction of a factor of four. Therefore doubling the frame rate (i.e. halving the sample period) would reduce the execution time of this algorithm by a factor of four. Thus, paradoxically, this algorithm is more likely to complete in the required sample time if the frame rate is higher.

In a system that has a reliable frame rate, the last known velocity of the object, rather than the last known position, can be used to centre the tracking window on the predicted position of the object. Using this method, the size of the tracking window can be reduced further as the error in the predicted position will depend only on the uncertainty in the measured velocity. The only risk associated with reducing the tracking window size relates to collisions, which can alter the velocity of the objects significantly. In robot soccer, the ball is the lightest and fastest object and so is at the greatest risk of being 'lost'.

Several techniques to overcome this problem have been proposed in the literature [40] based on predicting collisions of fast moving light objects. Figure 3.12 shows the movement of the tracking window as the robot moves.

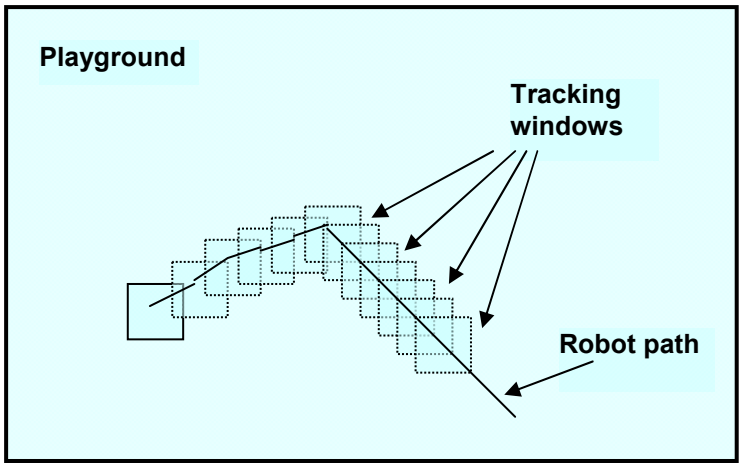


Figure 3.12 Robot path and the incremental tracking window

Adaptive Tracking Window Size

Enlarging the tracking window for the faster and lighter objects can produce reliable tracking results. A minimum tracking window size may be defined for a stationery object. As the object starts to move, the size of the tracking window will increase proportionately and adapt to the object's velocity. This will ensure that larger the velocity, bigger the tracking window size and lesser the possibility of 'losing' it. This will greatly increase the reliability of tracking. However, the downside is that even for a fixed number of objects, the number of pixels that need to be processed is not constant and can vary a lot depending on the individual object velocities. The variable number of pixels that will need to be processed creates an uncertainty in the total processing time per frame. This variability will have a detrimental effect on the control of the robot motion as the sample time for error correction will be variable.

3.6 Fast Access Colour Look-Up-Table

3.6.1 Limitations of Using RGB Colour Space

The blob detection algorithm can be implemented on any commodity image processing hardware. For example, the image digitization can be done using the FlashBus MV Pro frame grabber card which provides pixel colour information in RGB (Red, Green and Blue). A convex partition of the RGB colour space can be created for each colour identifier, as shown in Figure 3.13. This convex partition (colour subspace cube) is specified by a range of RGB values namely, MinR-MaxR, MinG-MaxG, and MinB-MaxB.

It has been shown in the literature that blob identification based on RGB colour space is not reliable in the face of varying light intensities as the luminance cannot be separated

from chrominance [31]. In order to cater to a wide variation of light intensity, the volume of the colour cube has to be extended. The drawback of doing this is that the colour cubes of different colours will overlap and encroach into each other's boundaries making it very difficult to segregate colours and at the same time detect them reliably. Instead a convex colour subspace, defined in the YUV colour space is often used providing substantially enhanced robustness and reliability of detection. The Y component independently corresponds to light intensity (luminance) of the colour and a wider threshold span can be set for it to cater to varying light intensities. U and V correspond to the chrominance of the colour.

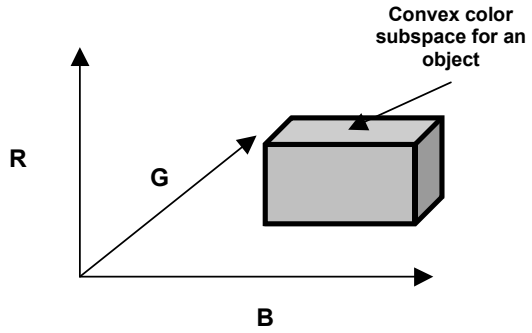


Figure 3.13 Convex RGB colour subspace

3.6.2 Defining YUV Thresholds

To define the YUV colour subspace, a sample of the image is captured and the colour of interest is zoomed in. In the zoomed image a rectangular region is defined, within which, each pixel is processed to calculate its YUV value using the standard colour space transformation matrix which is shown below (equation 3.7).

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

or alternatively as

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ U &= 0.565(B - Y) \\ V &= 0.713(R - Y) \end{aligned} \tag{3.7}$$

The relative position and orientation of the RGB colour cube in the YUV colour space is shown in Figure 3.14. From the computed YUV values, the MinY, MaxY, MinU, MaxU, MinV and MaxV are set. A user may manually fine-tune these thresholds using

the application's GUI. Usually a wider range of Y values are desirable giving it more bandwidth to account for varying light intensity.

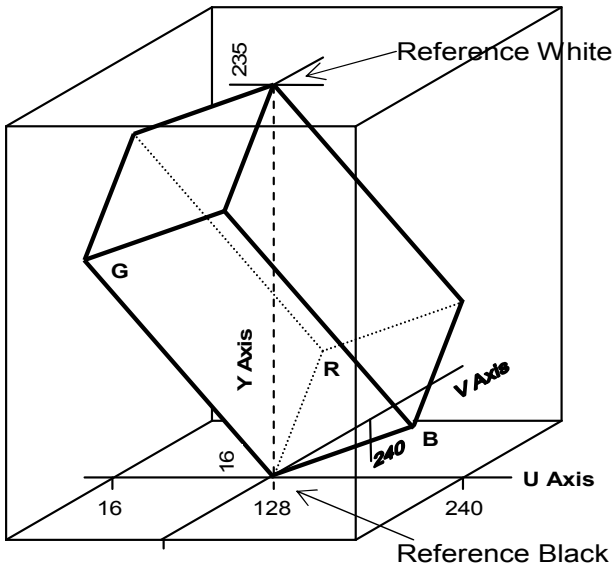


Figure 3.14 Relative positioning of YUV and RGB colour space

3.6.3 Membership Testing

For the two-pass sequential algorithm for blob-detection, each RGB colour pixel of an image needs to be tested to determine its colour sub-space membership. Since the colour boundaries are defined in YUV colour space, each pixel value would have to be converted from RGB to YUV, using equation (3.7), before testing membership using equation (3.8). This is a computationally intensive process and the overall performance could be quite low. Hence the mechanism used for thresholding warrants close scrutiny and requires careful efficiency consideration.

```
IF ( (Y >= MinY) AND (Y <= MaxY) AND
     (U >= MinU) AND (U <= MaxU) AND
     (V >= MinV) AND (V <= MaxV) )
THEN pixel_of_interest = TRUE                    (3.8)
```

Equation (3.7) requires 5 multiplications and the whole implementation of equation (3.8) may require up to 6 logical ANDing operations to determine whether a pixel belongs to a colour subspace and is thus of interest. To improve the computational efficiency, implementations using Boolean valued decomposition of the multidimensional threshold have been tested [31] on static images resulting in substantial reduction in processing time. This, however, still requires the colour space to

be transformed from RGB to YUV. This method had not been tested previously by any researcher on live images in real time.

3.6.4 One-dimensional Colour Look-Up-Table

An initial implementation in this research work used a large one-dimensional colour look-up-table (LUT) and an indexing technique based on the RGB value of the pixel. The *index* is created, using the equation (3.9) below, which is used to access the LUT.

$$index = R * 65536 + G * 256 + B \tag{3.9}$$

For a 24-bit RGB colour output from the frame grabber card, the maximum value of R, G or B is 255. Thus the size of the LUT is 256x256x256 bytes (16MB). For each RGB value, a unique *index* is created by the equation (3.9).

3.6.5 Posting the Look-Up-Table

Once the YUV thresholds have been defined for each colour, the LUT is posted with colour identities (IDs) for the entire RGB colour space as shown in the code segment in Figure 3.15.

The time it takes to update the LUT is not of any consequence as the update is done during the colour tuning phase. It is important that during the inspection time, the processing should not take unduly long and hence repeated multiplications and logical ANDing must be avoided.

```
for (r=0; r<256; r++)
  for (g=0; g<256; g++)
    for (b=0; b<256; b++)
      {
       y=(299*r+587*g+114*b+500)/1000;
       u=(565*(b-y) + 128000)/1000;
       v=(713*(r-y) + 128000)/1000;
       index = r*65536 + g*256 + b;

       //-- initialise on update --
       LUT[index] = NoCOL;

       //-- Reference Colour range --
       if ( (MinY<=y && y<=MaxY) &&
            (MinU<=u && u<=MaxU) &&
            (MinV<=v && v<=MaxV))
       {
         LUT[index] = RefCOL;
       }
     }
```

Figure 3.15 Posting the LUT with colour ID

3.6.6 Inspecting the Look-Up-Table

To test whether a pixel is in the YUV sub-space, given its RGB value, the index is calculated using equation (3.9) and the LUT content at that indexed location is tested as shown in the code segment in Figure 3.16.

To further improve the processing speed, the multiplications in equation (3.9) were replaced by shift-left operations as in equation (3.10).

$$index = R << 16 + G << 8 + B \tag{3.10}$$

```
index = r<<16 + g<<8 + b;
if ( LUT[index] == RefCol )
//-- it is a desired pixel
{
 //-- process the pixel
}
```

Figure 3.16 Inspecting the LUT

To classify each pixel in an image into one of a discrete number of colour classes, the index is created from the pixel's RGB values and the LUT is queried. The returned value indicates colour class membership.

The advantage of this method is that it does not require the RGB value of each pixel to be converted to YUV, which otherwise would take considerable processing time. However, this method has the following drawbacks-

- It takes very long to update the LUT. Using the experimental hardware setup detailed in section 3.2, it took 909 ms to update the LUT. This eliminates the possibility of updating the LUT in a real-time processing environment and hence the thresholds defined for each discrete colour class cannot be adapted to variations in light intensity.
- Because of the huge size of the LUT (16 Mbytes), the algorithm runs slower on a computer with a small cache memory, as there is frequent cache misses resulting in memory swapping. Nonetheless, in an image where the majority of the pixels belong to the background colour class, this is not a significant problem as the same part of the LUT will be accessed most of the time.

3.7 Discrete YUV Look-Up-Table

Performance issues have lead to research that focuses on effective classification in real-time. For simplicity of tuning, each colour is classified with a pair of thresholds on each of the Y, U, and V axes as illustrated in Figure 3.17.

Since each axis is independent, the large LUT may be decomposed into separate Y, U, and V LUTs of 256 elements each. The total size of all the discrete LUTs put together is only 768 bytes, greatly reducing the memory requirements compared to the LUT described in section 3.6.4. For each array element, one bit is used to represent each colour class as shown in Figure 3.18.

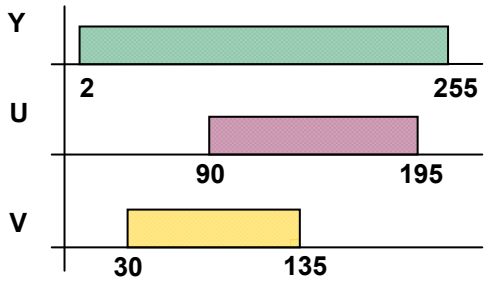


Figure 3.17 YUV Thresholds for Colour 3

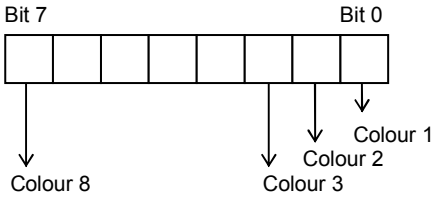


Figure 3.18 Colour representations in the LUT element

In programming terms, the colour IDs are defined as-

```
#define colour1_ID 0x01
#define colour2_ID 0x02
#define colour3_ID 0x04
#define colour4_ID 0x08
#define colour5_ID 0x10
```

Using arrays of bytes, 9 different colour classes can be represented (including 8 of interest and the background). To cater to more colours, the system can easily be scaled up to implement arrays of 16-bit or 32-bit integers.

Once the colour thresholds have been defined, the YUV LUTs are then posted with the colour IDs as shown in Figure 3.19. The shaded cells store a value of 1.

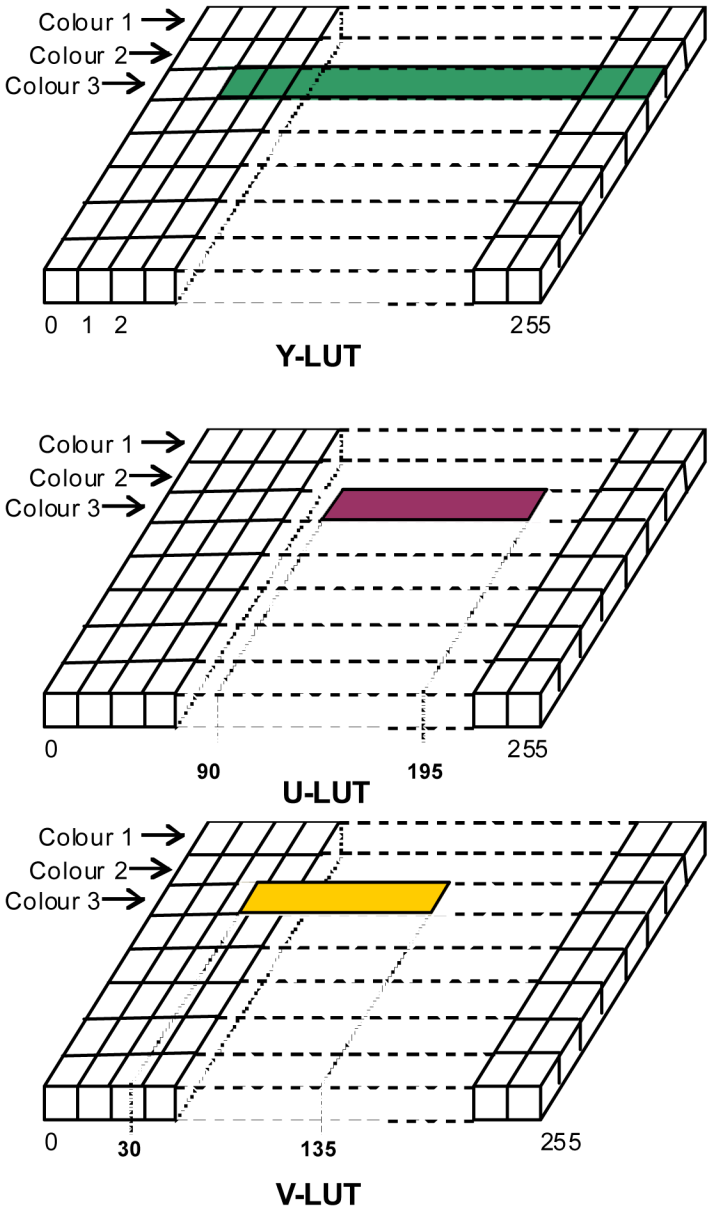


Figure 3.19 YUV LUT populated for colour 3 (not to exact scale)

3.7.1 Populating the Discrete YUV Look-Up-Table

After defining the YUV thresholds for every colour class, each LUT is posted individually with colour IDs. The code segment shown in Figure 3.20 updates the Y-LUT. The U- and V-LUTs are similarly posted.

```
for (y=0; y<=255; y++)
{
   if ((y >= Col1_MinY) &&(y <= Col1_MaxY))
     Y_LUT[y] |= Colour1_ID;
 //-- repeat for other colours
}
```

Figure 3.20 Posting the LUT with colour ID

To update the three LUTs it takes only 8.2 μs using the same experimental hardware setup. This enables the LUTs to be updated in real-time and hence may be used in implementing adaptive colour thresholding.

3.7.2 Testing Colour Class Membership

A pixel belongs to a colour class only if it is within all three Y, U, and V ranges. The colour class membership can therefore be computed as a bitwise AND of the elements of each component array. This is shown in the code segment in Figure 3.21.

```
if (Y_LUT[Y] & U_LUT[U] & V_LUT[V] & Colour1_ID)
{
  //the pixel belongs to Colour1 class
}
```

Figure 3.21 Testing colour class membership

The membership testing is very fast as the bitwise AND operation is computationally inexpensive. This method, however, requires the YUV values of each pixel to be available, which is often not the case for commodity hardware. Thus the advantage gained by using a smaller LUT is partly offset by the additional computation time required to map a pixel from RGB colour space to YUV colour space. In Section 3.7.3 two methods of speeding up this mapping are presented and the performance evaluation is detailed in Section 3.7.5.

3.7.3 Colour Space Transformation

The standard transformation matrix of equation (3.7) for mapping RGB to YUV involves several floating point multiplications, which are potentially very time-consuming. The floating point arithmetic may however be replaced by integer operations as shown in equations (3.11).

$$\begin{aligned} Y &= (299R + 587G + 114B)/1000 \\ U &= 565(B - Y)/1000 + 128 \\ V &= 713(R - Y)/1000 + 128 \end{aligned} \tag{3.11}$$

The U and V components are offset by 128 to bring them into positive range to facilitate array indexing. Equations (3.11) still use division operations. The division operations may be eliminated by scaling the coefficients by powers of 2 rather than powers of 10, allowing the use of a computationally less expensive shift-right operation as shown in equations (3.12).

$$\begin{aligned} Y &= (9798R + 19235G + 3736B) >> 15 \\ U &= 18514(B - Y) >> 15 + 128 \\ V &= 23364(R - Y) >> 15 + 128 \end{aligned} \tag{3.12}$$

3.7.4 A New Colour Space

This research makes a significant contribution by proposing a novel Y'U'V' colour space. This new colour space not only retains all the colours of the RGB colour cube, it actually increases the volume of the YUV colour cube, thereby enhancing the resolution of spatial colour separation. The proposed transformation is governed by equation (3.13).

$$\begin{bmatrix} Y' \\ U' \\ V' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.13}$$

The new Y'U'V' colour space has several advantages over the standard YUV space represented by equation (3.7).

- Computationally it is very inexpensive. Only integer additions and subtractions are involved; all floating point operations and logical shift operations have been completely eliminated.
- The white point corresponds to equal quantities of R, G, and B.
- The new Y'U'V' colour space provides better resolution. Without scaling, the Y' range is 0 to 765, U' range is from -510 to 510 and V' range is from -255 to 255. This enables colours that are closer together to be detected reliably.
- The Y', U' and V' axes are orthogonal, making the transformation back to RGB similarly simple. It also gives better decorrelation of the colour space for many images.

This larger LUTs (2298 elements as compared to 768) increase the LUT update time to 19.2 μs. This is still extremely fast and causes no concern for real-time update of the LUT.

3.7.5 Experimental Results and Discussion

The proposed improvements to the methods of transforming from RGB to YUV colour space and the new Y'U'V' colour space were evaluated using the robot soccer system, which offered the possibility of testing the algorithms for real-time processing with differing number of objects. A very high precision counter was implemented in the software which enabled measurement of time with an accuracy of a hundredth of a milli-second. To evaluate the robustness, tests were done with 4 objects (3 home robots and ball), 7 objects (3 home robots, 3 opponent robots and ball) and 11 objects (5 home robots, 5 opponent robots and ball) for incremental and full tracking. The tests were performed with constant light intensity. The test results are summarized in Table 3.1, and compared in Figures 3.22 and 3.23. The tracking times were measured for 1000 frames and averaged.

Table 3.1: Summary of vision test results for different look-up-tables

System	#Objects	Average Tracking Time (ms)		LUT update time
		Incremental	Full	
#1	4	5.27	15.54	909 ms
	7	5.62	20.97	
	11	5.95	28.34	
#2	4	5.38	21.89	8.2 μs
	7	5.64	30.37	
	11	5.99	41.72	
#3	4	5.34	17.22	8.2 μs
	7	5.56	23.41	
	11	5.85	31.68	
#4	4	5.15	14.34	19.2 μs
	7	5.38	19.22	
	11	5.68	25.76	

With respect to the data presented in Table 3.1, the various systems are as follows:

System #1: Large composite LUT indexed using RGB - equation (3.10)

System #2: Separate YUV LUTs, with integer division used to map RBG to YUV – equation (3.11)

System #3: Separate YUV LUTs using the >> operation to map RBG to YUV – equation (3.12)

System #4: New colour space (***Y'U'V'***) and separate **Y'U'V'** LUTs – equation (3.13)

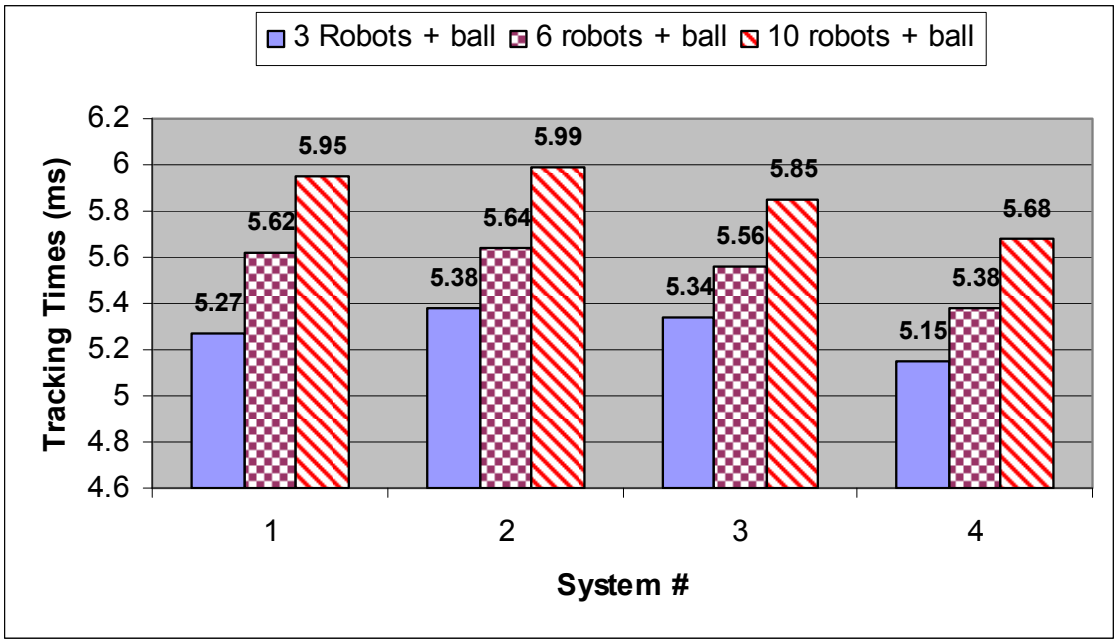


Figure 3.22 Comparison of incremental tracking time

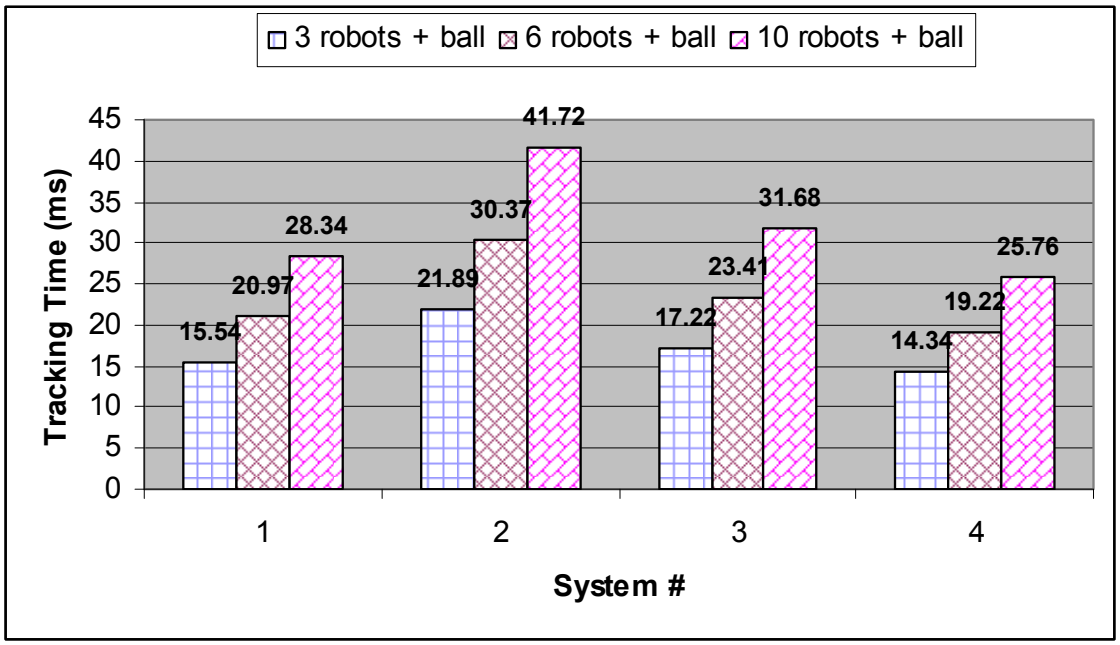


Figure 3.23 Comparison of full tracking time

It is worth noting that the spread of the data presented in Table 3.1 was negligible. During the measurement of the tracking times, the variation in the measured values occurred at most in the second place of decimal (i.e. a hundredth of a milli-second) for a particular system. The standard deviation over 1000 measurements was miniscule and negligible.

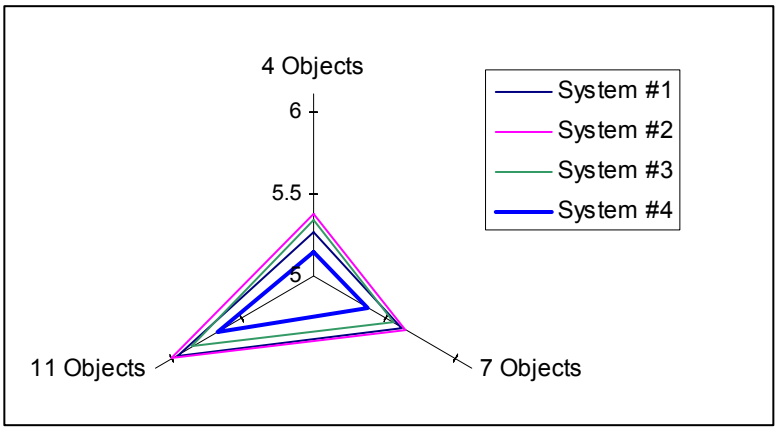


Figure 3.24 Chart showing the performance of the four systems for incremental tracking

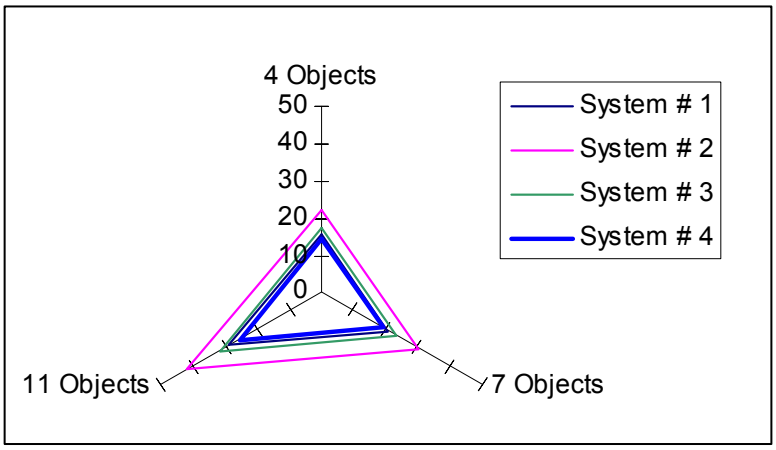
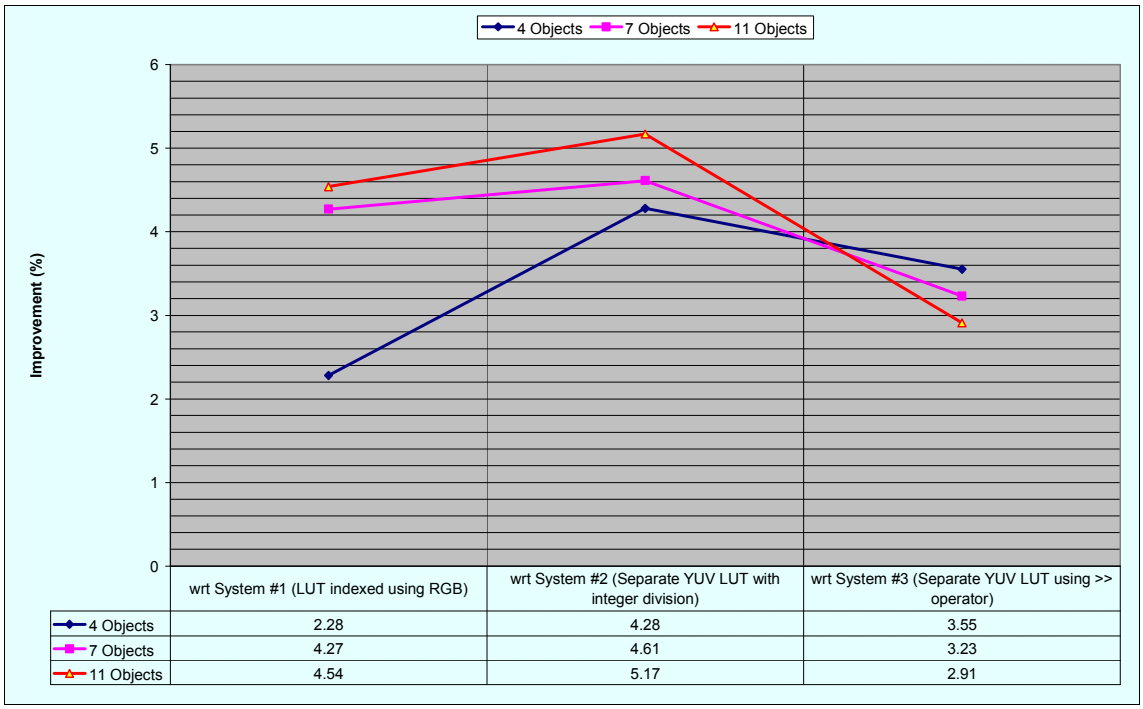
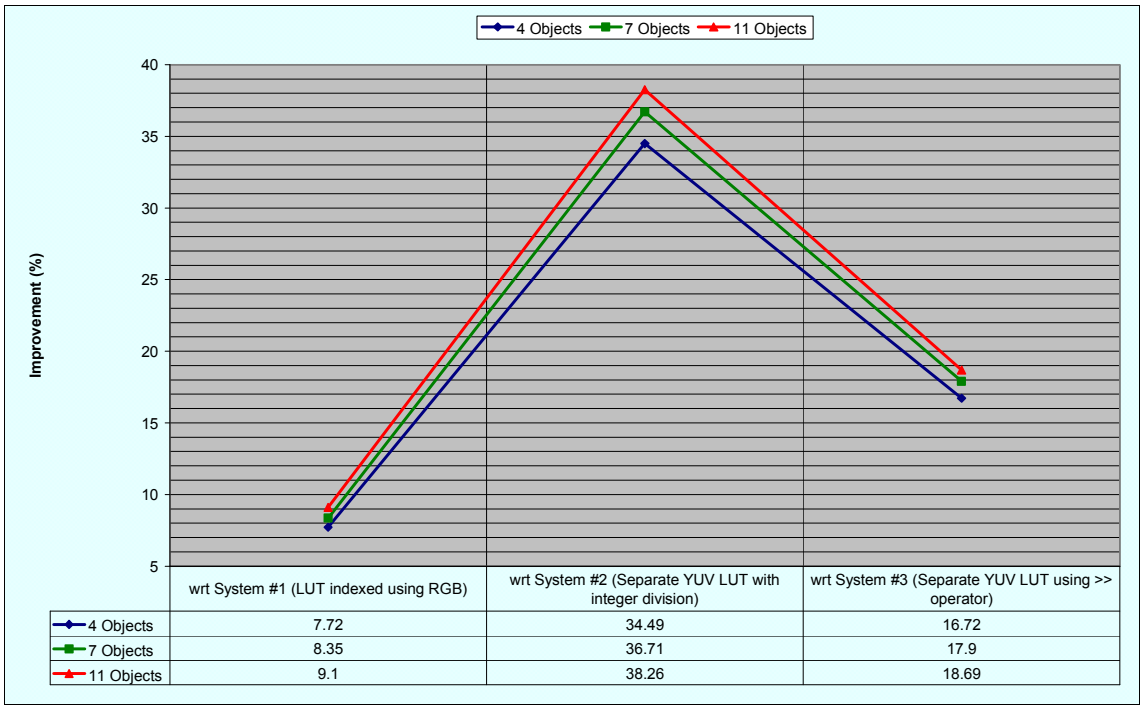


Figure 3.25 Chart showing the performance of the four systems for full tracking



	wrt System #1 (LUT indexed using RGB)	wrt System #2 (Separate YUV LUT with integer division)	wrt System #3 (Separate YUV LUT using >> operator)
4 Objects	2.28	4.28	3.55
7 Objects	4.27	4.61	3.23
11 Objects	4.54	5.17	2.91

Figure 3.26 Summary of improvements in processing speed (in %) achieved by Y'U'V' for incremental tracking



	wrt System #1 (LUT indexed using RGB)	wrt System #2 (Separate YUV LUT with integer division)	wrt System #3 (Separate YUV LUT using >> operator)
4 Objects	7.72	34.49	16.72
7 Objects	8.35	36.71	17.9
11 Objects	9.1	38.26	18.69

Figure 3.27 Summary of improvements in processing speed (in %) achieved by Y'U'V' for full tracking

The radar charts in Figures 3.24 and 3.25 show the vision processing time taken by the different systems for incremental and full tracking respectively for 4, 7 and 11 objects. The system using Y'U'V' colour space and discrete look-up-tables takes the least amount of time.

Figures 3.26 and 3.27 highlight the relative improvements achieved in the vision processing time by using the new Y'U'V' colour space for different number of objects. The improvements reported are with respect to the other three systems. As can be seen, the improvements are substantial.

To summarise the experimental results, it can be said that an efficient arrangement of discrete Y, U and V LUTs for fast colour segmentation has been proposed and tested for real-time vision processing applications. A significant reduction in LUT size (and hence the memory requirement) has been achieved. This translates into significant increase in program execution speed for incremental and full tracking of multiple objects, especially on processors with small cache. The time to update the discrete LUT is negligible (8.2 μs for YUV and 19.2 μs for Y'U'V') and hence is very suitable for real-time update. This is a vast improvement over the method that employs a large LUT with indexing using RGB and takes 909 ms to update the LUT. This has laid the foundation which will enable further work to be done to make the LUT 'adaptive' in order to cater to variation of light intensities and colour distortions, possibly due to reflections from nearby objects.

To enhance the gains derived from the discrete YUV LUTs, the proposed new colour space, Y'U'V', further simplifies the transformation from RGB to a YUV-like colour space. The time to fully track 7 objects, scattered on the filed in any position, reduces from 30.37 ms (using YUV LUT) to 19.22 ms (using Y'U'V' LUT), which is an improvement of 36.71%.

Furthermore, the Y'U'V' colour space allows better colour resolution, thereby increasing the robustness of colour classification. The results compare favourably to the colour threshold based approaches discussed in [34].

3.8 Distributed Vision Processing

A standard NTSC format provides a 525 line interlaced signal at 59.94 Hz which gives a maximum vertical resolution of 480 lines on most commodity image capture cards. Processing each field of the interlaced image independently gives an effective maximum resolution of 240x320 pixels. Some quantisation noise is introduced with this interlaced image which is resolved by filtering. Theoretically a higher resolution of 640 pixels can be achieved in the horizontal direction using commodity cards. However since this distorts the shape of the image, it does not provide any significant advantage.

The alternative methods for increasing the effective resolution of the images include the use of specialised vision equipment at a significantly higher cost, or combining the images from more than one camera using multiple capture cards. Each camera will view different parts of the field, giving a combined image of a higher resolution.

In multi-agent collaborative systems, the area of operation for the agents can be quite large, such as a big shop floor. Using a single camera to oversee the entire area is often not feasible. It may not be possible to mount the camera at a greater height to get a bigger field of view. At the same time, using wide angle lenses result in barrel distortions. A possible solution to this problem is to use multiple cameras, each looking at a portion of the field of operation. The data generated by processing individual images (taken from each camera sub-system) will then be combined and utilized in another server.

3.8.1 Distributed System Architecture

The structure of a two-camera system is illustrated in Figure 3.28. One camera views the left side of the field while the other views the right side. There is an overlap region that is viewed by both cameras. The image capture cards are installed in separate PCs enabling parallel processing of the two images.

The identified objects and positions are transferred over a high speed network connection to a strategy server, which amalgamates the two sets of data removing any

inconsistencies and taking the necessary action when any objects are missing. Essentially the strategy server is the higher level interface to the vision system, while each separate camera/capture card and image processing sub-system is a low level image analysis tool.

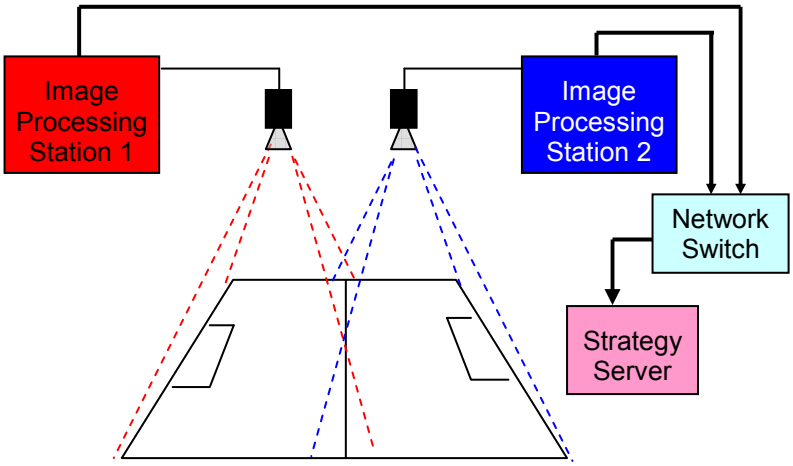


Figure 3.28 Overall system architecture of distributed vision processing

The planar calibration of the two-camera system is done using the boundary and centre line markings as reference points. With a suitable choice of lens and positioning of the cameras, the images are not significantly distorted or rotated. Standard translation and scaling operations give the transformation from image co-ordinates to real world co-ordinates [35].

3.8.2 Integrating the Distributed System

The integrated system builds on the reliable performance of the individual distributed image processing subsystems.

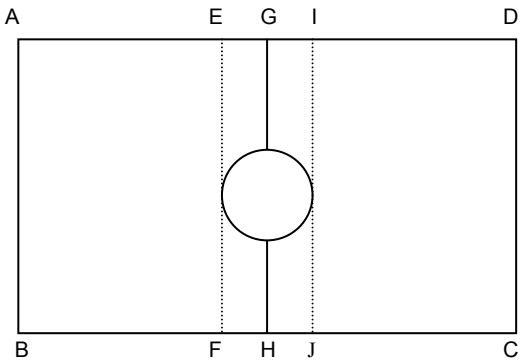


Figure 3.29 Overlap regions (region for full-tracking)

The two cameras overlooking the field have an overlap region (□ EFJI) as shown in Figure 3.29.

If both the cameras were in incremental tracking mode, the processing time would be well within 16.667 ms. However, a problem occurs when an object moves out of the view of the camera, at a certain vertical position, and later re-enters at a different vertical position. Having 'lost' the object, the vision system does not know the last position, which is required by incremental tracking to define the tracking window position in the next frame. Moreover, there is no communication between the two cameras; hence one camera cannot tell the other about the last know position of an object. One way to overcome this problem is by putting the camera in full tracking mode, so upon entry of an object in its view, it will be tracked. This, however, will increase the processing time.

In the proposed distributed system, a combination of incremental tracking and full tracking is employed. The overlap region is tracked fully in addition to incremental tracking of the complete view of a camera. For the camera looking at the left half of the field, the full tracking region is (□ EFJI) and the incremental tracking region is (□ ABJI). For the camera looking at the right half of the field, the full tracking region is the same (□ EFJI) and the incremental tracking region is (□ EFCD). The overlap region allows an object to be tracked in both windows. This adds to the overall processing time by a factor dependant on the size of the overlap region; however it improves the robustness of the system as any calibration error between the two cameras can be reduced.

Using this method, the processing time is 13ms. This is well within the limits of 16.667ms. Since the region where an object can enter the view of a camera is fully tracked, the object is never lost by the total system.

3.8.3 Role of the Strategy Server

In the presented system, an object can be in the overlapping region and picked up by both the cameras. Since the two camera systems are not synchronised, they may report slightly different positions to the strategy server. The strategy server resolves this problem by allocating responsibility for the position of an object to one of the subsystems. The transfer of responsibility is based on the distance from the centre line. This is implemented with a sensor hysteresis term which is larger than the sensor noise. This inhibits the system from rapidly switching responsibility between the subsystems.

In rare situations when both the cameras lose an object, the strategy server will issue commands to the two vision servers to initiate full tracking of the entire image to re-capture the object.

Figure 3.30 Image of the left half of the field

Figures 3.30 and 3.31 show the system in operation. Home team robots are labelled G and 1 - 4. The opposition robots are labelled O1 - O5. The ball is labelled B. Identical opposition robots in different windows can be given different labels since there are no distinguishing features of the opponent robots that can be easily identified. The consistent labelling of the opponent robots is ensured by the strategy server. Two opposition robots in identical positions are considered to be the same robot, so a single label would be given to them by the strategy server.

Figure 3.31 Image of the right half of the field

3.8.4 Test Results and Performance Tuning

Achieving the real time performance is a major challenge. The current full scanning algorithm cannot complete processing a 240x320 field within 16.667ms. The incremental tracking algorithm can track 11 objects within 10ms using tracking window sizes of 20x20 pixels for home and opponent robots and 40x40 pixels for the ball.

Adding full tracking in the overlap region (with a size of 15% of the field) raises this to 13ms in total, which is within the sample period of 16.667ms. Increasing the tracking window size to 40x40 pixels for all the objects resulted in the scanning time increasing to 15ms. This is still within 16.667ms. However incremental tracking cannot guarantee that it would reliably track a very high speed object passing through the overlap region, or an object that changes speed rapidly (as in a case of a high speed collision) or in the case where the tracked object becomes obscured. In these scenarios the only reliable solution would be to scan the whole image. Therefore there is an incentive to investigate different algorithms for scanning the whole image.

3.9 Catadioptric Stereo Vision System for Robotic Application

When an agent moves through space it is restricted in its movement by obstacles. The obstacles could be any other object and even other collaborating agents. For efficient movement through space the agent has to know the distance to obstacles and the boundaries of its 'world'. This is especially true for autonomous systems with local vision. Knowing the distance to an object allows for accurate path planning with the intention of avoiding obstacles. Obtaining the distance to objects, or depth information, for artificial systems has been the subject of many studies, and has resulted in many varied methods being proposed [41-53]. The multitude of research papers written on depth perception emphasises the importance of this research. Depth perception is the ability to estimate the distance, with a known accuracy, to other objects in an environment.

Amongst other techniques, stereo imaging has been suggested as an answer to the problem of depth perception for mobile robotics [41, 49]. It uses multiple images of the same scene taken from different camera locations. The multiple images are related in such a way as to provide disparity. Disparity is defined as the relative movement of an object between two or more views. In a stereo imaging system, the cameras are spatially separated, resulting in the disparity being a function of depth. The disparity is found by matching corresponding points in the input images as illustrated in Figures 3.32 and 3.33. Objects closer to the camera have a greater disparity of movement between two images, and this is used to calculate the distance to the objects.

A catadioptric stereo imaging system has been studied and developed for depth perception and is detailed in this section. The system uses one camera and two sets of planar mirrors to create two virtual cameras. A design procedure is presented with the aim of building a compact assembly. This has resulted in an inexpensive and compact system that achieves a depth resolution of 5.8 cm at a working distance of 2 m.

3.9.1 Review of Stereo Imaging

Some stereo methods that have been put forward are [41-53]:

- the two camera conventional stereo method, which places multiple cameras on a common axis focused on the same scene with a known baseline
- a single camera panning across the scene of interest taking multiple images through its arc
- a single camera moving through space and taking multiple images as it moves and
- Catadioptric stereo which is a single camera using mirrors and lenses to focus on a scene and produce multiple images on the same sensor.

A brief explanation of each of these methods follows.

Conventional Parallel Stereo

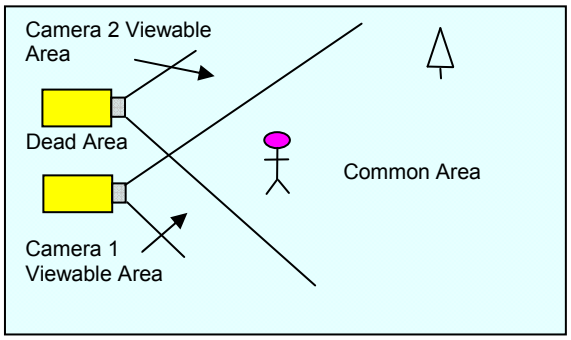


Figure 3.32 Two camera stereo system

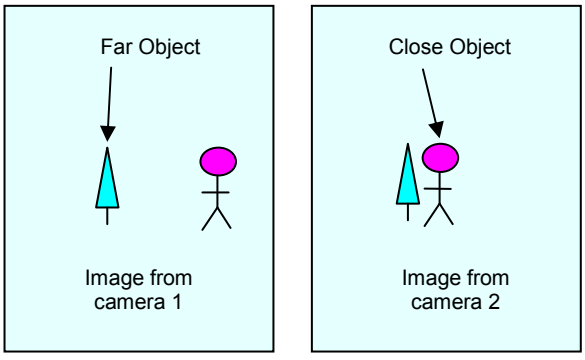


Figure 3.33 Close and far objects seen through a parallel stereo system with two cameras

Conventional stereo vision is usually achieved with two cameras that are mounted in a known relationship to each other and are synchronised to take images at the same instant.

The cameras are usually mounted in parallel (as in Figure 3.32) as this simplifies the geometry. With parallel cameras, an object point appearing in both images will be offset horizontally between the two images, with the offset, or disparity being inversely proportional to the range of the object point. By ensuring that the cameras are parallel, any perspective distortion will be common to the two images, and does not need to be considered in matching points in one image with those in the other. The main distortion in the images is only from the distortions of the camera lenses, so there is less complex post-processing required.

The parallel camera stereo vision is often the method of choice as it is relatively easy to set up. A lot of research has been done on this particular form of implementation, e.g. [41-43].

Stereo with Camera Panning

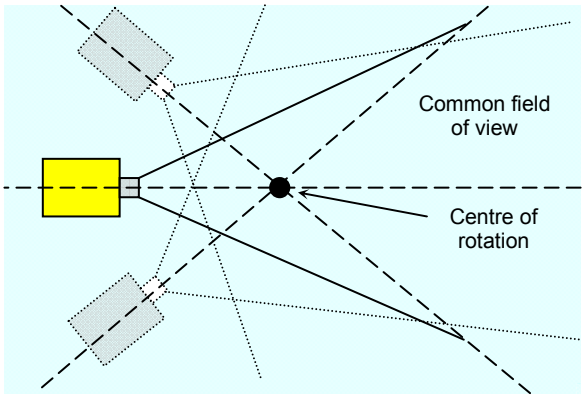


Figure 3.34 Panoramic camera system

Rather than use 2 cameras, an alternative approach is to use a single camera and pan the camera to obtain images from different views. In Figure 3.34, the camera is rotated about a point in front of it. With this system the line of sight converges, which results in a larger field of view common to both the images than with the parallel stereo [44, 45].

When the line of sight of the cameras is not parallel, perspective distortion must be taken into account when matching points in one image with those in the other image(s).

A disadvantage of panned stereo is that the images must be taken at different times. This will limit the speed with which the distances may be sampled.

Stereo from a Moving Platform

A related method, as shown in Figure 3.35, is a system that works on the camera being fixed and taking images as it travels through its environment. This system also works on

multiple images, and uses knowledge of the camera motion between frames to estimate the range to objects.

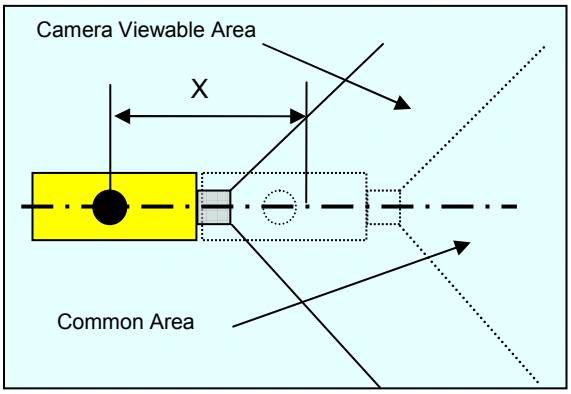


Figure 3.35 Single camera moving through known space system

Complex processing is required to find common pixels in both images, as the camera is usually moving in 3D space, and even if it is only moving in 2D space there are many dynamic distortions that can occur if the camera turns from moving directly ahead. When obtaining stereo images from a moving platform using sequential frames, the range estimation is further complicated if the objects are not stationary.

Catadioptric Stereo

A catadioptric system uses both lenses and mirrors as focusing elements. For stereo imaging the principle is that the lenses and mirrors are designed to act in a way that produces two images on the same sensor as shown in Figure 3.36 [46, 47].

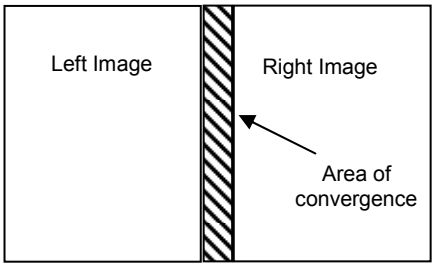


Figure 3.36 Map of the images on the sensor

The advantage of catadioptric stereo is that both images are automatically captured at the same time, giving a faster update rate. This method has several drawbacks. The sensor is split into two images so that approximately only half the resolution of the original sensor is available for each image. There is also an area of convergence between the two images, which means that some of the sensor area is lost.

Many catadioptric systems have been developed using rounded mirrors [48 - 50] to get uni-polar vision, so as to see all around the robot at once. There are also several other catadioptric configurations using planar mirrors, some of which are discussed in [41 - 53].

3.9.2 System Design

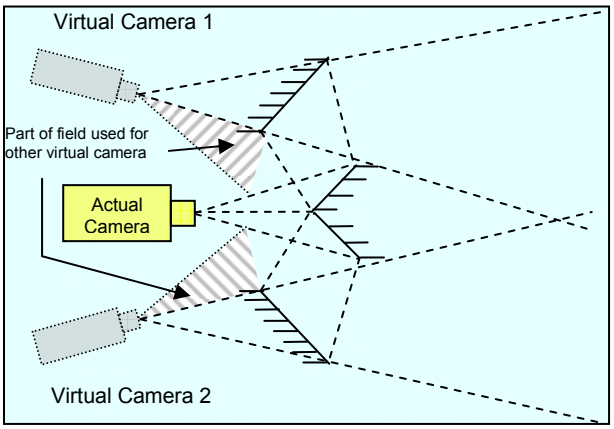


Figure 3.37 Catadioptric stereo vision system

The goal was to design a compact catadioptric stereo system for a mobile robot that has a field of view of 1 meter at a range of 2 meters. The target depth resolution is 5 cm. For a particular application, the design can be suitably modified to cater to a different field of view and depth resolution.

Figure 3.37 shows the system layout, as well as the position of the virtual cameras. Two mirrors are used to split the field of view, and another two to reflect the two new fields of view to overlap in the region of interest.

All the four mirrors are fixed to a base plate, on which the camera is also mounted. The mounting is machined, and designed in such a way that an intricate assembly process is not required to align the system.

System Geometry

The pinhole model can be used to describe the camera. This model has been used to design the mirror system. As there is symmetry in the system design, only half of the mirror assembly has to be designed, while the second half can be taken as an exact replica of the first half.

Figure 3.38 shows the system geometry. The following are the definitions of the symbols used:

ε = sensor size

f = focal length of the lens

θ = angle formed by the outside ray from the sensor that passes through the pinhole, where

$$\tan\theta = \frac{\varepsilon}{2 \cdot f} \tag{3.14}$$

η = distance of the pinhole from the vertex of the inside mirrors

ϕ = inside mirror angle

α = outside mirror angle

D = working distance

FOV = field of view at working distance

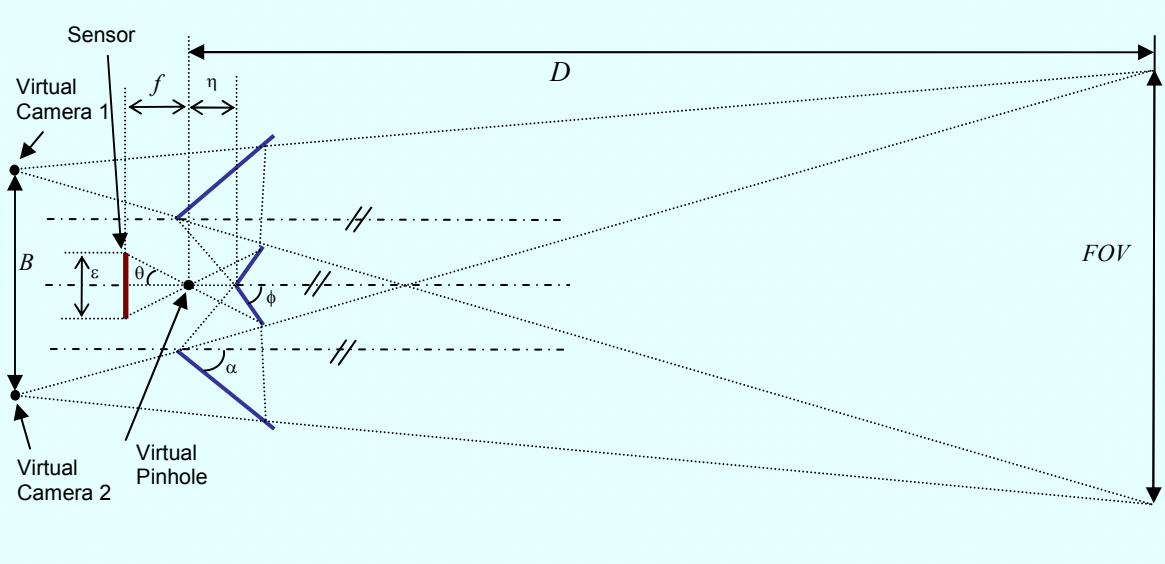


Figure 3.38 The catadioptric system showing the position of mirrors, pinhole and the CCD

For the most compact design, the central ray reflected off the inner and outer mirrors must just clear the outside edge of the inner mirror. To obtain the maximum field of view at the working distance for a particular configuration, the angle of the outside mirror is adjusted so that the inner ray from one view intersects the outer ray of the other view at the working distance.

The field of view is controlled by the angle of view of each virtual image. To a first approximation (and for small θ), the field of view at the working distance is given by:

$$FOV \approx D.\tan\theta \tag{3.15}$$

Rearranging this gives:

$$\tan\theta \approx \frac{FOV}{D} \tag{3.16}$$

By equating (3.14) and (3.16) one can solve for the focal length needed to achieve the desired field of view:

$$f = \frac{\varepsilon}{2\tan\theta} \approx \frac{\varepsilon D}{2FOV} \tag{3.17}$$

This allows the lens to be selected to achieve a particular working width for a given sensor size.

When designing a depth perception system it is important to design the system around the required depth accuracy. Let

B = baseline or the distance between the pinholes in the virtual cameras

λ = the width of one pixel on the sensor

The depth resolution d, at the working depth D can be defined as the change in range that results in a change in the disparity of 1 pixel in the image. Figure 3.39 shows this geometrical arrangement for calculating the depth resolution.

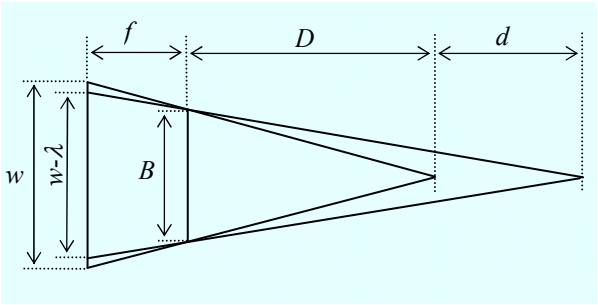


Figure 3.39 Moving an object point by *d* changes the disparity between the images by *λ*

From similar triangles

$$\frac{w}{D+f} = \frac{B}{D} \tag{3.18}$$

and

$$\frac{w-\lambda}{D+d+f} = \frac{B}{D+d} \tag{3.19}$$

Eliminating w from (3.18) and (3.19) and rearranging gives the minimum baseline required to achieve a particular depth resolution as

$$B = \frac{\lambda D}{fd}(D - d) \tag{3.20}$$

or for a given arrangement, the depth resolution is given by

$$d = \frac{\lambda D^2}{Bf - \lambda D} \tag{3.21}$$

Note that because the two virtual cameras have converging lines of sight, the above analysis is only an approximation. The introduction of perspective distortion will mean that the pixel resolution will depend on the actual location of the target point in the image. Depth resolution may also be improved by using sub-pixel matching techniques [54].

For the geometric arrangements of Figure 3.37 and 3.38, the lengths of the mirrors may be found by using the sine rule. The inner mirrors have length:

$$M_{in} = \eta . \frac{\sin\theta}{\sin(\phi\text{-}\theta)} \tag{3.22}$$

and the outer mirrors:

$$M_{out} = \eta . \frac{\sin\theta \sin\phi \sin(4\phi - 2\alpha - \theta)}{\sin(\phi - \theta)\sin(4\phi - 2\alpha)\sin(2\phi - \theta - \alpha)} \tag{3.23}$$

Similarly, the baseline can be derived to be:

$$B = 2\sin(2\phi - 2\alpha)\eta \times \left(1 + \frac{\sin\theta}{\sin(\phi - \theta)} \cdot \left(\frac{\sin\phi}{\sin(2\phi - 2\alpha)} + \frac{\sin(3\phi - 2\alpha + \sin\phi)}{\sin(4\phi - 2\alpha)}\right)\right) \tag{3.24}$$

The Design tradeoffs

The physical size of the system is determined by the application that it is used for. If the mirror assembly is not small enough to be mounted on a medium sized autonomous agent (400mm x 400 mm x 400 mm), then it would not be suitable for autonomous mobile robotic applications.

The use of catadioptric stereo ensures that the two virtual images are captured simultaneously. The major advantage of the single camera stereo system is its low cost. Here only a single camera and frame grabber card are required. The main disadvantage of the arrangement shown here is that the virtual cameras are not parallel. This

introduces perspective distortion, which is inherent to this configuration. If the two cameras are parallel, there would be no common area of view between the two images.

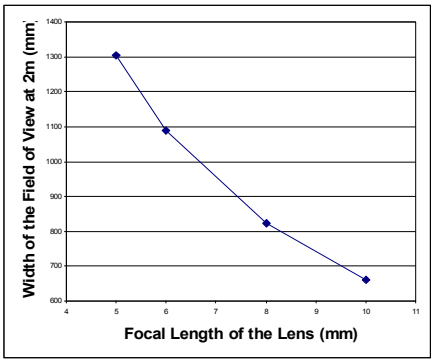


Figure 3.40 Effect of the lens focal length on the field of view for a 6.4 mm x 4.4 mm sensor with a working depth of 2 meters

Design Analysis

Equations (3.14) and (3.15) suggest that the primary factor affecting the field of view is the focal length. This was verified with the relationship shown in Figure 3.40.

A narrow field of view (FOV) will give improved spatial resolution. Since depth resolution also depends on spatial resolution, it will also improve with a narrow field of view.

If the field of view is too narrow then it would restrict the usefulness of the system on an autonomous mobile system, as the robot would not be able to see far enough into its periphery to avoid collision with other mobile systems. At the same time, too large field of view would reduce both the depth and spatial resolution.

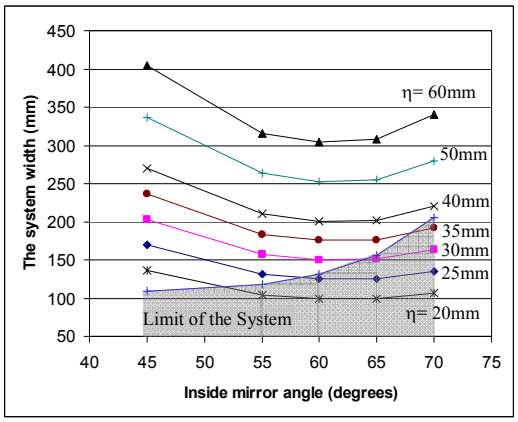


Figure 3.41 Effect of the distance of the pinhole from the vertex of the inside mirrors, on the system size. The shaded region is where the system is self-occluding

The size of the lens and the mirror system are directly proportional to the distance between the pinhole and the inner mirrors. This results from similar triangles and is reflected in equations (3.22) and (3.23). Increasing the angle of the inner mirror (Φ) has the potential to improve the compactness of the design. This is due to the fact that the length of the inner mirror can be decreased, allowing a narrower baseline. This effect is clearly illustrated in Figure 3.41. It can be seen that a mirror angle of approximately 60^o gives the most compact design, keeping the other parameters the same.

Angles greater than 45^o fold the optical path back behind the inner mirror. An additional constraint under these conditions is that the lens itself does not occlude the objects being viewed. The physical size of the lens will limit the distance from the pinhole to the inner mirrors as illustrated in Figure 3.42.

With reference to Figure 3.42, *d* is the width of the lens and *p* is the distance of the pinhole from the front of the lens.

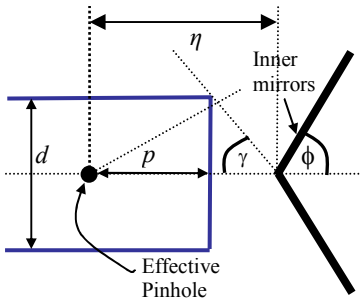


Figure 3.42 The lens and mirror arrangement to show how the size of the lens limits the distance between the pinhole and the vertex of the inside mirrors

The central ray is reflected off the mirrors with angle

$$\gamma = 180 - 2\phi \tag{3.25}$$

From trigonometry, the system is constrained such that

$$\tan\gamma > \frac{d/2}{\eta - p} \tag{3.26}$$

Substituting (3.25) into (3.26) and rearranging gives

$$\eta > p - \frac{d/2}{\tan 2\phi} \tag{3.27}$$

This constraint has been mapped onto Figure 3.41.

The use of a pinhole camera model becomes less valid as the pinhole to mirror distance is decreased. This appears in the image as a region of convergence, or blur, between the two views. The angle subtended by the joint between the mirrors also increases as the lens is moved closer to the mirrors.

3.9.3 Implementation and Calibration

High grade polished aluminium was used for the mirrors because this is less expensive than front silvered glass mirrors. Polished aluminium has the disadvantage of being very easily damaged, and once damaged it can not be easily repaired.

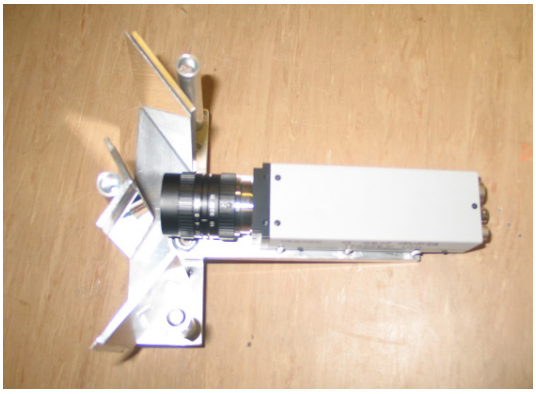


Figure 3.43 The completed catadioptric system

Aluminium was used to build the rest of the system with three 10mm plates being used as supporting structures. The structure can be seen in Figure 3.43.

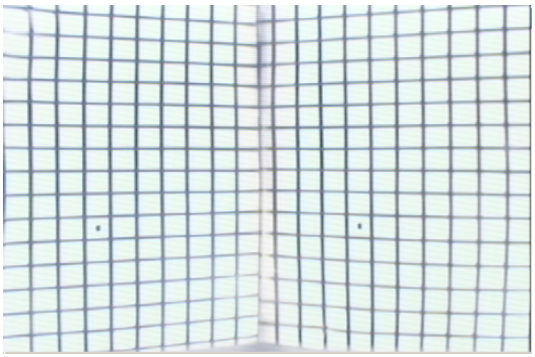


Figure 3.44 The calibration grid seen through the vision system

Figure 3.44 shows the image of a 10 cm spaced grid placed 2 m from the camera. This image may be used to calibrate the system to correct for distortions within the system. These arise from three main sources. The most obvious is the perspective distortion in each of the views. It results from the converging line of sight of the two virtual cameras, and is a direct result of the plane at 2 m not being parallel to the image plane within the virtual cameras. The second source of distortion is barrel lens distortion from the use of a wide angle lens. Magnification in the centre of the image is slightly greater than on the periphery, which results in the characteristic barrel shape of the vertical lines. The third

source of distortion is due to the minor imperfections on the surface of the polished aluminium mirrors. These lead to apparently random deviations or twists in the lines.

A calibration image can be used to provide information that is able to characterise and correct these distortions, allowing the true disparity between the two sides to be measured. The calibration process will result in a pair of images with no disparity for objects at the working distance. After correcting the distortions, a disparity map may then be produced by matching corresponding points between the left and right images. With the distortion removed, standard parallel line of sight stereo algorithms may be used. From the disparity map, a 2½-D depth map may be produced to provide data for robot navigation.

The system is inexpensive and is targeted for use on a mobile robot platform for collaborative tasks. The system achieves a depth resolution of 5.8 cm at a working distance of 2 m.

3.10 Discussion

This chapter discussed the vision related issues that affect the performance of robots in a collaborative system. The demands of a real time system, especially when the autonomous agents are very fast mobile robots, are enormous. The image processing has to be not only very fast but also accurate. Traditionally techniques such as incremental tracking, off-screen multi-buffered image capture and use of YUV colour space (instead of RGB colour space) have been used to improve the speed of image processing and to enhance the reliability of object detection. One of the drawbacks of commodity vision hardware is that they generally capture and output colour information in RGB. Before a pixel can be processed, its RGB values need to be converted to YUV, which is very time consuming. For an image of moderate resolution, say 320x240 pixels, and more than 10 objects to locate, it is impossible to complete the processing of a full scan in real time using even the most modern hardware. This backdrop was the motivation behind the research presented in this chapter.

A novel feature described in this chapter is the new Y'U'V' colour space which has been proposed and thoroughly tested with varying number of objects. Transformation from the RGB colour space to Y'U'V' colour space is very simple and computationally very inexpensive. It uses discrete look-up-tables for Y', U' and V' thresholds. The look-up-tables are very small, requiring very little memory and can be updated very fast. In the graphical user interface of the robot soccer game, shown in Figure 3.45, it can be seen that the vision processing time is 5.67ms. The comprehensive test results of the look-up-table update time indicate that using this colour space it will be possible to build an adaptive colour thresholding technique which will be of immense value in

varying lighting conditions. Also, because of the small memory and processing power requirements, the look-up-tables and the associated processing algorithms can easily be implemented on tiny microcontrollers. Figure 3.46 shows the dialog box to manually alter the lower and upper threshold of the Y', U' and V' for all the colour classes for a robot soccer application. The use of Y'U'V' colour space also enables detection of objects with very small colour patches. As shown in Figure 3.47, the vision system can be set up to detect colour patches with as few as 8 pixels.

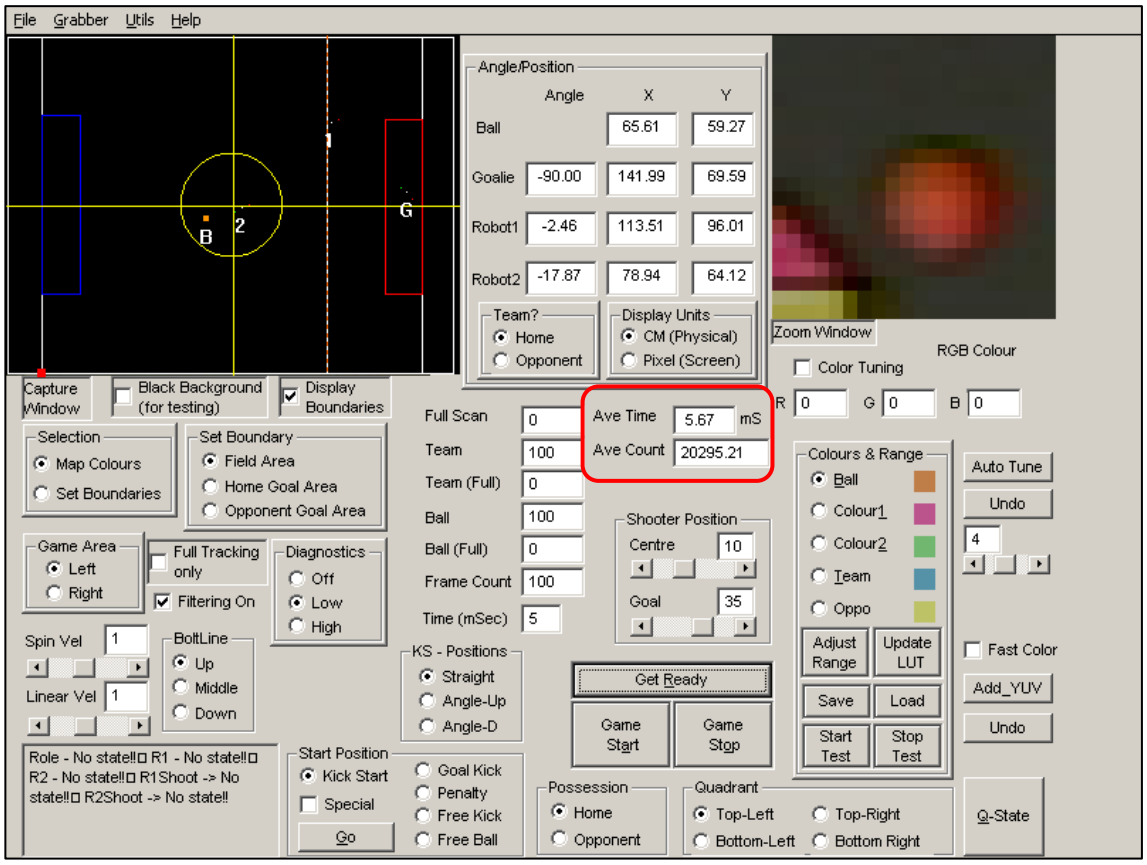


Figure 3.45 Measuring average tracking time

In a global vision based system, it is often not possible to oversee a large area using a single camera. The use of multiple cameras in a distributed architecture has also been presented in this chapter. Each camera looks at and processes part of the field. Multiple cameras, working in parallel, greatly improve the system throughput.

In many applications, autonomous robots must carry their own vision system. While in a global vision based system, the robots rely on the data from the central controller to avoid obstacles, in a local vision based system, the obstacle detection and avoidance must be done locally. In order to move efficiently, the robot must be able to perceive the distance to obstacles. The design of a catadioptric stereo vision system, using a single camera, has been presented. This eliminates the need to use two cameras for depth perception and thus reduces costs.

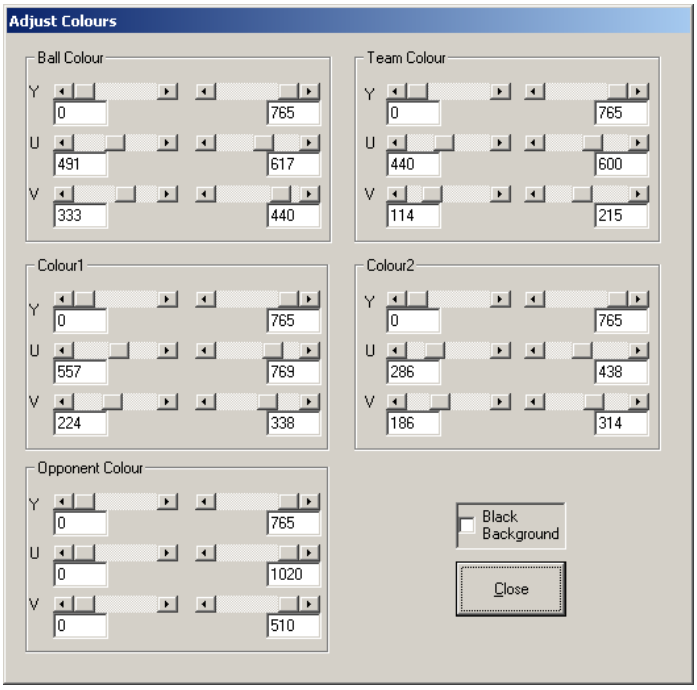


Figure 3.46 Dialog box to adjust the thresholds for colour classes

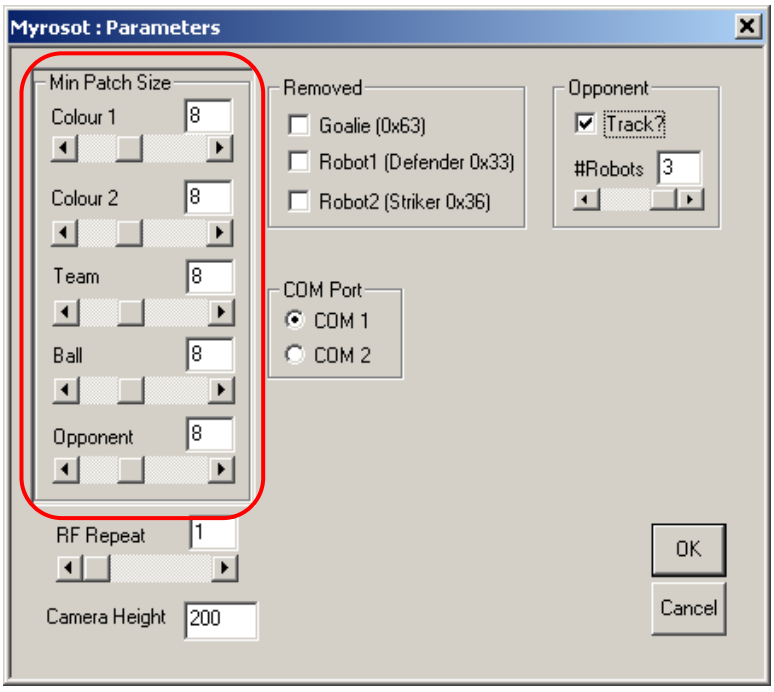


Figure 3.47 Dialog box to set minimum patch size for colour classes

Quantisation noise due to separate processing of odd and even scans of an interlaced image; inherent camera sensor noise and noise due to variation of ambient light intensity all stack up to adversely effect the accuracy of object detection. This translates into poor performance of the robots while cooperating to accomplish a task. To overcome the vision related noise, filters are required to be used. These are explained in Chapter 5.

4 Collaborative Operation Strategy: State Transition Based Control (STBC)

Often distributed control systems are employed in production processes in which robots must cooperate and collaborate with humans and/or other robotic systems. Complex distributed robot control systems consist of mechanical systems powered by actuators that are under the control of computer systems that rely on sensor input, such as vision, touch, infrared, ultra-sound and torque sensing. This complexity requires a control framework in which the various interacting components can be managed. This chapter introduces the state transition based control (STBC) techniques for managing the behaviour of collaborative robotic systems.

Generally, distributed systems are inherently more complex than individual isolated systems. The complexity is further aggravated by the various interactions that can occur between the distributed components. These interactions include explicit communication via message passing as well as implicit communication via physical interaction and resource contention. Managing this complexity is one of the most significant challenges in distributed robotic systems. As low-level control problems are solved, managing the high level supervisory control of behaviour becomes the main focus of attention.

Several approaches have been reported in the literature for managing complexity [55, 56]. Most of them are related to hierarchical control system architectures. That is, the detailed complexity is abstracted away in subsystems that are considered to be primitive or atomic. The advantage of the hierarchical approach is that at the higher level the complexity of the problem is reduced and so it is suitable for automatic solution or even manual solution. A disadvantage of rigorous abstraction using hierarchical approaches is that the optimal solution may be hidden by an inappropriate hierarchical decomposition. If an automated system has to create the hierarchical decomposition, the problem solution is not feasible.

In multi-agent collaborative systems, several approaches have been investigated for creating and analysing the hierarchical decomposition of the system. For example, [57] proposes a layered disclosure technique that allows the software agents in the system to act and present their actions at several levels of abstraction. An alternate approach is to adopt a simplified reactive architecture as in the subsumption method [58]. In this approach instinctive behaviour is developed as reactions to the current sensor input. Little modelling of the environment, in terms of any internal state, is required.

Essentially, the model of the world is available through the sensors so there is no need for it to be explicitly modelled.

In the context of robot soccer, several international teams have built the behaviour and control strategy using "conditional program structures". This makes the maintenance and upgrade of strategy a very time-consuming and tedious process. This is evident from the fact that the teams spend several hours between competition matches to change and tweak their strategy. This method is also error prone. A better approach and methodology is thus required.

To build complex behaviours relatively quickly and to manage the complexity in a dynamic collaborative system, a hierarchical state transition based control (STBC) method has been proposed and implemented. Amongst the many advantages of this approach is the data and process abstraction, which aids development of new functionalities by creating new states and building a hierarchy of such functionalities.

4.1 States: The Fundamental Component of STBC

The fundamental component of the State Transition Based Control (STBC) technique is the state itself. A state is represented by the current physical condition of the system (defined by the input sensors) as well as the previous condition (memory) of the system. The range of inputs is continuous and so infinite in variety, but the states are discrete and finite. The current state will determine the current action or behaviour. However as sensor input varies, the system will transit between states, thereby changing actions or behaviour.

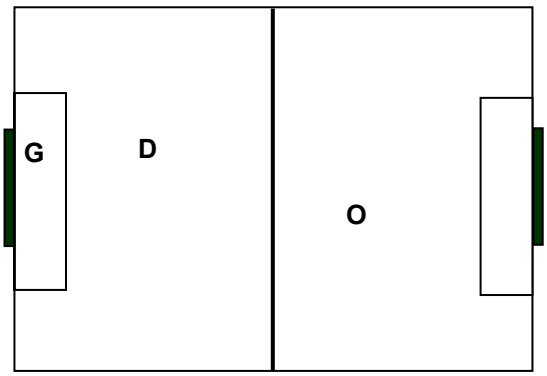


Figure 4.1 Goal area, defensive and offensive regions of a robot soccer field

This concept can be illustrated with a simplified example of the goalkeeper behaviour in robot soccer. The soccer field, shown in Figure 4.1, is divided into several regions which are marked as **G**, **D** and **O**. These are called the *decision regions*. The global vision system (camera + frame grabber) detects the position of the ball, which can be in

one of these regions. Depending on the position of the ball on the field, three different behaviours of the goalkeeper may be defined-

1. Clear the ball, when the ball is within the goal area (region **G**)
2. Track the ball position and position the robot accordingly (within the goal posts) when the ball is within the halfway line but outside the goal area (region **D**)
3. Idle in the goal centre position, when the ball is over the halfway line (region **O**)

A *state* encapsulates behaviour, hence these three behaviours represent the three *states* of the goalkeeper. The main determining point for the robot to be in a particular *state* is the position of the ball, but the action or behaviour that the robot was already engaged in is also important. If the robot is clearing the ball, the ball must be cleared fully (i.e. by a certain pre-determined distance), and the action must not terminate as soon as the ball has rolled out of the goal area (region **G**).

Therefore, conditions are required for the robot to enter a particular state, but the conditions for the robot to remain in a particular state and to leave it must also be defined. The transition conditions to enter and leave a state are often different and in general not symmetric.

4.2 State Transition Diagram

The different states for a particular behaviour can be viewed on a state transition diagram as shown in Figure 4.2 for the goalkeeper operation. It is usual to have an *initialisation state*, which is entered at the start of a particular operation or task. It is generally denoted as state 0 or S_0. The data that are required throughout the lifetime of the operation can be initialised in the initialisation state. The robot will leave the initialisation state and enter one of the valid *behavioural states* and thereafter transit from one state to another as the sensor input and internal conditions vary. In the example of the goalkeeper behaviour, the three behavioural states are *Clear Ball* (denoted as S_1), *Track Ball* (denoted as S_2) and *Idle* (denoted as S_3).

To change state, the state transition conditions have to be met. For the goalkeeper behaviour, the state transitions from the initialisation state are determined by the position of the ball. The various state transition conditions, that can be, are enumerated in Table 4.1. S_xS_y represents the condition to transit from state S_x to S_y.

As can be noted from the state transition diagram, there is no direct transition path from S_1 to S_3 and vice-versa. This is so because the ball cannot physically travel from the region **G** to region **O** without going through region **D** and the other way around. In this particular example, there is the possibility of only one state to move to from a particular state, except the initialisation state. However, in many other real world operations, it is

quite possible to transit from a behavioural state to one of several other states depending upon which state transition condition is satisfied.

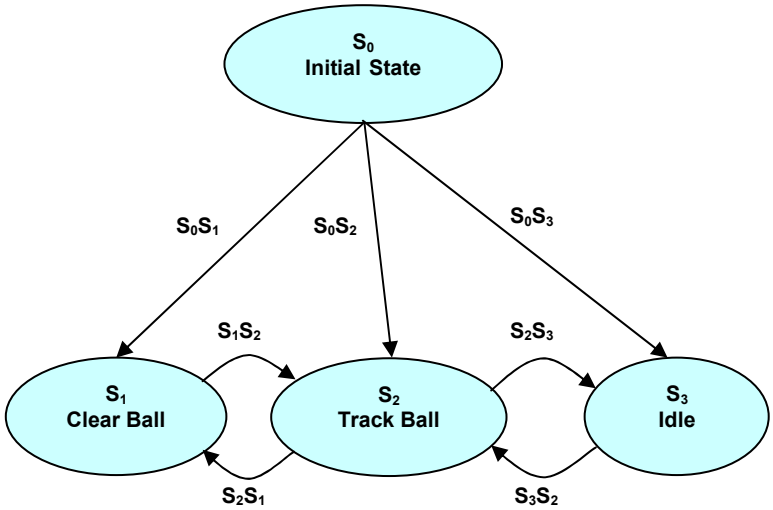


Figure 4.2 State transition diagram of goalkeeper action

Table 4.1: State transition conditions for goalkeeper behaviour

S_0S_1	Ball position is within the goal area
S_0S_2	Ball position is between the goal area and the centre line
S_0S_3	Ball position is beyond the centre line
S_1S_2	Ball position is outside the goal area
S_2S_3	Ball position is beyond the centre line (i.e. same as S_0S_3)
S_3S_2	Ball position is below the centre line
S_2S_1	Ball position is within the goal area (i.e. same as S_0S_1)

4.3 Maintaining State with Hysteresis

The conditions for state transitions listed in Table 4.1 are problematic when the ball is near the decision boundaries. Most real systems exhibit some measure of sensor noise. Even when the ball is not moving, there is a variation in its position that is measured by the vision system. The sensor noise may have the effect of the ball appearing to move between the defensive and goal regions (if it is on the boundaries of the penalty box) or defensive and attacking/offensive regions (if the ball is on the centre line). This would make the robot oscillate between the different states and hence switch rapidly between the actions required in the different states. To overcome this problem a hysteresis factor, larger than the sensor noise, is required. This hysteresis is termed as the *sensor hysteresis factor.*

The transitions between the states S_1 and S_2 are then modified versions of the transitions between S_0-S_1 and S_0-S_2. Similarly the transition conditions between S_2 and S_3 are modified to incorporate the sensor hysteresis factor. The sensor hysteresis factor for the

different decision boundaries on the field of operation will generally be the same. The modified state transition conditions are shown in Table 4.2.

Table 4.2: Modified state transition conditions for goalkeeper behaviour incorporating sensor hysteresis factor

S_1S_2	Ball position is outside the goal area **plus** a certain distance larger than the sensor noise
S_2S_1	Ball position is within the goal area **minus** a certain distance larger than the sensor noise
S_2S_3	Ball position is beyond the centre line **plus** a certain distance larger than the sensor noise
S_3S_2	Ball position is below the centre line **minus** a certain distance larger than the sensor noise

Another reason why a particular state might need to be maintained is when an action has to be completed before the transition to another state can be made. In the case of the goalkeeper's behaviour, this can be illustrated by the *clear ball* action. When the goalkeeper kicks the ball out of the goal area it would be dangerous to leave the ball as soon as it goes beyond the penalty box. So a *behavioural hysteresis factor* needs to be introduced. The state transition conditions incorporating behavioural hysteresis factor are shown in Table 4.3.

Table 4.3: Modified state transition conditions for goalkeeper behaviour incorporating sensor hysteresis factor and behavioural hysteresis factor

S_1S_2	Ball position is outside the goal area **plus** a sensor hysteresis factor, **plus** a safe kick-out distance (*behavioural hysteresis factor)*.
S_2S_1	Ball position is within the goal area **minus** a sensor hysteresis factor, **minus** a safe roll-in distance
S_2S_3	Ball position is beyond the centre line **plus** a sensor hysteresis factor **plus** a safe roll-over distance
S_3S_2	Ball position is below the centre line **minus** a sensor hysteresis factor, **minus** a safe roll-back distance

It should be noted that the behavioural hysteresis factor for the different boundary conditions need not be the same. It will largely depend on the overall operation and the actions that are performed in the states.

Using an object oriented approach it is possible to add the sensor hysteresis and behavioural hysteresis into a super class definition, so that the subclass can inherit this behaviour. This will greatly simplify the software design and implementation.

4.4 Hierarchy of States

Complex behaviour cannot easily be reduced to simple primitive actions. Normally the action that is required in each state is itself a complex behaviour. For example the *Clear*

Ball action in the goalkeeper behaviour cannot simply be to kick the ball, since this might result in scoring an own goal! The goalkeeper must always be careful in approaching the ball so that the ball is not accidentally knocked into its own goal. This can be achieved by creating a state based *Clear Ball* operation/action. The state transition diagram for the *Clear Ball* operation is shown in Figure 4.3 and the state transition conditions are tabulated in Table 4.4

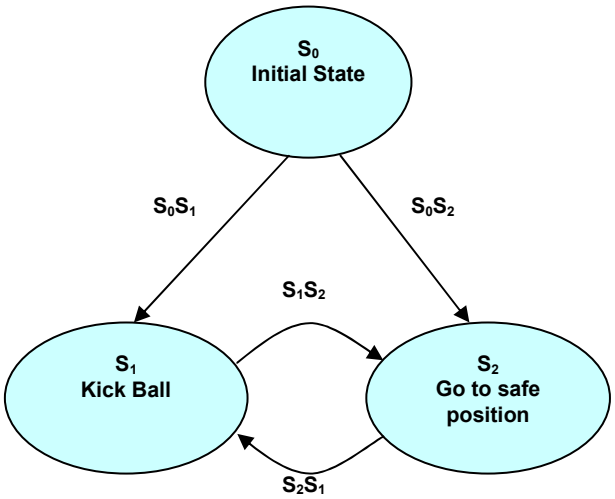


Figure 4.3 State transition diagram of the *Clear Ball* operation

Table 4.4: State transition conditions of *Clear Ball* operation of goalkeeper

S_0S_1	Robot behind the ball position
S_0S_2	Robot ahead of the ball position
S_1S_2	Same as S_0S_2
S_2S_1	Same as S_0S_1

The state based *Clear Ball* action can be coded using a similar methodology as that for the state based goalkeeper operation. The state transition conditions between S_0, S_1 and S_2 must be defined. Due care must be taken to set appropriate sensor and behavioural hysteresis factors. The actions required in each state must also be defined.

The actions "Kick Ball" and "Go to safe position" are low level behaviours implemented in a similar manner to [59]. Since each robot potentially has a large hierarchy of behavioural states, a single state variable is not enough; rather an array of state and sub-state variables must be maintained so that the robot can keep a record of what the current behaviour should be. A consequence of this hierarchy is that when a state transition occurs at a higher level, the sub-states would have to be initialised, since the sub-state behaviours are not normally carried over higher state transitions. An example of this is seen in the **`runAction`** function of the goalkeeper behaviour code in Appendix A.

4.5 Role Selection in Robot Soccer

Having described the STBC methodology, a complete role selection mechanism for collaborative tasks, such as those in a robot soccer system will be presented. This section describes the *Role Selection Layer* for two collaborating robots. In some applications it may be necessary to assign a fixed role to each robot. In the case of robot soccer, where the agents are physically homogenous, it is possible to allocate roles (goalkeeper, defence and offence) to them dynamically.

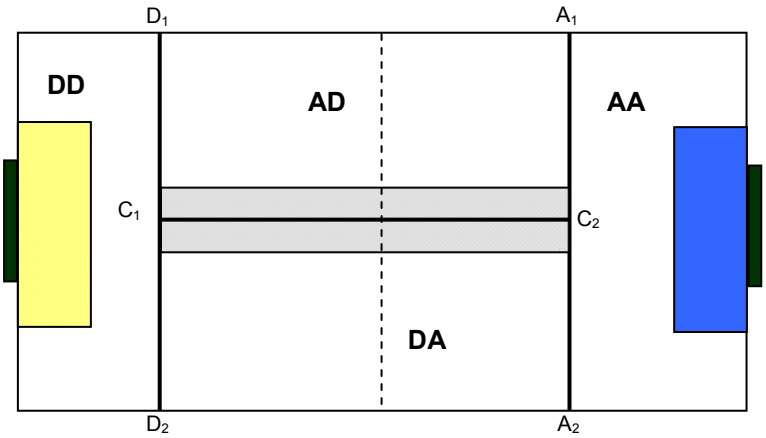


Figure 4.4 Boundaries for robot soccer role selection

Figure 4.4 shows the decision regions of the field of operation where both the robots adopt defensive behaviour (region marked **DD**), attacking behaviour (region marked **AA**) and where one acts defensively and the other attacks (regions marked **AD** and **DA**).

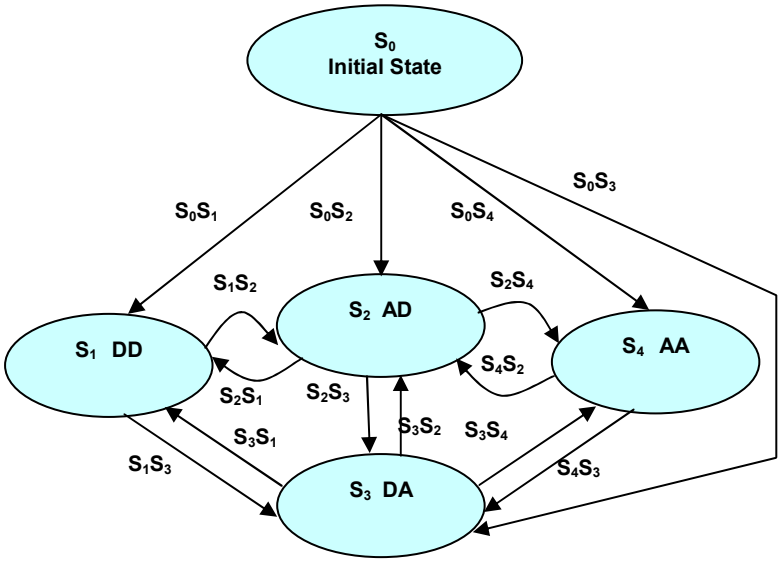


Figure 4.5 State transition diagram for role selection

The role selection is determined by the state transition diagram shown in Figure 4.5. The roles adopted by the two robots either as an attacker (A) or a defender (D) is given by the position of the ball. The shaded region around the horizontal centre line defines the hysteresis for transitions between **AD** and **DA** behaviours. In this example one robot is statically assigned the goalkeeper role. However, this formulation can be extended to the goalkeeper and it can also be dynamically assigned other roles.

The transitions between states are triggered by the movement of the ball across the region boundaries specified in Figure 4.4. A sensor hysteresis factor is applied to the boundaries as well as a behavioural hysteresis factor is applied that allows the attack or defence action to complete. The state transition conditions for role selection are given in Table 4.5. Sensor hysteresis factor and behavioural hysteresis factor are not shown but are implied.

Table 4.5: State transition conditions for role selection

S_0S_1	Ball position is below (i.e. left of) line D_1D_2
S_0S_2	NOT(S_0S_1) AND NOT(S_0S_4) AND ball position above line C_1C_2
S_0S_3	NOT(S_0S_1) AND NOT(S_0S_4) AND ball position below line C_1C_2
S_0S_4	Ball position is beyond (i.e. right of) line A_1A_2
S_1S_2	Ball position is beyond (i.e. right of) line D_1D_2 AND ball position above line C_1C_2
S_2S_1	S_0S_1
S_2S_3	NOT(S_2S_1) AND NOT(S_2S_4) AND (ball position below line C_1C_2 minus overlap distance)
S_3S_2	NOT(S_2S_1) AND NOT(S_2S_4) AND (ball position above line C_1C_2 plus overlap distance)
S_2S_4	S_0S_4
S_4S_2	Ball position is below (i.e. left of) line A_1A_2 AND ball position above line C_1C_2
S_3S_4	S_0S_4
S_4S_3	Ball position is below (i.e. left of) line A_1A_2 AND ball position below line C_1C_2
S_3S_1	S_0S_1
S_1S_3	Ball position is beyond (i.e. right of) line D_1D_2 AND ball position below line C_1C_2

The individual *Defensive* and *Offensive* behaviours of the robots are described below in sub-sections 4.5.1 and 4.5.2 respectively.

4.5.1 Defensive Behaviour

Figure 4.6 shows the decision regions of the field where the defensive robot adopts a *support-goalie* (region marked as **SG**) behaviour, a *defending behaviour* (region marked as **D**) and a *support-attacker* (region marked as **SA**) role.

Figure 4.7 shows the state transition diagram of the defensive behaviour. As well as the **SG**, **D** and **SA** behaviours, the state transition diagram shows a *backing-off* (**BO**) behaviour state. The transitions between state **SG**, **D** and **SA** are determined by the position of the ball as shown by the boundaries in Figure 4.6. A sensor hysteresis term is added to these transitions. The additional **BO** state is added so that there is no conflicting condition in which both the robots are trying to kick the ball. The transition S_2S_4 occurs when the other robot of the team has the best position to shoot or pass the ball, while the S_4S_2 transition occurs when the robot under discussion has the best position to shoot or pass. In this way any behavioural conflict is resolved.

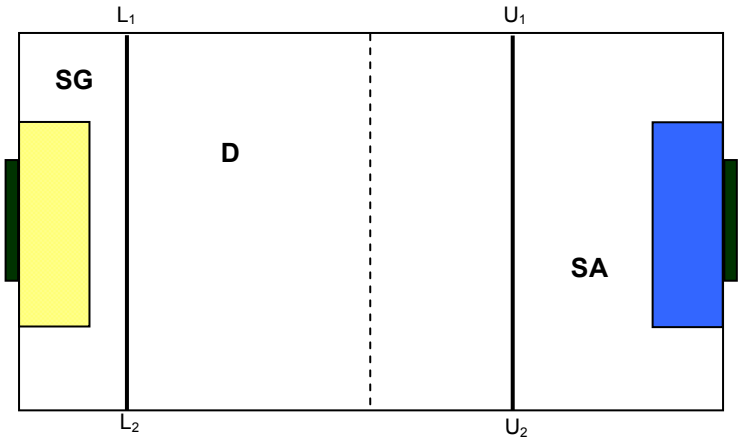


Figure 4.6 Boundaries for defensive behaviour

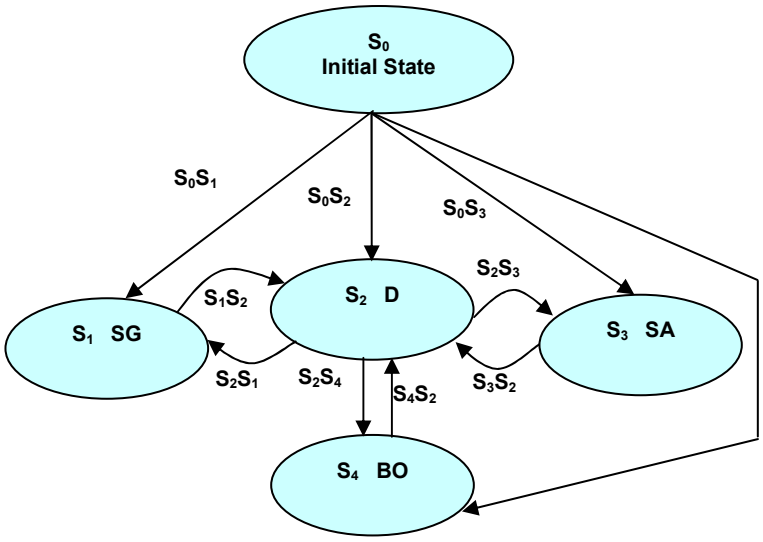


Figure 4.7 State transition diagram of the defensive behaviour

The **SG** state is a defensive state in which the robot idles in a position to support the goalkeeper. Depending on the position of the ball, this may be in line with the ball, waiting for the goalkeeper's pass, or away from the ball allowing the goalkeeper to kick

out the ball. The **SA** state allows the defender to support the action of the attacker. This will normally be a position behind the attacking robot and the ball.

4.5.2 Offensive Behaviour

Figure 4.8 shows the region of the field where the offensive robot adopts a *support-goalie and defender* (region marked as **SGD**) behaviour, *shoot* behaviour (region marked as **CShoot**) and a *leave ball* (region marked as **L**) behaviour.

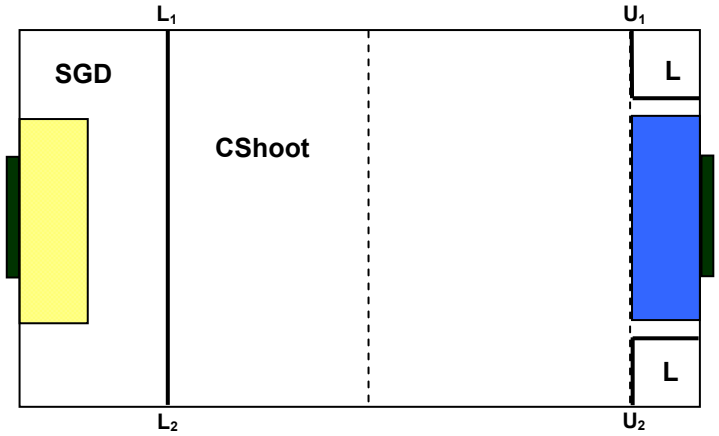


Figure 4.8 Boundaries for offensive behaviour

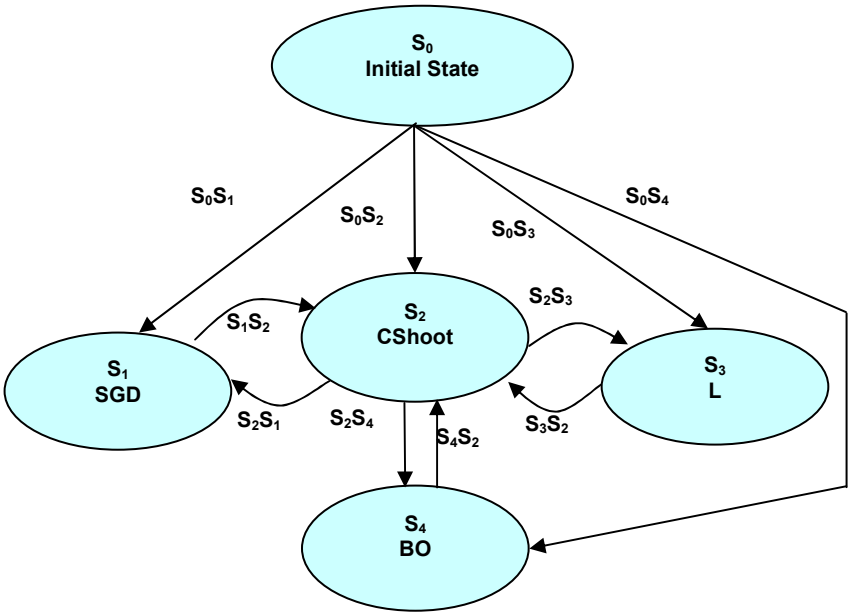


Figure 4.9 State transition diagram of the offensive behaviour

Figure 4.9 shows the state transition diagram of the offensive behaviour. Similar to the defensive behaviour, the offensive behaviour makes use of a sensor hysteresis term between the **SGD**, **CShoot** and **L** states. A behavioural hysteresis term inhibits the

transitions from the backing off (**BO**) state. This ensures that two or more robots do not adopt conflicting actions while shooting or passing the ball.

The **SGD** state is similar to the defenders **SG** state except the attacking robot is in a position clear of both the goalkeeper and the defender. Its role in some positions is to await a pass from the defender or goalkeeper and in other positions to disrupt the attacking flow of the opposition.

The **L** state is a situation in which the attacking robot is unlikely to score from. In this position the attacking robot idles allowing the opponent's goalkeeper or another opponent robot to clear the ball.

4.5.3 State Transition Based Shoot Action

The **CShoot** action itself is a complex behaviour and is implemented using a state transition based control. To take a good shot towards the goal, the robot must not only be positioned behind the ball, it must also be in a favourable orientation towards the target. If the robot is past the ball, it must go to a 'side position' before it can attain a good 'behind position'. Moreover, if the calculated 'behind position' is outside the boundaries of the play ground, the robot enters a state in which the action it performs is dribbling the ball along the wall. The states and the transition links are shown in Figure 4.10.

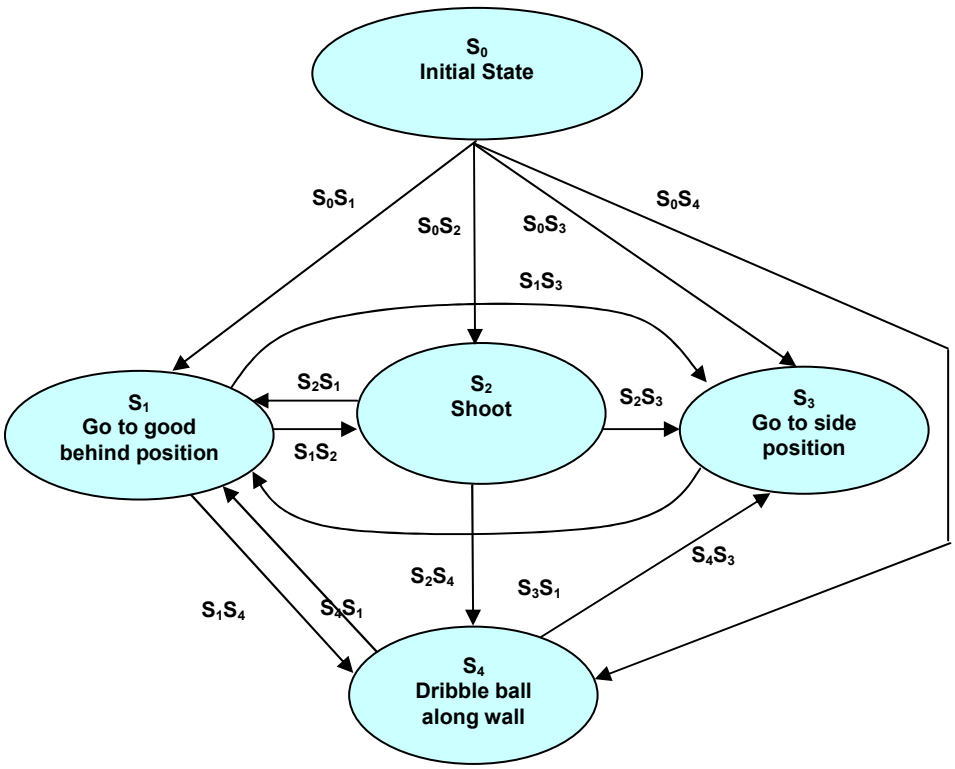


Figure 4.10 State transition diagram of *CShoot* action

The state transition conditions for the *CShoot* action are summarised in Table 4.6.

Table 4.6: State transition conditions for *CShoot* action

S_0S_1	NOT(S_0S_2) AND NOT(S_0S_3) AND NOT(S_0S_4)
S_0S_2	Robot behind the ball and good orientation towards target
S_0S_3	Robot past the ball
S_0S_4	The calculated '*behind position*' is outside the boundaries of the playground
S_1S_2	S_0S_2
S_2S_1	Robot-to-ball distance > 20 cm AND ball-to-target angle is more than ± 50^o
S_2S_3	S_0S_3
S_1S_3	S_0S_3
S_3S_1	Robot-to-ball distance > 10 cm AND ball-to-target angle is within ± 110^o
S_1S_4	S_0S_4
S_4S_1	'*behind position*' inside the boundaries of the playground
S_4S_3	S_0S_3 OR NOT(S_0S_4)
S_2S_4	S_0S_4

4.6 Auto Placement of Robots

In many collaborative tasks, agents have to take up predefined positions and orientation. This could be at the start of the cooperative effort or at the conclusion of a task. Such actions or behaviours can be very easily implemented using the STBC technique. The biggest advantage is that the complexity of the system effectively remains unchanged even when the number of agents increases. A simple state transition diagram, shown in Figure 4.11, will suffice for auto placement of any number of robots.

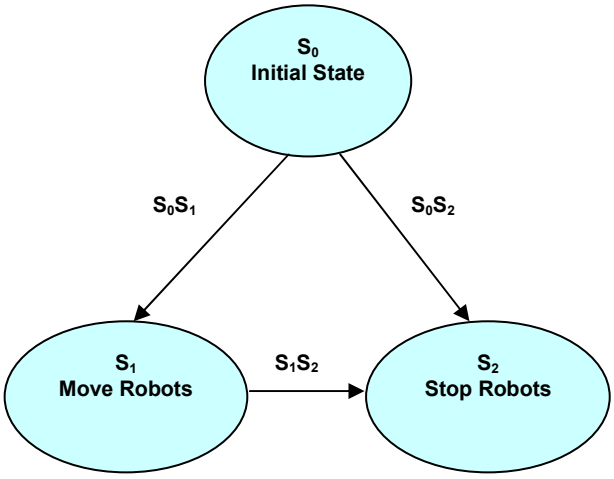


Figure 4.11 State transition diagram for auto placement of robots

The state transition conditions for auto placement of robots are given in table 4.7.

Table 4.7: State transition conditions for auto placement of robots

S_0S_1	Large position error or angle error of at least one robot
S_0S_2	Position error AND angle error below threshold
S_1S_2	S_0S_2

State S_2 is the 'stable state' which is entered once the robots have reached the final destination and attained the final orientation. In this state, the robots are stopped and no further action is required. There are no transition conditions that can change the state from S_2 to any other. The threshold values of position error and angle error can be suitably defined in the software. In this way the overall accuracy of the system can be controlled. The *Move Robots* behaviour is a complex action that avoids obstacles as it moves towards the target position.

4.7 The Formation and Marching Problem - Implementation

An important generic problem of agent interaction with the environment is the synchronization between an agent and the environment. In the context of multi-agent systems, this problem has been extensively studied and analyzed using robot soccer system as a test-bed. Entertainment robotics, involving multiple agents, is gaining importance as it is a useful research platform to test development techniques on systems ranging from high-level software based systems down to low-level robust hardware systems. A systematic comparative analysis of the synchronization problem remains an open challenge [60].

The formation and marching problems respectively require multiple robots to form up and move in a specified pattern. Solving these problems is quite interesting in terms of distributed algorithms, balancing between global and local knowledge and intrinsic information requirements for a given task. Solutions to formation and marching are also useful primitives for larger tasks, e.g., moving a large object by a group of robots or distributed sensing.

In a synchronized robotic parade, the situation on the ground is much less dynamic as compared to robot-soccer systems and the element of unpredictability, except in the case of hardware failure, is very low. The presented State Transition Based Control is therefore adequate and well suited for marching robots. The system being hierarchical in nature allows new behaviours to be coded and implemented quickly without increasing the complexity.

4.7.1 State Transition Diagram to Turn Robots

Figure 4.12 shows the State Transition Diagram to turn all the robots together, on the spot, to a final angle.

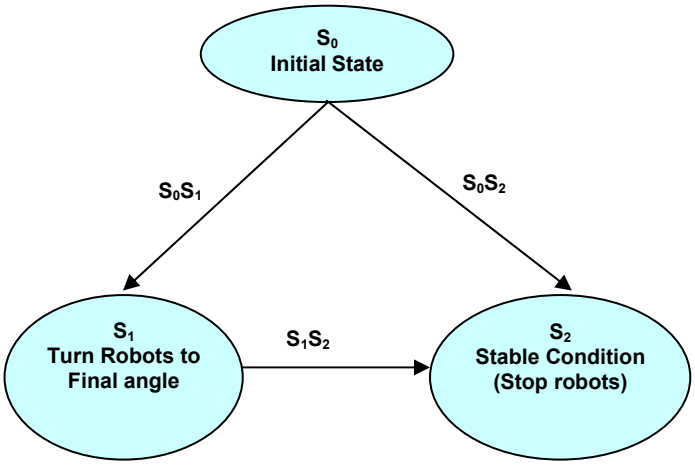


Figure 4.12 State transition diagram to turn robots on the spot

S_0, S_1 and S_2 are the three states in which the robots can be. The initialization state, S_0, is entered at the start of a particular operation. The robots will leave the initialization state and enter one of the valid behavioural states (S_1 and S_2) and thereafter transit between the states as the sensor input and the internal conditions vary. The state transition conditions are explained in Table 4.8.

Table 4.8: State transition conditions to turn robots on the spot

S_0S_1	$FinalangleError > \theta_{MAX}$
S_0S_2	$FinalangleError <= \theta_{MAX}$
S_1S_2	S_0S_2

Once the robots have turned to the final angle, they will enter the stable state S_2 in which the robots will stop. Having attained the final angle, the robots will not turn any more – hence S_2S_1 is always false. In this example, the robots need to be positioned with an accuracy of θ_{MAX}. In practice, this will largely depend on the resolution and accuracy of the vision system and the algorithms used for detecting colour blobs. Using the "two neighbouring pixel" algorithm explained in section 3.3 earlier, an accuracy of $\pm 3^o$ has been achieved.

4.7.2 Moving the Robot Contingent from One Place to Another

Figure 4.13 shows the initial and final position/orientation of four robots which need to move together. At the initial position the robots must first turn by the angle α to face the destination. This is the behaviour in state S_1 as shown in the state transition diagram in Figure 4.14. The group of robots will then travel along the line **AB** till it reaches the final destination. This is the behaviour in State S_2.

Along the way, while travelling in a straight line, i.e. in state S_2, it is possible that a robot may stray. As soon as it deviates from the travel direction by more than a set

threshold, it will transit back to the state S_1, in which the angle will be corrected so that the robot turns to face the destination gain; hence the state transition condition S_2S_1. When the robots reach the destination, they have to turn by the angle **β** (which may be different from **α**) to orient towards the final angle. This is the behaviour in state S_3. When the position errors and final angle errors of all the robots are below the set threshold, state S_4 will be entered. This is the stable state and there is no transition from this state.

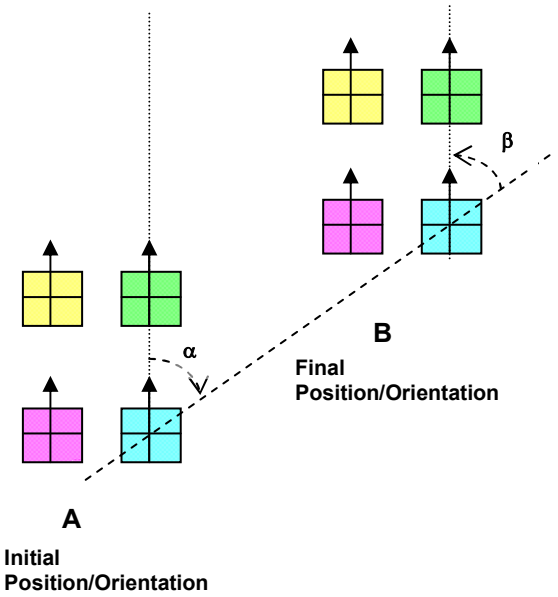


Figure 4.13 Initial and final position/orientation of the robot contingent

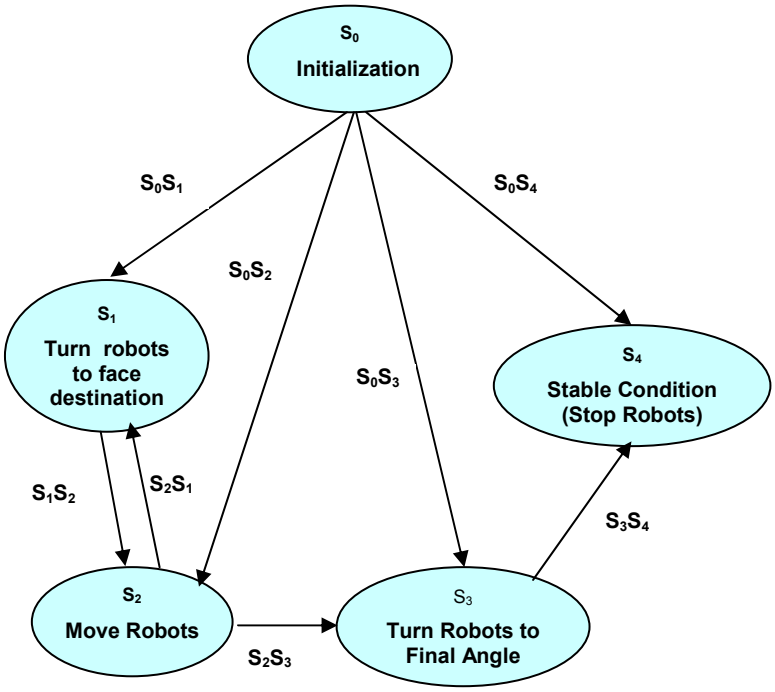


Figure 4.14 State transition diagram to position the robot contingent

With reference to the State Transition Diagram of Figure 4.14, the state transition conditions to position the robot contingent is shown in Table 4.9.

Table 4.9: State transition conditions to position the robot contingent

S_0S_1	*travelAngleError* > θ_{MAX} AND *positionError* > d_{MAX}
S_0S_2	*travelAngleError* <= θ_{MAX} AND *positionError* > d_{MAX}
S_0S_3	*positionError* <= d_{MAX} AND *finalAngleError* > θ_{MAX}
S_0S_4	*positionError* <= d_{MAX} AND *finalAngleError* <= θ_{MAX}
S_1S_2	(*travelAngleError* <= θ_{MAX} minus a hysteresis factor) AND *positionError* > d_{MAX}
S_2S_3	S_0S_3
S_3S_4	S_0S_4
S_2S_1	*travelAngleError* > θ_{MAX} plus a hysteresis factor

The *travelAngleError* is α and the *finalAngleError* is β. The *positionError* is the distance between the start and the end position of the robot.

As explained earlier, the sensor noise can make the state transition conditions problematic. Consider the case of transition from S_1 to S_2 and vice-versa. When the robot is on the verge of accumulating a *travelAngleError* of θ_{MAX}, the robot may oscillate between turning (S_1) and moving linearly (S_2). To overcome this problem a hysteresis factor is required which must be larger than the maximum sensor noise. The transitions between the states S_1 and S_2 are thus modified versions of the transitions between S_0-S_1 and S_0-S_2

4.7.3 Synchronised About Turn – Problems and Proposed Solution

The problem of turning the robots by 180^o using the state transition diagram shown in Figure 4.14 is rather tricky. In an about turn, a robot is free to turn clockwise or anti-clockwise. The problem is in the calculation of the robot's *finalAngleError*, which is normalized to keep it between $+180^o$ and -180^o. The problem actually originates at the vision sensor where noise results in an angular accuracy of $\pm 3^o$.

Figure 4.15 shows the position of two robots, which were commanded to position at an angle of 90^o. Because of the sensor noise, Robot1 is standing at 92^o while Robot2 is standing at 87^o.

If both the robots are now commanded to about turn and stand at -90^o, the *finalAngle*, then the following will happen:

Robot1FinalAngleError = -90 – 92 = -182

= 178^o (after normalization)

This will make the robot turn anti-clockwise.

Robot2FinalAngleError = -90 – 87

$= -177^{o}$

This will make the robot turn clockwise.

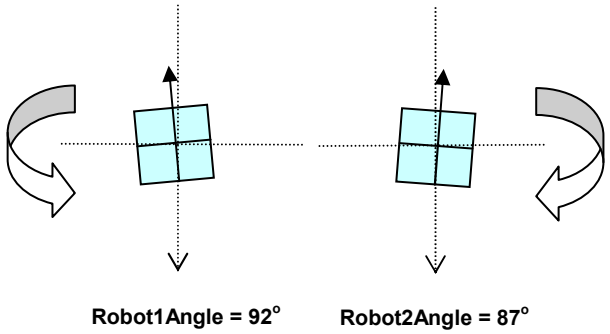


Figure 4.15 Two robots turning in opposite directions during an about turn operation

The solution is rather simple and it illustrates the ease with which behaviour can be modified using states. An additional state can be added to first turn the robot to a "*middle angle*" which is 90^{o} less than the final angle.

The state transition diagram for a 2-step turn is shown in Figure 4.16. It should be noted that there is no transition from S_0 to S_2. The state transition conditions are shown in table 4.10.

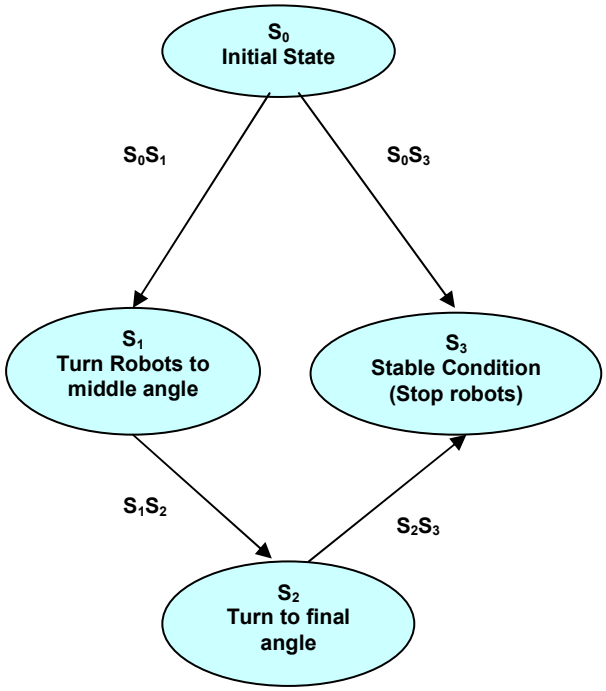


Figure 4.16 State transition diagram for a 2-step turn

Table 4.10: State transition conditions for a 2-step turn

S_0S_1	$MiddleAngleError > \theta_{MAX}$
S_0S_3	$FinalAngleError <= \theta_{MAX}$
S_1S_2	NOT(S_0S_1)
S_2S_3	S_0S_3

4.8 Obstacle Avoidance Using STBC Technique

Obstacle avoidance in robotics is an important discipline with the objective of moving agents on the basis of sensorial information. The challenges are many fold when the scenario is dynamic with an unpredictable behaviour. In many cases, the surroundings do not remain static, and thus the employed methods must continuously detect the changes and consequently adapt the motion trajectory. In a collaborative effort, the robots not only have to avoid obstacles, often they have to avoid collision with each other too.

Motion in very dense, cluttered and complex scenarios has been addressed by various techniques such as Nearness Diagram Navigation [61]. The method uses a 'divide and conquer' strategy based on situations to simplify the difficulty of the navigation. The proposed STBC technique, which has been successfully employed for robot positioning, behaviour programming and implementing collaborative strategies, can also be used for obstacle avoidance. In a working domain where the robot is not constrained to move in a particular direction, the STBC technique is very effective. Moreover, this method does not place any constraint on the robot's shape and kinematics. A simplified state transition diagram for obstacle avoidance is shown in Figure 4.17.

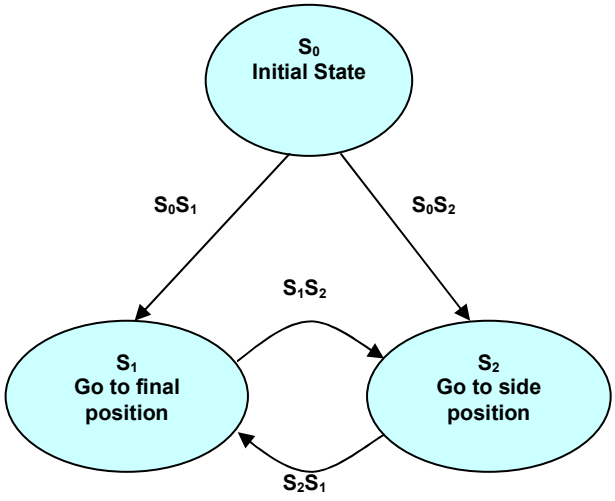


Figure 4.17 State transition diagram for obstacle avoidance

The state transition condition S_0S_2 detects whether there are obstacles in the way. If the outcome of this test is positive, the robot is commanded to move to a side position (State S_2), otherwise go to the final destination (State S_1). Condition S_1S_2 is the same as S_0S_2 and condition S_2S_1 is the same as S_0S_1. The side position is calculated to be a point, a predefined distance away, orthogonal to the side of the obstacle from the direction of the final position.

4.9 Discussion

Finite state machines are often employed to design sequential digital circuits. In general, the process consists of the following steps –

- Identify the states that the system may possibly be in and the corresponding outputs.
- Draw the transitions from one state to another. Transition back to the same state is also possible.
- Identify the inputs available in each state.
- Label the transitions. This essentially means determine the input conditions which result in a transition.
- Assign state variables to each of the states.
- Determine the required combinatorial logic to generate the state transitions.
- Determine the combinatorial logic required to generate the output variables.

A novel implementation of a similar finite state machine, albeit with some differences, has been detailed in this chapter for realising collaborative behaviour of robots. Scenarios presented are robot soccer game and synchronised formation and movement of a robot contingent. The steps involved are listed below-

- Identify the states that the system may possibly be in.
- Identify the actions that are required to be performed in each state.
- Draw the transitions from one state to another. At this stage it is not necessary to work out what causes these transitions. It is also not necessary to draw the transition, if any, back to the same state. As explained in the chapter, the STBC system will check if the conditions for transiting to the next possible state have been met or not. If not, it will continue to remain in the same state and perform the actions defined for that state.

- Identify the inputs available in each state. In the case of robot soccer, the inputs are generated by the vision processing software, namely the position of the ball and the position/orientation of the home and opponent robots.
- Label the transitions. This essentially means determine the state transition conditions.
- Assign a state variable to keep track of the current state of the system. In a hierarchical system, sub-state variables will also be required. In an object oriented implementation, the state variable can be created as a private data attribute of the class which implements the system (behaviour).

A big advantage of the proposed STBC methodology is the ease with which it can be implemented in an object-oriented paradigm to build complex behaviours for collaborative tasks. Behaviours can be easily enhanced and made more sophisticated by adding states and refining the state transition conditions. Also, complex behaviours can be built using a hierarchical approach in which each task can be decomposed into a state transition based system of primitive actions.

The state transition process can be coded in an object oriented class structure in which the Boolean '*state transition methods*' represent the conditions for a change of state to occur, while '*action methods*' encode the behaviour or action required in the given state.

The current state is stored so that only the transitions from the current state, to other allowed states, needs to be checked, thereby improving the processing efficiency of the system, yet another advantage of the STBC method. Since the outline of the state transition code is effectively standard, only the details of the state actions and the conditions that must be true for a state transition to occur need to be designed and coded. This effectively makes implementation of new behaviour fast and easy.

```
BOOL S0S1()
{
      if (BallPos.x <= NEARPOS)
      //-- ball in near position
            return true;
      else
            return false;
}
```

Figure 4.18 OOP implementation of a '*state transition method*'

Example of a '*state transition method*' is shown in Figure 4.18 while that of an '*action method*' in Figure 4.19.

```
void s2(void)
{
        finalPos.y = (float)(BallPos.y)
        //-- track the ball y

        position(RobotID, finalPos, 90);
}
```

Figure 4.19 OOP implementation of an '*action method*'

The complete listing of an object-oriented implementation of STBC-based goalkeeper behaviour is shown in Appendix A.

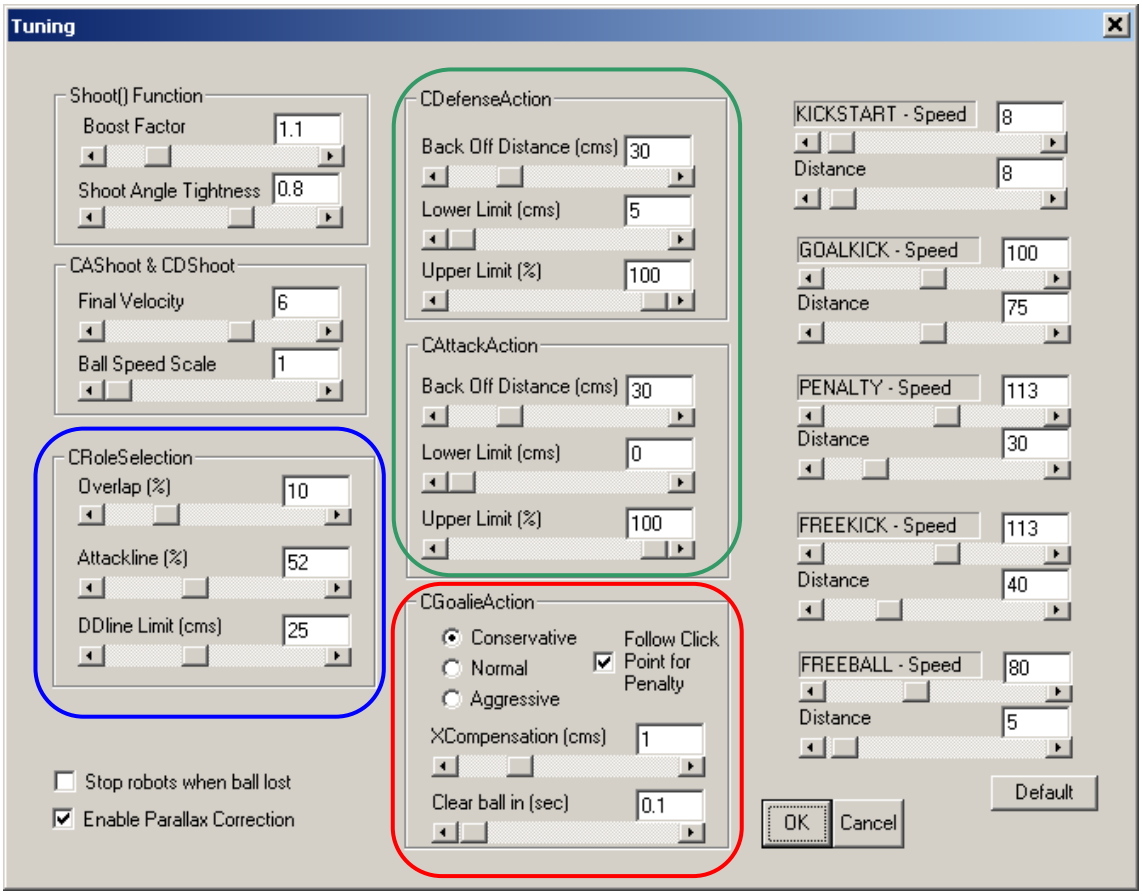


Figure 4.20 Graphical User Interface (GUI) for tuning robot behaviour

In a real time application, it would be a great advantage if it was possible to interact with the system and be able to modify behaviours without having to resort to changing code. Other than by adding or deleting states, behaviour can be changed simply by changing the state transition conditions or even disabling some altogether. For example, to change the behaviour of the goalkeeper in a robot soccer game, say from conservative to aggressive, one can do so using an interactive graphical user interface (GUI) to the application. Clicking a few radio buttons or check boxes effectively signals to the application program which set of state transition conditions to employ and what actions

to perform in a certain state. An example of this is shown in Figure 4.20 which allows different goalkeeper behaviour to be selected from the GUI (CGoalieAction).

For the role selection state machine described in section 4.5, the attack limit defined by line A_1A_2 (in Figure 4.4) can be easily changed from the GUI (CRoleSelection), using the scroll bar labelled **Attackline (%)** in Figure 4.20. Similarly the defence line D_1D_2 (in Figure 4.4) can be moved using the scroll bar labelled **DDline Limit (cms)** in Figure 4.20 and the overlap region around the horizontal centre line (in Figure 4.4) for the **AD** and **DA** states can be increased or decreased using the scroll bar labelled **Overlap (%)** in Figure 4.20.

The defensive behaviour, explained in section 4.5.1 can also be modified interactively through the GUI. Using the scroll bars in the group named **CDefenseAction** in Figure 4.20, one can change the robot's *back off* distance, the position of line L_1L_2 (Figure 4.6) and line U_1U_2 (Figure 4.6).

Likewise, the offensive behaviour, explained in section 4.5.2 can also be modified using the scroll bars in the group named **CAttackAction** in Figure 4.20. One can change the robot's *back off* distance, the position of line L_1L_2 (Figure 4.8) and line U_1U_2 (Figure 4.8).

To conclude, STBC methodology empowers one to quickly and easily implement complex collaborative behaviours and modify them interactively in a real time application.

5 Filtering and its Impact on Predictive Movement and Interception Control

The vision subsystem is one of the major components of a collaborative robotic system. As explained in Chapter 3, the interlaced images are a source of noise due to quantisation error. This has adverse effects on the overall performance of the agents in accomplishing a collaborative task. This chapter presents a case of applying a *g-h* filter to pre-process the data obtained from the vision system before passing it on to the predictive controller. The role of the filter and the predictive controller, and their impact on the overall system functionality and performance will be discussed in detail in this chapter.

The significance of the presented research and development is in that they show the importance of filtering in improving the system performance and accuracy of a vision based intelligent controller. When developing an intelligent control system, the filter design cannot be left as an after thought but must be integrated into the system. To date, many automated learning based intelligent control techniques such as neural networks; genetic algorithms and genetic programming make use of raw sensor data in the expectation that the algorithm will learn the noise characteristics of the data. The presented study shows that explicitly developing a filter and evolving it along with the controller is preferable, rather than relying purely on the intelligent control algorithm.

The experimental part of the research presented in this chapter was based again on the use of the robotic soccer system. Soccer robots must exhibit actions like positioning at a designated location, moving to the ball, blocking an opponent, turning to an advantageous angle, circling the ball and shooting a goal. Highly complex game-play, combined with the agility and speed of the robots, requires accurate path planning and prediction of moving targets. The path plans must be updated to reflect the changing game situations, meaning that they must be created in real-time [62]. Several techniques have been tested for robot soccer path planning including genetic algorithms [07], fuzzy logic [08, 63], neural networks [64], potential fields [64, 65] and State Transition Based Control (STBC) presented in Chapter 4. The genetic algorithm approach [07] is limited by the need to evolve the robotic control on a real system. This is time consuming and subjects the robots to significant wear and tear. The fuzzy logic approach [08, 63] suffers because a large number of rules must be specified for the large number of variables. This makes it impractical to use fuzzy logic unless tools are provided for automatically generating the control rules. The potential field approach [65], though

simple to implement, often results in systems with undesirable limit cycles and unpredictable behaviour due to zeros in the potential field. A STBC approach, however, can be designed quickly and intuitively, enabling rapid development and testing of new robot behaviours. This approach forms the basis of the predictive control system presented in this chapter.

To evaluate the performance of the STBC system, the work detailed in this chapter employs a case study of a robot goalkeeper behaviour in a robot soccer game. The goalkeeper intercepts the ball, clears the ball from the goal area or sits idle when the ball is in the opponent's half of the field. This implementation also demonstrates robotic targeting (a shoot function in robot soccer). The targeting system uses filtering to substantially improve shooting accuracy on both a stationary and moving object, in this case, the ball. Several studies have implemented real-time image processing [66, 67], ball interception [68] and robot localisation [69] in the robot soccer domain. However, they have not managed to achieve the target interception success rate reported in this work. The proposed technique has been developed and successfully tested for a fast moving target at speeds of up to 1.3 m/s. The system successfully predicts the trajectory of the target through the robot's workspace, plans the robot's motion to reach the target at a given angle and executes that plan. The system is robust in environments with substantial noise generated by variation in ambient light, shadows and reflections.

Alternatives to the Kalman filter that have been investigated for mobile robots include fuzzy filters [70], particle filtering [71] and probabilistic data association filters [72]. Also extensions to Kalman filtering, such as distributed Kalman filtering, have been considered [73]. Many studies present results in environments that have a significantly higher level of noise (as is the case when a sonar is the main sensor) than in the reported experiment. Particle filtering [71] techniques applied to mobile robot vision problems have been shown to have errors up to 30 cm. This is much larger than the errors observed in the unfiltered system reported in this chapter. Probabilistic data association filters [72] have been shown to be better than Kalman filters and particle filters in scenarios where a Gaussian does not correctly model the noise characteristics of the system or occlusions lead to degenerative cases. It is likely that this approach will improve the performance of the system in scenarios where the robot moves around an obstacle (or another robot) since the predictive model of the robot's position should have bimodal characteristics (the robot cannot move through the obstacle, it has to pass to one of two sides). The distributed Kalman filter has been used to improve the localization of a group of robots [73], however this technique does not improve the reported system as the position of all the robots are available from the single global vision system, rather than a set of distributed sensors.

Kalman filtering [12, 13] has proved its usefulness in autonomous and assisted navigation and guidance systems. It is also very effective in radar tracking of moving objects. The Kalman filter is represented by a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimates of present and future states based on past observations and can do so even when the precise nature of the modelled system is unknown. Kalman filtering is also a computationally efficient algorithm which generates an optimal estimate from a sequence of noisy observations.

Robotic interception of moving objects in industrial settings has gained much attention in recent years. Among the most efficient approaches is the Active Prediction, Planning and Execution (APPE) system [11]. The key feature of this system is the ability of a robot to perform a task autonomously without complete a priori information. The APPE system has been tested for an object moving at a maximum velocity of 4.5 cm/s. While APPE is effective, the resulting speed of the robot system is limited. This led to the investigation of a predictive control system based on STBC which is shown to reliably control a mobile robot at significantly higher speeds, in excess of 70 cm/s.

5.1 Object Interception

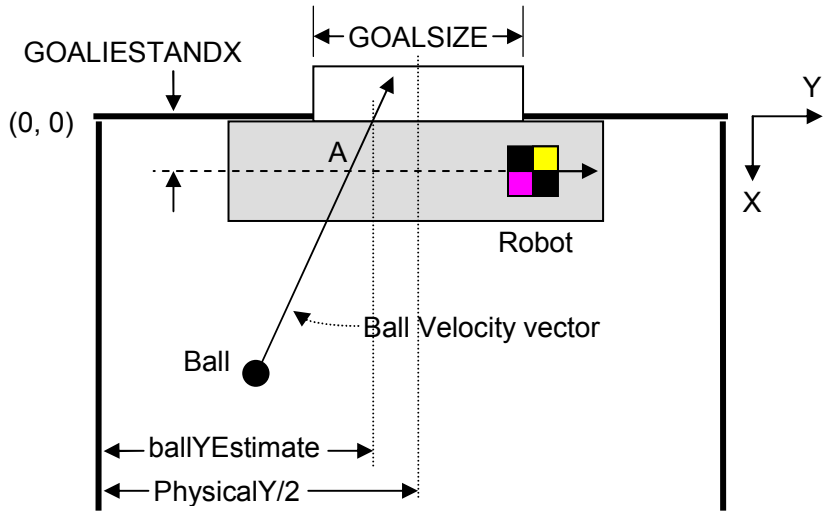


Figure 5.1 Calculating the ball interception position

The robot must reach the expected position as quickly as possible in order to intercept the object. Unlike object targeting (discussed in section 5.2), object interception does not require the robot to approach the object from a given direction. In robot soccer, one of the most important object interception tasks is to block the ball as it heads toward the goal. From the vision data, the velocity of the ball is calculated and updated in each

frame. The accuracy of the system can be evaluated by the success rate of ball interception by the goalkeeper. This is done when the ball is moving towards the goal and is predicted to be entering the goal as shown in Figure 5.1.

The flow chart of the algorithm to predict where the robot must be positioned to intercept the ball is shown in Figure 5.2.

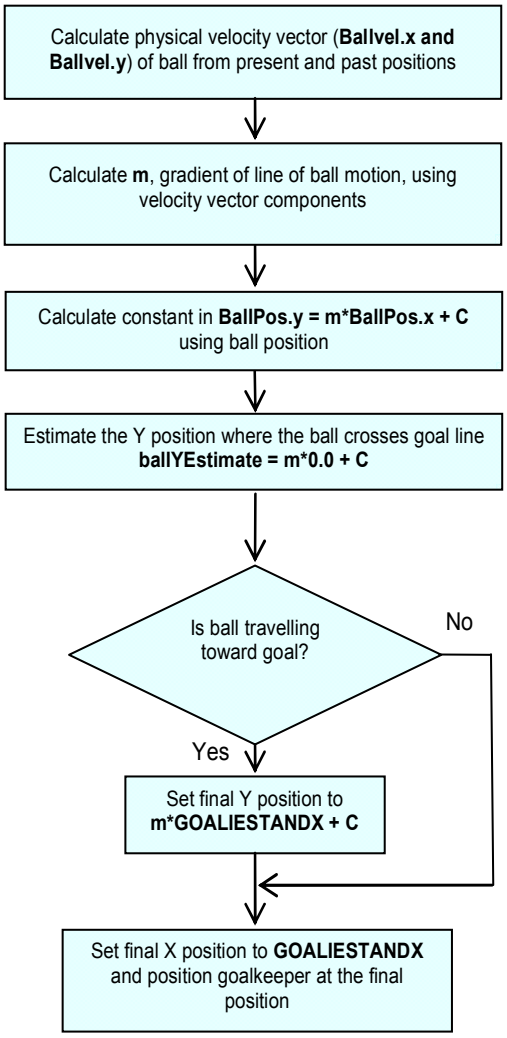


Figure 5.2 Flow chart for predicting the point of interception

If the ball position is not detected accurately, the robot is unlikely to intercept the ball. This is even more so when the ball is travelling at very high or very low speed. At a high speed the time available for the robot to reach the target position is very short. On the other hand, at low speeds the numerical accuracy of finding the angle of the direction of the ball is low, making the calculation of the target position noisy. To overcome this problem predictive filtering has to be used.

5.2 Control Algorithm for Object Targeting

A targeting (shoot) function is implemented by providing the robot with a target position (in line with the ball and the goal) from where it will progressively approach and then guide and/or hit the object (the ball) towards the target (the goal). This is position **P** in Figure 5.3, a point behind the stationary object (the ball), in line with the target. The velocities of the left and right wheels of the robot, V_L and V_R, which move it to position **P** and orient it to face the target, are calculated by the targeting (shoot) algorithm.

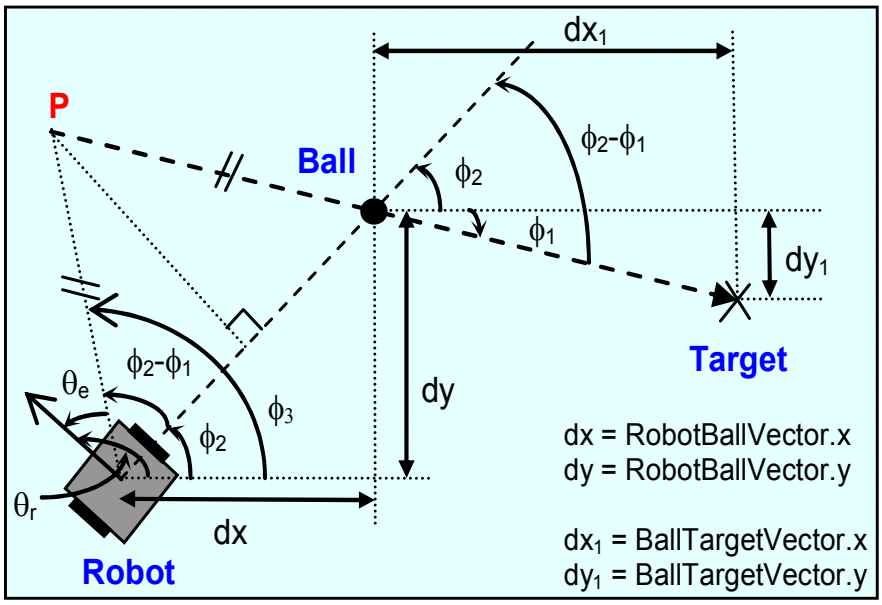


Figure 5.3 Calculating wheel velocities for stationary object targeting

The angles shown in Figure 5.3 are:

ϕ_1 - *TargetAngle* (negative value in the case under consideration)

ϕ_2 - *BallAngle* (positive value)

ϕ_2-ϕ_1 - BallTargetAngle

ϕ_3 - DesiredDirectionAngle

θ_e - AngleError, and

θ_r - RobotAngle

The robot's desired direction angle ϕ_3 is

$$\phi_3 = \phi_2 + (\phi_2 - \phi_1) = 2\phi_2 - \phi_1 \tag{5.1}$$

The angle error θ_e between the required direction and the current robot direction is

$$\theta_e = \phi_3 - \theta_r = 2\phi_2 - \phi_1 - \theta_r \tag{5.2}$$

The wheel velocities are calculated using the following pair of equations:

$$\begin{aligned} V_L &= V_C - Ka_p\theta_e = V_C - Ka_p(2\phi_2 - \phi_1 - \theta_r) \\ V_R &= V_C + Ka_p\theta_e = V_C + Ka_p(2\phi_2 - \phi_1 - \theta_r) \end{aligned} \tag{5.3}$$

Ka_p is the constant proportional angular gain. When the robot is in line with the target the angle error θ_e is equal to zero. The robot then moves straight towards the target with a constant velocity, V_C.

The movements achieved using the control algorithm (equation 5.3) resulted in very reliable stationary targeting. The target (i.e. the ball) is consistently struck. However, the level of *directional accuracy* is not high. Thus the ball's resulting direction of motion is not sufficiently reliable. As the image is processed in two parts – the odd and even scan fields separately – the position of the stationary ball could shift by as much as 1 pixel. Figure 5.4 shows the shift in the ball position in the odd and even scan fields. Depending on the resolution of the captured image and the size of the operation field, this could correspond to an error as high as 0.62cm. This results in incorrect calculation of wheel velocities.

The problem becomes more severe when the ball is close to the target. This is due to the fact that a small change in the ball position alters the *TargetAngle* (ϕ_1) by a large amount. From one odd or even scan field to the next, the difference in angle error $\delta\theta_e$ is large, resulting in wider variations of V_L and V_R as the robot takes the shot. As such, the trajectory followed by the robot is not smooth. To overcome this problem filtering is needed.

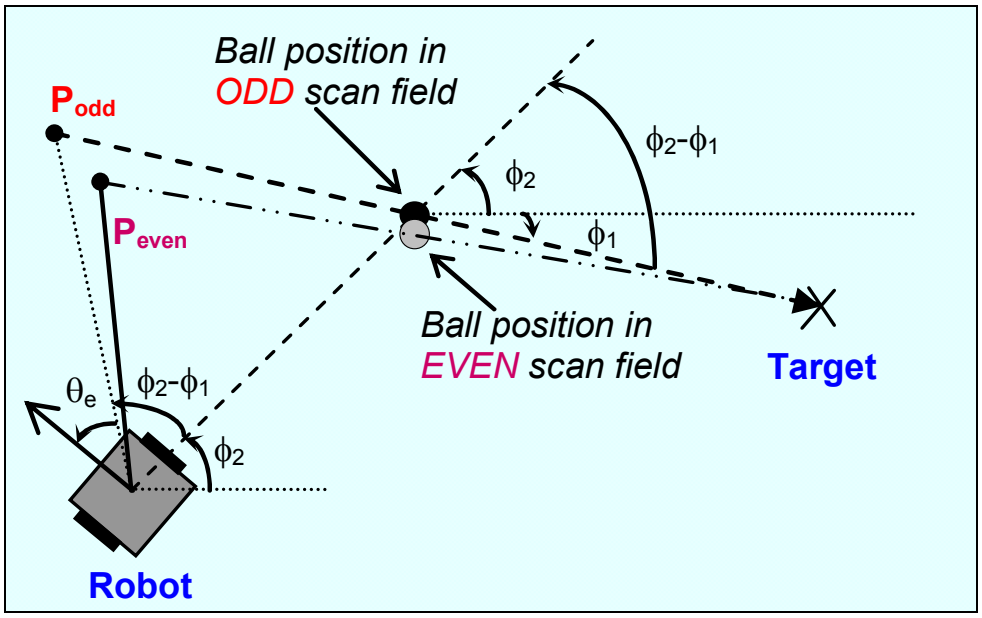


Figure 5.4 Object position in odd and even scan fields

5.3 Application of *g-h* Tracking Filter

With reference to Figure 5.4, reducing the variation of the ball position from one scan field to another will reduce $\delta\theta_e$, the change in *AngleError*, and hence increase the stability of the wheel velocity calculations using equation (5.3). Application of a *g-h* filter to the data (i.e. position of ball/robots and robot orientation) obtained from the vision system was explored with the aim of significantly reducing the variation in calculated position **P**. In this section, the *g-h* filter is explained first and the experimental results are detailed in Section 5.5.

The filter can be described by the following equations:

$$\bar{\dot{x}}_n = \hat{\dot{x}}_n + \frac{h}{T}\left(y_n - \hat{x}_n\right) \tag{5.4}$$

$$\bar{x}_n = \hat{x}_n + g\left(y_n - \hat{x}_n\right) \tag{5.5}$$

where,

$\bar{\dot{x}}_n$ is the filtered (smoothed) estimate of velocity at discrete time *n*

$\hat{\dot{x}}_n$ is the predicted estimate of velocity at time *n* based on the sensor values at time *n-1* and all preceding times

$\bar{x}_n$ is the filtered (smoothed) estimate of position at time *n*

$\hat{x}_n$ is the predicted estimate of position at time *n* based on the sensor values at time *n-1* and all preceding times

T is the sample time (scan-to-scan period) for the system

y_n is the sensor reading at time *n*

h and g are the filter parameters

Equations (5.4) and (5.5) provide an updated estimate of the present object velocity and position based on the present measurement of object position y_n as well as prior measurements.

The predicted velocity and position at time *n* are calculated based on a constant velocity model:

$$\hat{\dot{x}}_n = \bar{\dot{x}}_{n-1} \tag{5.6}$$

$$\hat{x}_n = \bar{x}_{n-1} + T\bar{\dot{x}}_{n-1} = \bar{x}_{n-1} + T\hat{\dot{x}}_n \tag{5.7}$$

Equations (5.6) and (5.7) allow transition from the velocity and position at time *n-1* to the velocity and position at time *n*. These *transition equations* together with equations (5.4) and (5.5) allow tracking of an object and are combined to give just two tracking filter equations:

$$\hat{\dot{x}}_n = \hat{\dot{x}}_{n-1} + \frac{h}{T}\left(y_{n-1} - \hat{x}_{n-1}\right) \tag{5.8}$$

$$\hat{x}_n = \hat{x}_{n-1} + T\hat{\dot{x}}_n + g\left(y_{n-1} - \hat{x}_{n-1}\right) \tag{5.9}$$

Equations (5.8) and (5.9) are used to obtain running estimates of target velocity and position.

Filtering addresses two major issues. It lessens the vision quantization error due to separate processing of odd and even scan fields and it significantly reduces the negative effect of the variation in the calculation of ball position due to light intensity fluctuations.

To further improve the efficiency of the robot movement, the target position **P** (Figure 5.3) can be adjusted to be closer to the ball. This can be achieved by adding a scale factor k_{sf}, thus modifying equation (5.3) as follows:

$$V_L = V_C - Ka_p(\phi_2 + k_{sf}(\phi_2 - \phi_1) - \theta_r) \tag{5.10}$$

$$V_R = V_C + Ka_p(\phi_2 + k_{sf}(\phi_2 - \phi_1) - \theta_r) \tag{5.11}$$

The result of the simulation of the modified algorithm is shown in Figure 5.5. It can be seen that the robot travels a shorter distance and reaches the shooting position faster. The value of scale factor k_{sf} can be optimized for different angles and distances from the interception point to set the intermediate target angle of the controller.

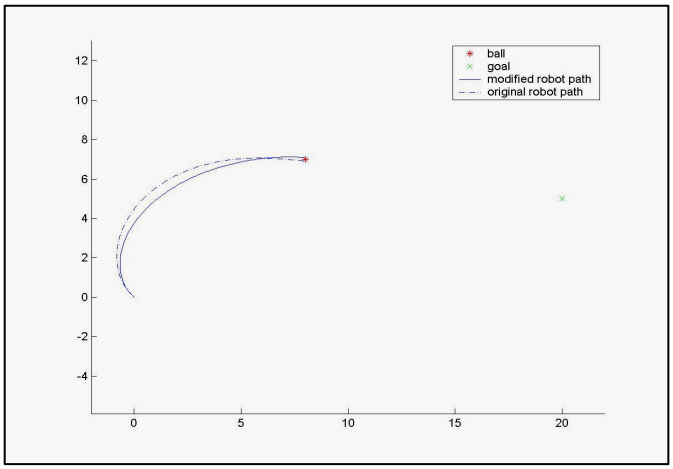


Figure 5.5 Simulated path using original and modified control algorithms

5.4 Prediction of an Object's Position

In order to approach a moving target (e.g., the ball in a robot soccer game), its future position must be predicted. Filtering can be used to achieve much greater accuracy of target prediction and interception. The ball and robots' positions are predicted using a constant velocity model represented by the equations below:

$$\hat{x}^b_{n+t} = \hat{x}^b_n + t * T\bar{\dot{x}}^b_{n-1} \tag{5.12}$$

$$\hat{x}^r_{n+t} = \hat{x}^r_n + t * T\bar{\dot{x}}^r_{n-1} \tag{5.13}$$

where,

$\hat{x}^b_{n+t}$ is the predicted estimate of ball position at time $n+t$ based on the sensor values at time $n-1$ and all preceding times

$\hat{x}^b_n$ is the predicted estimate of ball position at time n based on the sensor values at time $n-1$ and all preceding times

$\bar{\dot{x}}^b_{n-1}$ is the filtered (smoothed) estimate of ball velocity at discrete time $n-1$

t is the elapsed time

T is the sample time (scan-to-scan period) for the system

In equation (5.13), the corresponding variables are for the robot.

The constant velocity model is realistic for a fast moving ball that does not decelerate significantly. However the constant velocity model only provides an approximation of the motion of an accelerating robot or a robot moving on a circular path.

The position and time of the intersection between the ball and robot can be found by solving the equations (5.12) and (5.13) simultaneously. In general, if the two paths intersect, there is only one real valued solution. Positive ***t*** values are required for the collision points that will occur in the future while negative values indicate collision points that could have occurred in the past. When there is no common solution to the equations, the robot and ball paths are predicted to miss each other; that is, their paths do not converge.

The predicted intersection points are used to calculate the robot's desired angle (equation 5.1) and angle error (equation 5.2), which are required for the calculation of the control action (equation 5.3) or (equations 5.10 and 5.11).

5.5 Experimental Results

This section details experimental results in three categories – stationary object, robot motion trajectory, and intercepting (or blocking) of a moving target.

5.5.1 Stationary Object

Ball Position. The position of a stationary object (the ball) was measured for different filter parameters. The value of *g* was varied from 0.1 to 0.9 in steps of 0.1 and for each value of *g*, the value of *h* was varied from 0 to 1.0 in steps of 0.1. In effect, 99 different combinations of *g* and *h* were used. For each set of filter parameters, the data were recorded for 500 frames.

The variation in ball position, *dx* and *dy*, over time, was calculated as:

$$dx^b = x_n^b - x_{n-1}^b$$
$$dy^b = y_n^b - y_{n-1}^b \qquad (5.14)$$

Here (x_n^b, y_n^b) and (x_{n-1}^b, y_{n-1}^b) are the position of the ball in the current and previous frames respectively. For data plotted in Figures 5.6 to 5.8, the filter parameters were set to *g*=0.8 and *h*=0.2. Figure 5.6 shows the variation (*dx*) of the ball's non-filtered and filtered position (X-coordinate) from one frame to the next, Figure 5.7 plots the ball's measured and filtered position (X-coordinate) and Figure 5.8 shows the difference in the actual and filtered position. (The segment of data shown in the plots is the region in which the maximum *d*x occurred.) Likewise, Figures 5.9 to 5.11 are for filter parameters set to *g*=0.6 and *h*=0.4 and Figures 5.12 to 5.14 are for filter parameters set to *g*=0.1 and *h*=0.9.

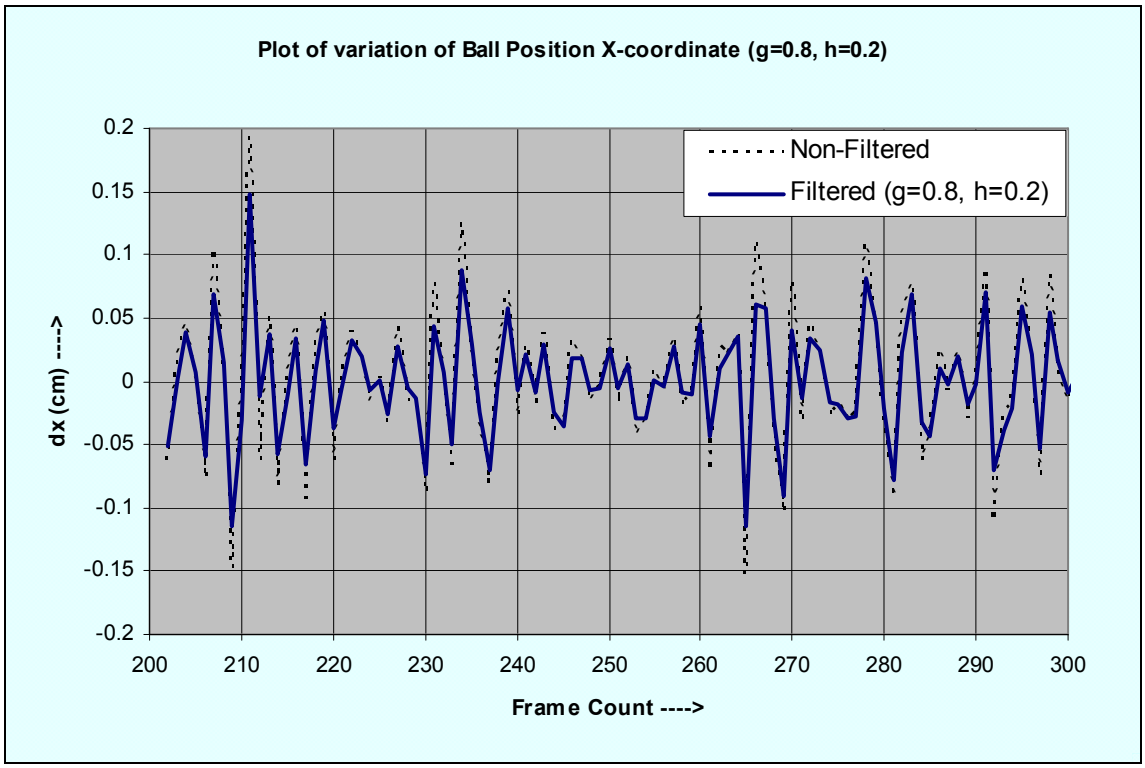


Figure 5.6 Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.8, h=0.2*)

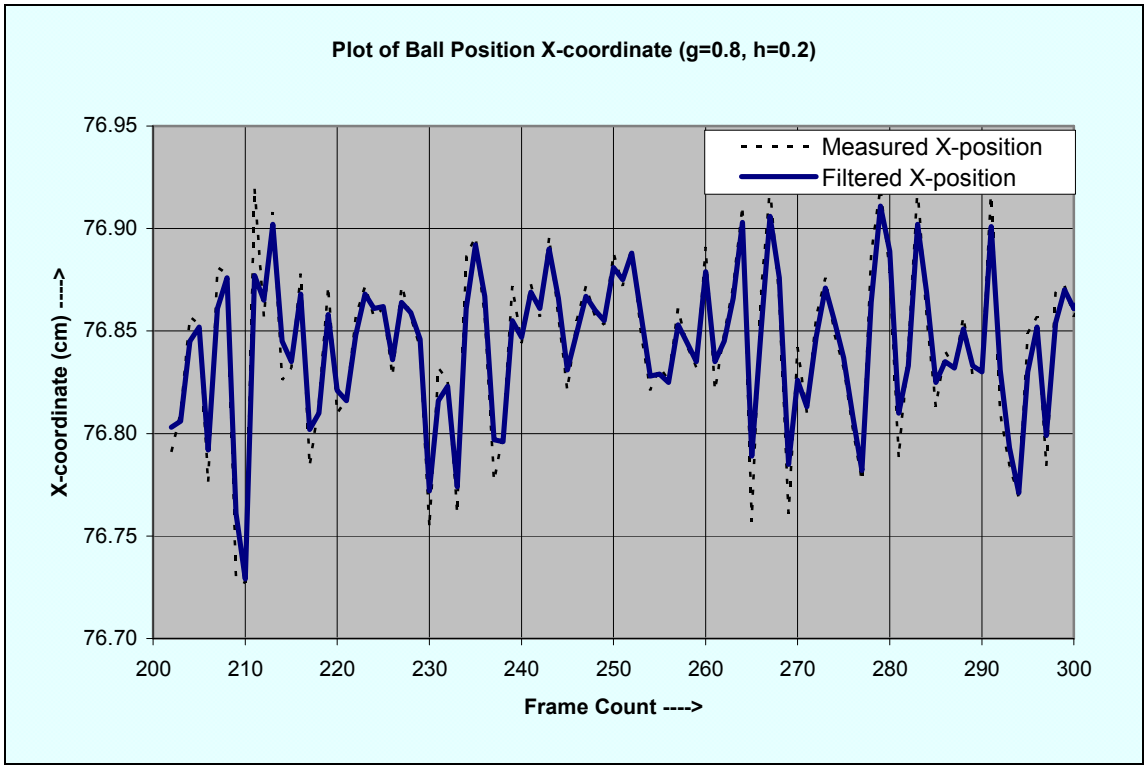


Figure 5.7 Plot of a stationary ball's X-position (*g=0.8, h=0.2*)

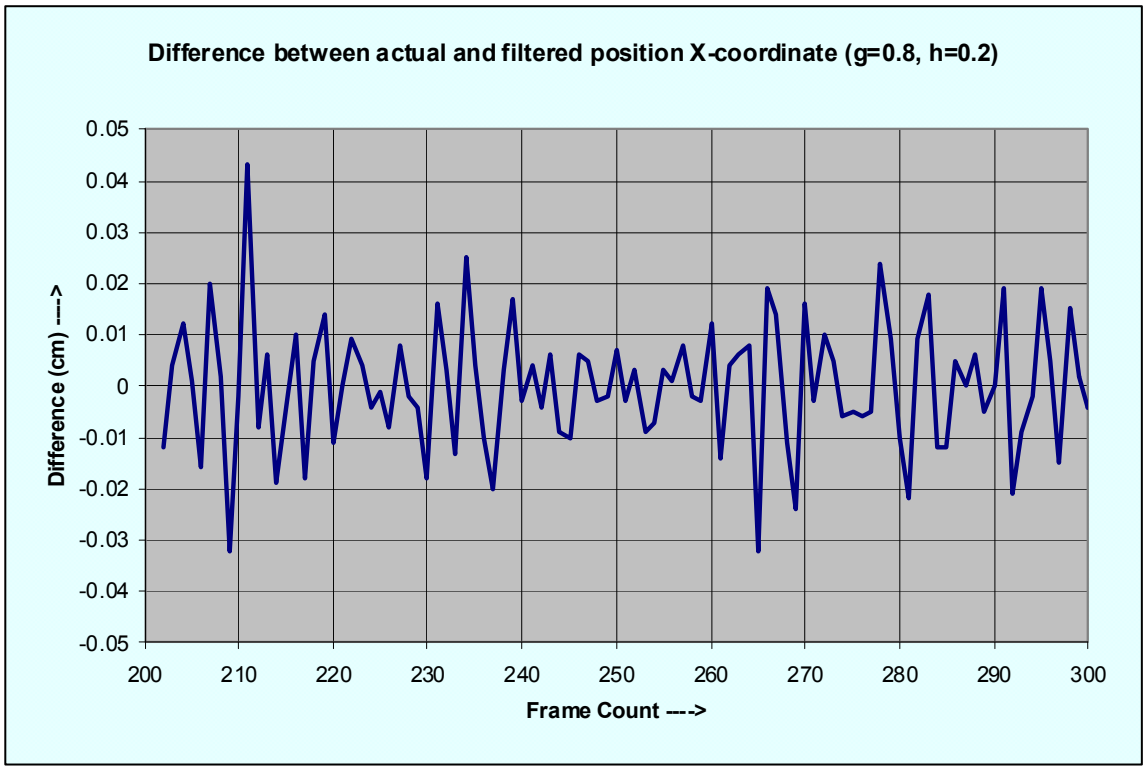


Figure 5.8 Plot of the difference between actual and filtered X-position (*g=0.8, h=0.2*)

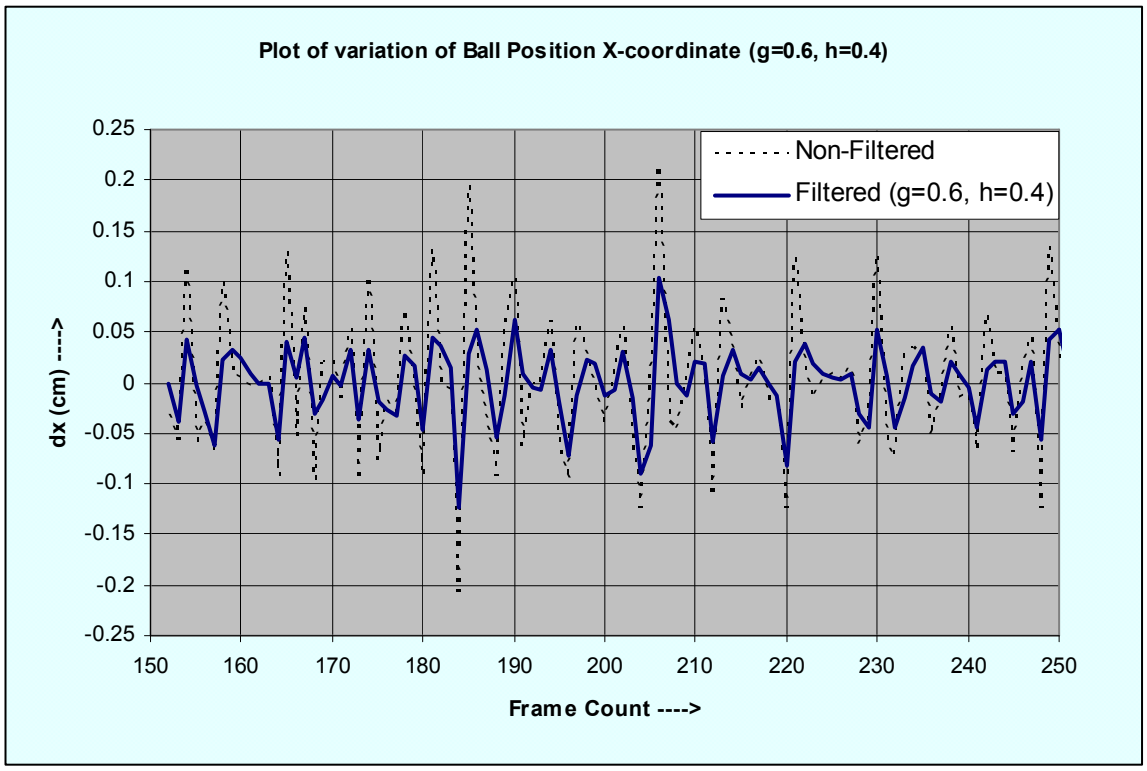


Figure 5.9 Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.6, h=0.4*)

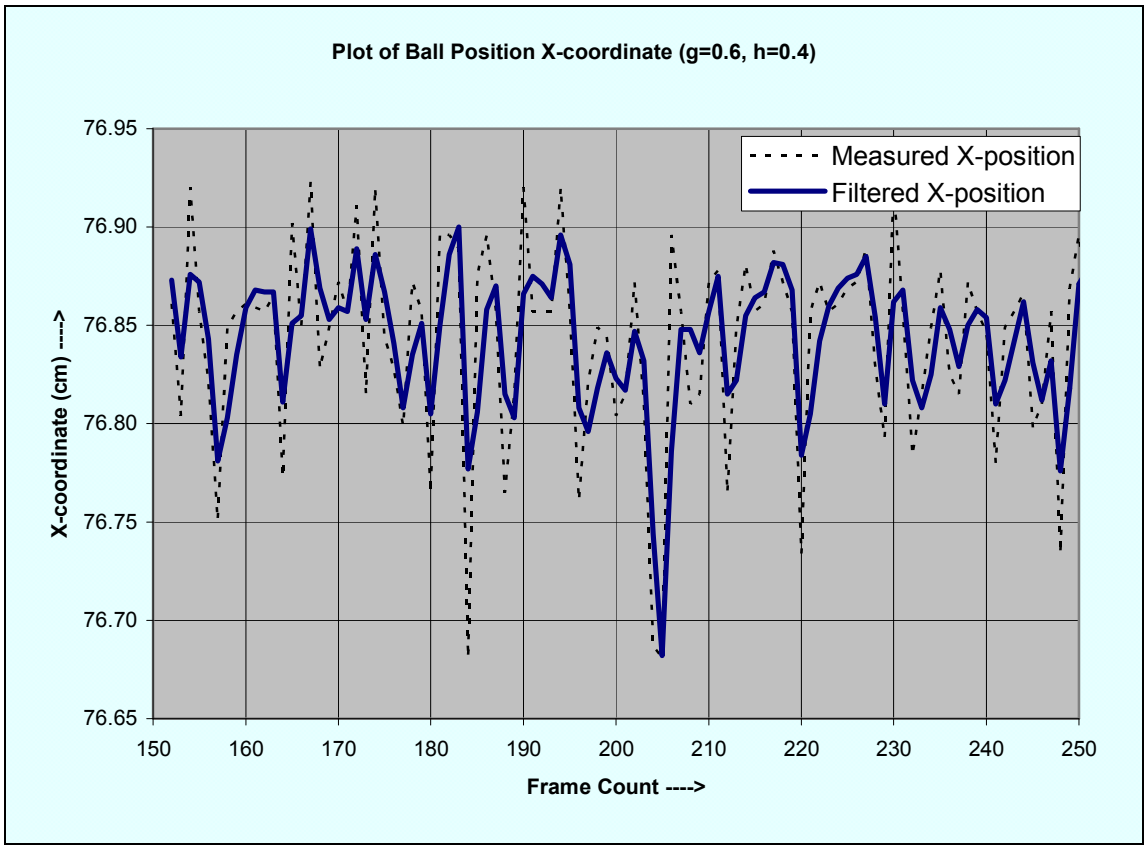


Figure 5.10 Plot of a stationary ball's X-position (*g=0.6, h=0.4*)

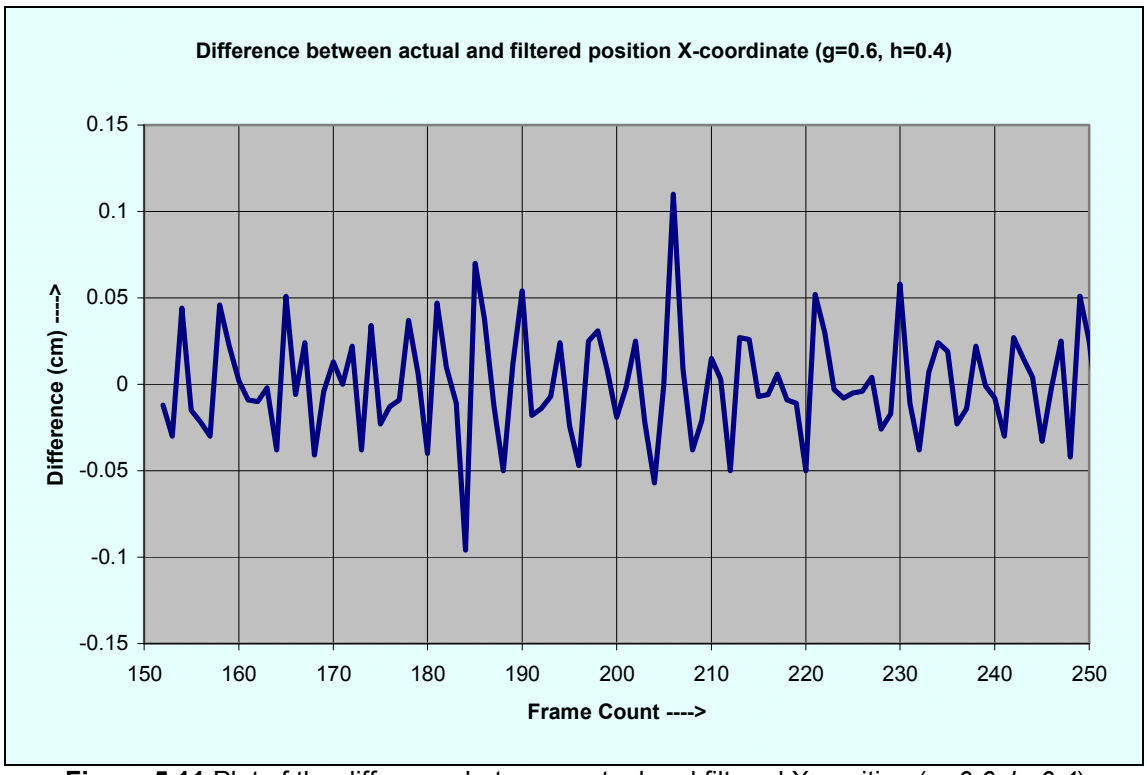


Figure 5.11 Plot of the difference between actual and filtered X-position (*g=0.6, h=0.4*)

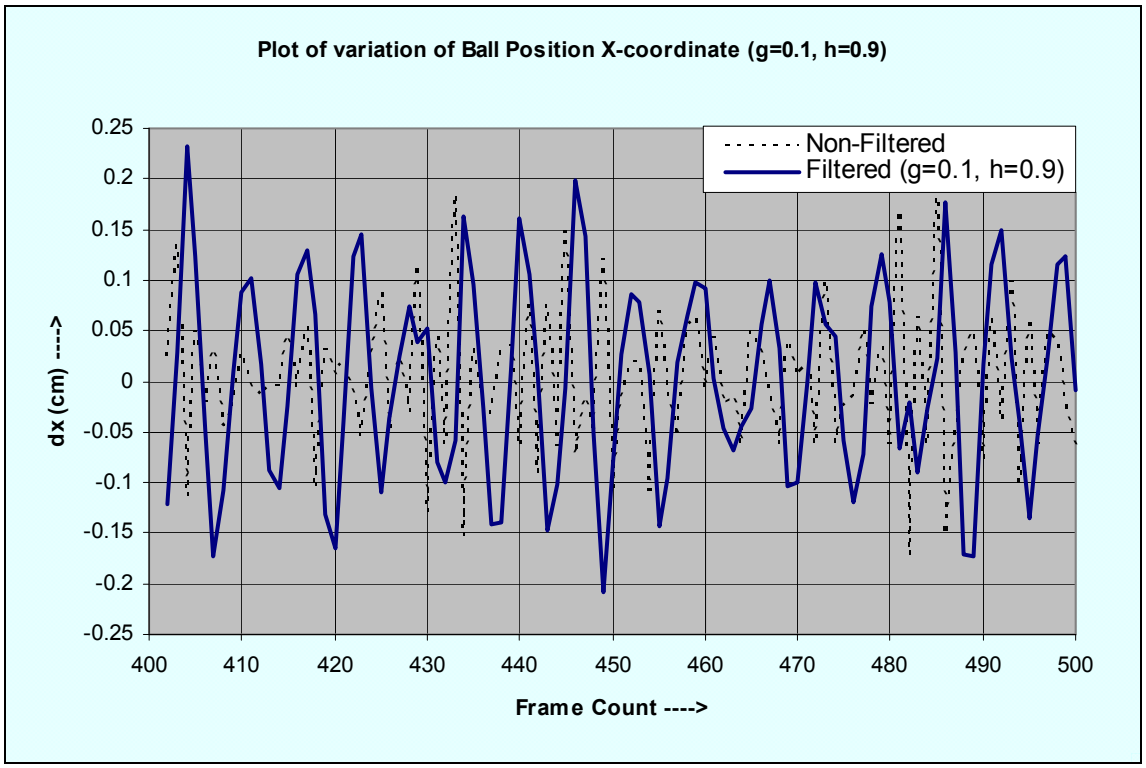


Figure 5.12 Frame-to-frame variation of X-coordinate of a stationary ball (*g=0.1, h=0.9*)

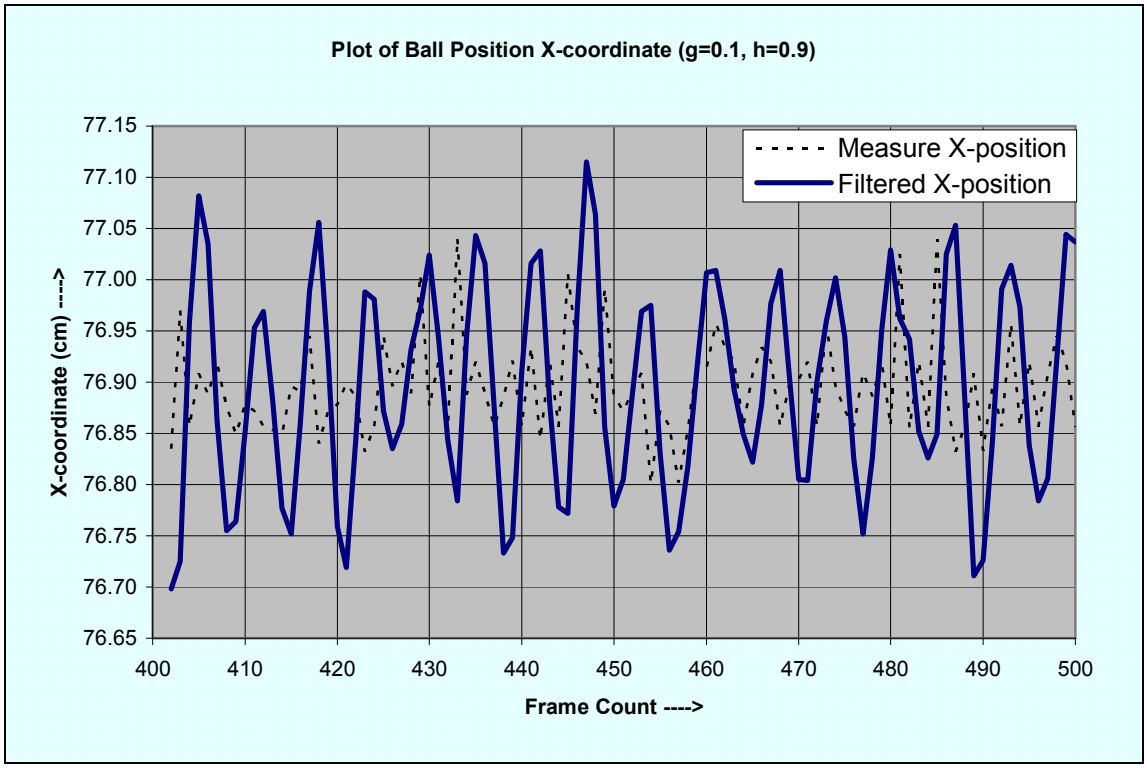


Figure 5.13 Plot of a stationary ball's X-position (*g=0.1, h=0.9*)

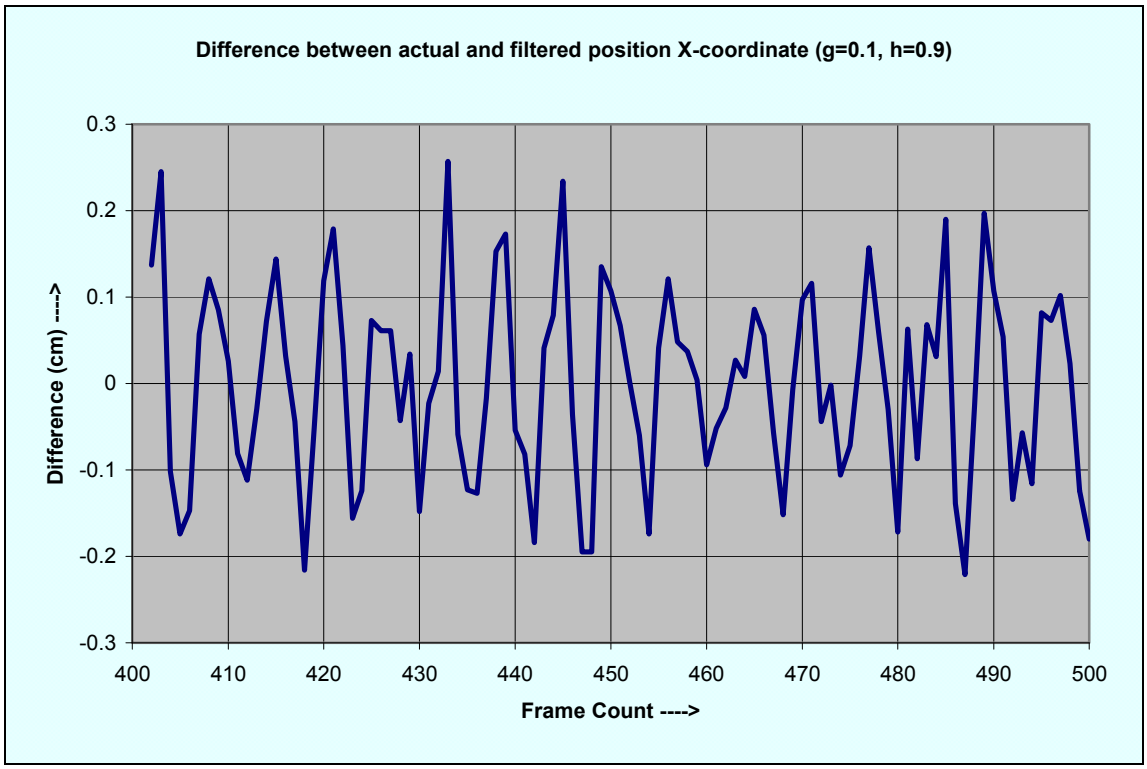


Figure 5.14 Plot of the difference between actual and filtered X-position (*g=0.1, h=0.9*)

Table 5.1 Filter evaluation and comparison data for ball position (X coordinate)

Filter Parameters		Maximum |*dx*| (mm)		Average |*dx*| (mm)		|difference between actual and filtered X coordinate| (mm)	
		Non-Filtered	Filtered	Non-Filtered	Filtered	Maximum	Average
1	*g=0.9, h=0.1*	2.32	1.91	0.47	0.40	0.23	0.05
2	*g=0.8, h=0.2*	1.93	1.48	0.45	0.33	0.43	0.09
3	*g=0.7, h=0.3*	1.85	1.20	0.44	0.27	0.55	0.14
4	*g=0.6, h=0.4*	2.15	1.23	0.49	0.25	1.10	0.22
5	*g=0.5, h=0.5*	2.60	1.20	0.44	0.21	1.44	0.26
6	*g=0.4, h=0.6*	1.70	0.8	0.47	0.22	1.36	0.33
7	*g=0.3, h=0.7*	1.85	1.61	0.46	0.29	2.11	0.43
8	*g=0.2, h=0.8*	2.15	1.47	0.62	0.44	2.35	0.59
9	*g=0.1, h=0.9*	1.84	2.33	0.59	0.73	2.97	0.83

Table 5.1 shows a sub-set of data from the experiments done using 99 different combinations of *g* and *h*. It summarises the data for X coordinate for 9 different sets of filter parameters. Figure 5.15 plots the average filtered *dx* (i.e. variation of X position of the stationary ball from one frame to another) and the average difference between actual and filtered X coordinate for the different filters. The intersection of the two plotted curves gives the desired value of *g* and *h*. It occurs between filter #4 and filter #5, closer to filter #4 (*g=0.6, h=0.4*).

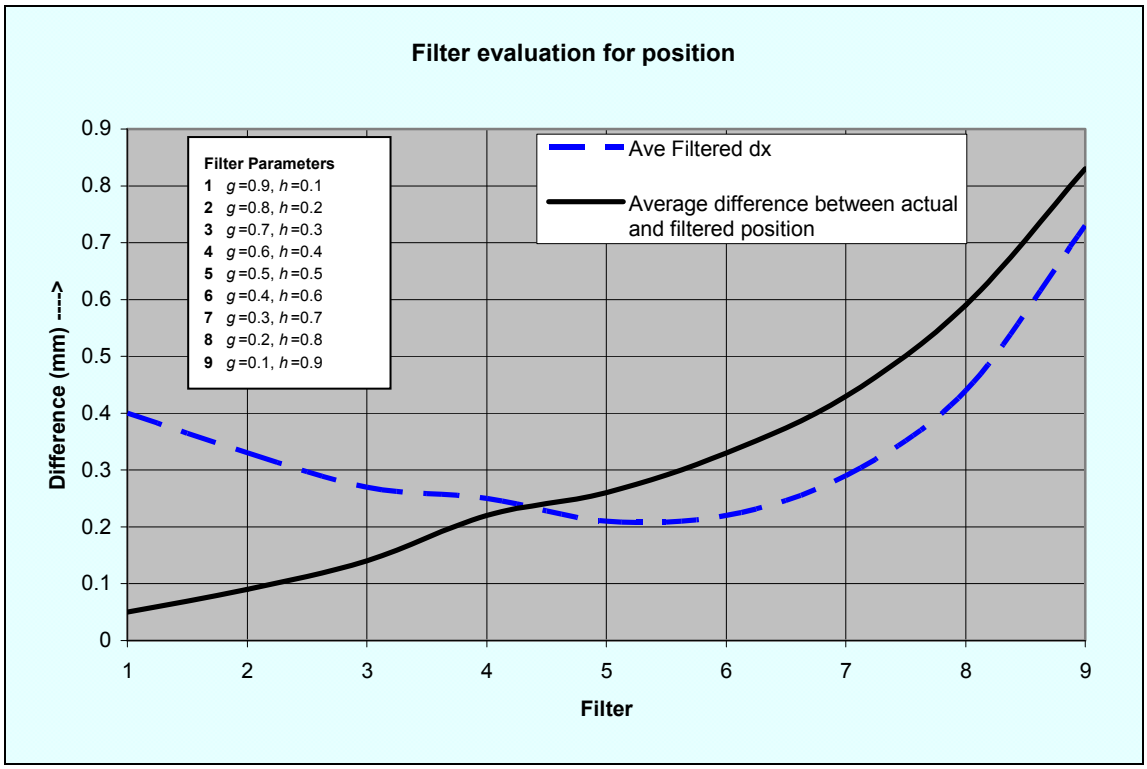


Figure 5.15 Filter evaluation based on measurement of position (X coordinate)

Table 5.2 Filter evaluation and comparison data for ball position (Y coordinate)

Filter Parameters		Maximum |*dy*| (mm)		Average |*dy*| (mm)		|difference between actual and filtered Y coordinate| (mm)	
		Non-Filtered	Filtered	Non-Filtered	Filtered	Maximum	Average
1	*g=0.9, h=0.1*	5.56	4.69	3.09	2.50	0.55	0.29
2	*g=0.8, h=0.2*	5.73	3.97	3.17	2.02	1.13	0.58
3	*g=0.7, h=0.3*	5.50	3.02	3.21	1.54	1.60	0.84
4	*g=0.6, h=0.4*	5.67	2.42	3.17	1.06	2.21	1.06
5	*g=0.5, h=0.5*	5.26	1.60	3.17	0.64	2.46	1.27
6	*g=0.4, h=0.6*	5.56	1.28	3.21	0.35	3.36	1.49
7	*g=0.3, h=0.7*	5.73	1.43	3.42	0.37	4.22	1.78
8	*g=0.2, h=0.8*	5.10	2.27	2.92	0.66	4.50	1.70
9	*g=0.1, h=0.9*	5.38	3.57	3.00	1.02	5.41	1.92

Table 5.2 summarises the experimental data for the Y coordinate for 9 different sets of filter parameters. Figure 5.16 plots the average filtered *dy* (i.e. variation of Y position of the stationary ball from one frame to another) and the average difference between actual and filtered Y coordinate for the different filters. The intersection of the two plotted curves gives the desired value of g and h. It occurs, as the graph shows, for filter #4 ($g=0.6$, $h=0.4$).

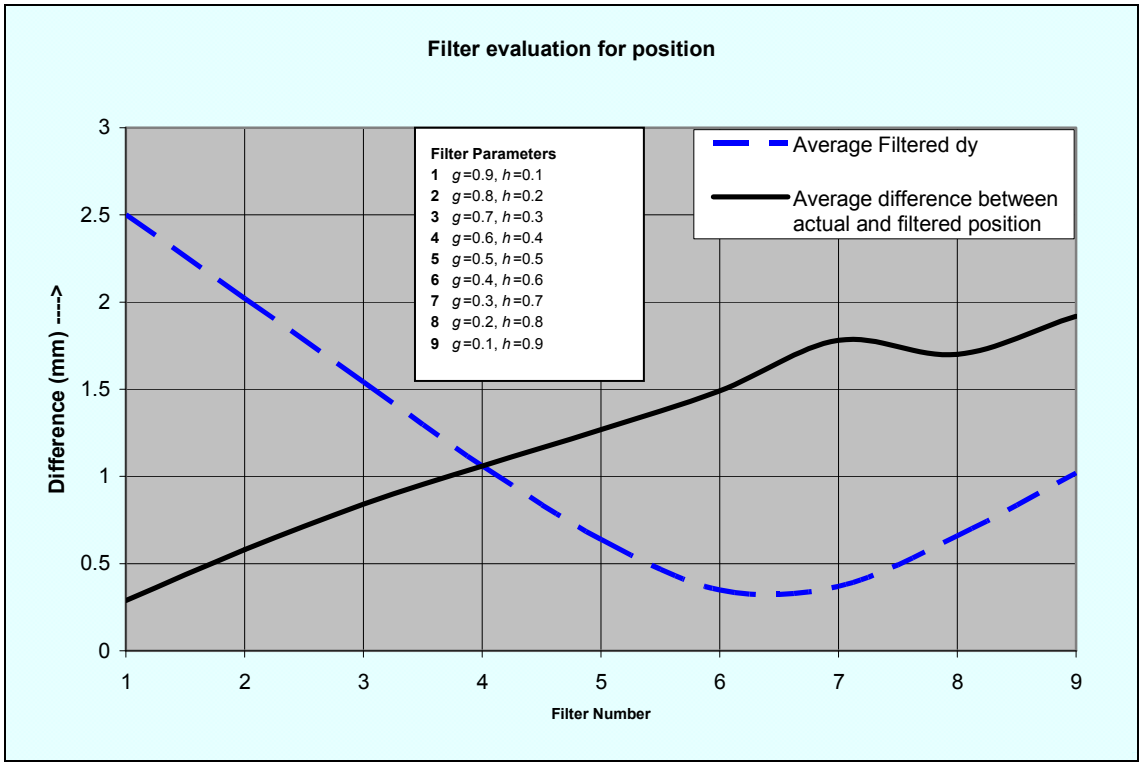


Figure 5.16 Filter evaluation based on measurement of position (Y coordinate)

Ball Interception Position of Goalkeeper. The vision sensor error significantly affects the performance of the object interception system because the predicted position of the object is used to calculate the interception point. A typical situation is when the goalkeeper needs to respond very quickly as a penalty shot is taken by the opponent striker (Figure 5.17). To realise an accurate and fast interception, its algorithm must predict a few frames ahead of the future position of the mobile object and fast and accurately calculate the required robot position. The robot's position along the Y-axis is calculated from the object's position and velocity (equation 5.15); while its position along the X-axis is kept constant (i.e. the robot moves in a straight line, parallel to the Y-axis, with minimal deviation).

$$Y^g_{n+1} = y^b_n + f_c \dot{y}^b_n$$
$$X^g_{n+1} = x^g_n \qquad (5.15)$$

Here (X^g_{n+1}, Y^g_{n+1}) is the target position for the goalkeeper, (x^g_n, y^g_n) is the current position of the goalkeeper, (x^b_n, y^b_n) is the current ball position, $(\dot{x}^b_n, \dot{y}^b_n)$ is the current ball velocity and f_c is the frame correction factor.

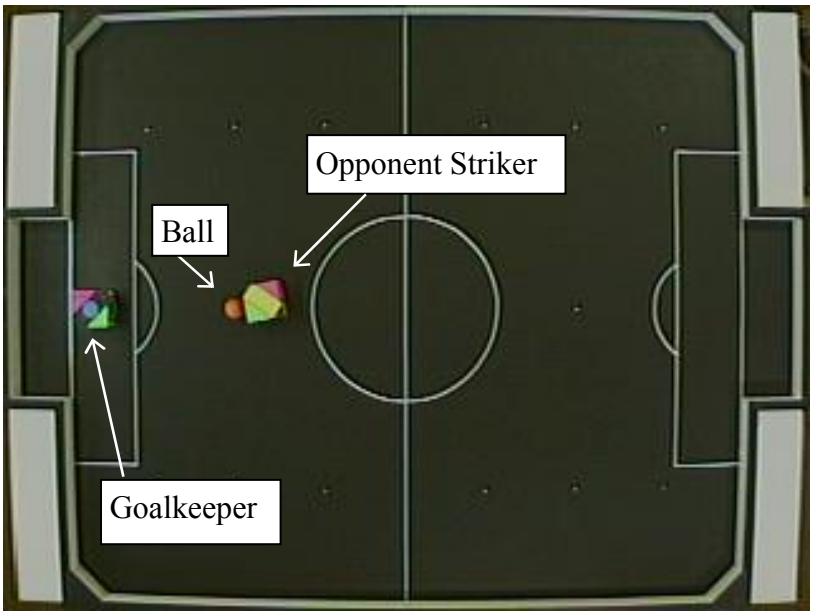


Figure 5.17 Penalty shot taken by the opponent striker

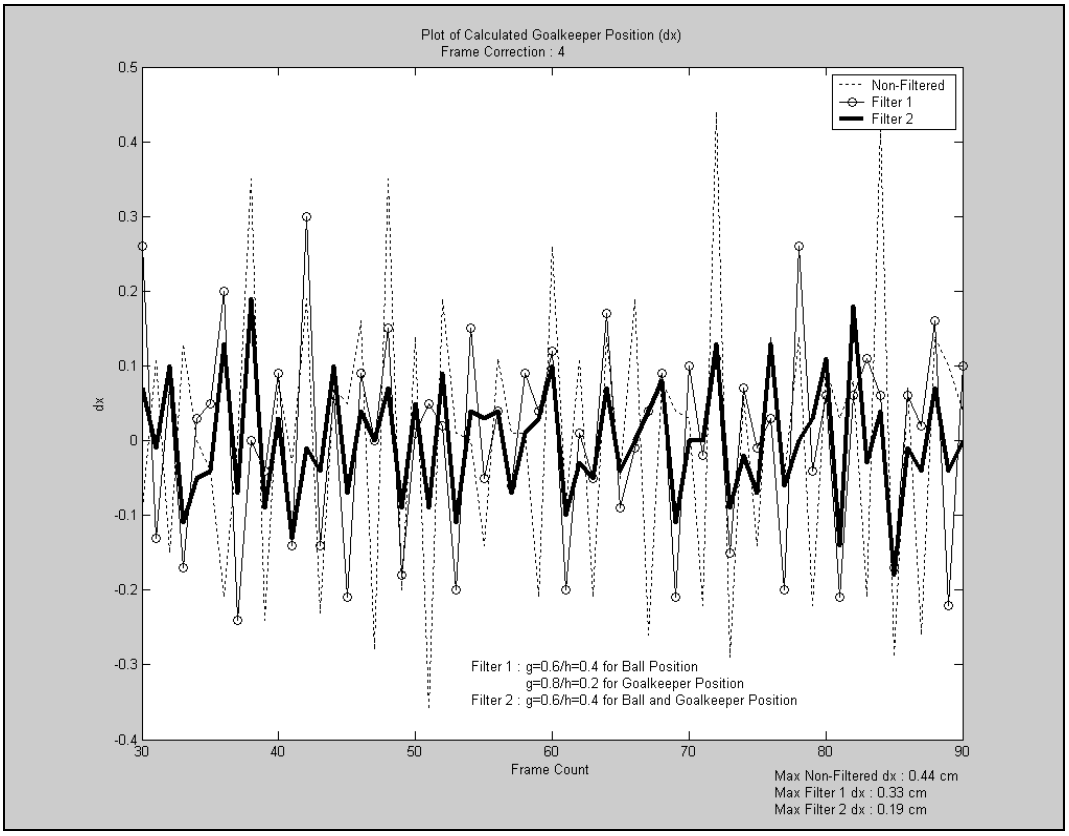


Figure 5.18 Plot of variation of calculated goalkeeper position (*dx*) for a penalty shot

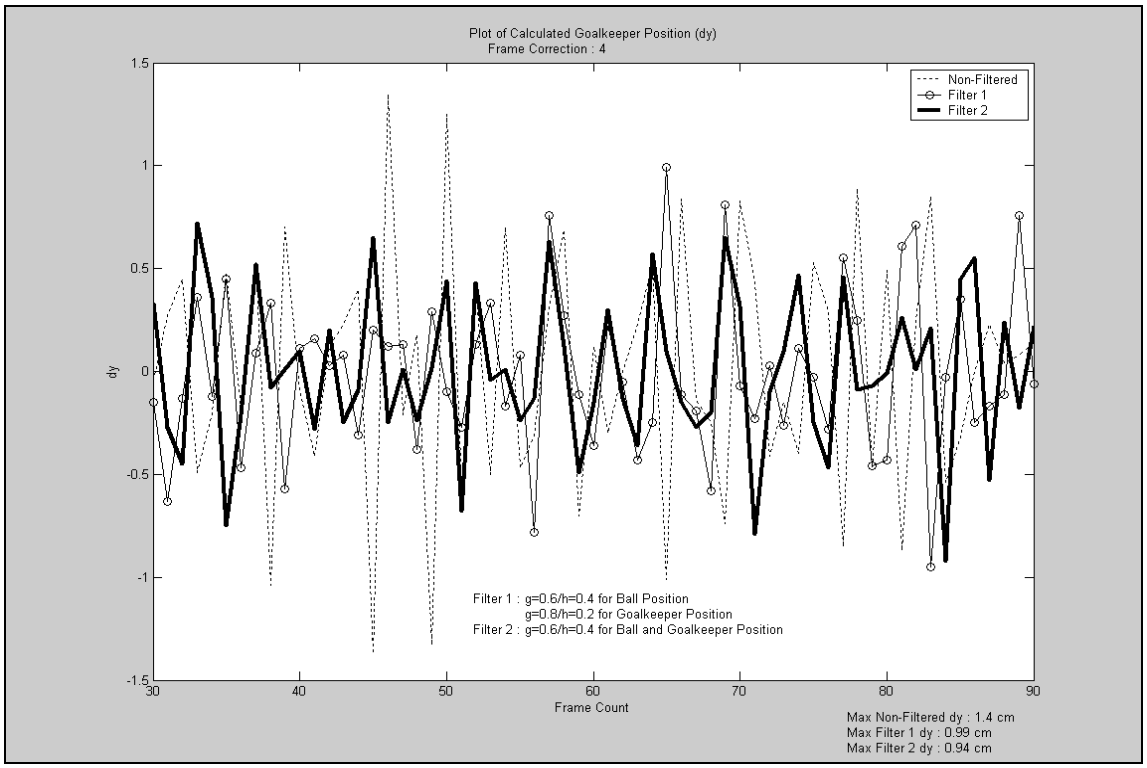


Figure 5.19 Plot of variation of calculated goalkeeper position (*dy*) for a penalty shot

For a stationary ball (as is the case for a penalty shot) the noise on the ball's position is interpreted as a velocity, resulting in large variations in the calculated position of the goalkeeper for interception. The filters significantly improve the performance of the goalkeeper facing a penalty shot. The position of a static ball, placed on the penalty spot, was measured and the required interception position of the goalkeeper was calculated. The point of interception was calculated using the algorithm shown in the flow chart of Figure 5.2. The frame-to-frame variation in the calculated position of the goalkeeper, *d*x and *d*y, using two different filters are presented in Figures 5.18 and 5.19 respectively. For filter parameters *g=0.6, h=0.4*, the maximum non-filtered *dx* value was 0.44 cm, which was reduced to 0.19 cm, while the maximum non-filtered *dy* value was 1.40 cm, which was lowered down to 0.94 cm. This significantly dampened the goalkeeper's movement when the ball was stationary, thereby improving its initial defensive position to guard the goal against the penalty shot.

Robot Angle. The filter performance was also evaluated by measuring the variation of angle of a stationary robot. The angle of a stationary robot was measured for different filter parameters. The value of *g* was varied from 0.1 to 0.9 in steps of 0.1 and for each value of *g*, the value of *h* was varied from 0 to 1.0 in steps of 0.1, resulting in 99 different combinations of *g* and *h*. For each set of filter parameters, the data were recorded for 500 frames.

The effects of filtering (with four different sets of filter parameters) on the variation of angle of the robot are shown in Figures 5.20 to 5.31. The variation of angle, over time, was recorded as:

$$d\theta^r = \theta_n^r - \theta_{n-1}^r \tag{5.16}$$

Here θ_n^r and θ_{n-1}^r are the robot angles in the current and previous frames respectively. The tracking filter, equations (5.8) and (5.9), were used to obtain running estimates of the target angular velocity and orientation.

For data plotted in Figures 5.20 to 5.22, the filter parameters were set to g=0.8, h=0.2. Figure 5.20 shows the variation ($d\theta$) of the robot's non-filtered and filtered angle from one frame to the next, Figure 5.21 plots the robot's measured and filtered angle in each frame and Figure 5.22 shows the difference in the actual and filtered angle. (The segment of data shown in the plots is the region in which the maximum $d\theta$ occurred.) Likewise, Figures 5.23 to 5.25 are for filter parameters set to g=0.7, h=0.3, Figures 5.26 to 5.28 are for filter parameters set to g=0.6, h=0.4 and Figures 5.29 to 5.31 are for filter parameters set to g=0.1, h=0.9.

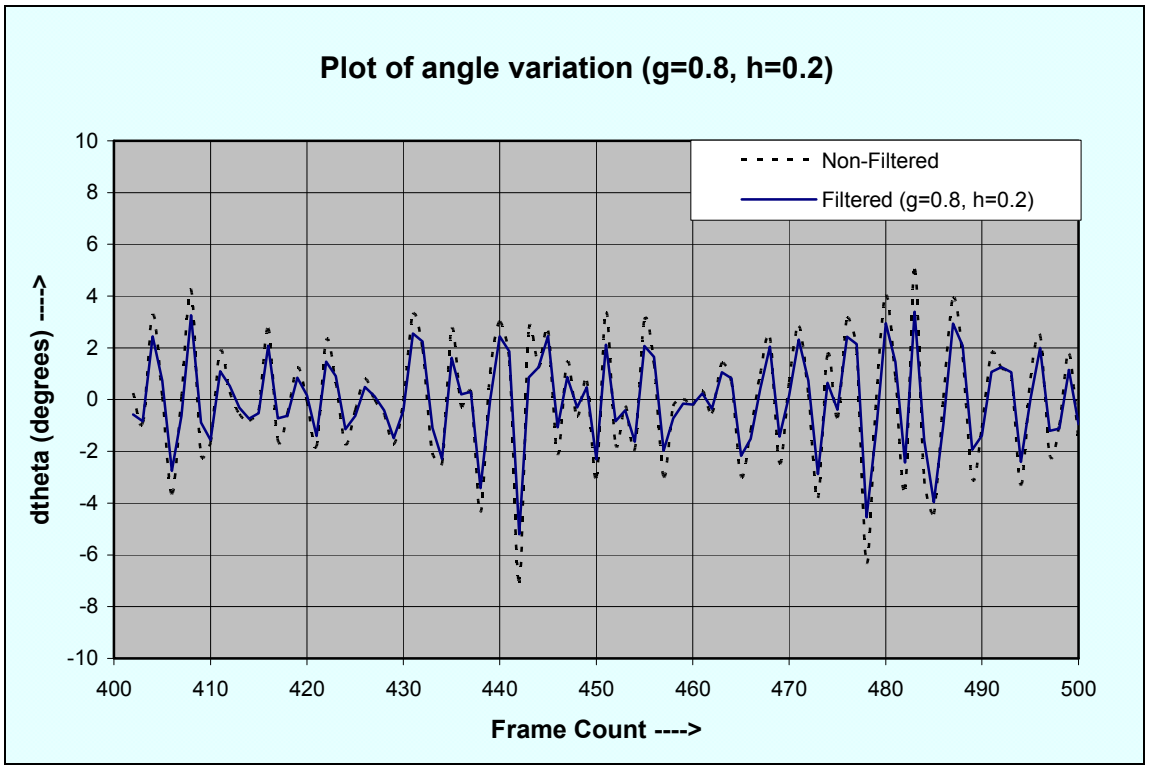


Figure 5.20 Frame-to-frame angle variation of a stationary robot (*g=0.8, h=0.2*)

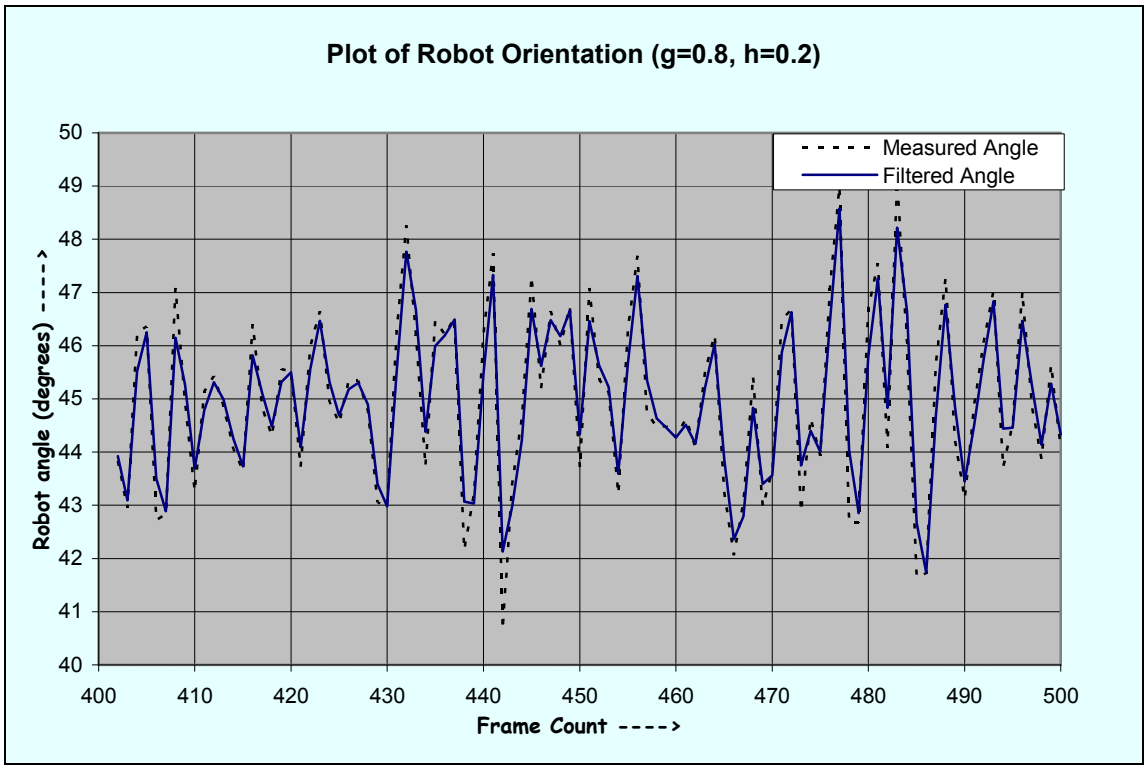


Figure 5.21 Plot of a stationary robot's angle (*g=0.8, h=0.2*)

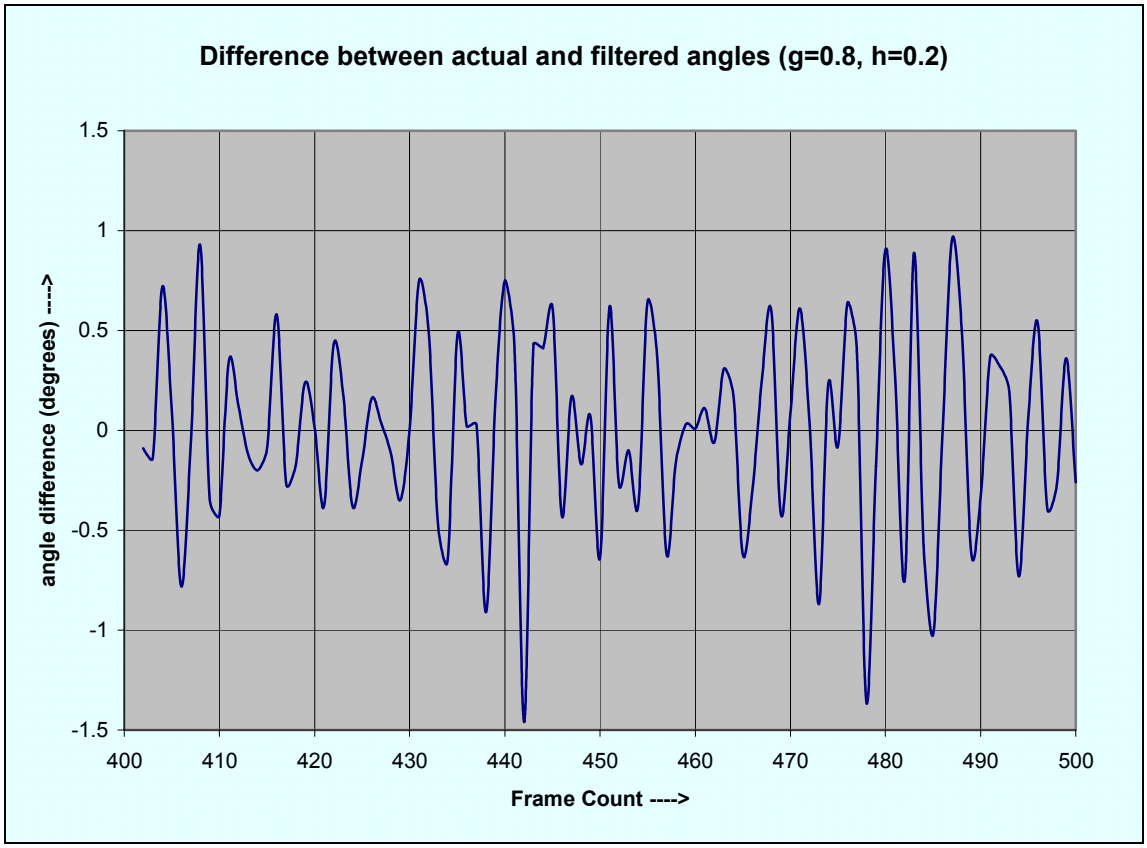


Figure 5.22 Plot of the difference between actual and filtered angles (*g=0.8, h=0.2*)

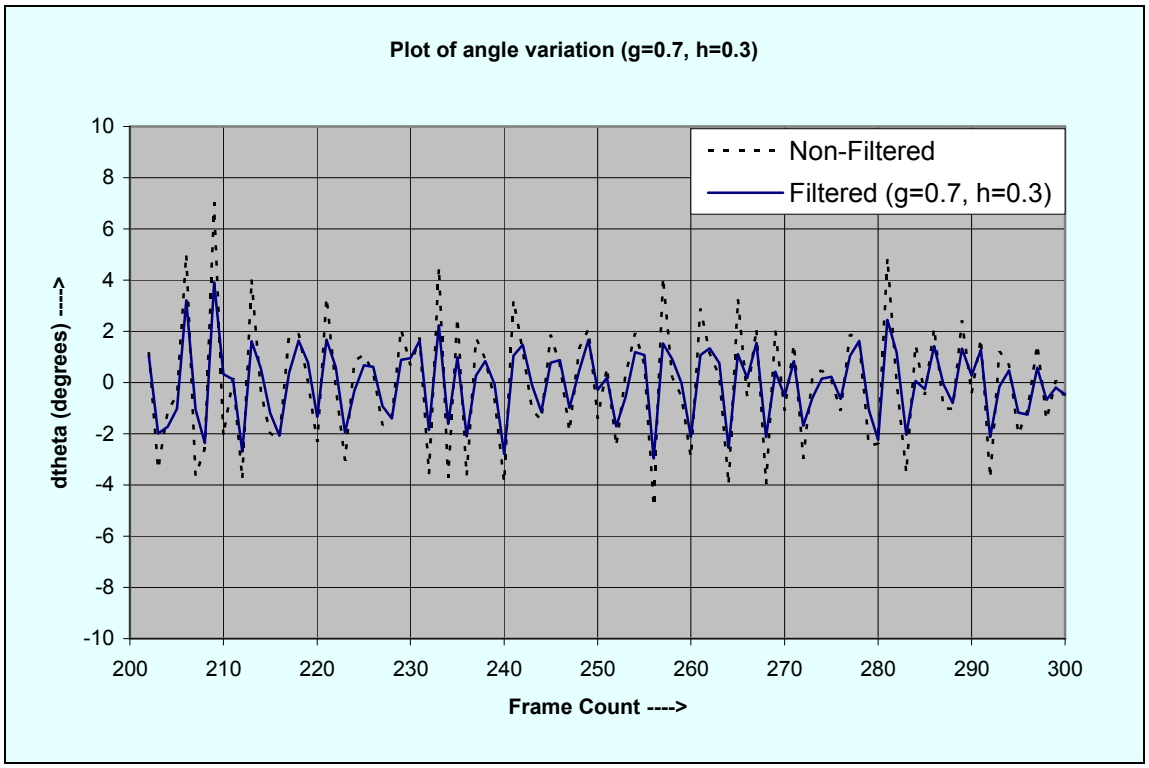


Figure 5.23 Frame-to-frame angle variation of a stationary robot (*g=0.7, h=0.3*)

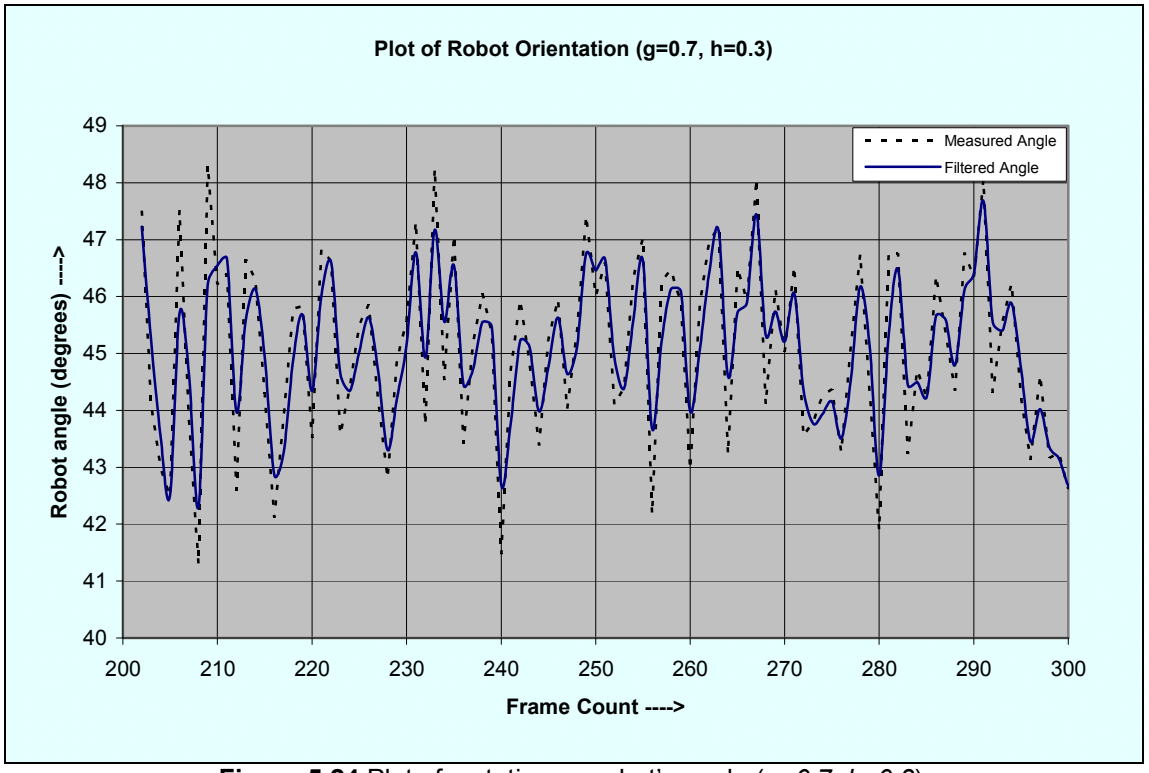


Figure 5.24 Plot of a stationary robot's angle (*g=0.7, h=0.3*)

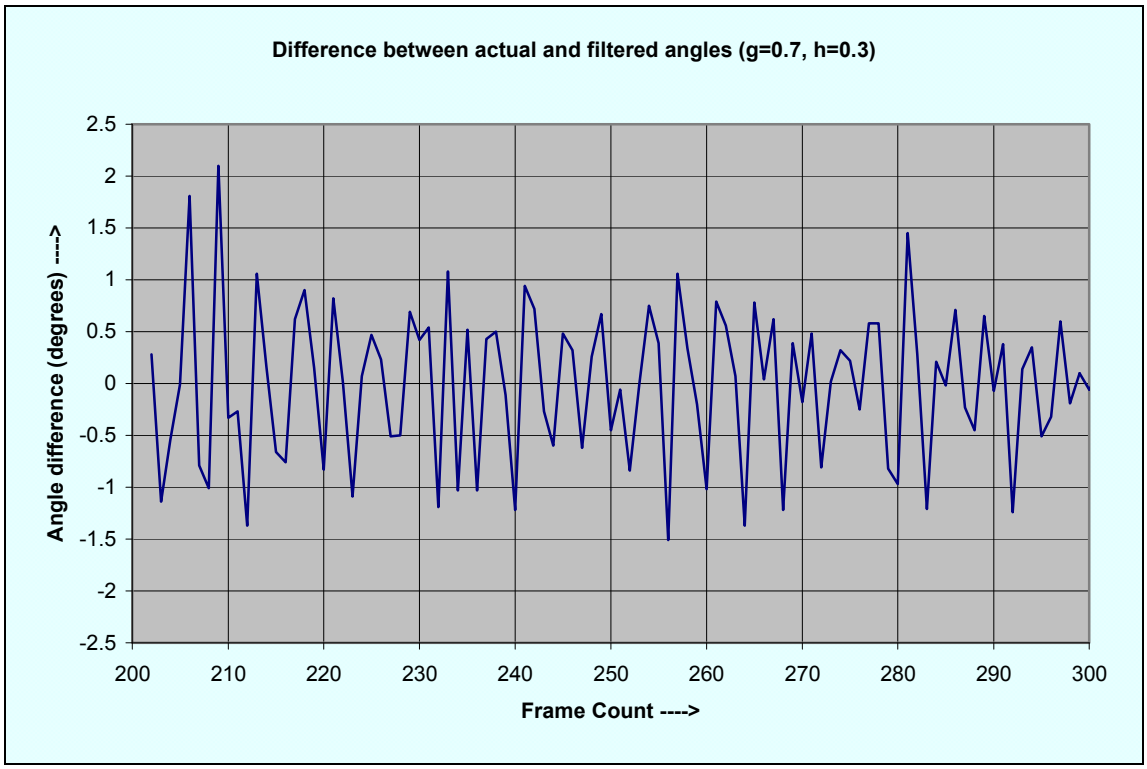


Figure 5.25 Plot of the difference between actual and filtered angles (*g=0.7, h=0.3*)

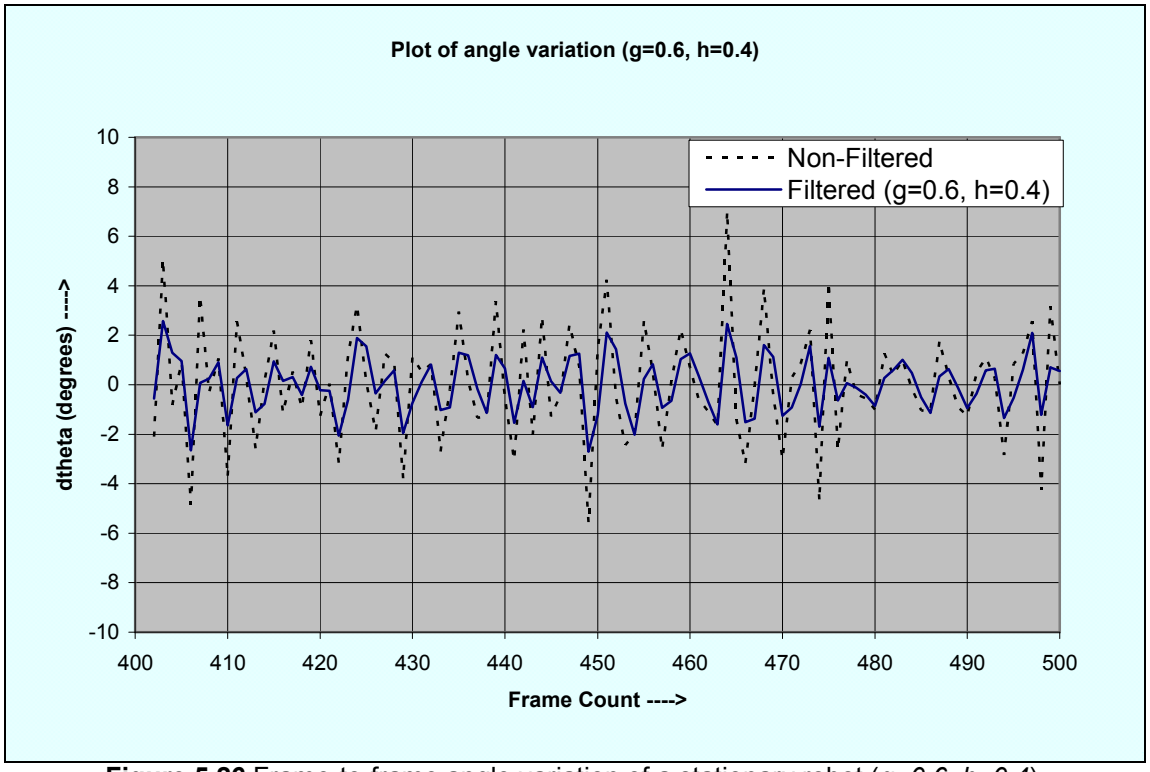


Figure 5.26 Frame-to-frame angle variation of a stationary robot (*g=0.6, h=0.4*)

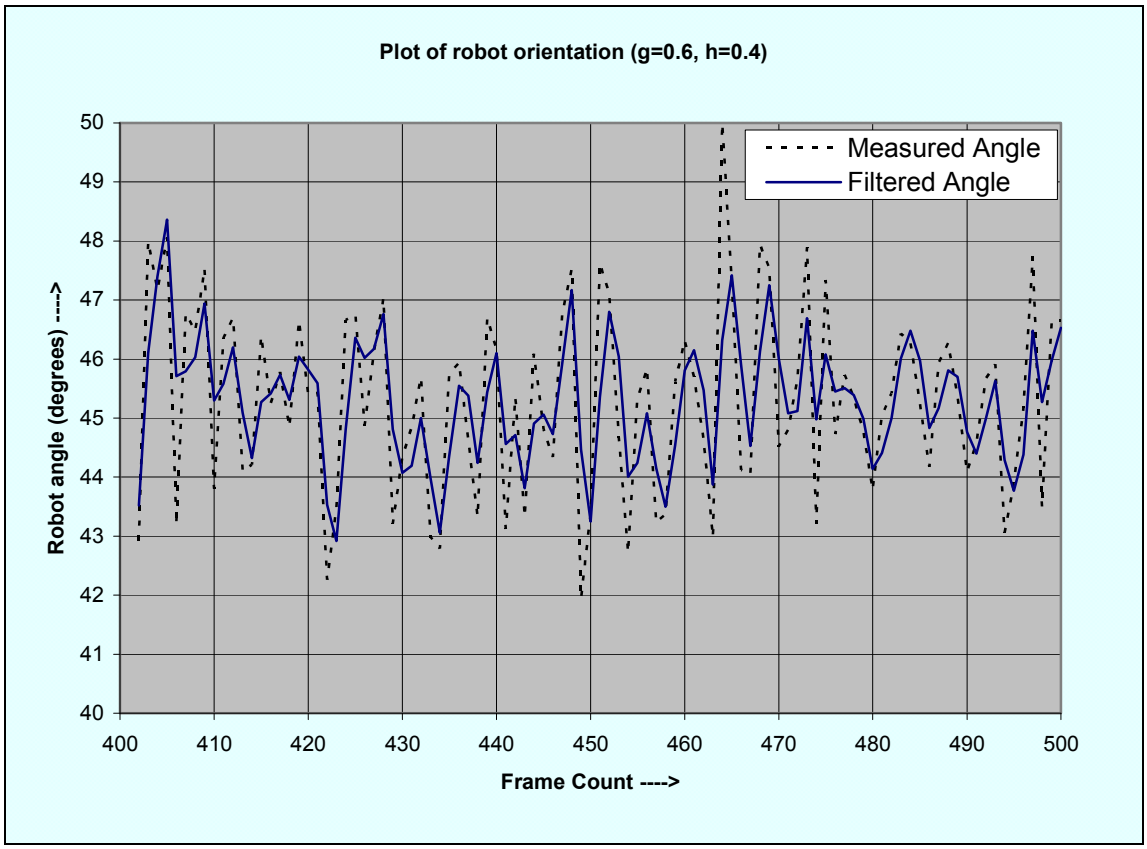


Figure 5.27 Plot of a stationary robot's angle (*g=0.6, h=0.4*)

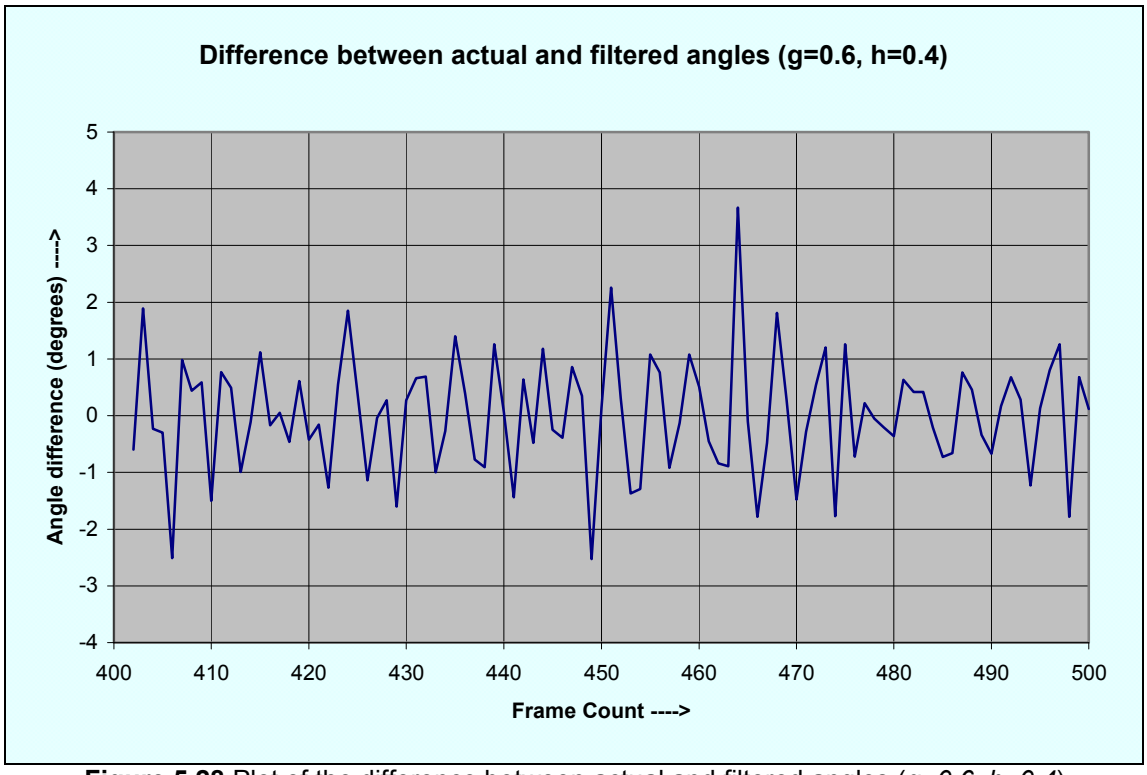


Figure 5.28 Plot of the difference between actual and filtered angles (*g=0.6, h=0.4*)

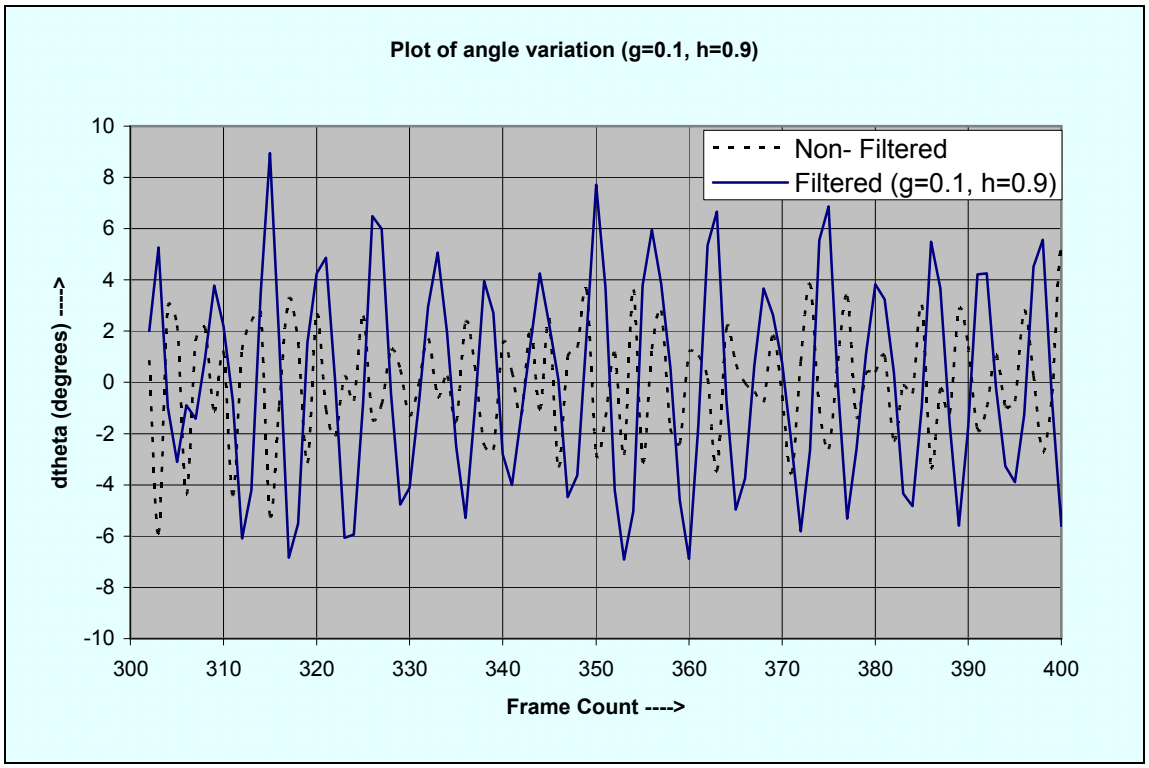


Figure 5.29 Frame-to-frame angle variation of a stationary robot (*g=0.1, h=0.9*)

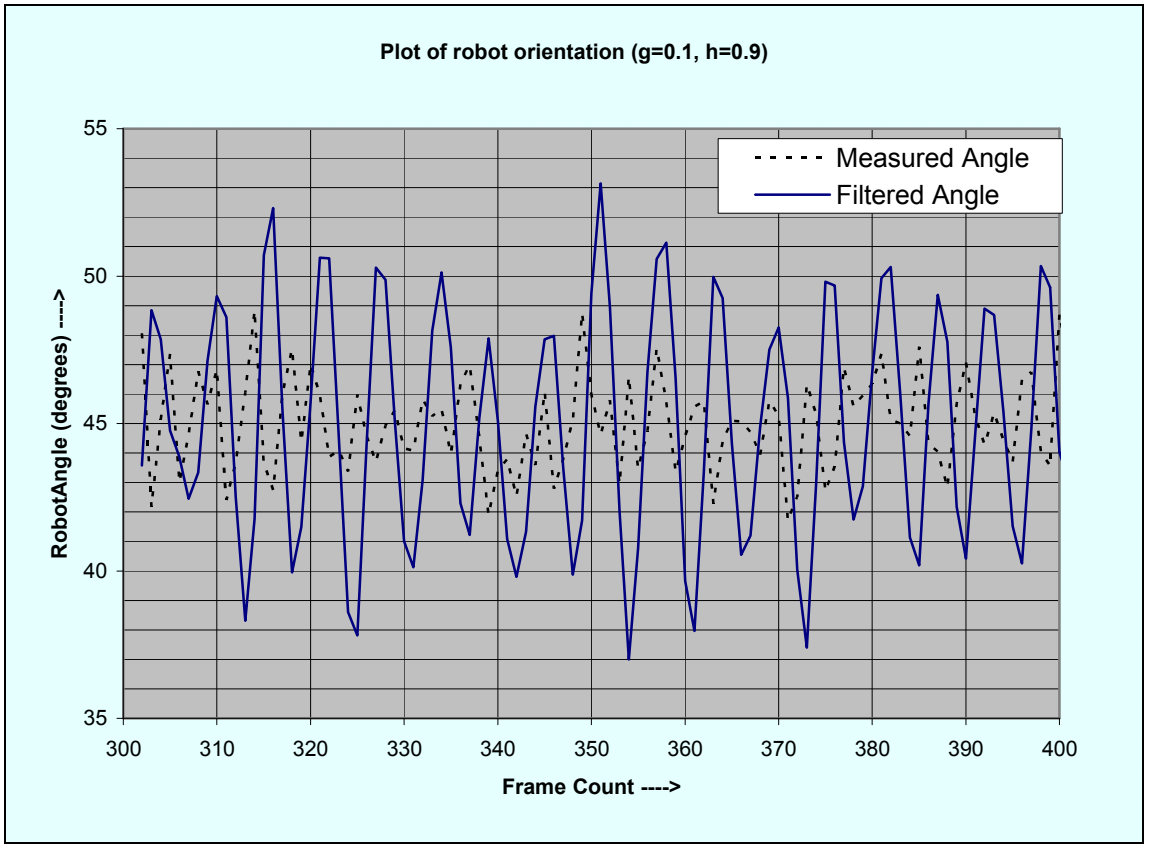


Figure 5.30 Plot of a stationary robot's angle (*g=0.1, h=0.9*)

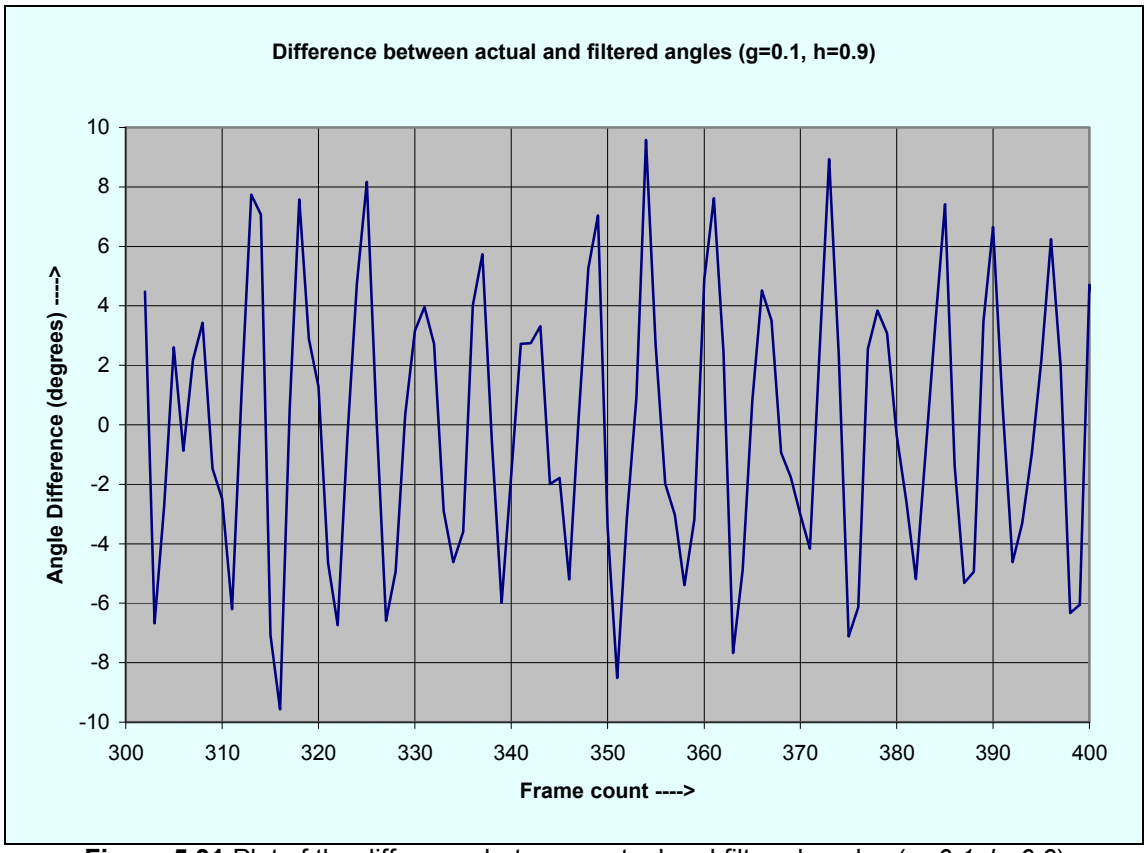


Figure 5.31 Plot of the difference between actual and filtered angles (*g=0.1, h=0.9*)

Table 5.3 Filter evaluation and comparison data for angle variation

Filter Parameters		Maximum $\lvert d\theta \rvert$		Average $\lvert d\theta \rvert$		$\lvert$Difference between actual and filtered angle$\rvert$	
		Non-Filtered	Filtered	Non-Filtered	Filtered	Maximum	Average
1	*g=0.9, h=0.1*	7.01°	5.76°	1.71°	1.47°	0.70°	0.17°
2	*g=0.8, h=0.2*	7.07°	5.20°	1.78°	1.34°	1.46°	0.37°
3	*g=0.7, h=0.3*	7.02°	3.91°	1.80°	1.11°	2.10°	0.58°
4	*g=0.6, h=0.4*	7.01°	3.34°	1.81°	0.92°	3.67°	0.79°
5	*g=0.5, h=0.5*	7.03°	3.13°	1.80°	0.80°	3.70°	1.03°
6	*g=0.4, h=0.6*	7.04°	3.09°	1.95°	0.89°	4.84°	1.40°
7	*g=0.3, h=0.7*	6.98°	4.90°	1.76°	1.25°	6.73°	1.74°
8	*g=0.2, h=0.8*	7.00°	5.05°	1.72°	1.55°	7.55°	2.04°
9	*g=0.1, h=0.9*	7.01°	8.95°	1.80°	2.28°	9.66°	2.81°

Table 5.3 summarises the experimental data for 9 different sub-sets of filter parameters. Figure 5.32 plots the average filtered $d\theta$ and the average difference between actual and filtered angle for the different filters. The intersection of the two plotted curves gives the first approximate value of g and h. It occurs between filter #4 and filter #5, closer to filter #4 (*g=0.6, h=0.4*).

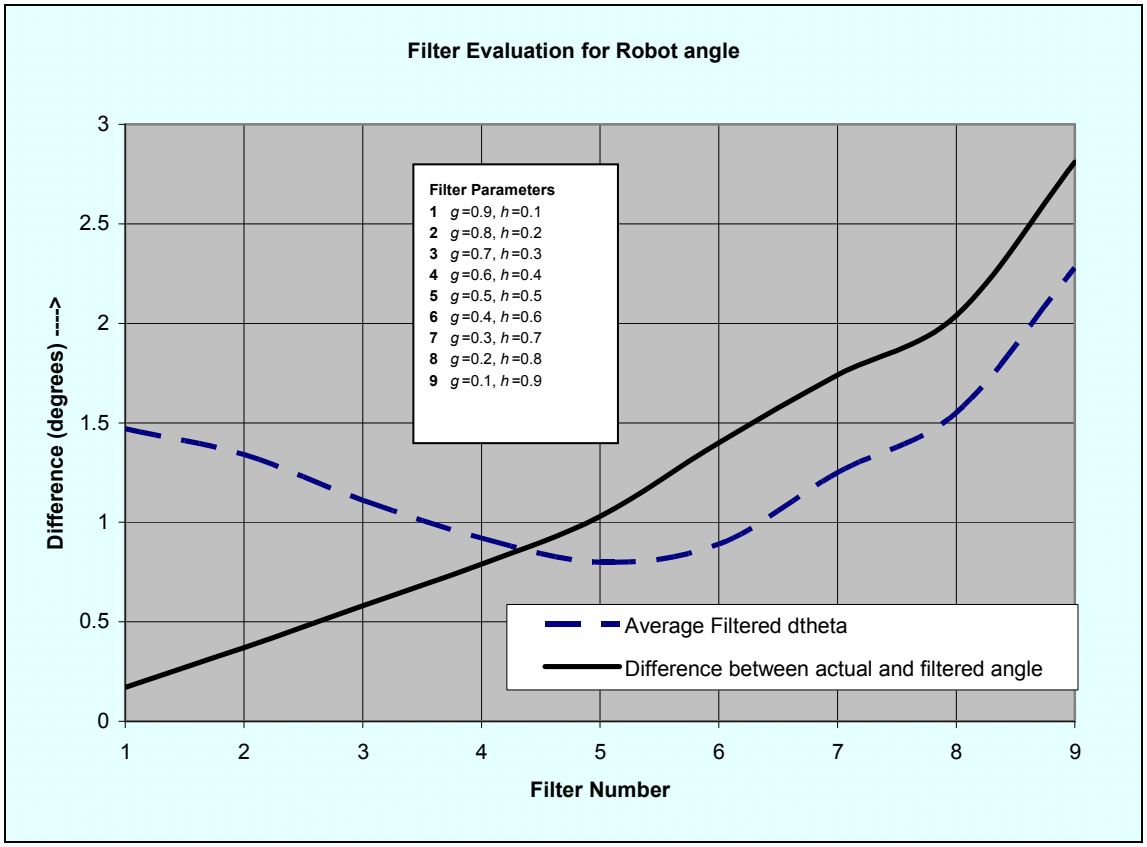


Figure 5.32 Filter evaluation based on measurement of angle

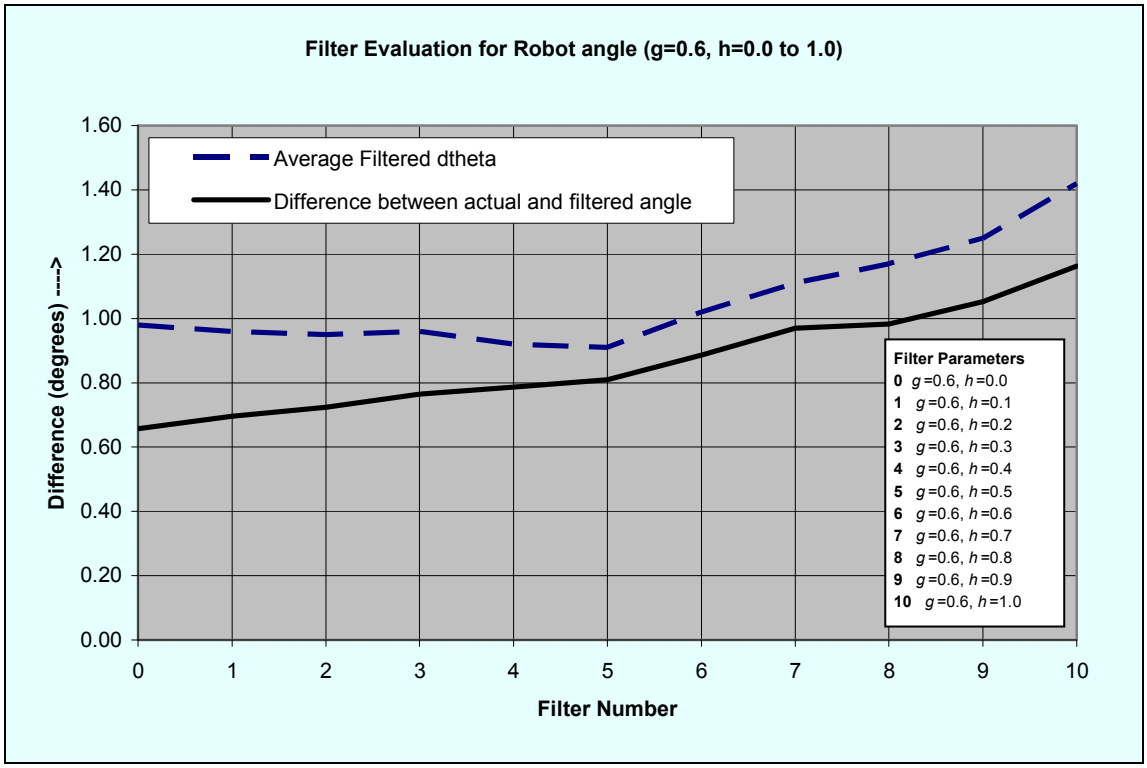


Figure 5.33 Filter evaluation based on measurement of angle (*g=0.6, h=0.0 to 1.0*)

Based on the data plotted in Figure 5.32, the optimal filter parameters are approximated to be *g=0.6, h=0.4*. For a finer practical judgement of the filter parameters, the data were plotted for *g*=0.6 and *h* varying from 0.0 to 1.0, which is shown in Figure 5.33. The corresponding data are shown in Table 5.4. The inference drawn from the plot of Figure 5.33 is that the best filter parameters for tracking the angle of a static object is *g=0.6, h=0.5* for which the average variation of angle from one frame to the next is the minimum and the average difference between the actual and filtered angle is the least. There is however not much to differentiate between the filters (*g=0.6, h=0.4*) and (*g=0.6, h=0.5*) because the average $d\theta$ for both is practically the same (0.92^o and 0.91^o respectively).

Table 5.4 Filter evaluation and comparison data for angle variation (*g*=0.6, *h*=0.0 to 1.0)

Filter Parameters		Maximum $\lvert d\theta \rvert$		Average $\lvert d\theta \rvert$		$\lvert$Difference between actual and filtered angle$\rvert$	
		Non-Filtered	Filtered	Non-Filtered	Filtered	Maximum	Average
0	*g=0.6, h=0.0*	6.72 ^{o}	3.73 ^{o}	1.79 ^{o}	0.98 ^{o}	2.48 ^{o}	0.657 ^{o}
1	*g=0.6, h=0.1*	6.67 ^{o}	3.26 ^{o}	1.84 ^{o}	0.96 ^{o}	2.36 ^{o}	0.696 ^{o}
2	*g=0.6, h=0.2*	6.64 ^{o}	3.12 ^{o}	1.82 ^{o}	0.95 ^{o}	2.44 ^{o}	0.724 ^{o}
3	*g=0.6, h=0.3*	6.14 ^{o}	2.98 ^{o}	1.80 ^{o}	0.96 ^{o}	2.33 ^{o}	0.765 ^{o}
4	*g=0.6, h=0.4*	7.01 ^{o}	3.34 ^{o}	1.81 ^{o}	0.92 ^{o}	3.67 ^{o}	0.787 ^{o}
5	*g=0.6, h=0.5*	5.96 ^{o}	2.86 ^{o}	1.73 ^{o}	0.91 ^{o}	2.55 ^{o}	0.809 ^{o}
6	*g=0.6, h=0.6*	5.75 ^{o}	3.08 ^{o}	1.78 ^{o}	1.02 ^{o}	3.14 ^{o}	0.886 ^{o}
7	*g=0.6, h=0.7*	8.02 ^{o}	3.81 ^{o}	1.82 ^{o}	1.11 ^{o}	3.38 ^{o}	0.970 ^{o}
8	*g=0.6, h=0.8*	6.92 ^{o}	3.97 ^{o}	1.74 ^{o}	1.17 ^{o}	3.61 ^{o}	0.983 ^{o}
9	*g=0.6, h=0.9*	6.28 ^{o}	3.97 ^{o}	1.80 ^{o}	1.25 ^{o}	3.47 ^{o}	1.052 ^{o}
10	*g=0.6, h=1.0*	6.03 ^{o}	4.93 ^{o}	1.85 ^{o}	1.42 ^{o}	4.37 ^{o}	1.163 ^{o}

Using the complete set of 99 measurement points and plotting the intersection of the 3D surface curves (of the average filtered $d\theta$ and the average difference between actual and filtered angle) will not reveal any additional useful information which will help to arrive at a better practical value of the filter parameters.

5.5.2 Robot Motion Trajectory

This section details the outcome of the investigation done on the effect of filtering on the robot's dynamic performance. In the first phase of the experiment, the robot was commanded to move from point (15.0, 65.0) to point (130.0, 65.0) which entails only linear motion (no angular motion), resulting in a change in the X coordinate only. Figure 5.34 shows the performance of the system with and without filtering for a *scaleFactor* of 0.8 (i.e. reduction in velocities of 20%).At the 60^{th} frame, the robot's X-

coordinate is 6.77 cm more than it was without filtering. The response is faster with filtering, yet the system shows no over-shoot.

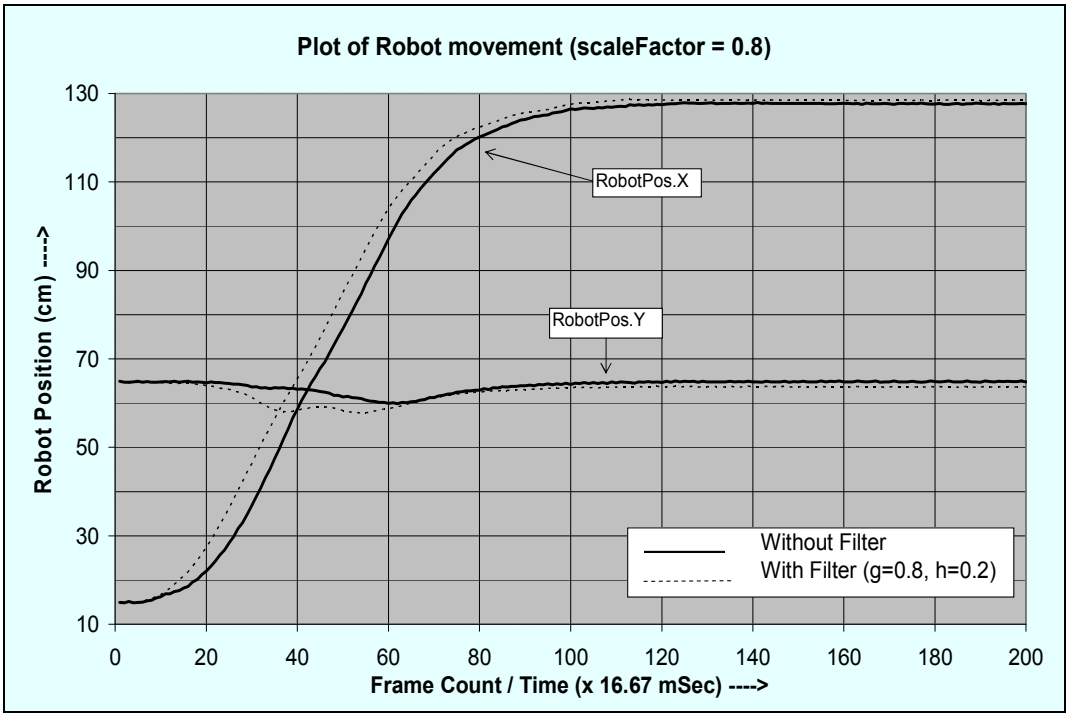


Figure 5.34 Robot motion trajectories with and without filtering (*scaleFactor* = 0.8)

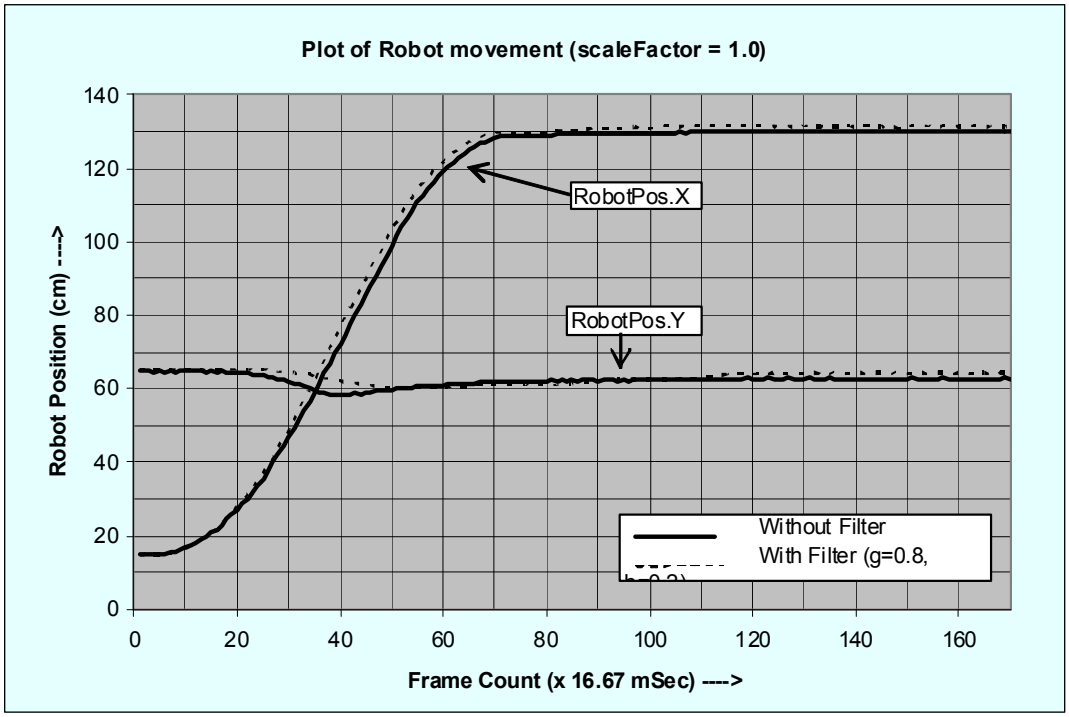


Figure 5.35 Robot motion trajectories with and without filtering (*scaleFactor* = 1.0)

For a *scaleFactor* of 1.0 (i.e. normal speed), the robot's movements are shown in Figure 5.35. The robot is much faster and reaches its destination earlier. At the 60th frame, the robot's x-coordinate is approximately 120 cm, i.e. it has traversed a distance of 105 cm. For the *scaleFactor* of 0.8 (Figure 5.34), the robot's x-coordinate at the 60th frame is approximately 100 cm, i.e. the robot has traversed a distance of 85 cm.

In the second phase of the experiment, the robot was placed at location (20.0, 20.0) at an angle of 90^{o}. It was commanded to move to point (130.0, 110.0) with a final orientation of 0^{o}. Figure 5.36 shows the spatial movement of the robot for different values of filter parameters. As can be seen in Figure 5.36, the path taken by the robot to move from its initial position to the final position is wobbly without any filtering. For the data plotted in Figure 5.36, the actual distance traversed (*d*) by the robot to move from the start position (20.0, 20.0) to the end position (130.0, 110.0) was measured as a sum of the distance travelled per sample (equation 5.17) and is summarised in Table 5.5 along with the distance error in position at the destination. The robot performs best with filter parameters of *g*=0.8 and *h*=0.2.

$$d = \sum_{n=0}^{N} \sqrt{\left(x_n^r - x_{n-1}^r\right)^2 + \left(y_n^r - y_{n-1}^r\right)^2} \tag{5.17}$$

Here (x_n^r, y_n^r) and (x_{n-1}^r, y_{n-1}^r) are the position of the robot in the current and previous frames respectively.

Table 5.5 Actual distance travelled by the robot (*scaleFactor* = 1)

Filter Parameters	Actual distance travelled (*d*)	Position Error at destination
g=0.8, h=0.2	**155.14 cm**	**1.478 cm**
g=0.6, h=0.4	158.42 cm	1.919 cm
Without filter	158.43 cm	2.197 cm

The experiment of moving the robot from start position (20.0, 20.0) to the end position (130.0, 110.0) was repeated many times and the actual distance travelled was measured. Table 5.6 shows the average distance travelled and the standard deviation for 10 runs. As can be seen from the data, the robot performs best with filter parameters of *g*=0.8 and *h*=0.2 for which the average distance travelled and the standard deviation is the least.

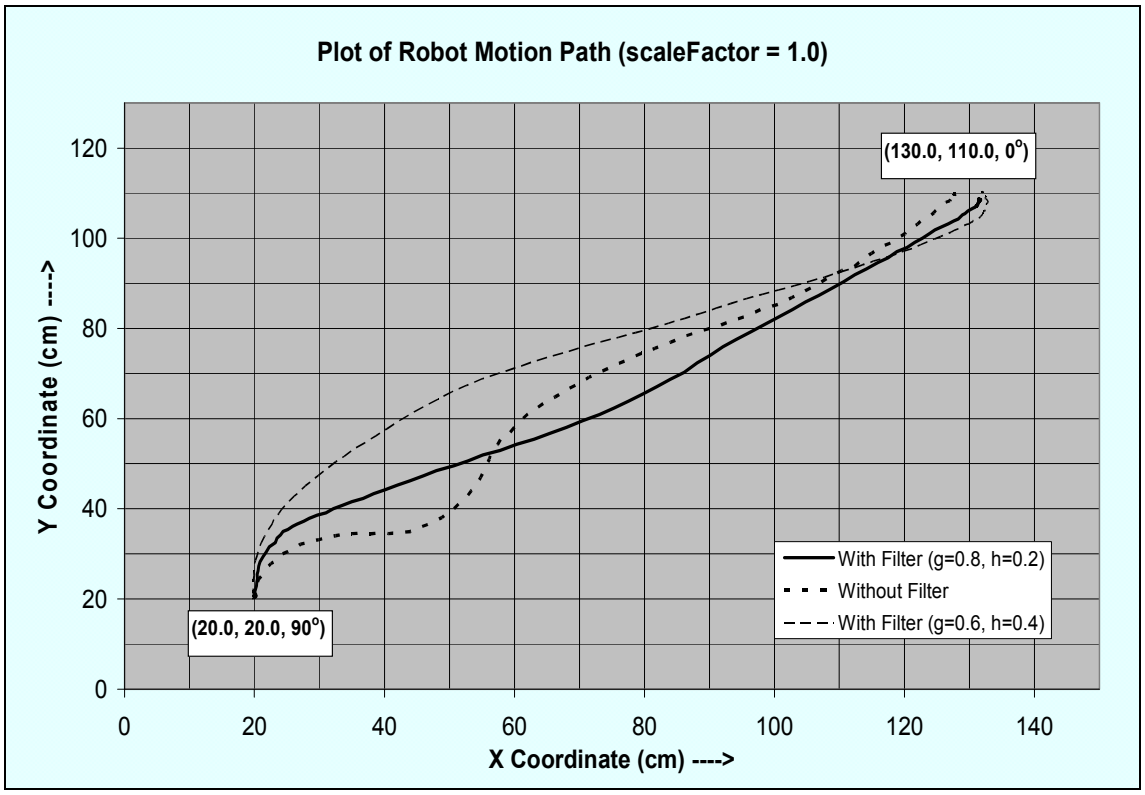


Figure 5.36 Robot motion trajectories with and without filtering

Table 5.6 Analysis of actual distance travelled by the robot

Run	Without Filter	*g=0.6, h=0.4*	*g=0.8, h=0.2*
1	158.43 cm	158.42 cm	155.14 cm
2	157.96 cm	157.78 cm	154.98 cm
3	158.89 cm	157.76 cm	154.82 cm
4	158.08 cm	158.06 cm	155.06 cm
5	157.67 cm	157.67 cm	154.86 cm
6	157.78 cm	158.32 cm	154.89 cm
7	158.14 cm	158.16 cm	155.12 cm
8	158.66 cm	157.89 cm	155.14 cm
9	158.55 cm	157.78 cm	154.86 cm
10	157.72 cm	157.90 cm	154.92 cm
Average	*158.19 cm*	*157.97 cm*	***154.98 cm***
Standard Deviation	*0.424913*	*0.255308*	***0.126179***

It should be noted that the shortest distance between points (20.0, 20.0) and (130.0, 110.0) is 142.13 cm. [$\sqrt{(130-20)^2+(110-20)^2}$]. Hence the robot always traverses a distance which is greater than the shortest. For the robot to reach the destination by traversing the shortest path, it would have to turn on the spot at the start location to orient towards the destination, move along a straight line and on reaching the destination turn again on the spot to orient itself to the final angle. This *turn-move-turn* approach is potentially detrimental in a dynamic environment where the destination (target) is constantly changing; the robot may end up wasting a lot of time turning to orient with the direction of the (moving) target. Hence the robot is set to start moving as soon as it is within a set limit of the target angle which results in a 'loopy' trajectory. This is explained in detail in Chapter 6 and an advanced motion control algorithm, called the Triangular Targeting Algorithm (TTA), is proposed which enables the robot to approach the target from behind with a desired angle.

Table 5.7 shows the positional error at the destination and the standard deviation for 10 runs.

Table 5.7 Analysis of positional error of the robot at the destination

Run	Without Filter	*g=0.6, h=0.4*	*g=0.8, h=0.2*
1	2.20 cm	1.92 cm	1.48 cm
2	2.21 cm	1.99 cm	1.52 cm
3	2.46 cm	1.78 cm	1.60 cm
4	2.52 cm	1.86 cm	1.49 cm
5	2.33 cm	1.90 cm	1.55 cm
6	2.35 cm	1.82 cm	1.56 cm
7	2.22 cm	1.87 cm	1.44 cm
8	2.45 cm	1.92 cm	1.51 cm
9	2.09 cm	2.01 cm	1.49 cm
10	2.38 cm	1.89 cm	1.48 cm
Average	*2.321 cm*	*1.896 cm*	***1.512 cm***
Standard Deviation	*0.137635*	*0.070111*	***0.046857***

The effect of filtering, or the lack of it, at substantially higher speeds were also measured and analysed. The robot's wheel velocities were scaled up by 20% and a set of readings taken. The robot's trajectory is plotted in Figure 5.37. Table 5.8 summarises the actual distance travelled and the position error at the destination. As can be seen

from the data, there is not much difference in the actual distance travelled; however the positional error at the destination is the minimal for filter parameters of *g=0.8, h=0.2*.

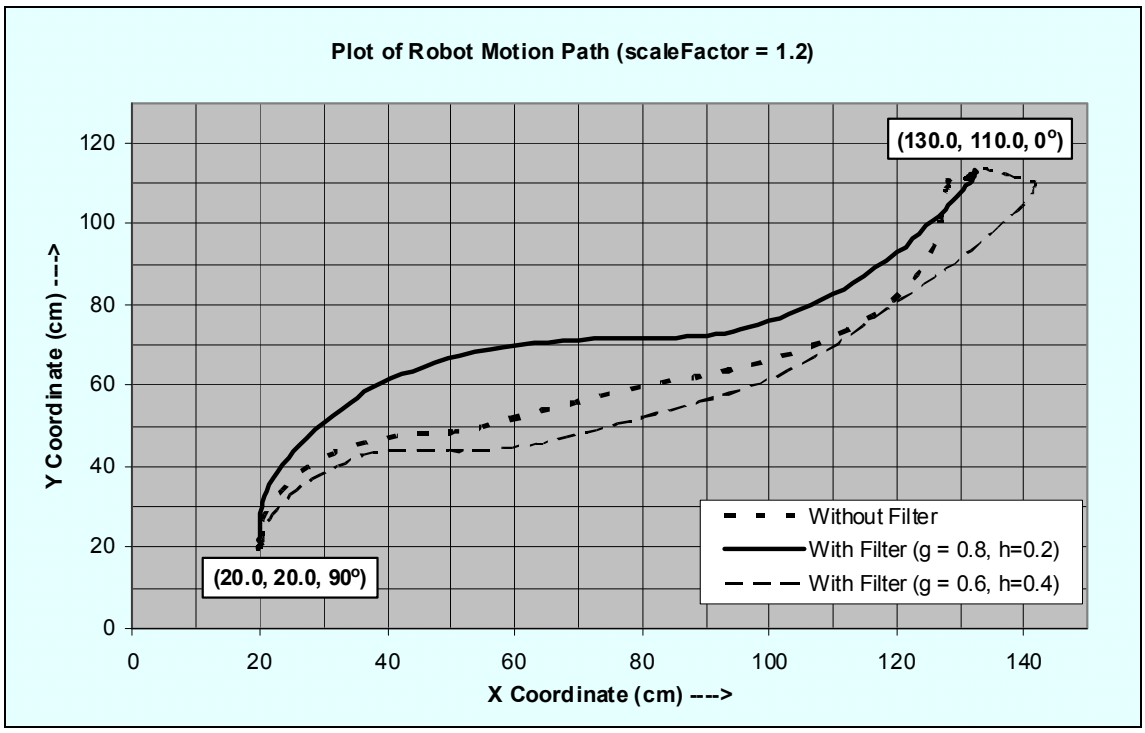


Figure 5.37 Robot motion trajectories with and without filtering at higher speeds

Table 5.8 Actual Distance travelled by the robot (scaleFactor = 1.2)

Filter Parameters	Actual distance travelled (*d*)	*Position Error at destination*
g=0.8, h=0.2	171.12 cm	**1.62 cm**
g=0.6, h=0.4	171.38 cm	2.43 cm
Without filter	169.91 cm	1.75 cm

5.5.3 Intercepting or Blocking a Moving Object

The prediction algorithm of the controller was validated by experiments conducted on a moving target (ball in this case) that was intercepted by a robot (goalkeeper). The target velocity, initial distance between robot and target and the target trajectory was varied and the system's performance recorded.

In Figure 5.38, the target is moving at a high speed of 1.318 m/s. The robot is still able to intercept, though barely. The average robot speed is 25.44 cm/s and the time to intercept is 47 frames (783.5 ms). A slower moving target is intercepted better as shown in Figure 5.39. The target speed is 53.44 cm/s and the average speed of the robot is 9.4 cm/s. The time to intercept is 100 frames (1667 ms).

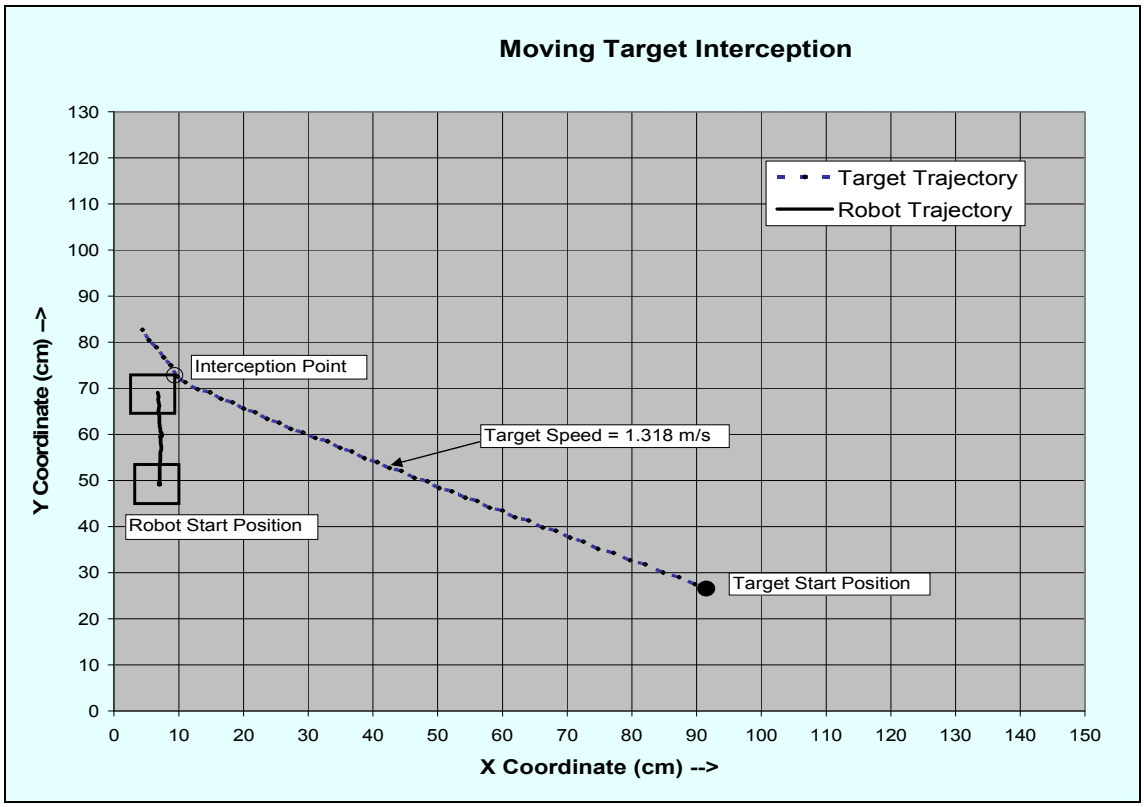


Figure 5.38 Goalkeeper intercepting a fast moving target

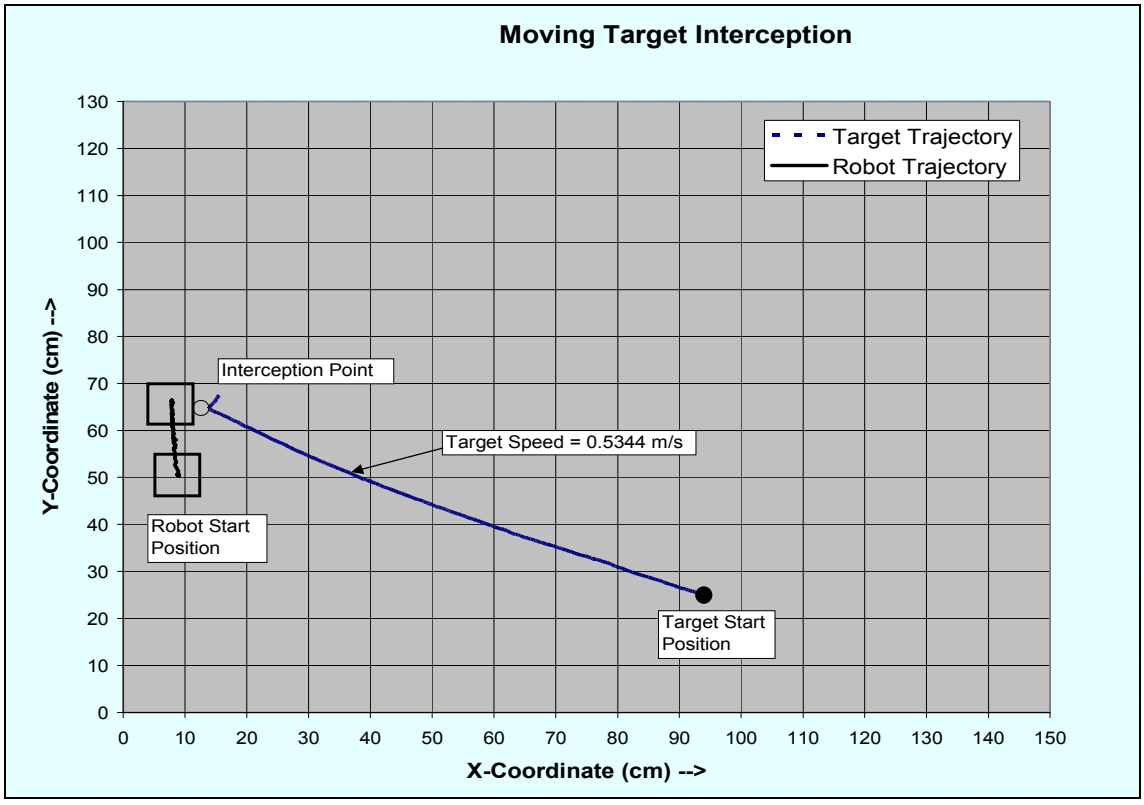


Figure 5.39 Goalkeeper intercepting a slow moving target

An experiment was conducted to evaluate the performance of the controller when the target is rebound from the boundary walls of the field resulting in a sudden change of direction and a sharp drop in velocity. The ball trajectory and the interception point are shown in Figure 5.40. The target was intercepted successfully. After rebound, the target speed dropped drastically from 1.93 m/s to 1.0m/s.

With uniform lighting conditions and the initial position of the robot in line with the goal line, the robot is able to intercept the ball on every shot (at low ball speeds and speeds of up to 1.3m/sec), thus proving the robustness of the vision tracking system and the predictive controller.

Figure 5.41 and 5.42 show two more examples of target interception with the target and robot starting at different locations and at varying distance from each other. For the experimental results of Figure 5.38 to 5.42, the filter parameters were set to *g=0.8, h=0.2*. The performance of the controller was not satisfactory when the filtering and prediction was turned off. It can be seen in Figure 5.43, the robot failed to intercept the target. The position of the target and the robot at the 80th and 110th frames are shown.

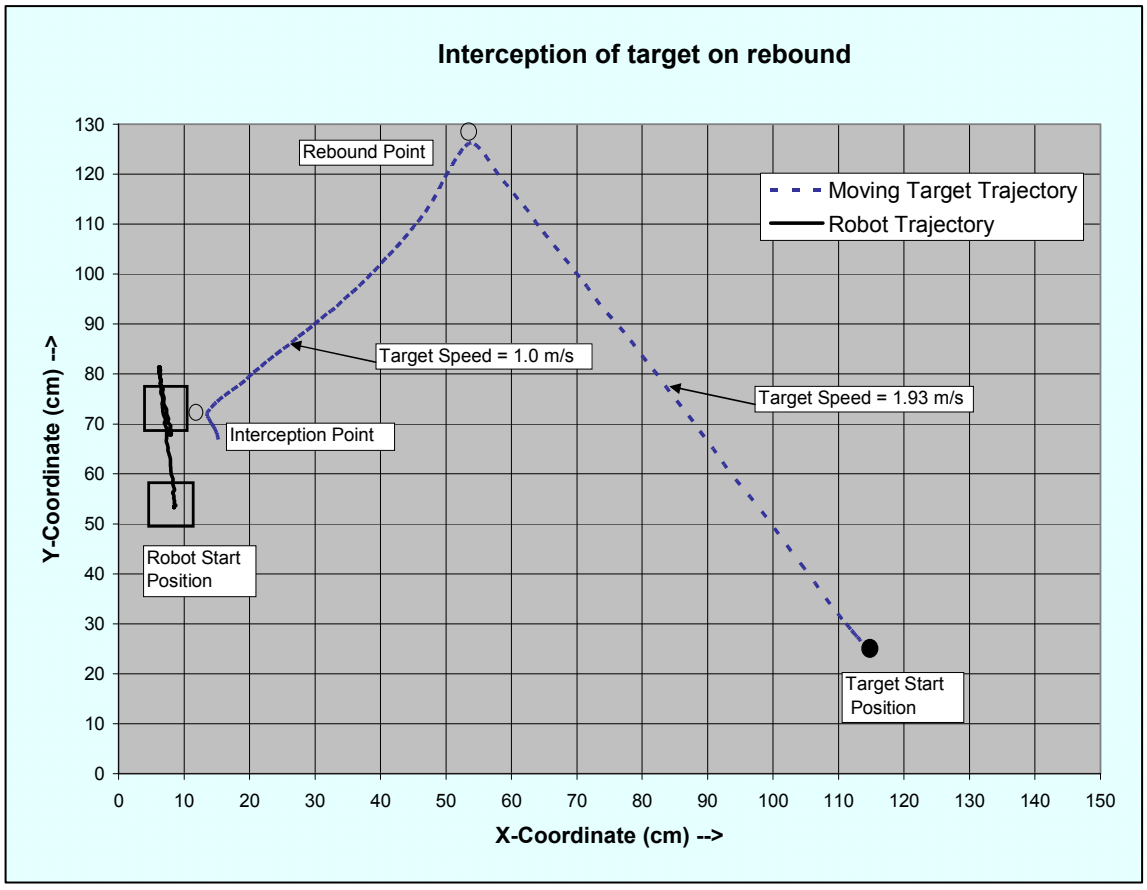


Figure 5.40 Goalkeeper intercepting a target on rebound

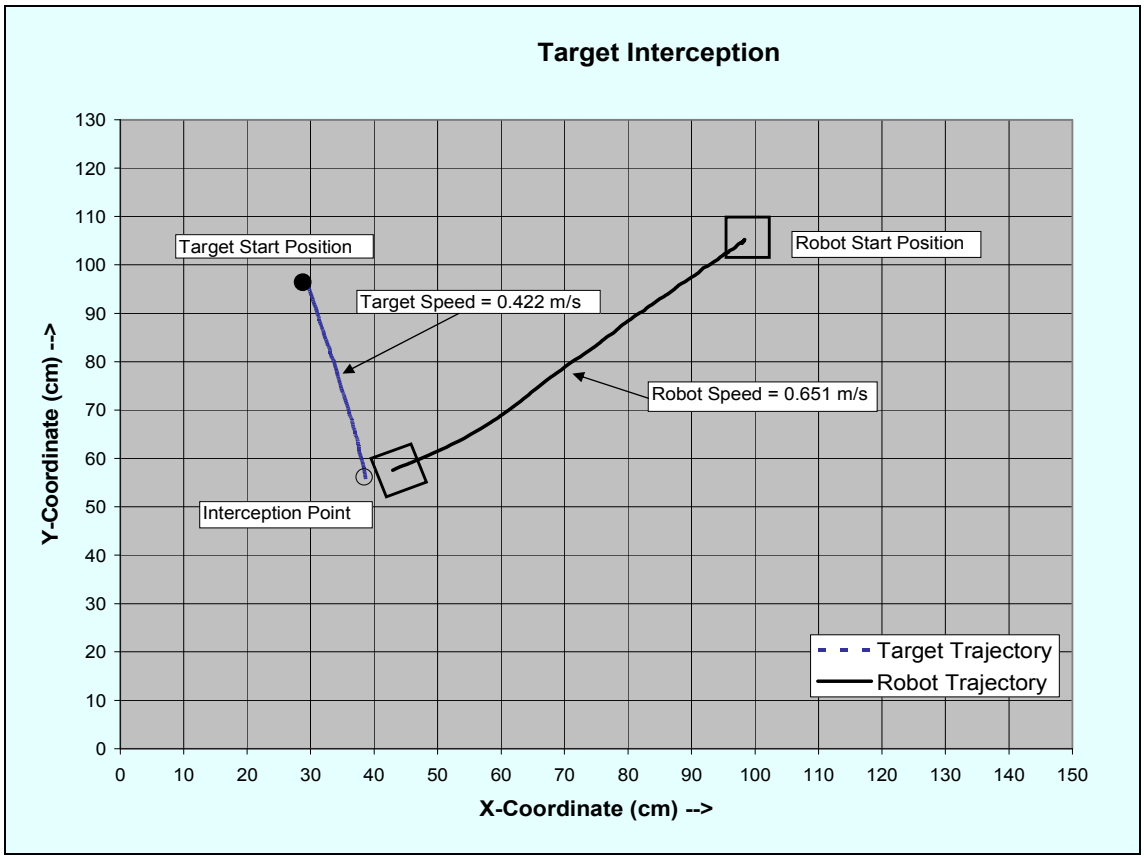


Figure 5.41 Robot intercepting a moving target (target moving away from the robot)

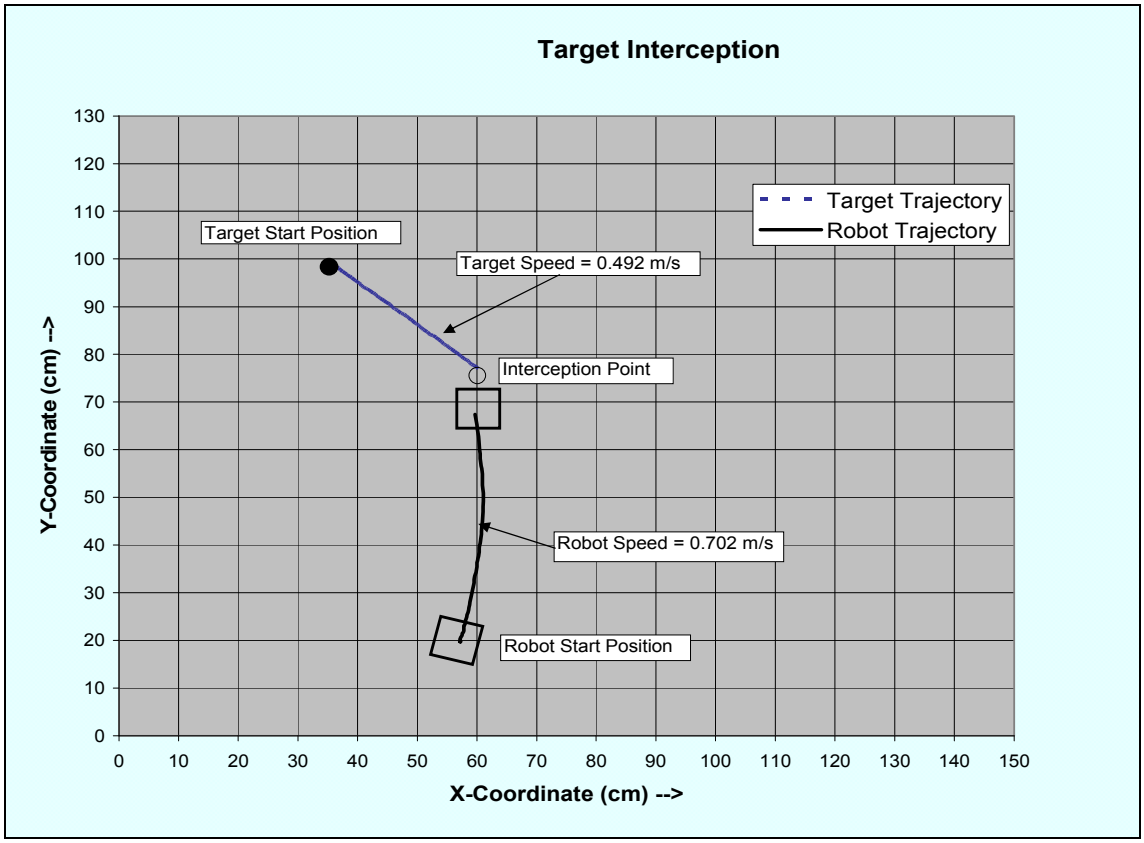


Figure 5.42 Robot intercepting a moving target (target moving towards the robot)

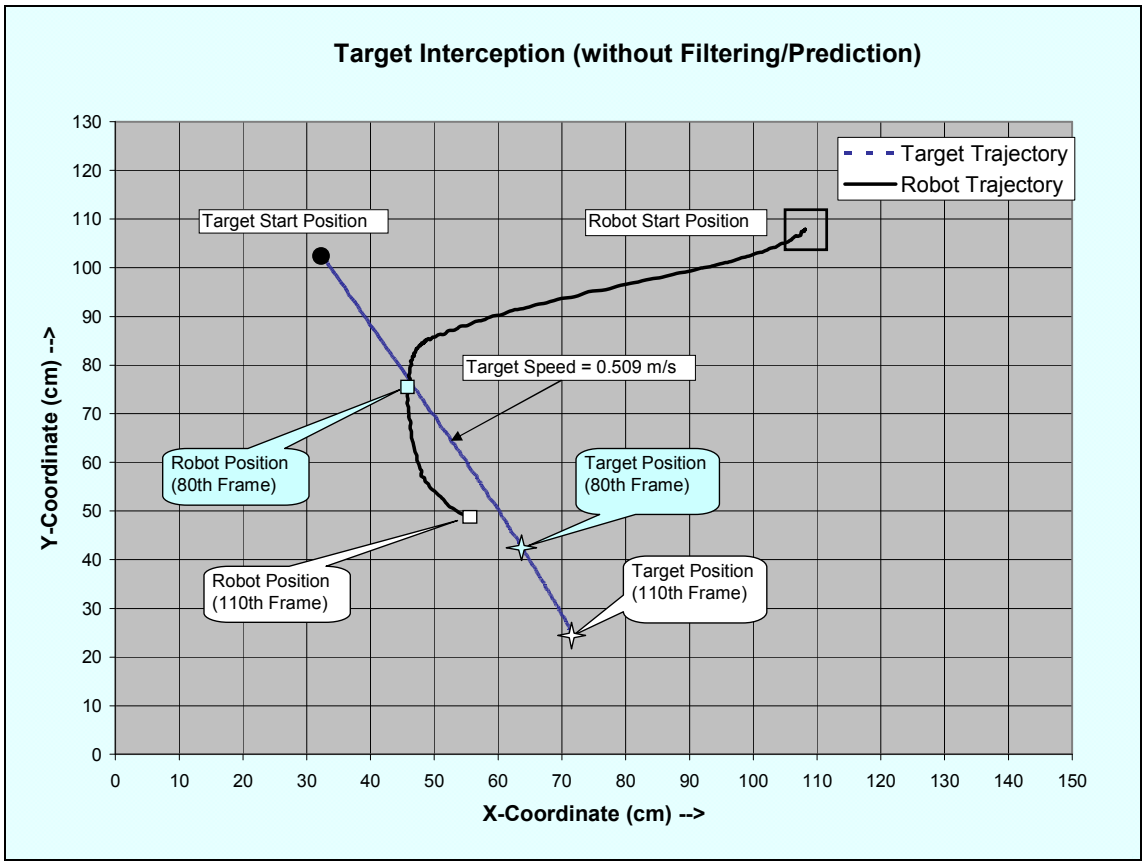


Figure 5.43 Failure to intercept a moving target without filtering/prediction

5.6 Discussion

The constant velocity model is suitable for the ball moving on a smooth hard surface (e.g. wood), as it does not accelerate or decelerate significantly unless it hits an obstacle. However, tracking the ball rolling on a soft surface (e.g. carpet where it decelerates significantly), using a constant velocity model results in a steady state tracking error. Similar problem exists when tracking the robots because the robots regularly accelerate and decelerate or change direction.

To overcome this limitation, a constant acceleration model should be used to filter the robot's position. The filter is described by:

$$\bar{\dot{x}}_n = \hat{\dot{x}}_n + \frac{h}{T}\left(y_n - \hat{x}_n\right)$$

$$\bar{x}_n = \hat{x}_n + g\left(y_n - \hat{x}_n\right)$$

$$\bar{\ddot{x}}_n = \hat{\ddot{x}}_n + \frac{2k}{T^2}\left(y_n - \hat{x}_n\right) \qquad (5.18)$$

where,

k is the filter parameter

$\bar{\bar{\ddot{x}}}_n$ is the filtered estimate of acceleration at discrete time n

$\hat{\bar{\ddot{x}}}_n$ is the predicted estimate of acceleration at time n based on sensor values at time $n-1$ and all preceding times.

Based on a constant acceleration model, the *transition equations* for prediction of position, velocity and acceleration are:

$$\hat{\bar{\ddot{x}}}_n = \bar{\bar{\ddot{x}}}_{n-1} \tag{5.19}$$

$$\hat{\bar{\dot{x}}}_n = \bar{\dot{x}}_{n-1} + T\bar{\bar{\ddot{x}}}_{n-1} \tag{5.20}$$

$$\hat{\bar{x}}_n = \bar{x}_{n-1} + T\bar{\dot{x}}_{n-1} + \frac{T^2}{2}\bar{\bar{\ddot{x}}}_{n-1} \tag{5.21}$$

The constant acceleration object is then tracked with zero lag error in the steady state. The ball position can be predicted using the constant velocity model (equation 5.12) while the robot's position is predicted using a constant acceleration model (equation 5.22):

$$\hat{\bar{x}}^r_{n+t} = \hat{\bar{x}}^r_n + t*T\bar{\dot{x}}^r_{n-1} + \left(t*T\right)^2 \bar{\bar{\ddot{x}}}^r_{n-1} / 2 \tag{5.22}$$

Equations (5.12) and (5.22) are solved simultaneously to calculate the future collision point.

The system successfully predicted the trajectory of a ball through the robot's workspace. Depending on the distance of the robot from the ball (i.e. the magnitude of the *RobotBallVector* in Figure 5.3) a ball prediction of 4 to 10 frames ahead was tested for shooting a moving ball with good success rate. Failures occurred when the target (the ball) was near an obstacle (another robot) or the edge of the soccer field. In these scenarios a higher level behaviour selection module is required to select alternative strategies. Further, the velocity calculation (equation 5.3) can be enhanced to incorporate a derivative of the angle error to dampen sudden swings in the robot's angular position. The modified equations are:

$$\begin{aligned} V_L &= V_C - Ka_p\theta_e + Ka_d\dot{\theta}_r \\ V_R &= V_C + Ka_p\theta_e - Ka_d\dot{\theta}_r \end{aligned} \tag{5.23}$$

where Ka_p and Ka_d are the proportional and derivative gains, respectively.

In each frame, the robot's present orientation is calculated and the angular velocity computed using equation:

$$\dot{\theta}_r = \theta_r\big|_n - \theta_r\big|_{n-1} \tag{5.24}$$

As discussed in section 3.1.1, the image can be processed as two separate fields – odd and even scan fields – or as a composite frame comprising all the scan lines. The performance of the filters for odd-even scans and full scans were recorded. Figure 5.44 shows the variation of a stationary ball's Y coordinate for filter parameters *g=0.8, h=0.2* when odd and even fields are processed separately. The Y coordinate swings by as much as 5.73 mm when no filtering is used. With filtering, the maximum *dy* is reduced to 3.97. As is expected, the sign of *dy* alternates from one frame to another. For the same filter parameters, i.e. *g=0.8, h=0.2,* for full scanning, the maximum non-filtered *dy* is 2.4 mm which is reduced to 1.7 mm. This is shown in Figure 5.45.

Similarly, Figures 5.46 and 5.47 are for filter parameters *g=0.6, h=0.4*. Table 5.9 tabulates the maximum *dy* and the improvements achieved for the two filters for full scan and separate odd-even scans.

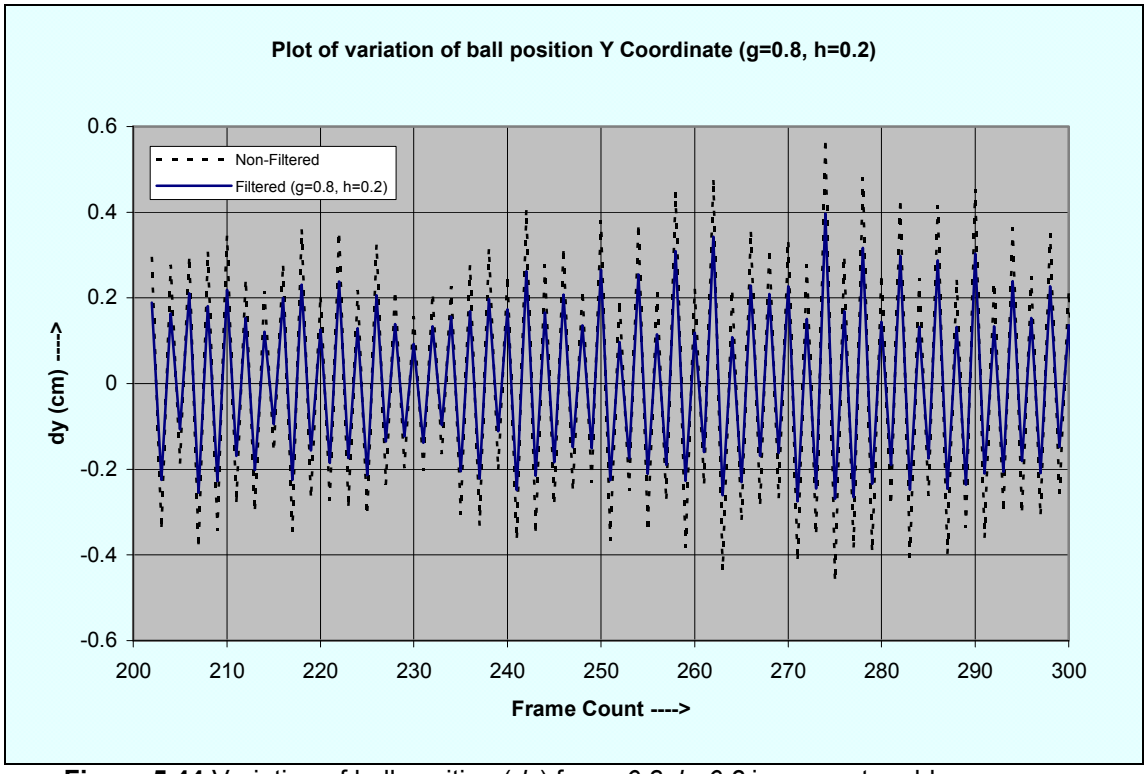


Figure 5.44 Variation of ball position (*dy*) for *g=0.8, h=0.2* in separate odd-even scans

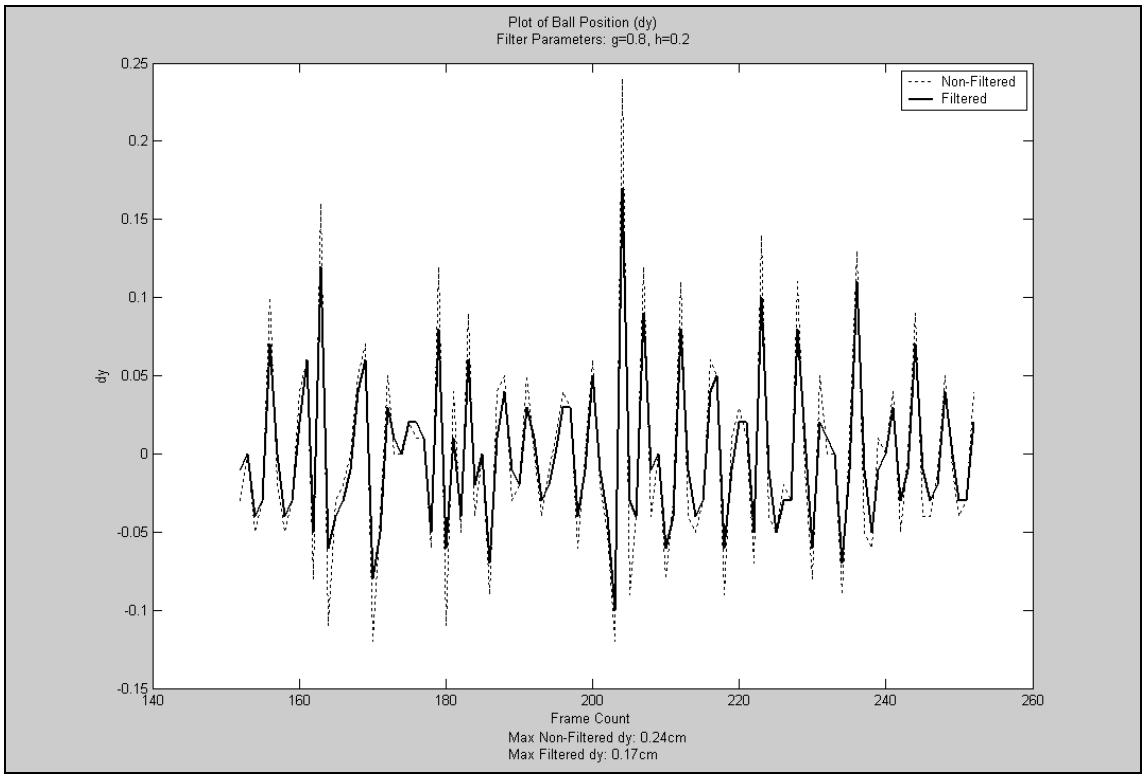


Figure 5.45 Variation of ball position (*dy*) for *g=0.8, h=0.2* in full scan mode

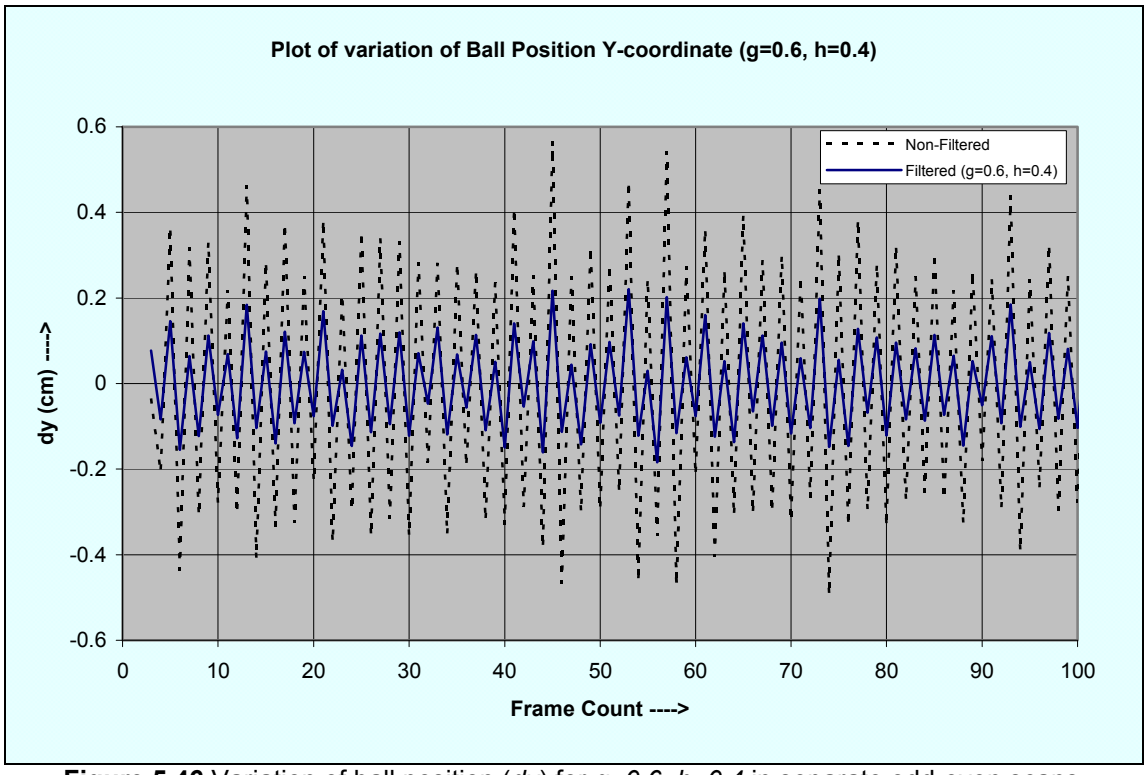


Figure 5.46 Variation of ball position (*dy*) for *g=0.6, h=0.4* in separate odd-even scans

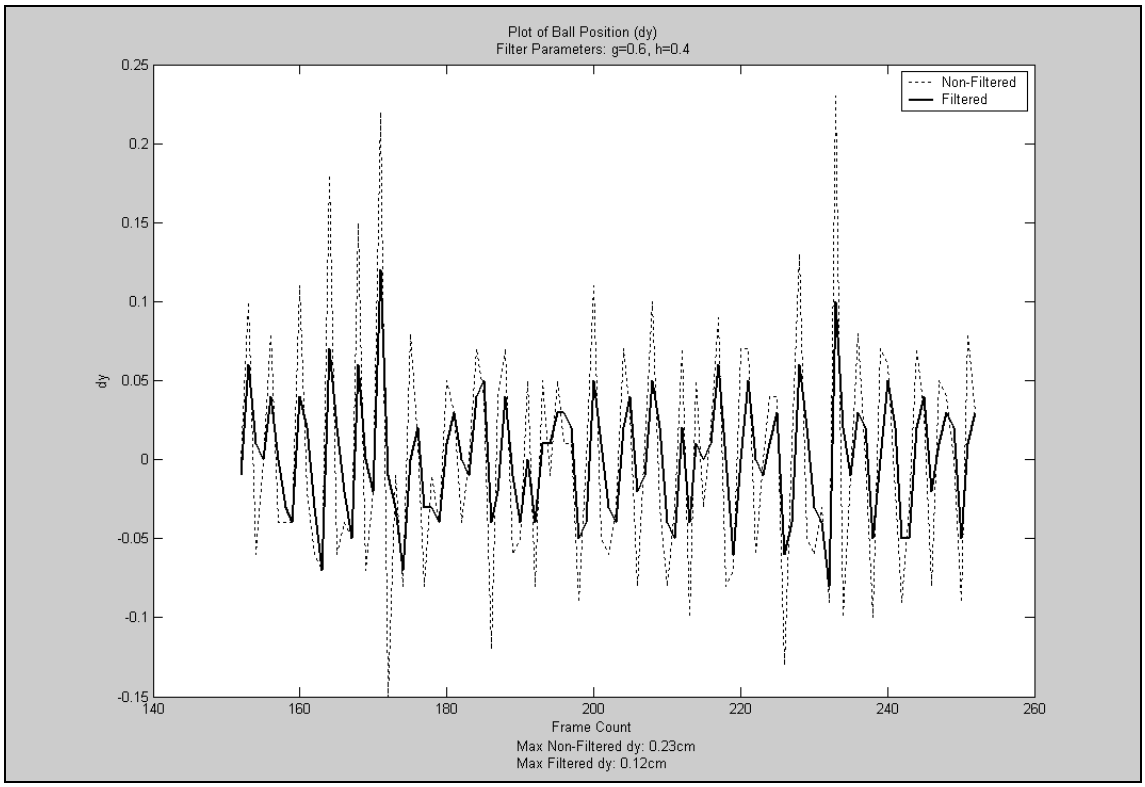


Figure 5.47 Variation of ball position (*dy*) for *g=0.6, h=0.4* in full scan mode

Table 5.9 Evaluation matrix for two filters for different scan modes

Filter Parameters	Max |*dy*| for separate Odd-Even Scans (mm)			Max |*dy*| for full scan (mm)		
	Non-Filtered	Filtered	% Improvement	Non-Filtered	Filtered	% Improvement
g=0.8, h=0.2	5.73	3.97	30.71	2.4	1.7	29.17
g=0.6, h=0.4	5.67	2.42	57.32	2.3	1.2	47.83

The *g-h* tracking filters have been implemented as classes in an object-oriented paradigm. The filter parameters are passed to the constructor at the time of instantiating a filter object. The implementation is shown in Appendix B.

6 Advanced Motion Control Algorithm for Fast Mobile Robots

In a number of autonomous robotic applications, the robots must possess two basic capabilities – to turn towards a target and to move to a given location. These basic functions may be combined to implement other functionalities and behaviours like intercepting a moving object, shooting a projectile in a given direction at a target and synchronized movements in a collaborative environment to accomplish a common task. In most applications, the accuracy of movement, whether angular or linear, is of critical importance. Improved accuracy gains further significance and poses added challenges when dealing with very fast moving robots.

Several methods of calculating wheel velocities of a fast moving robot can be found in the literature. A simple nonlinear velocity estimator for high-performance motion control has been detailed in [74]. A classical PID (Proportional-Integral-Derivative) control has commonly been employed for the control of robot orientation and position [75, 76]. The biggest challenge in PID control is the tuning of the control parameters, namely the gain factors. For faster system response, the gain can be scheduled to be a different value for a range of error values [77, 78]. In systems where the robots are guided using vision, the complexities are many fold due to noise from the sensor and use of interlaced scan cameras. In Chapter 5, use of filtering to minimize the noise and implement a predictive controller has been discussed.

In this chapter a control algorithm to calculate the robot wheel velocities that is independent of gain parameters is detailed. The system works by defining an intermediate 'virtual' target position for the robot to go to in its quest to reach the destination. The intermediate position is such that it assists the robot to reach the target with the desired final angle and is continually updated as the robot moves through its work space. The selection of the virtual target position is influenced by several parameters such as the path curvature, robot orientation, distance of the robot to the final position and the robot-to-target angle.

An important and integral component of the position control methodology is the velocity profiling which has been incorporated to ensure rapid and accurate response to robot positioning. The velocity profiling has three components- angular, linear and cross-effect profiling. These are explained in Section 6.6.

The results of implementing this algorithm on actual robots in a robot soccer platform are presented in Section 6.7. The proposed algorithm was tested on fast moving robots exceeding speeds of 1.5m/s.

6.1 Classical PID Control for Robot Positioning and Orientation

Feedback from the global-vision system is used to form a closed-loop control system and the data generated by the vision sub-system is ideally suited for implementation of a classic proportional-integral-derivative (PID) control for robot positioning and orientation. The feedback mechanism in the closed-loop system enhances the ability to correctly orient (or position) the robot. In this technique, the output of the system is measured and compared to the desired input and corrective actions are taken, based on the error *e*, to achieve the desired result.

The control function is calculated using the following equation:

$$v(t) = v_{ss} + K_p e + K_i \int e(t)dt + K_d \left[\frac{de(t)}{dt} \right] \tag{6.1}$$

where K_p, Ki *and* K_d are the proportional, integral and derivative gains respectively, and v_{ss} is the steady state value. Larger values of the proportional gain will increase the system response time but may introduce large oscillations and instability in the system. The derivative component will help to dampen the oscillations where as the integral component will reduce the steady state error. Optimum system performance requires that the coefficients, K_p, K_i, and K_d be tuned for a given combination of motion mechanics and payload inertias.

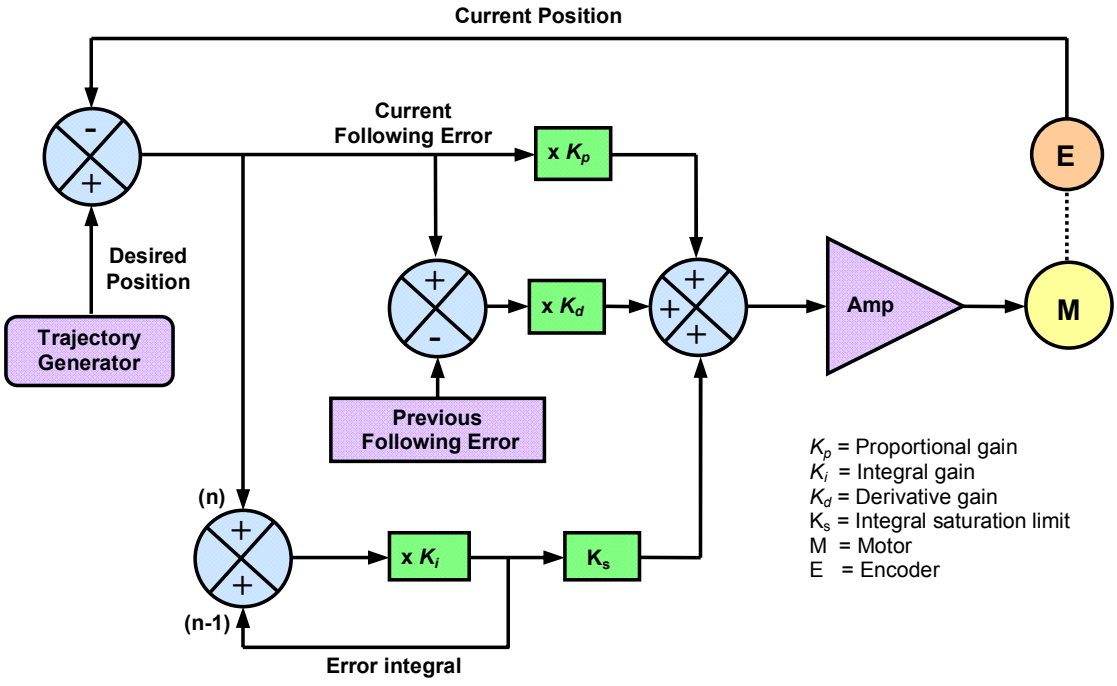


Figure 6.1 Block diagram of a PID controller for motor position control

Figure 6.1 shows the block diagram of a PID controller for a motor position control. In practical applications where the system dynamics are such that the reference for the controller changes rapidly and frequently, the integral effort of the controller is often omitted. This is because the system does not have enough time to reach a steady state before the reference changes. The benefits of the integral component to minimise the steady state error is thus not realised. Moreover, in a very dynamic system, there is a risk of the controller quickly reaching the integral saturation limit (**Ks** in Figure 6.1) as the errors are accumulated. This necessitates extra care to be taken in designing the controller. Hence, in practice, a PD controller is often used for highly dynamic systems and the control function is calculated using the following equation:

$$v(t) = v_{ss} + K_p e(t) + K_d \left[\frac{de(t)}{dt} \right] \tag{6.2}$$

6.1.1 Robot Orientation Control

This section first develops the control algorithm for the wheel velocities of a bi-wheeled autonomous robot using a PD controller. Figure 6.2 shows the robot's current and final direction.

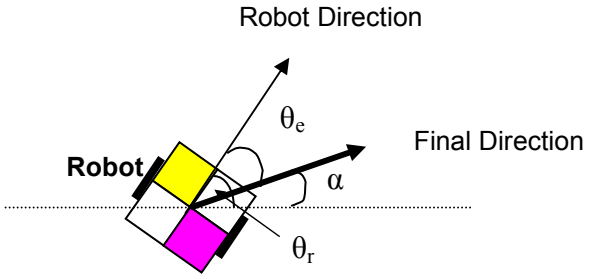


Figure 6.2 Orientation control - robot's current and final direction

In Figure 6.2, the angles shown are as follows:

θ_r – *RobotAngle*

α - *DesiredAngle*

θ_e – *AngleError* (negative value in the case shown in the figure)

$$\theta_e = \alpha - \theta_r \tag{6.3}$$

The robot needs to turn, from its present orientation, to the final direction and the velocities of the left and right wheels, V_L and V_R, are calculated using the following equations:

$$V_L = -Ka_p\theta_e + Ka_d\theta_r$$
$$V_R = -V_L \quad (6.4)$$

Ka_p and Ka_d are the proportional and derivative gains, respectively. In each frame, the robot's orientation is calculated and the angular velocity, $\dot{\theta}_r$, computed using the following equation:

$$\dot{\theta}_r = \theta_r(t) - \theta_r(t-1) \quad (6.5)$$

When the robot starts to turn towards the desired direction, the angle error decreases and hence the velocity reduces. When the robot is finally in line with the desired direction, the angle error is zero, thereby bringing the proportional control factor, $\boldsymbol{Ka_p\theta_e}$, in equation (6.4) to zero. The derivative control has the effect of damping the velocity.

However, in the absence of an accurate kinematic model of the robot, it is very difficult to theoretically calculate the gains for optimal motion control. Improperly tuned systems are often either over damped, which results in sluggish performance, or under damped, which results in severe oscillations around the steady state. A very large set of experiments are required to tune the system well. The process of choosing the right value for Ka_p and Ka_d is tedious.

6.1.2 Gain Scheduled Proportional Control

In a classical PD controller, the coefficients, K_p (and K_d) are constant over the entire range of the error value. This has the effect of having the correction factor being directly proportional to the error - larger the error, greater the control effort. In the case of a robot approaching a target, it will gradually slow down as it nears the target and eventually stop when it reaches the target. In the case of a robot turning towards a certain direction, the angular velocity will gradually and *uniformly* decrease till it becomes zero (i.e. the robot will stop turning) when it reaches the desired direction. The drawback of this approach is that it does not give finer control over the system's response for different ranges of error values. In some applications it may be necessary to have different proportional gains for different ranges of error values. For example, it may be required to make a robot, such as a goalkeeper in a robot soccer game, move at a fast pace, irrespective of the error value (i.e. the distance from the target position), yet

be able to stop quickly at the target position without overshoot. So for most of its travel distance, a high value of proportional gain can be used which can be reduced to a much smaller value when the distance error is below a certain threshold. A similar situation exists when the robot is turning.

Hence, it is proposed that instead of having a constant proportional gain, a multi-tiered gain schedule be created based on the absolute value of the angle error. This will enable finer control over the response of the system and based on the application it may be tuned for optimal performance in a desired range of operation. Moreover, different robots in a collaborative environment may be tuned to perform differently over the same range of error values.

In the real world, there exist many robots in which there is no differentiation between the robot's front and rear ends as they are symmetric and can move in both the directions, forward and reverse, without having to turn. In such cases, the maximum angle by which the robot will ever need to turn is 90^o. Hence the maximum angle error is $\pm 90^o$.

For a multi-valued proportional gain, the control equations are as shown below:

$$V_L = -Ka_p(\theta_e - BaseAngle) - offset + Ka_d\dot{\theta}_r$$
$$V_R = -V_L \tag{6.6}$$

The Ka_p, *BaseAngle* and *Offset* depend on the tier in which the *AngleError* (θ_e) falls. Equation (6.6) is explained using Figure 6.3 which shows the multi-valued Ka_p.

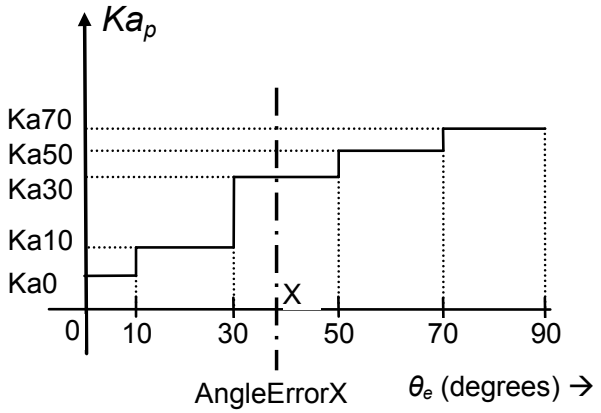


Figure 6.3 Multi-tiered proportional gains, Ka_p

If the angle error, *X,* is between 30^o and 50^o, (Figure 6.3) the following values are used and substituted in equation (6.6):

$BaseAngle = 30$

$Offset = (30 - 10)*\text{Ka10} + 10*\text{Ka0}$

$Ka_p = \text{Ka30}$

The wheel velocities are thus:

$$V_L = -Ka30(X - BaseAngle) - [(30-10)Ka10 + 10 * Ka0] + Ka_d\dot{\theta}_r$$

$$V_R = -V_L$$

This gain scheduled Ka_p, together with the *Offset* calculation, results in a smooth change of V_L and V_R without discontinuities at the boundaries of the steps.

6.1.3 Effect of Scaling Ka_p on the System Response

As mentioned earlier, different robots in a collaborative environment may be tuned to perform differently over the same range of error values. Not only that, the multi-tiered gain coefficients may be scaled up or down to change the overall response of the robot.

A scaling factor was introduced which affects uniformly all the Ka_p values and equation (6.6) was modified as shown below:

$$V_L = -(Ka_p(\theta_e - BaseAngle) + offset)scaleFactor + Ka_d\dot{\theta}_r$$

$$V_R = -V_L \qquad (6.7)$$

The effect of different scaling factors is seen in Figure 6.4. The robot, oriented towards 0^o is given a stimulus to turn to 90^o. The best response is achieved when the scaling factor is 0.85; the system is close to being critically damped. For the scaling factor of 0.8, the robot does not reach the steady state value of 90^o; the system is under-damped. For values of scaling factor greater than 0.85, the system exhibits substantial overshoots; it is under-damped. The steady state conditions are reached within 25 frames, i.e. <0.42 Sec, which is very fast.

Table 6.1 Multi-tiered proportional gains used for experimentation

Ka70	0.2857
Ka50	0.3429
Ka30	0.4000
Ka10	0.4570
Ka5	0.2857
Ka2	0.1500
Ka0	0.1000

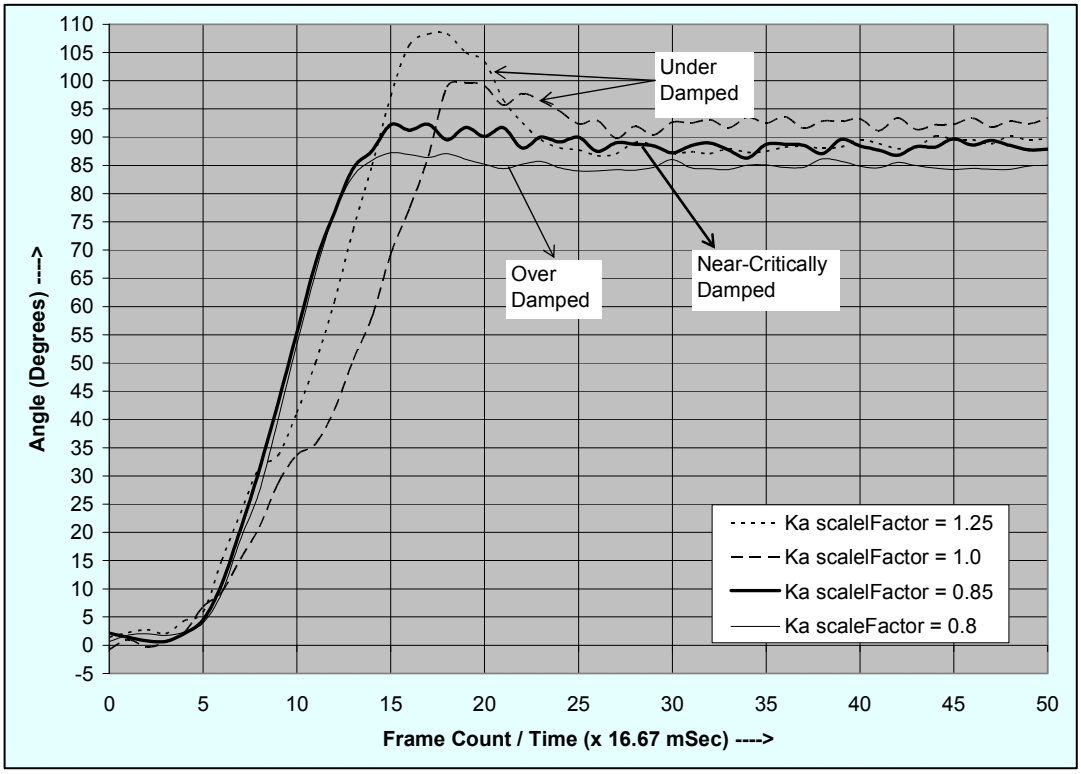


Figure 6.4 Effect of scaling proportional gains, Ka_p

6.1.4 Fine Tuning the Proportional Gain, Ka_p

It is possible to further improve the system performance by tuning the shape of the gain scheduled values of Ka_p. By changing the values, even by a small amount, significant performance improvements can be made, as shown in Figure 6.5.

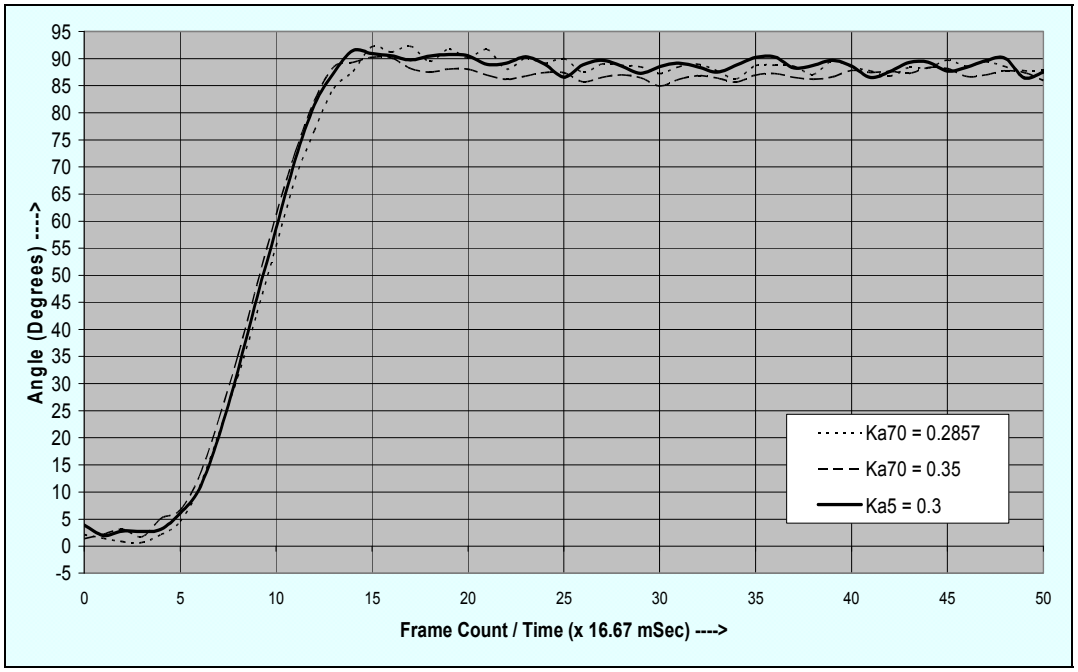


Figure 6.5 Fine-tuning proportional gain, Ka_p

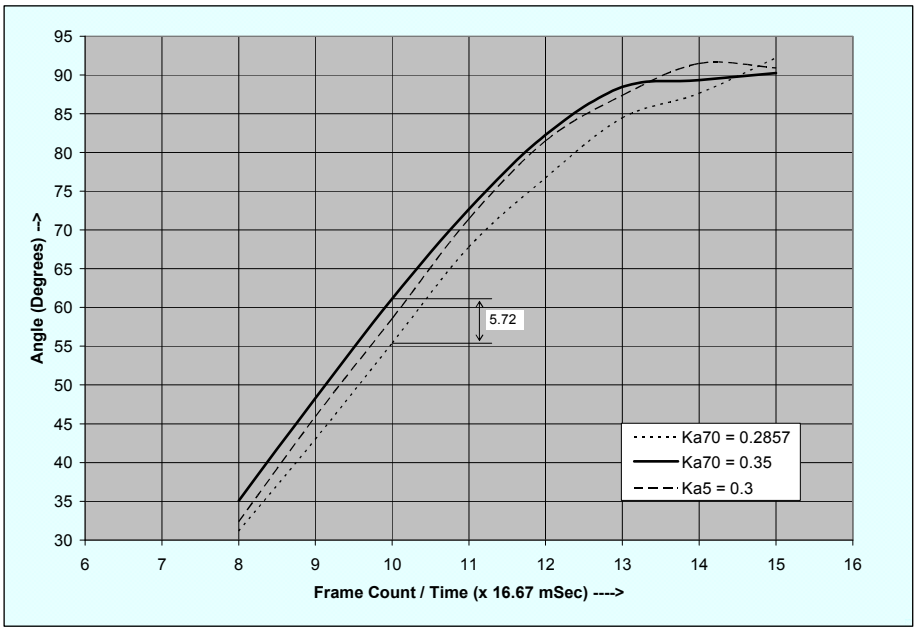


Figure 6.6 Tweaking proportional gains for faster system response

By changing the proportional gain Ka70 from 0.2857 to 0.35 and Ka5 from 0.2857 to 0.3, the robot responds significantly faster. This is shown in Figure 6.6 which plots the response from frame number 6 to 16. At the 10th frame count (166.7 ms), the robot turns by an extra 5.72 degrees which is an improvement of 6.35%, given that the final angle of the robot is 90^o starting from 0^o.

6.1.5 The Effect of Derivative Gain, Ka_d

Just as the proportional gain has been multi-tiered, the derivative gain, Ka_d, may be gain scheduled too. Figure 6.7 shows the effect of Ka_d on the system response using a given proportional gain.

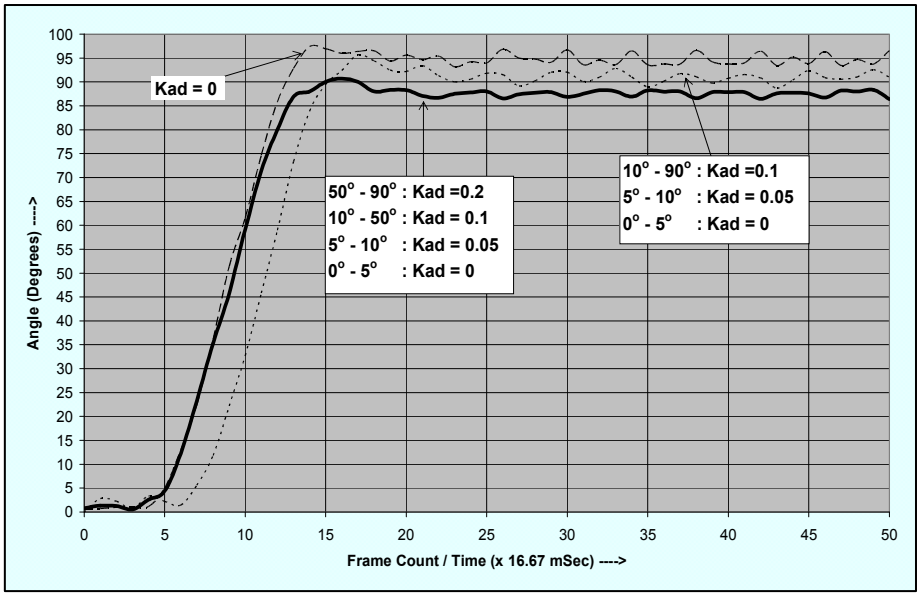


Figure 6.7 Gain Scheduled Derivative gains, Ka_d

6.1.6 Combining g-h Filter and PD Control

The affect of combining the g-h filter together with the PD control was studied. Figure 6.8 shows the effect of filtering on the system response with the PD control.

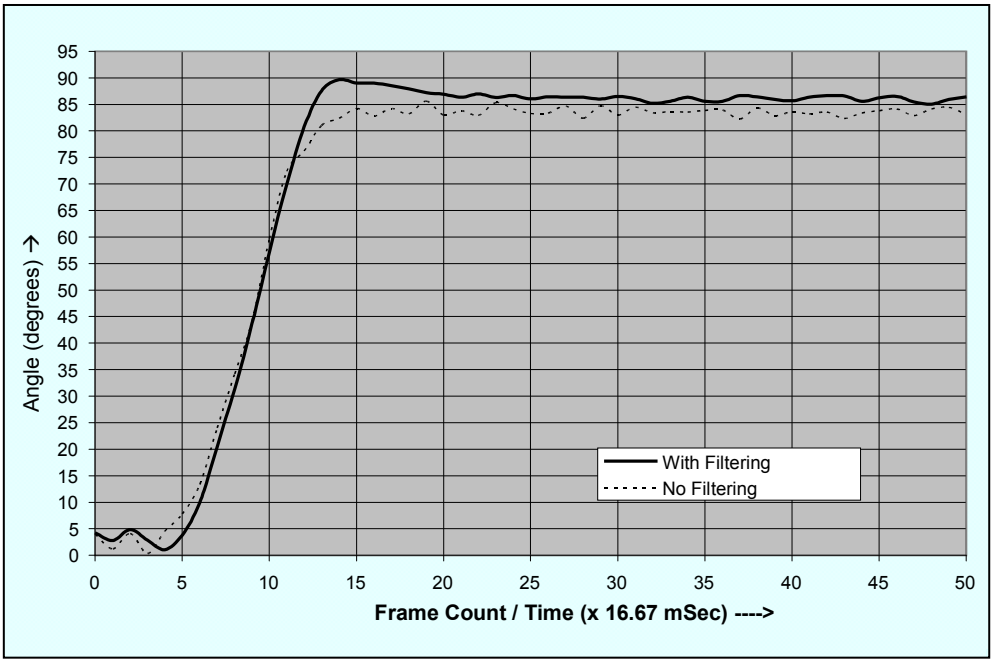


Figure 6.8 Filtering with derivative control

The parameters used to filter the robot's angle were g=0.8 and h=0.2. Without filtering, the system is slightly over-damped. The oscillations in the angle are due to the controller's response to vision sensor noise.

Figure 6.9 shows the system response for several different filter parameters, but without derivative gain (only proportional control).

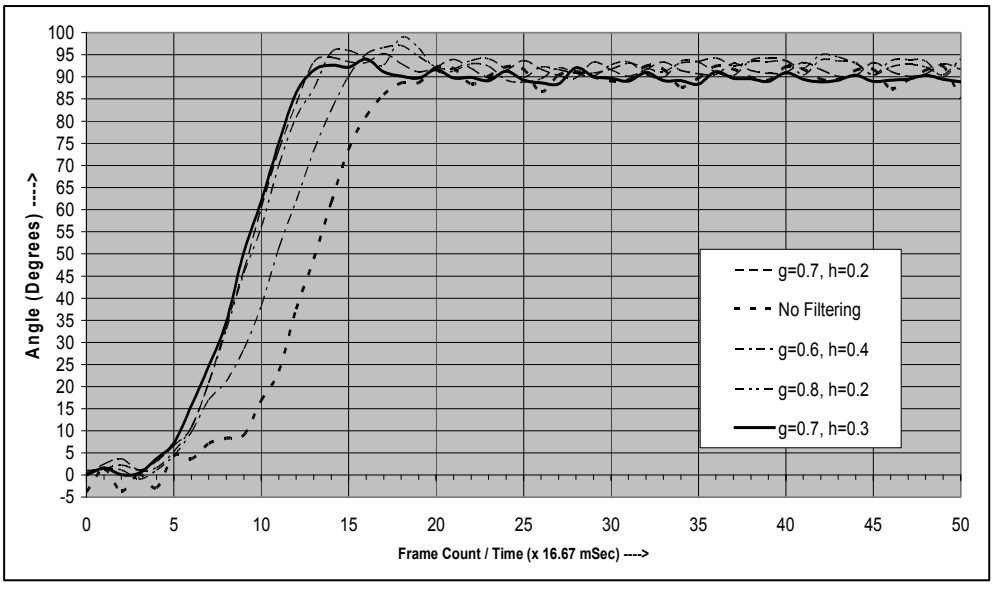


Figure 6.9 Filtering without derivative control

The best system performance is achieved for g=0.7, h=0.3. It can be seen that the filtered vision data without derivative control provides the best system response. The derivative controller, when used together with a g-h filter, can be improved by increasing the proportional gain for small angular error.

6.2 Classical PD Control for Robot Position

This section details the PD controller for robot positioning. It follows a similar approach to robot orientation discussed in section 6.1. The wheel velocity has two components, one for linear motion and another for angular motion.

6.2.1 Robot Position Control

Figure 6.10 shows the initial and final position of the robot. The robot has to traverse the linear distance and when it reaches the destination, it has to orient towards the final direction.

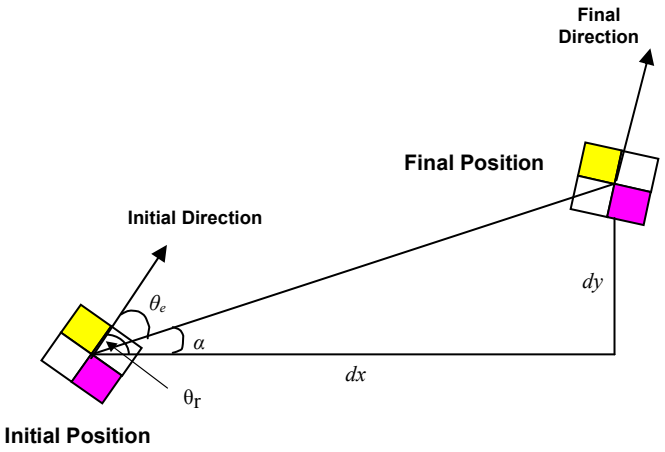


Figure 6.10 Movement control – robot's current and final position/orientation

In Figure 6.10, the angles shown are:

θ_r – *RobotAngle*

α - *DesiredAngle*

θ_e – *AngleError* (Negative value for the case shown in Figure 6.10)

$$\theta_e = \alpha - \theta_r \tag{6.8}$$

Distance error, d_e, and speed error, S_e, are calculated using the following equations:

$$dx = finalPos.x - initialPos.x$$

$$dy = finalPos.y - initialPos.y$$

$$d_e = \sqrt{dx^2 + dy^2}$$

$$S_e = finalVel - currentVel \tag{6.9}$$

For the robot to move from its initial position to the final position, the velocities of the left and right wheels, V_L and V_R, are calculated using the following equations:

$$V_L = (Kd_p d_e + Kd_d S_e)d - Ka_p \theta_e$$
$$V_R = (Kd_p d_e + Kd_d S_e)d + Ka_p \theta_e \quad (6.10)$$

Kd_p and Kd_d are the proportional and derivative gains, respectively, for position control. Ka_p is the proportional gain for orientation (angle) control. The parameter d is the 'direction factor', which can be used to switch off or on the velocity component for linear motion.

$d = 0$ No linear motion (robot is however free to turn)

$d = 1$ Robot moves in the forward direction (coupled with angular motion if required)

$d = -1$ Robot moves in the reverse direction (coupled with angular motion if required)

Figure 6.11 shows the value of the parameter d which depends on the angle error θ_e.

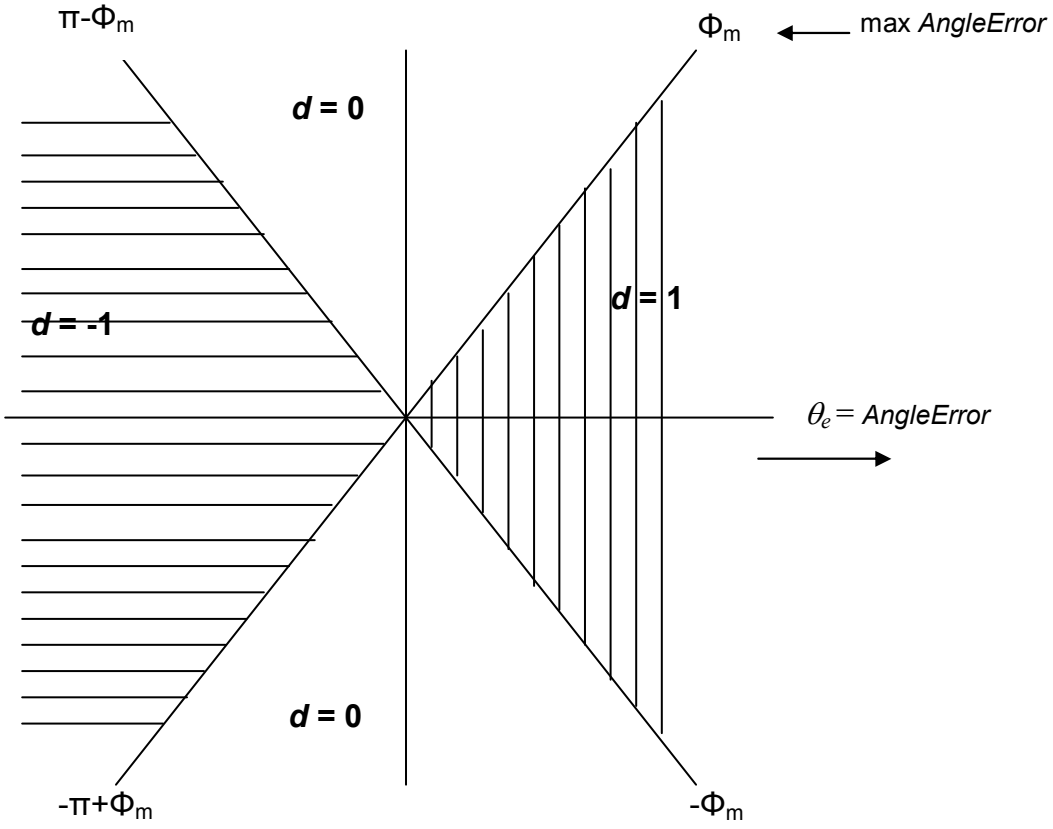


Figure 6.11 Determining 'direction factor'***d*** for robot movement

In Figure 6.11, Φm is the maximum *AngleError*. The value of d is calculated using the equations shown below:

$$d = 1 \text{ for } (-\Phi_m < \theta_e < \Phi_m)$$
$$d = -1 \text{ for } ((\Pi >= \theta_e > \Pi - \Phi_m) \text{ or } (-\Pi < \theta_e <= -\Pi + \Phi_m))$$
$$d = 0, \text{ otherwise} \qquad (6.11)$$

The value of Φm can be set to a different value for individual robots in a collaborative task. It essentially decides at what point the linear component of the velocity kicks in when the robot moves from its present position to a destination. As the robot starts turning towards the desired direction, the angle error is decreasing and hence the velocity component, $Ka_p\ \theta_e$, is reduced (equation 6.10). When the robot is in line with the desired direction, α, the angle error, θ_e, is zero, thereby bringing the velocity component, $Ka_p\ \theta_e$, to zero.

Choosing the optimal values for Kd_p, Kd_d and Ka_p is a tedious process.

6.2.2 Multi-tiered Kd_p for Robot Positioning

Much like Ka_p, the proportional gain for position control, Kd_p, has also been scheduled. For the case under study, the maximum distance that the robot has to travel is 2m. The gain is scheduled based on the distance error, d_e.

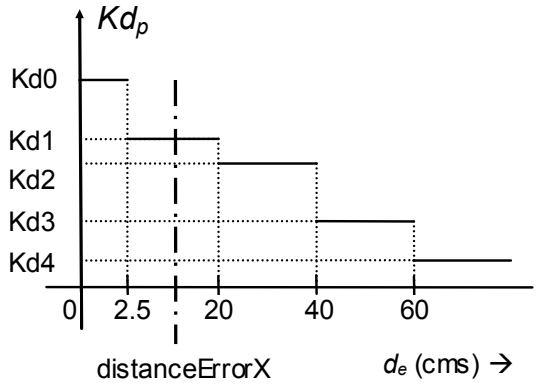


Figure 6.12 Multi-tiered proportional gains, Kd_p

If the distance error is between 2.5cm and 20cm, say *distanceErrorX* (Figure 6.12), the wheel velocities are calculated as follows-

$$Offset = Kd0 * 2.5$$
$$Kd_p = Kd1$$
$$V_L = (Kd_p(distanceErrorX - 2.5) + Offset + Kd_d S_e)d - Ka_p\theta_e$$
$$V_R = (Kd_p(distanceErrorX - 2.5) + Offset + Kd_d S_e)d + Ka_p\theta_e \qquad (6.12)$$

This multi-tiered Kd_p, together with the *Offset* calculation, results in a smooth change of V_L and V_R without discontinuities at the boundaries of the steps.

Apart from the proposed multi-tiered Kd_p, a scaling factor was introduced which uniformly affects all the Kd_p values in the same way as it does to the Ka_p values (equation 6.7). In the experimental setup, the robot was commanded to move from point (15.0, 65.0) to point (130.0, 65.0) which entails no angular motion, only linear motion, resulting in a change in the X coordinate only. The effect of different scaling factors is seen in Figure 6.13.

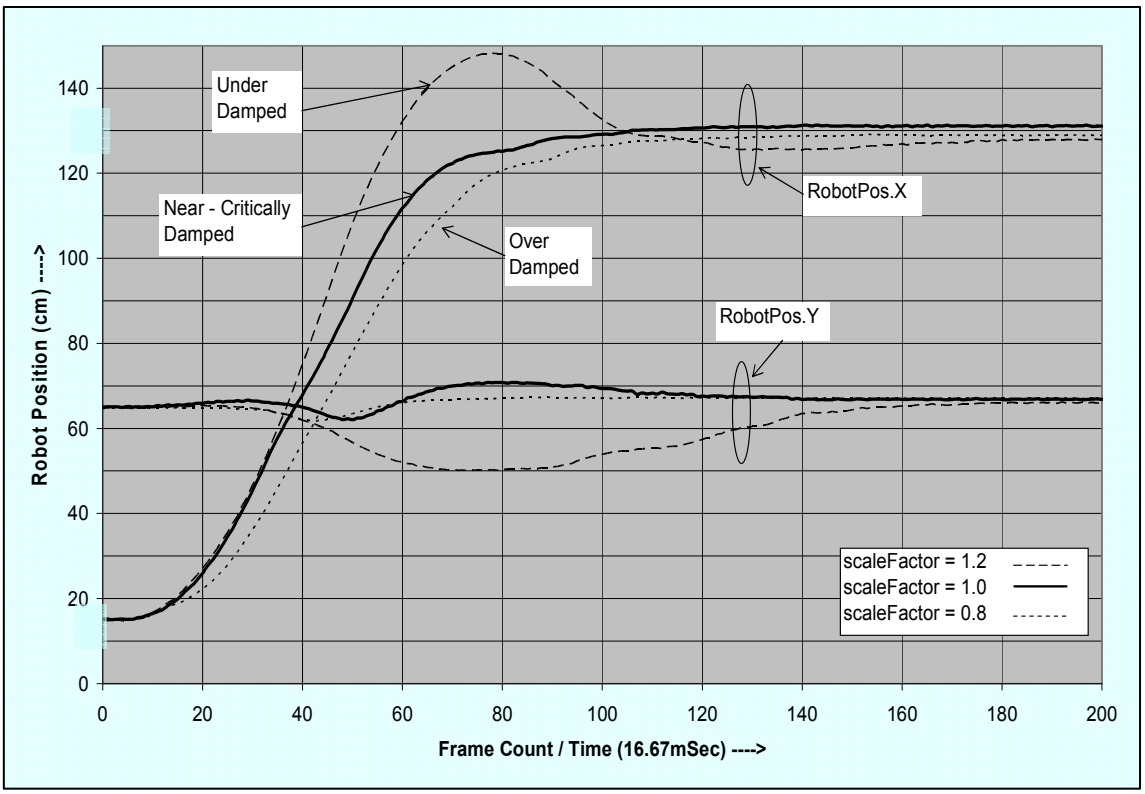


Figure 6.13 Effect of increasing proportional gain, Kd_p

For high gain (*scaleFactor* = 1.2), the system exhibits under-damped oscillatory characteristics while for low gain (*scaleFactor* = 0.8), the system exhibits over-damped sluggish characteristics. Based on the application and the maximum distance that a robot has to travel, the *scaleFactor* may thus be adjusted accordingly. In a collaborative environment, different robots can have different *scaleFactor* depending on the response required from the robot.

6.3 Advanced Robot Positioning Algorithm

In the proposed algorithm, the position function can be decomposed into three sections- pre-processing inputs, 'retargeting' system and velocity profiling, as shown in Figure 6.14.

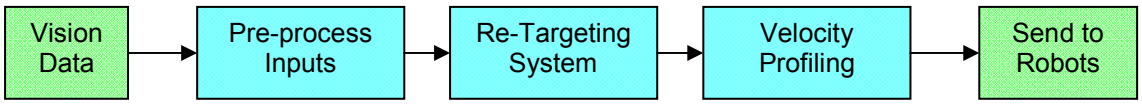


Figure 6.14 Robot Positioning Process

6.3.1 Pre-Processing Inputs

The vision layer generates data pertaining to the position, orientation, angular and linear velocities of the robots. At the pre-processing section, the robot-to-target distance, the robot-to-target angle and the alignment error are calculated. Also, all the angles are normalized. Figure 6.15 shows the various angles and linear distances which are pre-processed.

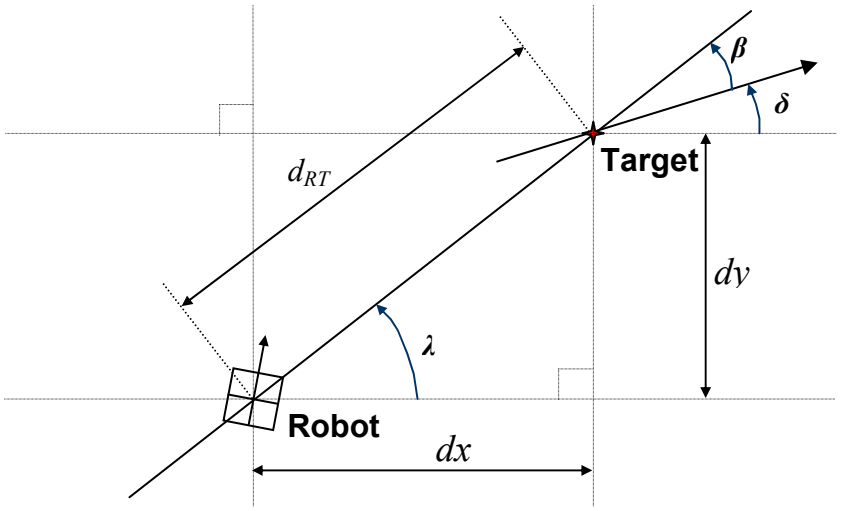


Figure 6.15 Pre-processing inputs

λ is the robot-to-target angle

δ is the final robot angle at the target position

β is the alignment error

d_{RT} is the distance error (distance between robot and target positions)

The robot-to-target angle, alignment error and the distance error are calculated using the equations shown below:

$$\lambda = \tan^{-1}\left(dy \big/ dx\right)$$

$$\beta = \lambda - \delta$$

$$d_{\mathrm{RT}} = \sqrt{dy^2 + dx^2} \tag{6.13}$$

6.3.2 Re-targeting System

An important issue with targeting systems is that the robot must approach the target in-line with the final angle. If the target is a ball, as is the case in robot soccer, it would be hit in an undesired direction if not approached in line with the final angle, which would usually be towards the opponent's goal or another collaborating robot of the team to which the ball has to be passed.

Figure 6.16 shows a simple positioning methodology in which the robot first aligns with the target angle, moves to the target, and then aligns with the final angle (*turn-move-turn* approach). This approach may suffice in some applications, such as moving loads on a production floor, in which the robot stops on reaching the target position. In fact in some applications such as search and rescue, foraging and scavenging etc. such an approach is highly desirable. However, in many general forms of conveyance the target is continually updated. For example, in a robot soccer game, the ball, which is the target, is always moving and its position relative to the opponent's goal is continually changing. In such dynamic scenarios, the *turn-move-turn* approach will be very inefficient; the robot may end up just turning most of the time, trying to align with the target, before it can enter the *move* phase.

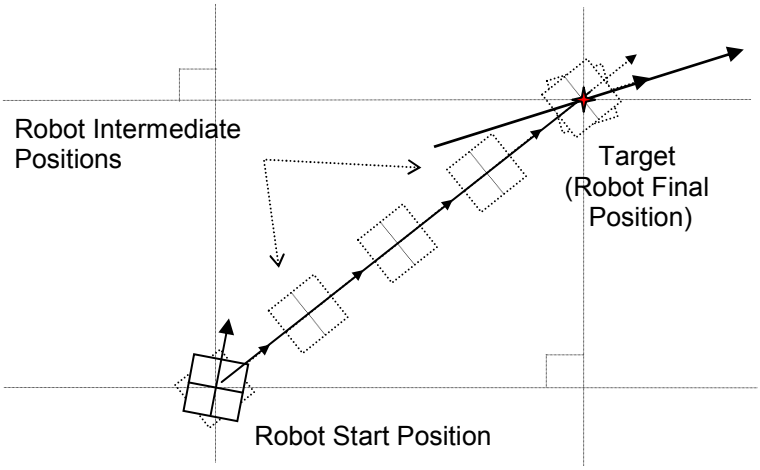


Figure 6.16 *Turn-move-turn* approach to reach a target

To approach the target in line with the final angle, the proposed re-targeting system creates a new virtual target to control robot positioning when approaching the real target. Various algorithms can be used to calculate the most appropriate virtual target so as to give the robot more stable and accurate control [79]. The presented re-targeting method has two distinct parts- one is based on a 'triangular targeting algorithm' (TTA) and the other is a 'proximity positioning algorithm' (PPA). The new target position calculated by the re-targeting system is used in the profiling stage to derive the wheel velocities.

The most appropriate re-targeting algorithm is selected based on the distance between the target and the robot. A circular area, with a very small diameter, around the target defines the transition boundaries to switch between TTA and PPA as shown in Figure 6.17. As long as the robot is outside the transition boundary, TTA remains in force to position the robot. When the robot is very close to the target it is often struggling to maintain stability – it not only has to reach the target precisely (i.e. zero distance error) with a linear motion, it also has to align with the final direction (i.e. zero alignment error) with an angular motion. In the presented system, as soon as the robot enters the circle, PPA takes over. The algorithms are explained in detail in later sections (6.4 and 6.5). The use of two different algorithms to control the robot's targeting system increases the accuracy of positioning and alignment, and increases the stability of the robot when it reaches the target. Hysteresis at the transition boundary is defined to prevent oscillations and smooth switching between the TTA and PPA methods.

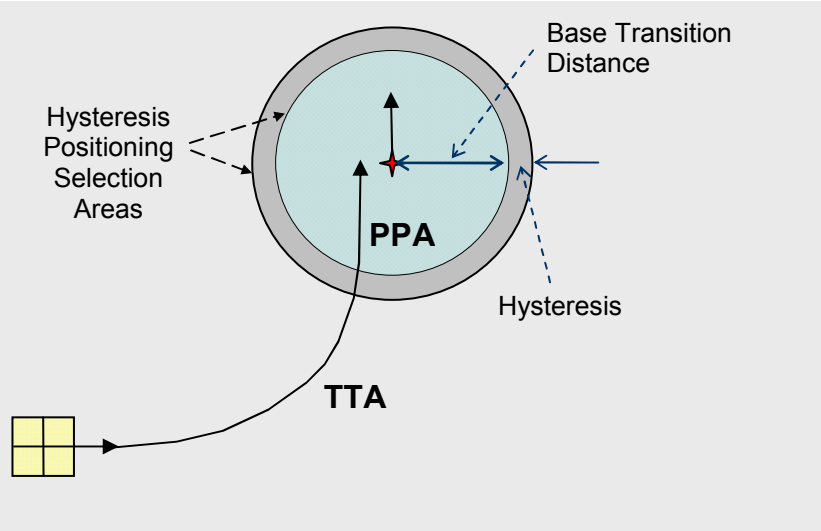


Figure 6.17 Selecting the most appropriate re-targeting algorithm

6.4 Triangular Targeting Algorithm

The Triangular Targeting Algorithm (TTA) allows the robot to traverse around objects and come in behind a set target position which is aligned with the final angle. This is especially good for pushing objects in a desired direction in a collaborative task. This also gives the robot angular stability because the angular velocity of the robot is reduced to near zero as it approaches the target.

The robot's path, as it moves towards a target, is shown in Figure 6.18. The robot is commanded to go to a '*virtual target*'. The robot position, target position and the position of the virtual target form the three vertices of a triangle, hence the name Triangular Targeting Algorithm (TTA). In the presented algorithm, the height of the triangle is calculated based on the alignment error β (Figure 6.15 and 6.19). As the robot starts to move towards the virtual target position (indicated by the star at the top of the triangle in Figure 6.18), the alignment error reduces. This in turn reduces the height of the triangle. The end result is that the robot approaches the target from behind facing the desired final angle.

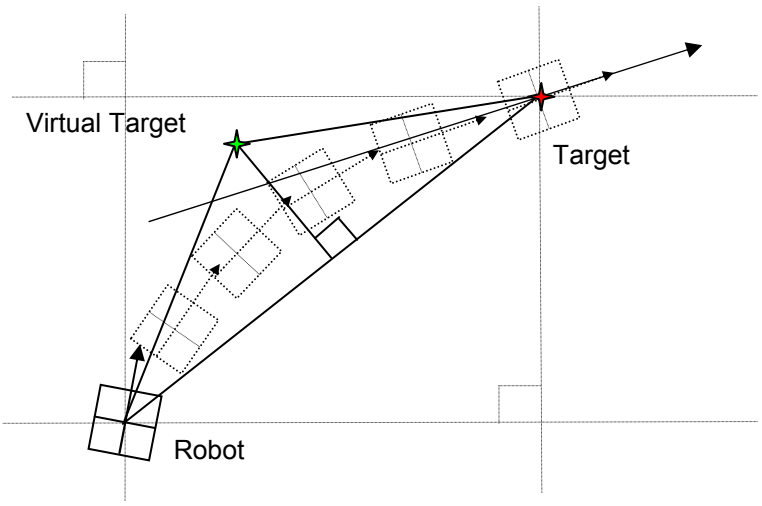


Figure 6.18 Triangular Targeting Algorithm: Robot's path towards the target

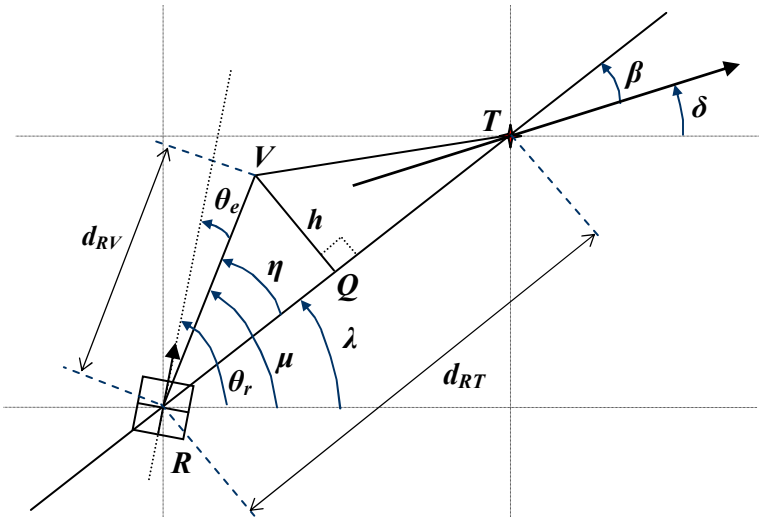


Figure 6.19 Various angles and distances for TTA

In Figure 6.19 *R, V* and *T* denote the position of the robot, virtual target and the real target respectively. The various angles and distances used in the TTA are as follows:

d_{RV} is the distance of the robot to the virtual target

d_{RT} is the distance between the robot and the target

h is the triangle height (VQ)

μ is the robot-to-virtual target angle

η is the inner triangle angle

θ_r is the robot angle

θ_e is the angle error

λ is the robot-to-target angle

δ is the final robot angle at the target position

β is the alignment error

The alignment error is calculated using the following equation:

$$\beta = \lambda - \delta \tag{6.14}$$

The triangle height is an exponential function of the alignment error and is given by the following equation:

$$h = d \left[1 - e^{-\left| \frac{\beta}{triCur} \right|} \right] h_{\max} \tag{6.15}$$

where d is the triangle protrusion direction and is dependent on the value of β, the alignment error; d is -1 if $\beta >= 0$, otherwise it is 1. The value of β determines how far the robot has to arc around to align with the target and the final angle; it is zero when the robot is perfectly aligned with the final direction, at which stage the triangle height, h, is zero too.

h_{max} is the maximum triangle height and is related to the distance, d_{RT}, by the following equation:

$$h_{\max} = \frac{triMax}{triOff} * d_{RT} \tag{6.16}$$

The maximum triangle height is clamped at *triMax*. From the equations (6.15) and (6.16), it can be deduced that the triangle height is proportionately small when the robot is closer to the target. This prevents the robot from traversing an expensive arc even if the alignment error is large. The value of h_{max} needs to be clamped; otherwise the arc traversed by the robot will keep getting bigger as the robot goes farther away from the target. The relationship between the maximum triangle height and the robot-to-target distance is illustrated in Figure 6.20.

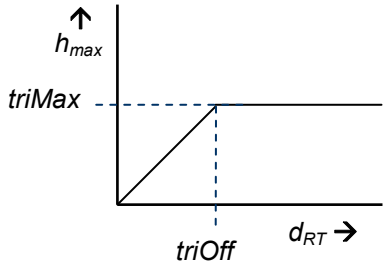


Figure 6.20 Relationship between the maximum triangle height and the distance error

For a given alignment error, β, the curvature, *triCur*, determines the rate at which the robot will converge behind the target, aligned with the final angle. The smaller the value of *triCur*, the steeper will be the curve that relates triangle height, h, to the alignment error, β. This is shown in Figure 6.21. If the curve is too steep, the robot will oscillate along the final angle vector as it approaches the target. If the curve is too shallow, the robot will slowly align with the final angle vector and may not reach the final angle when it reaches the target. This is depicted in Figure 6.22. Figure 6.23 shows the triangle heights for different robot positions in a workspace of dimension 150 cm x 150 cm. The target is at location (0, 0) and the final angle is 0^o.

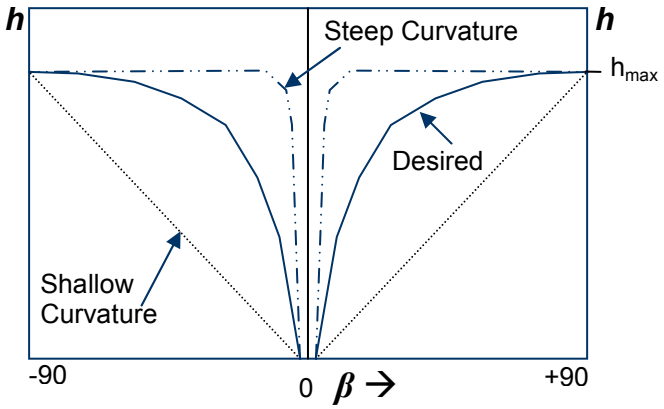


Figure 6.21 Exponential profiling of triangle height

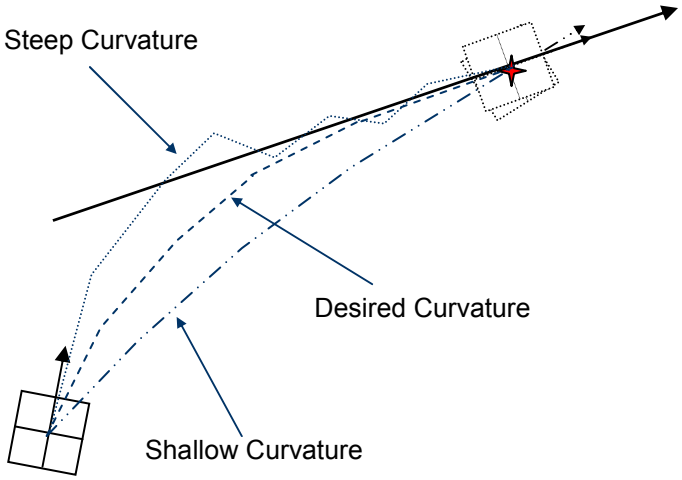


Figure 6.22 Effects of curvature on robot movement

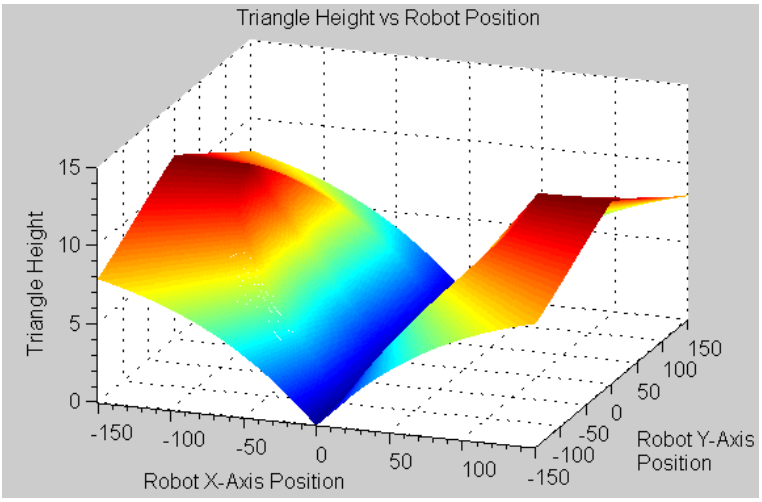


Figure 6.23 Triangle heights for different robot positions

6.5 Proximity Positioning Algorithm

As explained in section 6.3.2, when the robot is very near the target, within a pre-defined circular zone, the positioning algorithm switches over to PPA (Proximity Positioning Algorithm). The purpose is to increase the absolute positioning accuracy. It also helps to overcome the noise in the vision system.

The algorithm works by locking the robot's angle to the final angle when the robot is in close proximity of the target. Figure 6.24 shows how the algorithm works. Line AB is orthogonal to the final direction vector, *F*. For reference, X-axis is defined along the direction vector *F* and so the Y-axis is along the line AB. When the robot enters the inner circle, it turns to align with the final direction vector and continues to move (with pure linear motion) till it reaches the line AB. This reduces the final angle error and the X-axis error of the robot's position with respect to the target. However, there is a possibility of the robot reaching the target with a positional error along the Y-axis. In practice, this error is very small since the diameter of the circle (i.e. length of line AB) is kept very small, generally much less than half the robot width. The algorithm is very accurate when used in conjunction with Triangular Targeting Algorithm (TTA) and greatly improves the stability of the robot when it reaches the target. The hysteresis helps to overcome the oscillatory effects of the sensor noise at the transition boundary.

The diameter of the circle, which is the boundary for positioning algorithm to switch from TTA to PPA, and the hysteresis can be changed using the Graphical User Interface (GUI) shown in Figure 6.44.

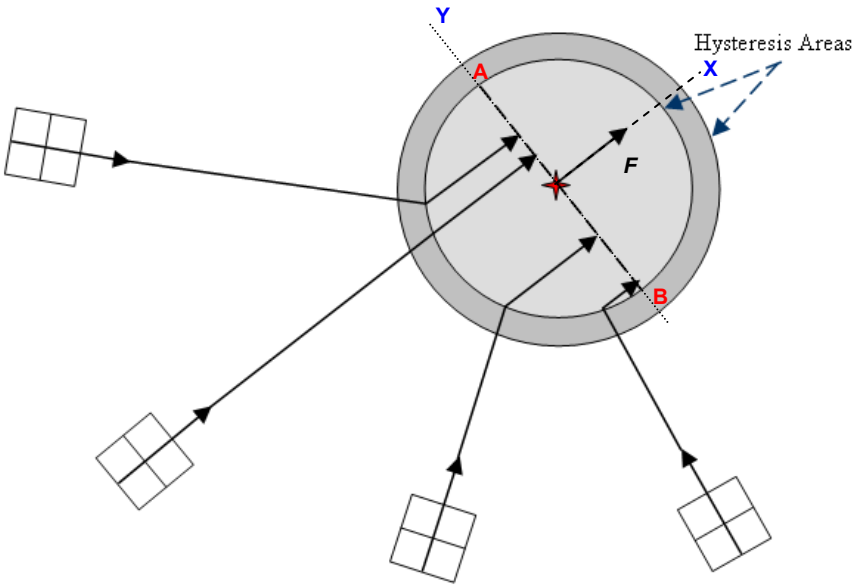


Figure 6.24 PPA algorithm at work

6.6 Velocity Profiling

With reference to Figure 6.19, the robot has to move the distance d_{RV} at an angle μ. These are given by the equations (6.17) and (6.18). The wheel velocities are calculated to move the robot to the virtual target and are a function of the distance error (d_e) and angle error (θ_e).

$$d_e = d_{RV} = \sqrt{h^2 + \left(\frac{d_{RT}}{2}\right)^2} \tag{6.17}$$

$$\mu = \lambda + \tan^{-1}\left(\frac{h}{d_{RT}/2}\right) \tag{6.18}$$

$$\theta_e = \theta_r - \mu \tag{6.19}$$

Velocity profiling ensures rapid and accurate response to robot positioning. This system has three parts - angular, linear and cross-effects profiling.

6.6.1 Angular Velocity Profiling

Angular profiling controls the turning performance of the robot. It is desirable that the robot's angular velocity be high when there is a large turning angle and low when the turning angle is small so that the desired angle is not overshot. The general profiling function used here is an exponential curve as shown in Figure 6.25.

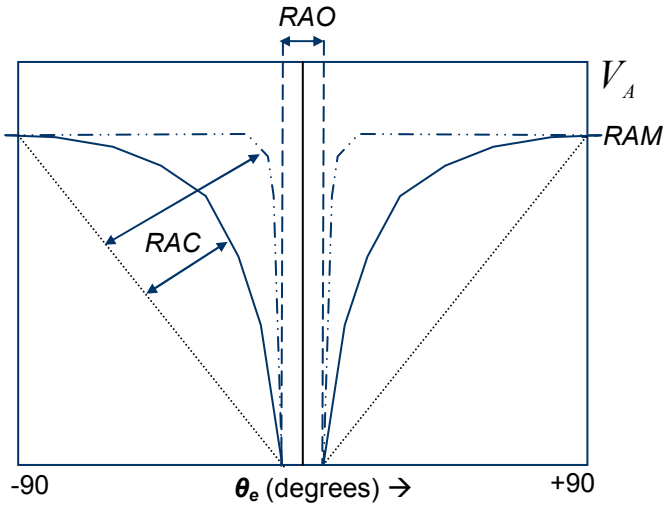


Figure 6.25 Exponential profiling of angular velocity

The angular velocity, $V_{A,}$ is given by the following equation:

$$V_A = 0 \Big|_{|\theta_e| < RAO}$$

$$V_A = \left(1 - e^{-\left(\frac{|\theta_e| - RAO}{RAC}\right)}\right) RAM \tag{6.20}$$

For angle errors less than *RAO*, the angular velocity is zero. The angular velocity is clamped at *RAM*, the maximum angular velocity of the robot. Exponential profiling is very effective in quickly correcting the robot's orientation. The parameters Robot Angular Curvature (*RAC*), Robot Angular Offset from zero (*RAO*) and the Robot's Angular Maximum turning velocity (*RAM*) can be efficiently controlled by this technique. If the curvature is set too high (low value of *RAC*) then the system becomes unstable as small angle errors will generate high angular velocity (V_A). Offset from zero is also required to be small to keep angle accuracy high, but large enough to overcome the inaccuracies of the vision system.

6.6.2 Linear Velocity Profiling

It is desired that the linear velocity be large when the robot is far from the target and small when the robot is close to it. Exponential profiling is also useful here as it provides rapid convergence to the desired position. When the robot is closer to the target position the speed is reduced rapidly to avoid overshoot. The linear velocity profiling curve is shown in Figure 6.26.

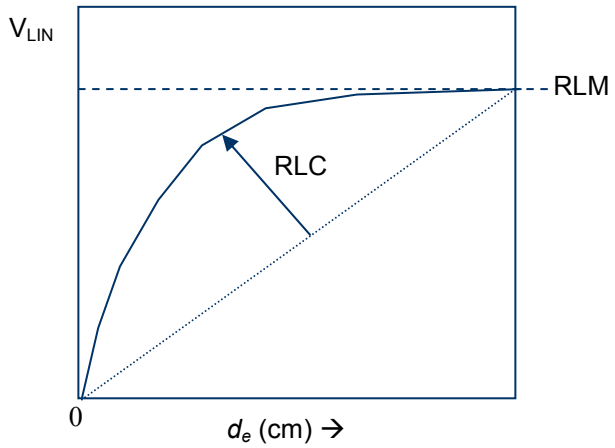


Figure 6.26 Exponential profiling of linear velocity

The equation to calculate linear velocity, V_{LIN}, is:

$$V_{LIN} = \left(1 - e^{\frac{-|d_e|}{RLC}}\right) RLM \tag{6.21}$$

RLM is the maximum linear velocity and *RLC* is the curvature.

6.6.3 Cross-Effects Profiling

A major problem arises when the angle error and the distance error are large at the same time. This results in large angular and linear velocities. Because of the large linear velocity component added to the angular velocity, the robot rotates too fast and overshoots resulting in a wobbly motion to the target. This 'cross effect' of angular and linear velocities is a tricky problem to manage, especially for fast moving robots. For higher angle errors, the linear velocity component has to be reduced and as the robot orients towards the target, i.e. the angle error reduces, the linear velocity component should increase. This is managed by an exponential 'cross effect curve' shown in Figure 6.27.

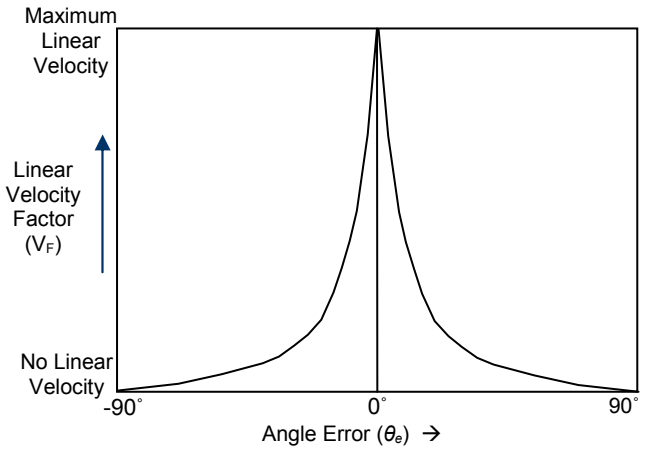


Figure 6.27 Exponential Cross-effect curve

For a given angle error, a linear velocity factor, V_F, is calculated using the following equation:

$$V_F = 1 - \left(1 - e^{\frac{-|\theta_e|}{RCEC}} \right) \tag{6.22}$$

In equation (6.22), RCEC is the curvature of the cross-effect curve. The final robot wheel velocity is calculated by combining the angular velocity (calculated using equation 6.20) and linear velocity (calculated using equation 6.21). It is given by the following equation:

$$V = V_A + V_{LIN} V_F \tag{6.23}$$

For larger values of angle error (θ_e), the linear velocity factor (V_F) is small. In other words, when the angular velocity is large, the linear velocity component of V in equation (6.23) is small. As the angle error reduces, V_F increases thereby increasing the linear velocity component.

6.7 Experimental Results

The algorithms were tested on the soccer robots.

6.7.1 Angular Motion Function

To evaluate the angular performance, the robot was rotated on the spot by 90^o using only the angular velocity profiling described by equation (6.20). The results are compared with those of a PD controller. The comparison is shown in Figure 6.28. The performances are virtually indistinguishable, though the PD control is steeper initially. In either case, the robot reaches its final angle of 90^o in 16 frames which is 266.5ms. There are practically no over shoots and the steady state errors are very small.

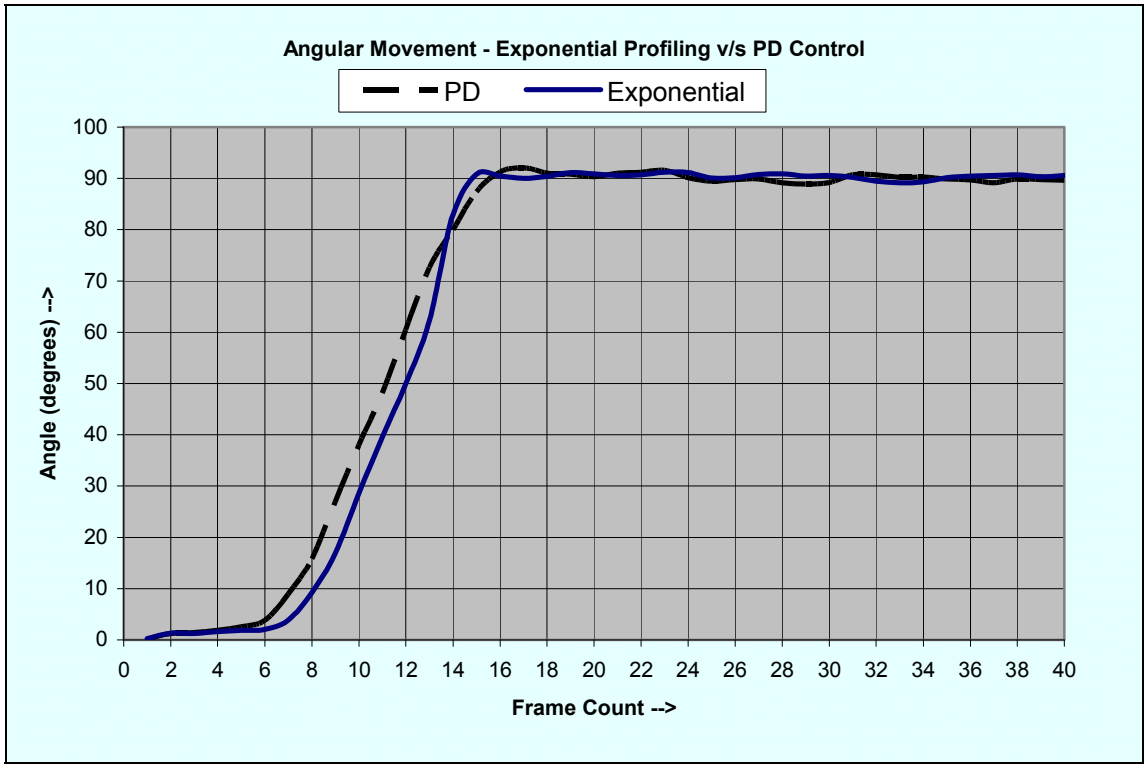


Figure 6.28 Evaluation of angular movement

6.7.2 Linear Motion Function

To evaluate the linear performance, the robot was moved in a straight line along the X-axis only from location (15, 65) to (130, 65). The linear velocity profiling described by equation (6.21) was used. The results are compared with those of a PD controller. The comparison is shown in Figure 6.29.

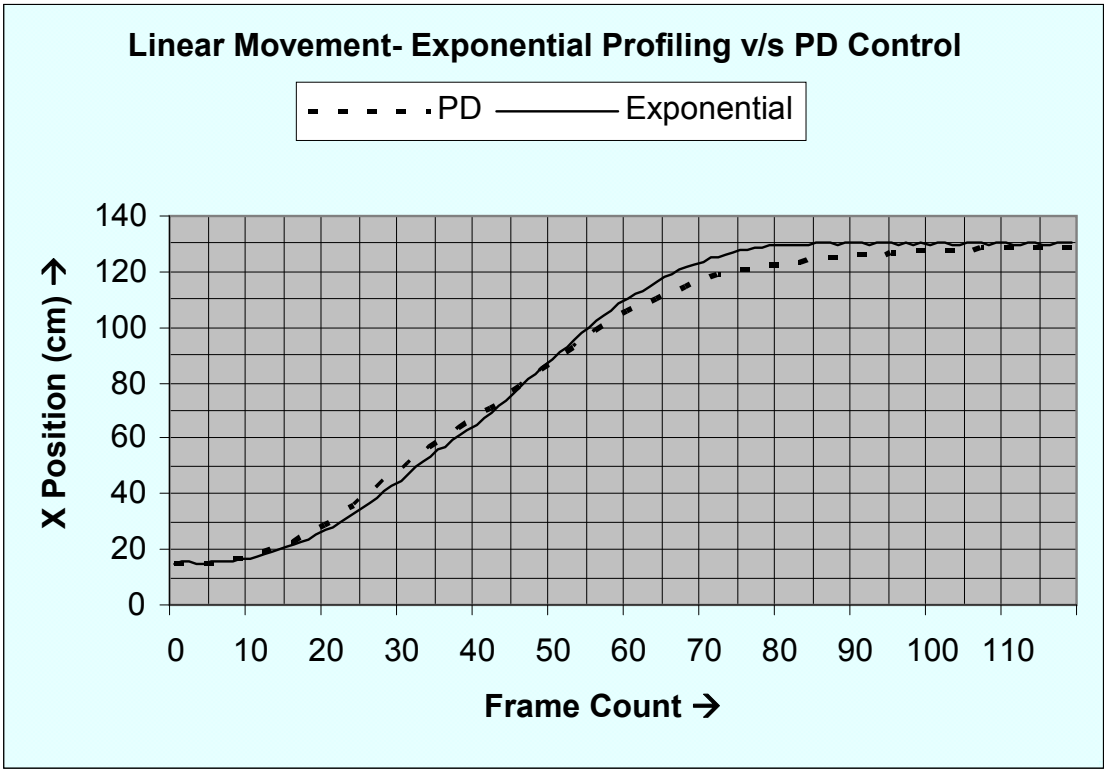


Figure 6.29 Evaluation of linear movement

As seen in Figure 6.29, the robot reaches the destination much quicker using exponential profiling. Using exponential profiling the robot takes 80 frames (1.333 s) to reach the destination while using a PD controller it takes 120 frames (1.992 s).

6.7.3 Cross-Effects Optimization

To test the cross-effects profiling (Figure 6.27 and equation 6.22), the robot is moved diagonally across the test platform from location (20, 20) to (130, 110). The final angle is 90^o off the start angle. A diagonal path was deliberately chosen so that the angular and linear velocity components are both present. Triangular Targeting Algorithm (TTA), described in section 6.4 earlier, was turned off so that no virtual target position is calculated and the robot straight heads for the actual target position. The performance of the algorithm in merging the linear and angular profiles was measured for two cases

– optimised and non-optimised cross-effect curves. The paths taken by the robot are shown in Figure 6.30.

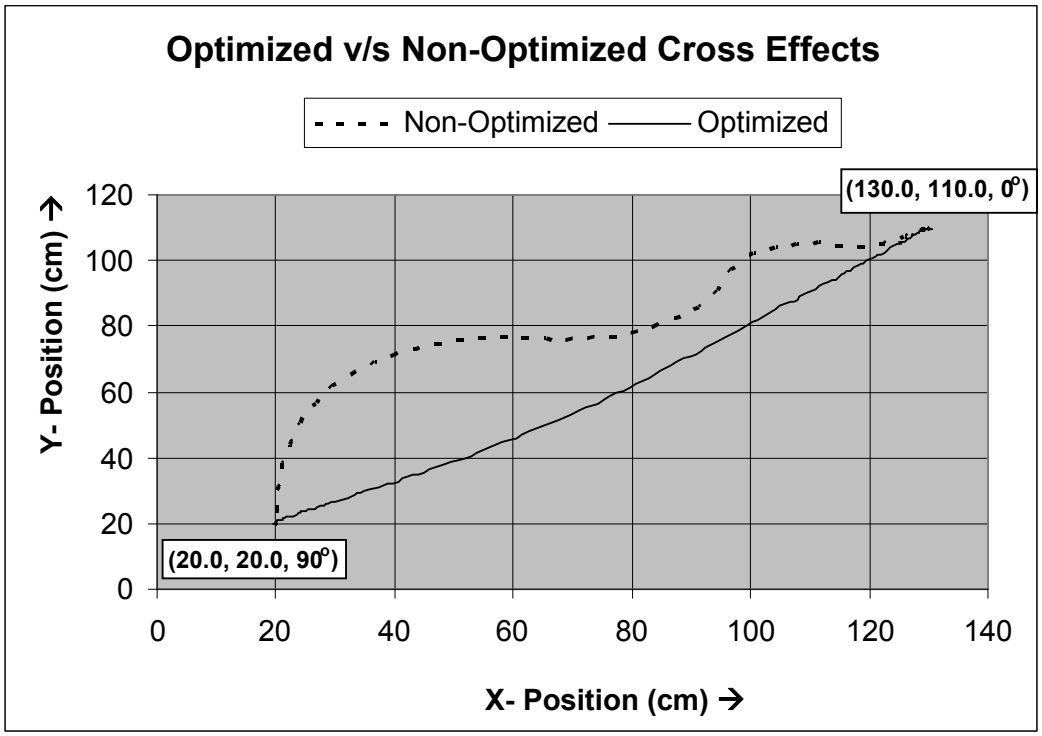


Figure 6.30 Evaluation of cross-effect curve

The non-optimized case refers to the use of the exponential cross-effects curve of Figure 6.27. It was observed that the algorithm suffers from a few drawbacks, namely duck tailing (angular overshoots), presence of linear velocity even at 90^o angle error and large swing in linear velocity due to a small change in angle error. To overcome these problems a modified/optimized cross-effects curve was used. The optimized 'fuzzy' cross-effect curve is shown in Figure 6.31 and is used to calculate the linear velocity factor (V_F). The linear velocity component is further clamped using the curve shown in Figure 6.32. As seen in Figure 6.30, the robot travels a much shorter path for the optimized case.

6.7.4 Testing Triangular Targeting Algorithm

The TTA algorithm was extensively tested. The robot was commanded to move from different starting positions to reach a target. The series of graphs shown in Figures 6.33 to 6.43 depict the robot movement path and the virtual target path for various starting positions of the robot and target respectively.

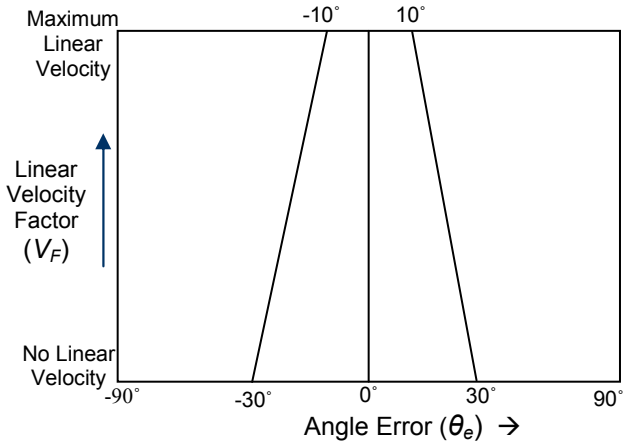


Figure 6.31 Optimized 'fuzzy' cross-effect curves

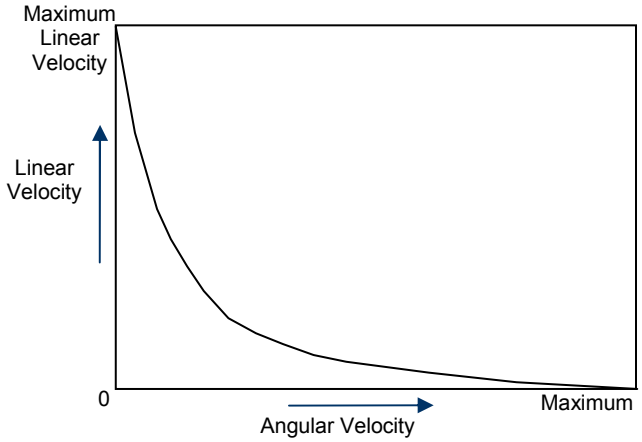


Figure 6.32 Relating linear velocity to angular velocity (velocity clamping curve)

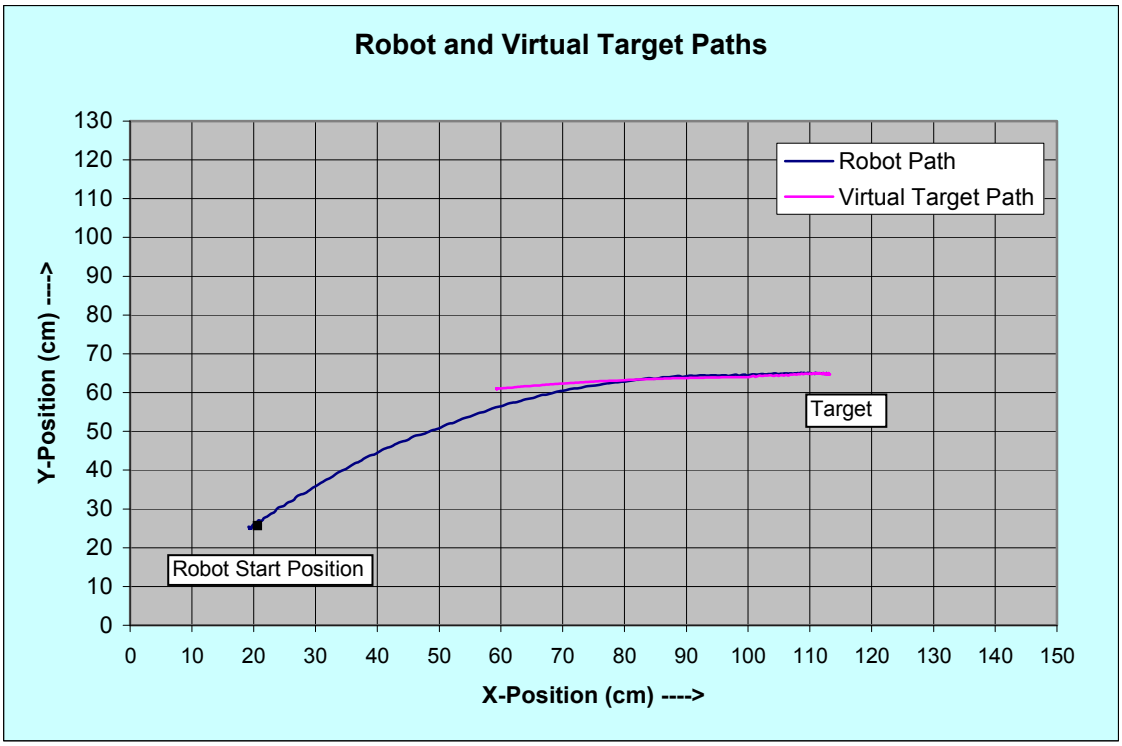


Figure 6.33 Robot starting posture (20, 25, 90°), end posture (115, 65, 0°)

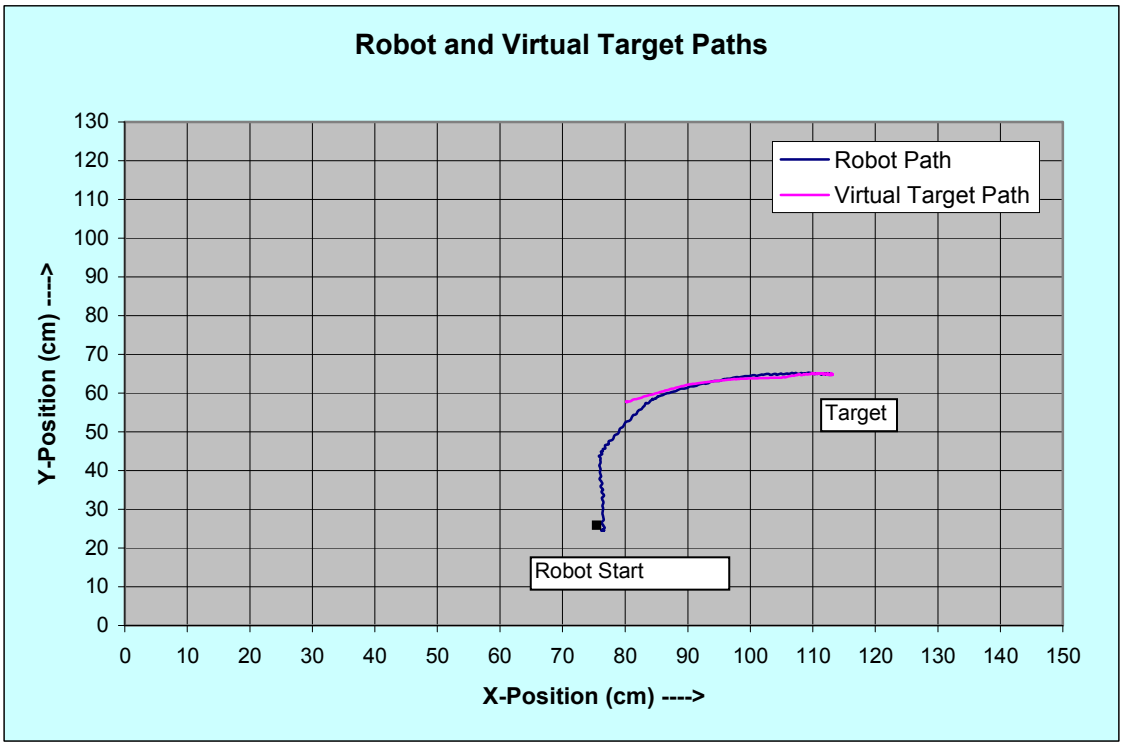


Figure 6.34 Robot starting posture (75, 25, 90^{o}), end posture (115, 65, 0^{o})

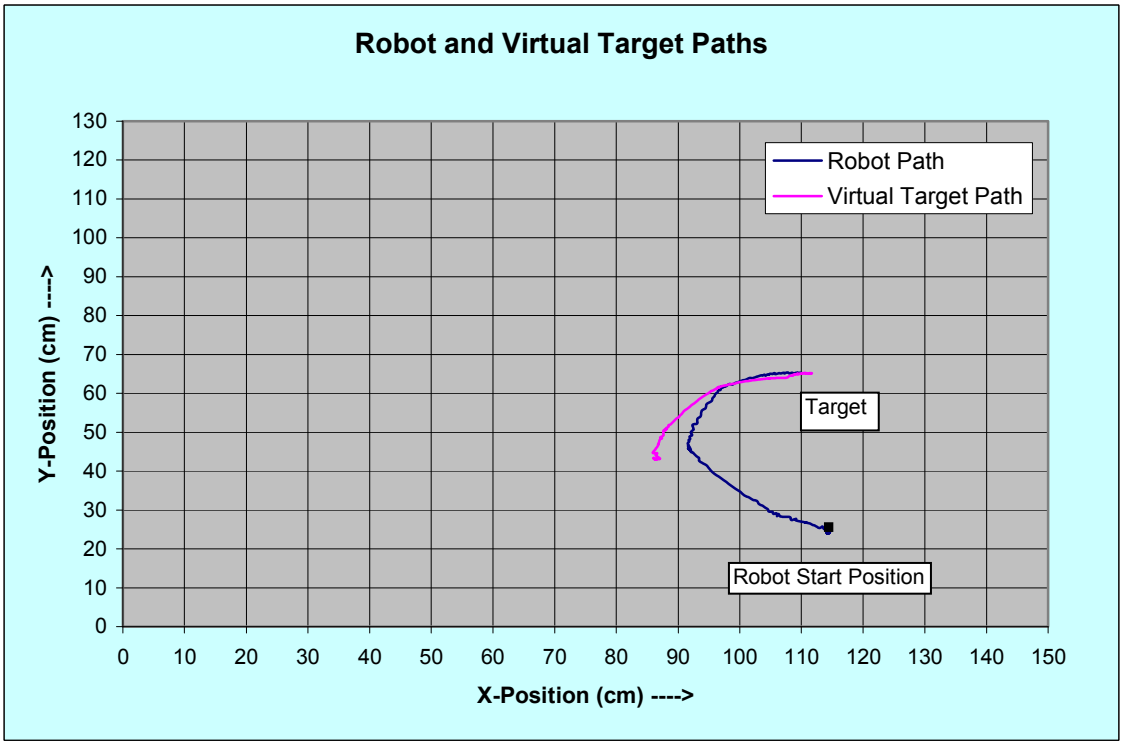


Figure 6.35 Robot starting posture (115, 25, 90^{o}), end posture (110, 65, 0^{o})

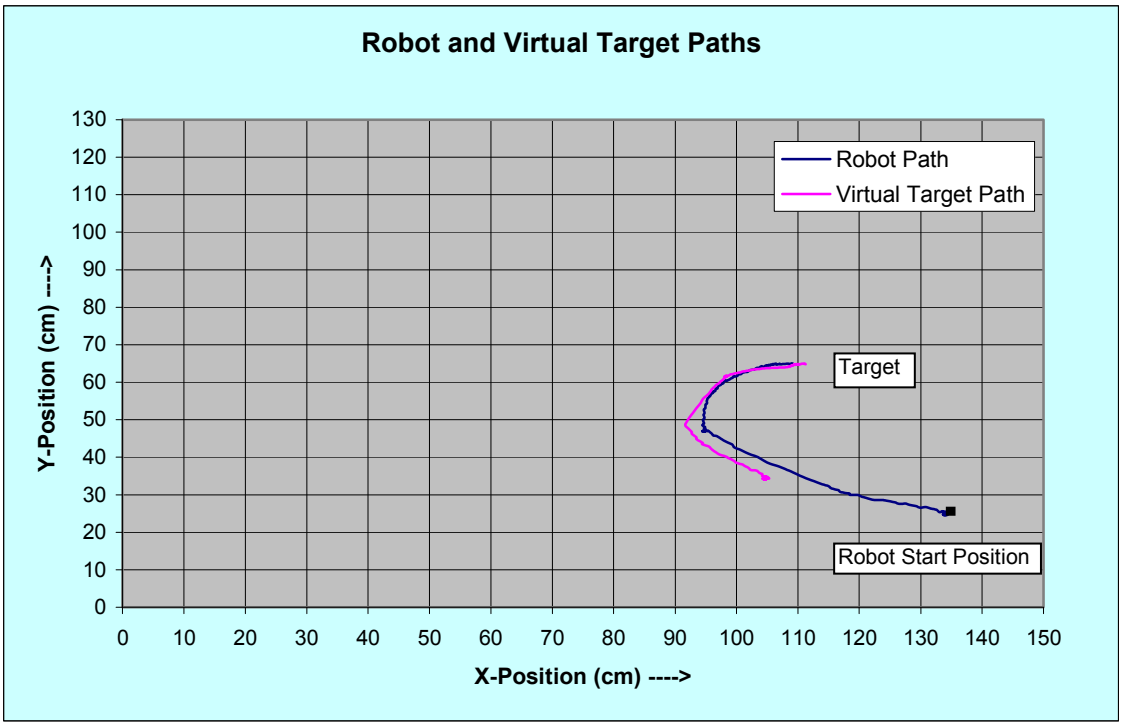


Figure 6.36 Robot starting posture (135, 25, 90°), end posture (110, 65, 0°)

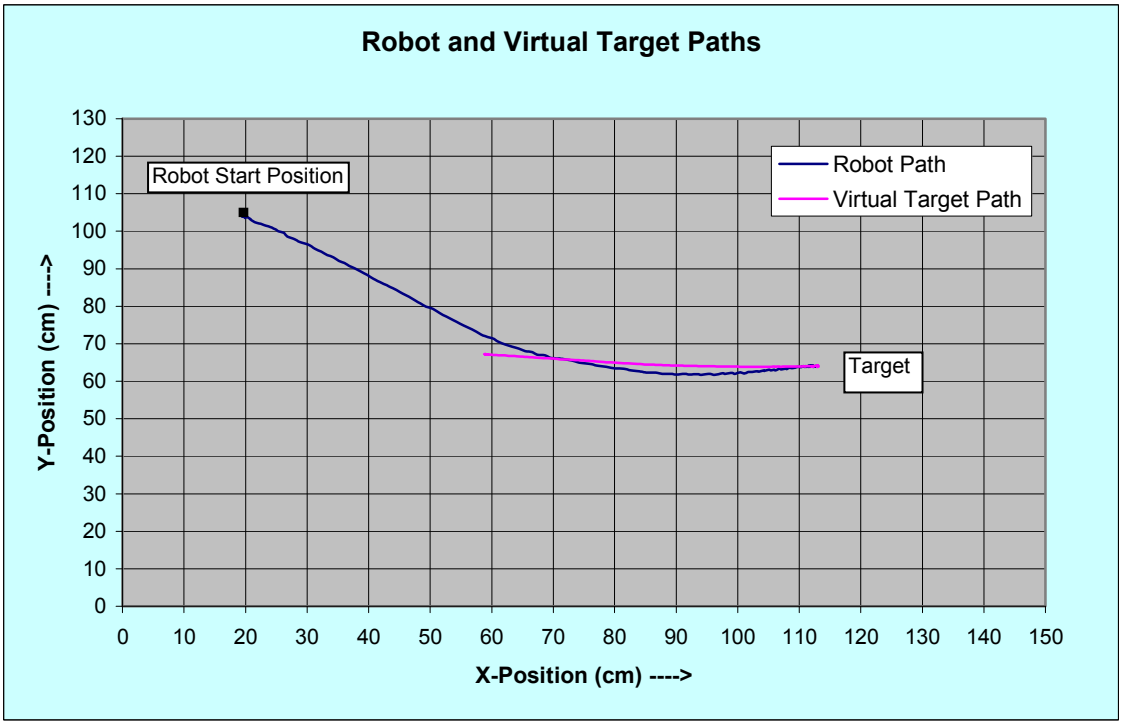


Figure 6.37 Robot starting posture (20, 105, 90°), end posture (115, 65, 0°)

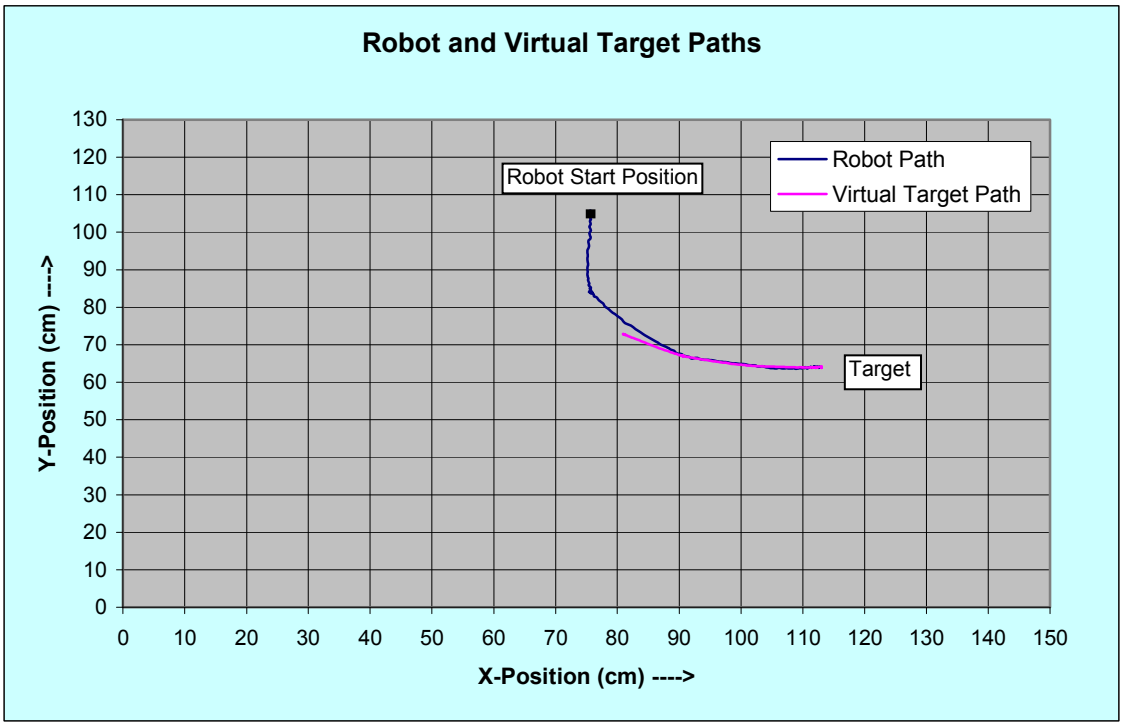


Figure 6.38 Robot starting posture (75, 105, 90^{o}), end posture (115, 65, 0^{o})

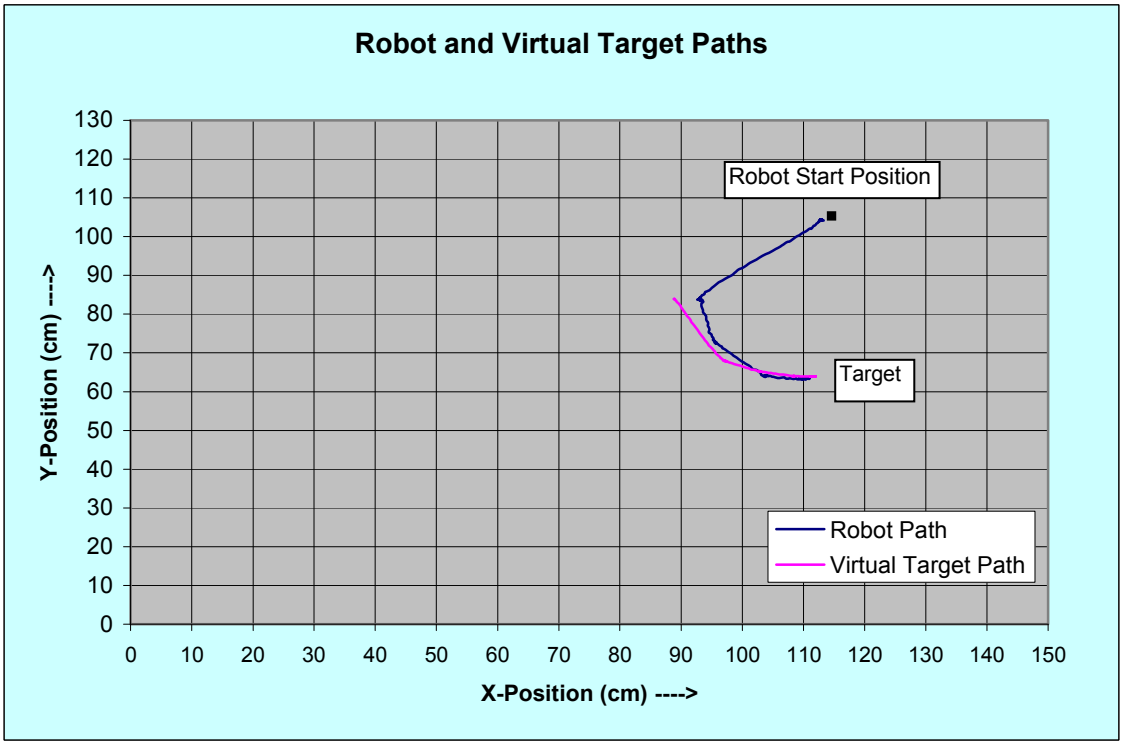


Figure 6.39 Robot starting posture (115, 105, 90^{o}), end posture (110, 65, 0^{o})

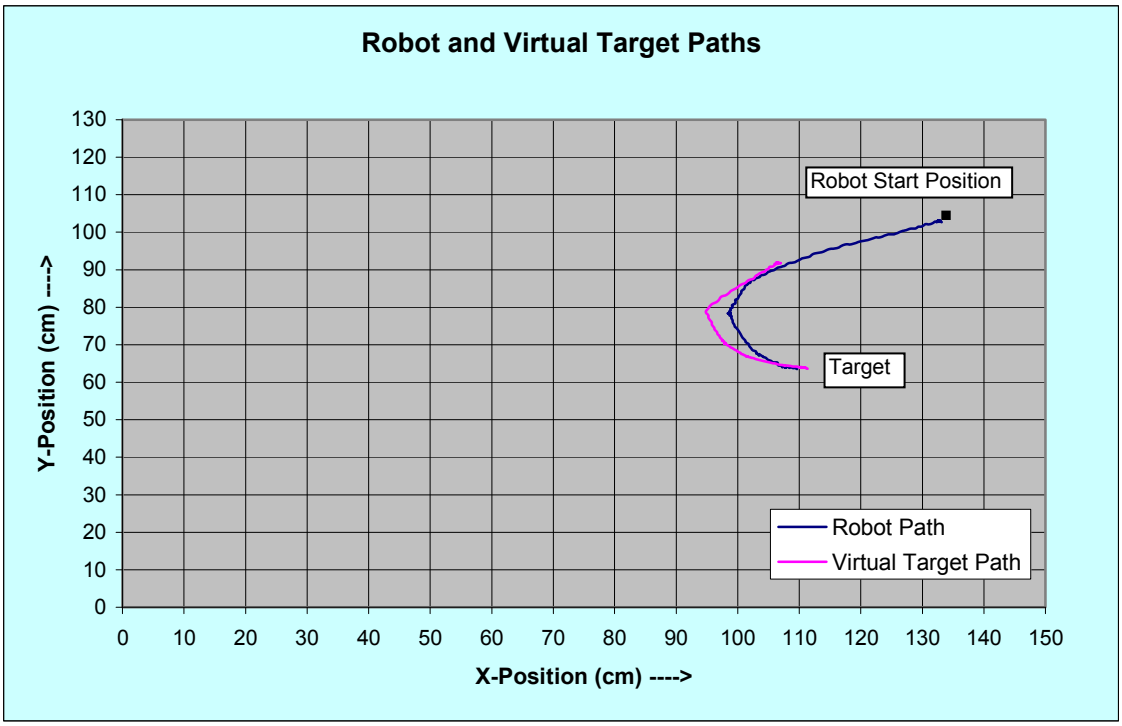


Figure 6.40 Robot starting posture (135, 105, 90°), end posture (110, 65, 0°)

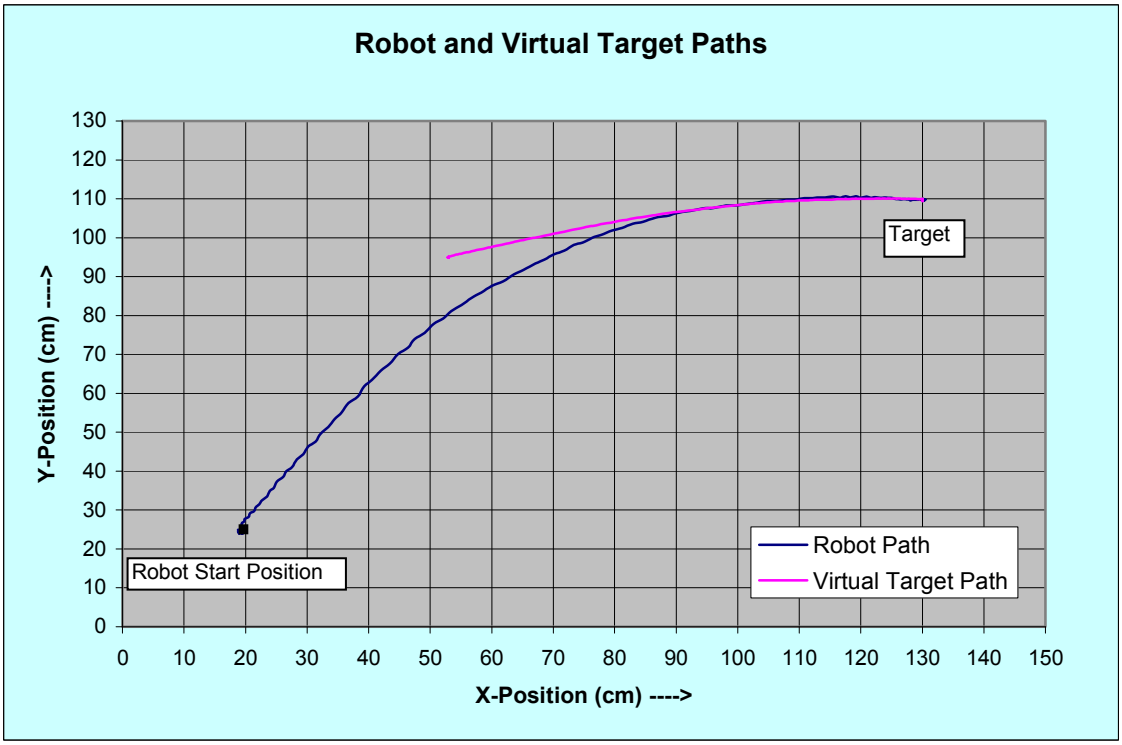


Figure 6.41 Robot starting posture (20, 25, 90°), end posture (130, 110, 0°)

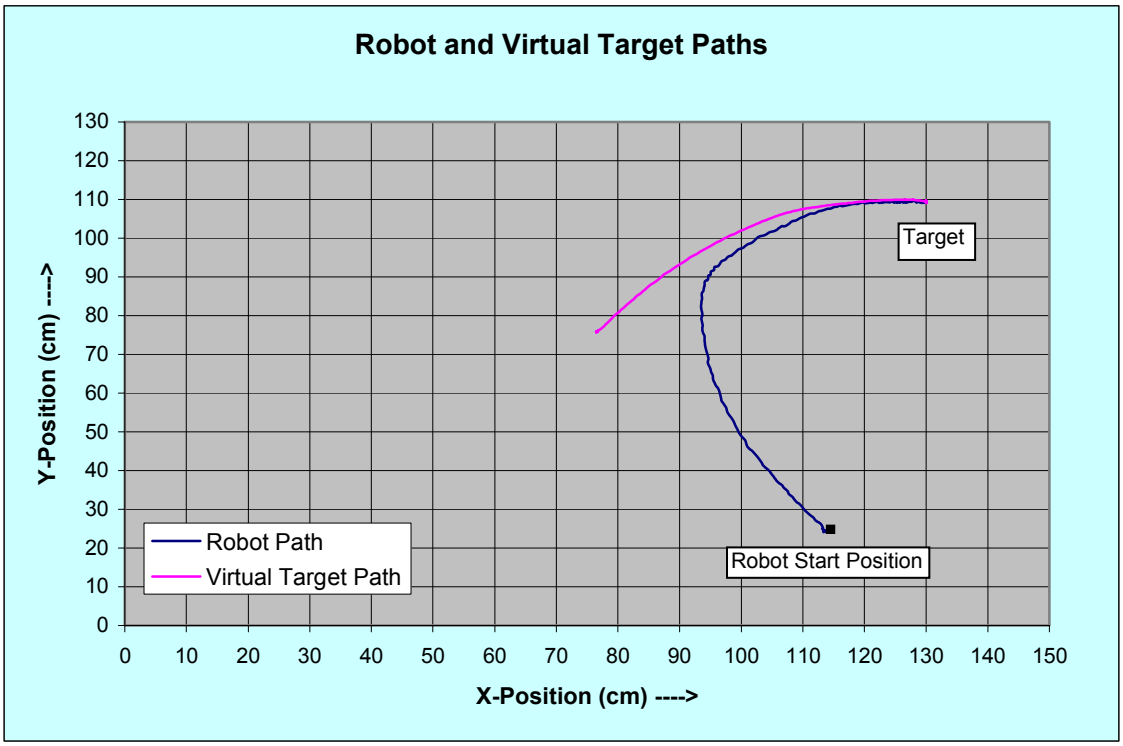


Figure 6.42 Robot starting posture (115, 25, 90^{o}), end posture (130, 110, 0^{o})

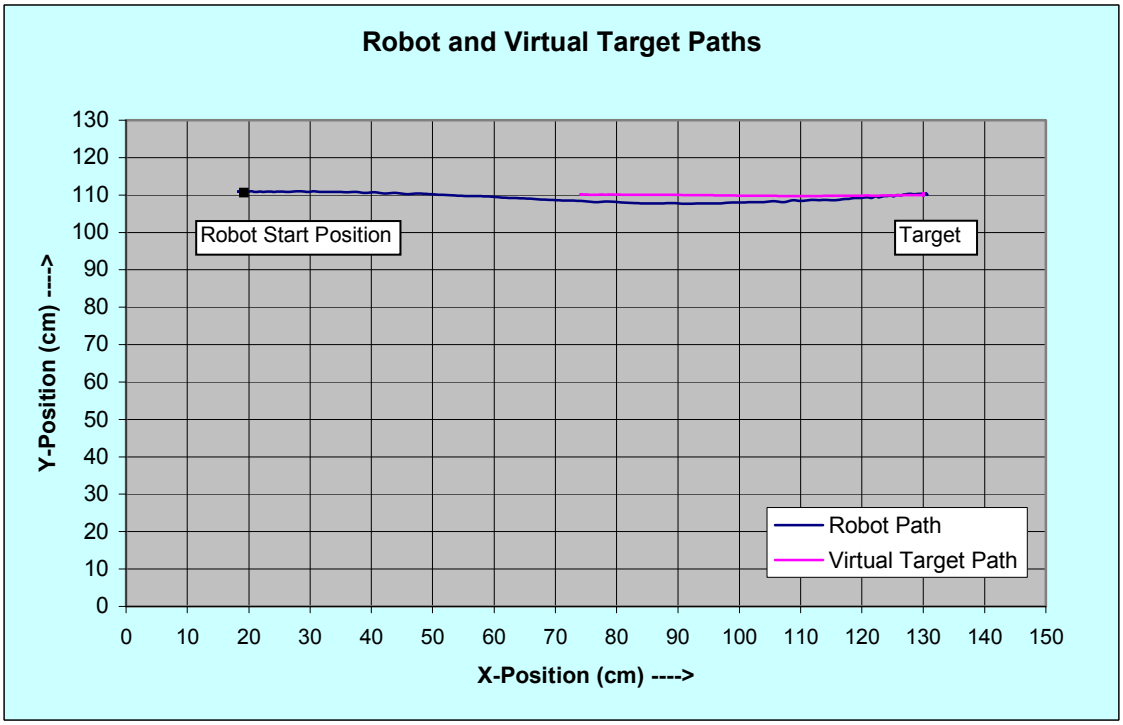


Figure 6.43 Robot starting posture (20, 110, 0^{o}), end posture (130, 110, 0^{o})

6.8 Discussion

A novel Triangular Targeting Algorithm (TTA) for accurate motion control of fast mobile robots has been proposed and tested. The algorithm works by moving the robot to an intermediate virtual target on its way to the final destination. The position of this virtual target is continually updated as the robot traverses through the workspace. When the robot is within a predetermined range of the target, the control switches to Proximity Positioning Algorithm (PPA) which is a positioning algorithm to minimize the position error at the destination. The robot is accurately positioned at the target, oriented towards the final direction. The experimental results confirm that the proposed algorithm significantly improves the performance of the robot when compared to standard methods such as PD controllers which are commonly used. The added advantage of this algorithm is that a separate path generation process (followed by position controller) is not required. The algorithm creates the trajectory and the wheel velocities required to follow it. The exponential functions in the developed algorithms are evaluated only once every frame, hence they do not add any significant overhead to the computation time.

The accuracy and repeatability of the algorithm was established by performing the experiments (documented in Figures 6.33 to 6.43) ten times or more. Sample results are presented in the Tables 6.2 and 6.3. Table 6.2 summarises the experimental results of moving the robot from an initial posture of (135, 105, 90^o) to an end posture of (110, 65, 0^o) which is plotted in Figure 6.40.

Table 6.2 Performance matrix for TTA: start posture (135, 105, 90^o), end posture (110, 65, 0^o)

Run	Distance Travelled	*Position Error*	*Angle Error*
1	79.54 cm	1.67 cm	-2.37 ^{o}
2	78.11 cm	2.08 cm	-2.22 ^{o}
3	80.08 cm	1.81 cm	1.63 ^{o}
4	78.67 cm	0.79 cm	1.40 ^{o}
5	79.60 cm	1.55 cm	-0.92 ^{o}
6	78.01 cm	1.39 cm	2.77 ^{o}
7	79. 82 cm	1.68 cm	2.48 ^{o}
8	79. 66 cm	1.09 cm	1.84 ^{o}
9	78.83 cm	1.11 cm	-1.18 ^{o}
10	79. 17 cm	1.45 cm	-1.29 ^{o}
Maximum	*80.08 cm*	*2.08 cm*	*2.77 ^{o}*
Average (Modulus)	*78.98 cm*	*1.46 cm*	*1.81 ^{o}*
Standard Deviation	*0.78786*	*0.383081*	*0.624767*

Table 6.3 Performance matrix for TTA: start posture (20, 25, 90°), end posture (130, 110, 0°)

Run	Distance Travelled	*Position Error*	*Angle Error*
1	153.43 cm	1.22 cm	-1.73 °
2	153.36 cm	1.87 cm	-2.22 °
3	154.12 cm	0.89 cm	1.89 °
4	153.70 cm	1.88 cm	1.04 °
5	153.12 cm	1.26 cm	0.84 °
6	154.78 cm	1.55 cm	-2.63 °
7	152.61 cm	1.66 cm	-1.88 °
8	153.66 cm	1.12 cm	2.33 °
9	152.20 cm	1.80 cm	-2.40 °
10	153.72 cm	0.90 cm	-1.62 °
Maximum	*154.78 cm*	*1.88 cm*	*2.63 °*
Average (Modulus)	*153.47 cm*	*1.42 cm*	*1.89 °*
Standard Deviation	*0.728682*	*0.385811*	*0.579613*

Table 6.3 summarises the experimental results of moving the robot from an initial posture of (20, 25, 90°) to an end posture of (130, 110, 0°) which is plotted in Figure 6.41. The path and the distance traversed in each run are very close. Maximum positional error at the destination is 2.08 cm and the maximum angle error at the destination is 2.77°.

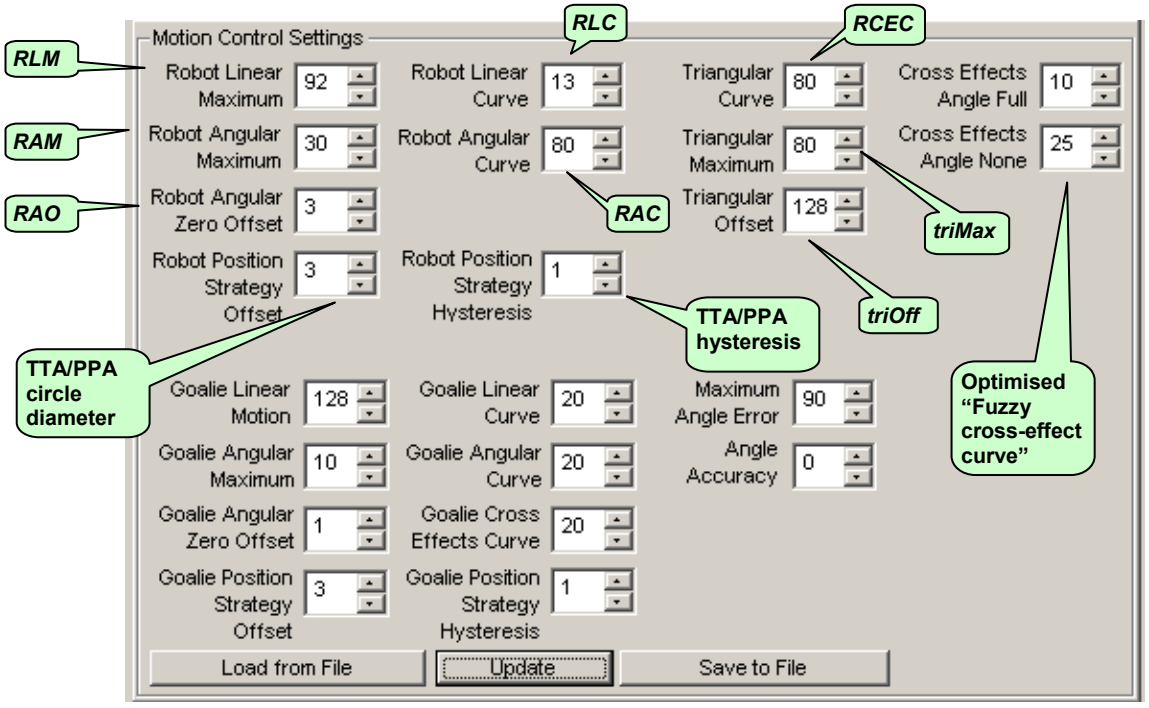


Figure 6.44 GUI for tuning TTA parameters

Figure 6.44 shows the graphical use interface which can be used to tune individual robots. The tuning process starts with fixing certain parameters such as *RLM*, *RAM*, *RAO*, *triOff*, *triMax*, the TTA/PPA circle diameter and hysteresis. This is a one-time effort. Subsequently, the various curvatures are tuned starting with the angular velocity profiling curvature, *RAC*. This is done by making the robot turn on the spot by 90^o as detailed in section 6.7.1. The angular curvature is increased (i.e. the value of *RAC* is reduced) to a point where the robots just starts exhibiting overshoots. Next, the robot is made to move in a straight line as detailed in section 6.7.2. The linear curvature is increased (i.e. *RLC* is reduced) to a point where the robot just starts exhibiting overshoots. To tune the cross-effects curvature, the robot is moved diagonally as detailed in section 6.7.3. The value of *RCEC* is increased or decreased to suit the performance desired from the robot.

Some experiments were done to investigate if the TTA parameters can be fine tuned for two different distances to the target – a short range and a long range. The algorithm is quite efficient for both short and long ranges. However the algorithm can be adjusted to perform better at either range.

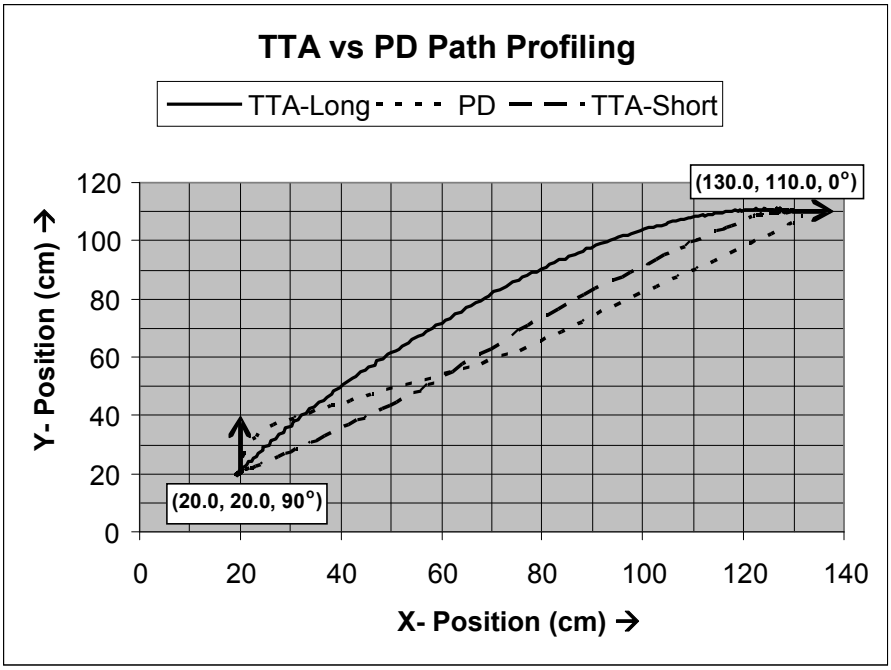


Figure 6.45 Performance evaluation of long and short range TTA

TTA-Long demonstrates a fairly optimal path for long ranges such as the one shown in Figure 6.45. However in close range, TTA-Long induces large and inefficient arcs. TTA-Short has been tuned for short range and is commonly used as it is comparatively efficient at long ranges and very efficient at close range.

7 Summary and Future Research

Many mobile tasks can be most efficiently accomplished when a group of robots is employed. Some tasks are almost impossible without multiple robots. Complex group behaviours can be formed from simple interactions between individual robots. The type of organisation and the level of coordination, and communication within a team of robots affect the type of tasks that can be done. In *swarm* approaches, usually large groups of robots execute the same simple strategies with no explicit communication. In *cooperative* approaches, usually smaller groups of robots can have different strategies and behaviour – some robots may perform rudimentary tasks while others may be assigned expert roles. This heterogeneity at behaviour level is often imperative in solving complex tasks. The term *collaborative* robotics has come to mean cooperative behaviour where robots explicitly communicate with one another or with a central command and control unit.

Homogeneous teams of robots have one clear advantage – there is built-in redundancy. If a team member fails for any reason, the rest of the team can still go on and be successful. Tasks that cannot be reasonably done by a single agent should lead to heterogeneity. The need for heterogeneity also depends on the number of skill sets required by the domain. The robot soccer platform is a good example of a domain requiring agents to have multiple skill sets. Heterogeneity can be at two levels – the hardware level where the robots are equipped with different sets of sensors and/or actuators, or at behaviour level where each robot is capable of exhibiting different behaviour or role. In an unstructured domain, being able to allocate roles dynamically is a big advantage in solving complex tasks. The research presented in this thesis deals with robotic herds which are homogeneous at hardware level but heterogeneous at behaviour level. The herd is shepherded by a central command and control unit which relies on the inputs from a global vision system that oversees the workspace of the herd.

Robotic herding behaviour is not new; it was first described in [80]. Many researchers, working in the area of artificial intelligence, have used groups of robots to evolve cooperative behaviours [81 - 83]. However, all the reported research is simulation based and to date no work has been reported that has used actual robots. Even the collaborative and adversarial learning using robot soccer reported in [84] is based on simulated robot models that simulate real-world responses.

Implementation of cooperative behaviour using real robots in a real-world environment poses several significant challenges. Simulators model real-world responses such as friction, conservation of momentum etc *as closely as possible* [84]. Sensor noise of variable magnitude is included to model *more or less* precise real systems. However,

simulators cannot accurately model the dynamic nature of the robots' workspace. For example the unpredictable and momentary disturbances created in the colour thresholds in the vision system by reflective surfaces around the robot's operating environment such as a spectator walking past the robot soccer platform. The precise nature of wheel slippage and its variation due to deterioration of tyre surface is very complex to model. Nonetheless this has profound influence on the accuracy of the robots movements.

The challenges in implementing cooperative behaviour using fast moving real robots are shown in Figure 7.1. These are devising fast image capture and vision processing algorithms for real-time applications; easily and quickly building complex behaviour for collaborative operations; eliminating or significantly reducing noise from vision systems to improve predictive movement and interception control; and developing algorithms for accurate posture control of fast mobile robots. This thesis has addressed all these challenges. Science and engineering has still not made progress to the extent where the dexterity of human hand and finger movements can be replicated by hardware. However, it is envisaged that in future, anthropomorphic robotic arms will play an increased role alongside human beings in teleoperated collaborative tasks. The thesis has thus addressed the issue of the development of an anthropomorphic robotic arm with force feedback with an eye on the future role of man-machine interaction and collaboration.

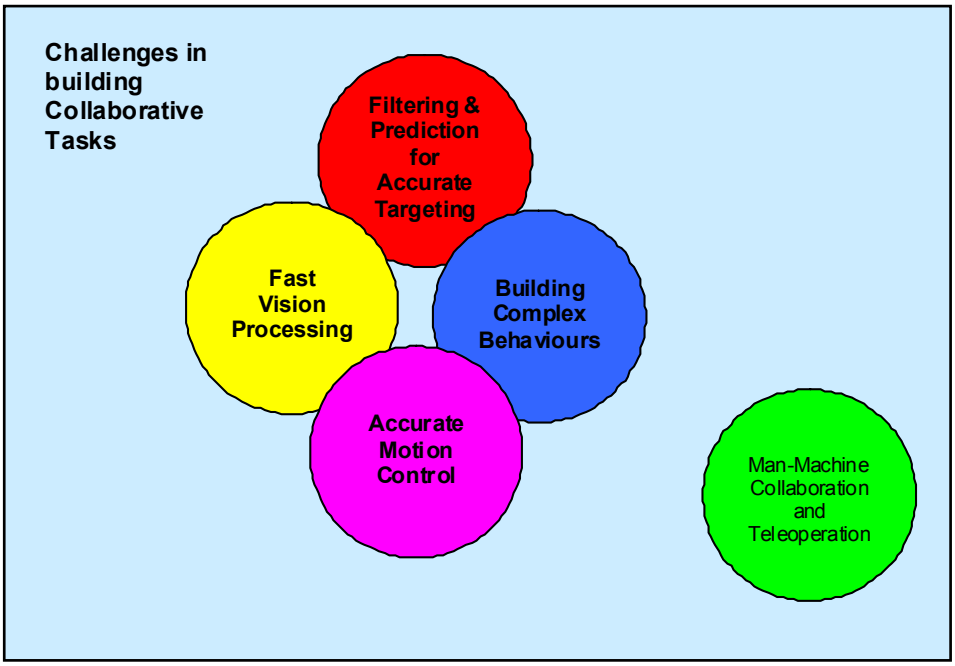


Figure 7.1 Factors significantly affecting collaborative behaviour in a vision based domain

Fast Vision Processing. The thesis examined the methods to improve the speed of image capture and subsequent processing to achieve real-time performance. While

standard methods, such as off-screen multi-buffered capture and incremental tracking were used to increase processing speed, a novel method for creating discrete YUV look-up-tables has been proposed and evaluated. The look-up-tables are very small in size requiring very small memory (768 bytes) and can be updated very quickly (8.2 μs). Using this look-up-table, the testing of colour class membership is a breeze using logical AND operations. This however requires the colour information to be available in YUV which is often not the case with commodity hardware. Various methods to speed up conversion from RGB to YUV have been proposed and their efficacy documented. These include elimination of computationally expensive floating point multiplications and replacing division operations with shift-right operations.

Another significant contribution of the reported research is the formulation of a new colour space which has been proposed and evaluated for not only increasing the image processing speed but also the robustness of object detection. This new colour space, called Y'U'V', completely eliminates all floating point and logical shift operations and uses only integer additions and subtractions. It still uses the same discrete look-up-tables as for the standard YUV colour space though the size is slightly bigger (2298 bytes). The update time of the look–up-table is still very small (19.2 μs). Using this new colour space, significant improvements have been achieved in the overall image processing speeds for incremental and full tracking. In the presented system the vision processing was completed in 5.68 ms for 11 objects. This allowed a margin of over 10ms to complete the subsequent strategy and control calculations. The reported work has laid the foundation for implementing *adaptive thresholding*.

An architecture for distributed vision processing, using two cameras to oversee the field of operation, has been described. A strategy server to combine images from two separate cameras, with overlap regions, has been presented.

As more and more robots get equipped with local vision systems, the onus of obstacle avoidance will move from the central command and control centre to the realm of the robots themselves. Effective obstacle avoidance requires depth perception of objects in the real world for which a catadioptric stereo vision system has been proposed. The system uses a single camera and a set of planar mirrors. The use of a single camera reduces the cost and size of a robot which is essential for a herd of many robots.

Building Complex Behaviours. Multi-agent collaborative systems are inherently more complex due to the distributed nature of the system. Interactions that can occur between the agents aggravate the complexity and managing this complexity is a challenging issue. A common technique to build complex behaviour and manage agent interaction (i.e. strategy for cooperation) uses reactive architecture in which instinctive behaviours are developed as reactions to current sensor inputs. Such systems are limited in

capabilities when the number of agents is large and there is a huge heterogeneity in behaviour. It is also not suitable for rapid prototyping of new behaviours or quickly changing an existing behaviour.

A state transition based control (STBC) technique has been proposed in this thesis. It has been shown that complex behaviours and strategy for collaboration can be easily built using this technique. The technique is similar to the finite state machines commonly used for the design of sequential digital circuits [85], albeit with some differences. This is the first known utilisation and adaptation of STBC for collaborative systems. The thesis reports how the technique has been used successfully to build heterogeneous agent behaviours and role selection in a robot soccer game, auto placement of robots and obstacle avoidance. Formation and marching problems of a robot contingent has been solved using STBC.

Implementing the state controller in an object-oriented class structure makes it very easy to encode and prototype behaviours. Amongst the many advantages of this approach is the data and process abstraction, which aids development of new functionalities by creating new states and building a hierarchy of such functionalities using class inheritances. The STBC strategy implementation is computationally efficient even for sophisticated behaviours and completes within 1 ms using the experimental platform. It has been shown that behaviours can be altered interactively using graphical user interfaces in a real time application.

Filtering and Prediction for Accurate Targeting. While fast vision processing and use of STBC to build complex behaviour certainly enhance the performance of a collaborative system, noise from input sensors can adversely affect the system and deteriorate the performance. In a global vision based system the sources of noise are many, namely the quantisation error due to separate processing of odd and even scans of an interlaced image, inherent noise from the camera and frame grabber electronics and the variation of light intensity (real-world uncertainties). It is thus imperative that some filtering is used to eliminate or significantly reduce noise before the vision data are used by the State Transition Based Controller (for strategy) and the motion controller (for robot positioning and object targeting). It is an added advantage if the filters are predictive so that anticipatory postures can be taken up by the robots such as by a goalkeeper in a robot soccer game to block the ball. Prediction also increases the success rate of target interception.

A *g-h* tracking filter has been proposed and extensively evaluated in this thesis. The filter parameters have been experimentally tuned for stationary and moving objects. It has been applied to a target's position as well as orientation. Using various filters, a robot's motion trajectory was recorded. The prediction algorithm was validated by

experiments conducted on a moving target that was intercepted by a robot. The performance of the filter in reducing quantisation noise due to separate processing of odd and even scan fields of an interlaced image is very significant. For the completeness of the study of the filters' effect, it was also evaluated for full scan images. It has been shown that the performance of the agents improves significantly when the filters are employed.

The filters have also been implemented as classes in an object-oriented paradigm. Filter parameters are passed to the constructor at the time of instantiating a filter object. New filters can be easily built through inheritance mechanism of object classes.

Accurate Motion Control. The state transition based controller uses the filtered data from vision layer to implement the robot behaviours and strategy for collaboration. Actions generated by the strategy layer are then translated into primitive tasks of robot posturing and movements which are implemented by generating the required wheeled velocities. Traditionally PID controllers are difficult to tune because of which a novel algorithm called Triangular Targeting Algorithm (TTA) has been proposed in this thesis. It combines with a Proximity Positioning Algorithm (PPA) to successfully steer a robot behind an object aligned with the final orientation. This is very useful and essential in many collaborative tasks. The algorithm works by defining an intermediate 'virtual' target position for the robot to go to in its quest to reach the destination. The intermediate position is such that it assists the robot to reach the target with the desired final angle and is continually updated as the robot moves through its work space. The selection of the virtual target position is influenced by several parameters such as the path curvature, robot orientation, distance of the robot to the final position and the robot-to-target angle.

An important and integral component of the position control methodology is the velocity profiling which has been incorporated to ensure rapid and accurate response to robot positioning. The velocity profiling has three components- *angular*, *linear* and *cross-effect* profiling. The effect of each component has firstly been evaluated separately and then the combined effect on the trajectory of a robot. The robot successfully traverses an accurate path to reach the object and ends up aligned with the desired final angle. The performance matrices show that the traversed path is repeatable and the deviation from the intended path and the variation in the final robot posture (position and orientation) are negligible.

Man-Machine Collaboration. In future, many tasks will be done by robots and human beings while collaborating with each other. Teleoperated robotic manipulators will assist human beings in many applications such as assisted surgery, object manipulation in hazardous conditions, shop-floor automation etc. Future work involves design and

development of a low cost anthropomorphic robotic arm which will be operated using a master-slave controller. For dexterous manipulation of objects, the robotic arm must be equipped with some form of force feedback or haptic sensing, the importance of which can be seen from the research presented in [86]. Several ways of force measurement, such as using force sensors and measuring motor current, will have to be explored. While force sensors may be suitable for the gripper, the force on other joints may be accurately measured using motor currents. For teleoperation of the robotic arm, a design using radio frequency wireless communication will need to be implemented and tested.

The thesis has laid the foundation for future research in the following areas-

- *Adaptive colour thresholding* for robust vision processing.
- Implementations of the proposed vision processing techniques, such as the new colour space and discrete Y, U and V look-up-tables on FPGA based hardware for local vision systems of autonomous robots.
- Stereo vision systems for depth perception in humanoid robots.
- Implementation of Triangular Targeting Algorithm for different types of robots (bi-wheeled, traction type locomotion, omni-wheeled robot etc).
- Development of anthropomorphic robotic arms with haptic sensing.
- *Web enabled robots* for collaborative tasks in a home (for eldercare), office or shop floor.

8 References

[01] K.H. Choi, S.C. Kim, S.H. Yook, "Multi-agent hybrid shop floor control system", International Journal of Production Research, Vol. 38, Issue 17, November 2000, pp. 4193 - 4203

[02] A. Wagner, R. Arkin, "Multi-robot communication-sensitive reconnaissance", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '04), Vol. 5, 2004, pp. 4674 - 4681

[03] M.A. Diftler, R.O. Ambrose, K.S. Tyree, S.M. Goza, E.L. Huber, "A mobile autonomous humanoid assistant", 4th IEEE/RAS International Conference on Humanoid Robots, Vol. 1, Nov. 2004, pp. 133 - 148

[04] M. Fujita, "Digital creatures for future entertainment robotics", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '00), Vol. 1, April 2000, San Francisco, CA, USA, ISBN: 0-7803-5886-4, pp. 801-806

[05] M. Uchiyama, "Multirobots and Cooperative Systems", In B. Siciliano, M. Thoma, K.P. Valavanis (Eds.), Control Problems in Robotics and Automation, Series: Lecture Notes in Control and Information Sciences, Vol. 230, Springer-Verlag, January 1998, pp.19-34

[06] C.H. Messom, "Robot Soccer – Sensing, Planning, Strategy and Control, a distributed real time intelligent system approach", AROB, Oita, Japan, 1998, pp. 422 – 426.

[07] S. Piao, B. Hong, "Robot path planning using genetic algorithms", Journal of Harbin Institute of Technology, Vol. 8, No. 3, September 2001, pp. 215 – 217

[08] Q. Meng, B. Yin, X. Jianshe, "Intelligent learning technique based on fuzzy logic for multi-robot path planning", Journal of Harbin Institute of Technology, Vol. 8, No. 3, September 2001, pp. 222 – 227

[09] A. Konno, R. Uchikura T. Ishihara, T. Tsujita, T. Sugimura, J. Deguchi, M. Koyanagi, M. Uchiyama, "Development of a High Speed Vision System for Mobile Robots", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 1372 - 1377

[10] C.H. Messom, S. Demidenko, K. Subramaniam, G. Sen Gupta, "Size/Position Identification in Real-Time Image processing using Run Length Encoding", Proceedings of the IEEE Instrumentation and Measurement Technology Conference (IMTC 2002), Alaska, USA, 2002, pp 1055-1059

[11] D. Hujic et el, "The Robotic Interception of Moving Objects in Industrial Settings: Strategy Development and Experiment", IEEE/ASME Transactions on Mechatronics, Vol. 3, No. 3, September 1998, pp. 225 – 239

[12] E. Ivanjko, M. Vasak, I. Petrovic, "Kalman filter theory based mobile robot pose tracking using occupancy grid maps", International Conference on Control and Automation (ICCA '05), Vol. 2, June 2005, pp. 869 - 874

[13] R.E. Kalman, “A new approach to linear filtering and prediction problems”, Trans. ASME J. Basic Eng., Vol. 82D, 1960, pp. 35 – 45

[14] G. Sen Gupta, C.H. Messom, H.L. Sng, “State Transition Based Supervisory Control for a Robot Soccer System”, Proceedings of the 1st IEEE International Workshop on Electronic Design Test and Applications (DELTA 2002), January 2002, Christchurch, New Zealand, pp. 338-342

[15] Y.U. Cao, A.S. Fukunaga, A.B. Kahng, “Cooperative Mobile Robotics: Antecedents and Directions”, Autonomous Robots, Vol. 4, No. 1, 1997, pp.7-27

[16] D. Barnes, J. Gray, “Behaviour synthesis for cooperant mobile robot control”, International Conference on Control, 1991, pp. 1135–1140

[17] M. Mataric, “Interaction and Intelligent Behaviour”, PhD thesis, MIT, EECS, May 1994.

[18] S. Premvuti, S. Yuta, “Consideration on the cooperation of multiple autonomous mobile robots”, IEEE/RSJ IROS, 1990, pp. 59–63

[19] H. R. Everett, D. W. Gage, “From Laboratory to Warehouse: Security Robots Meet the Real World”, The International Journal of Robotics Research, Vol. 18, No. 7, 1999, pp. 760-768

[20] P. Santana, J. Barata, H. Cruz, A. Mestre, J. Lisboa, L. Flores, “A multi-robot system for landmine detection”, 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005), Sept. 2005, Vol. 1, ISBN: 0-7803-9401-1, pp. 721-728

[21] R. Cassinis, “Landmine detection methods using swarms of simple robots”, In E. Pagello et al. (Eds.) Intelligent Autonomous Systems, IOS Press, 2000, pp. 212-218

[22] R.G. Simmons, D. Apfelbaum, W. Burgard, D. Fox, M. Moors, S. Thrun, H.L.S. Younes, “Coordination for multi-robot exploration and mapping”, Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Application of Artificial Intelligence, 2000, ISBN: 0-262-51112-6, pp. 852-858

[23] L.E. Parker, “ALLIANCE: an architecture for fault tolerant multirobot cooperation”, IEEE Transactions on Robotics and Automation, Apr 1998, Vol. 14, Issue: 2, ISSN: 1042-296X, pp. 220-240

[24] B. Astrand, A-J Baerveldt, “An agricultural Mobile Robot with Vision-Based perception for Mechanical Weed Control”, Autonomous Robots, Vol. 13, No. 1, 2002, Springer Netherlands, pp. 21-35

[25] J.S. Jennings, G. Whelan, W.F. Evans, “Cooperative search and rescue with a team of mobile robots”, Proceedings of the 8th International Conference on Advanced Robotics (ICAR '97), Monterey, CA, USA, ISBN: 0-7803-4160-0, pp. 193-200

[26] K. Sugawara, “Collective Motions and Formations of multi-robots based on simple dynamics”, IEEE/ICME International Conference on Complex Medical Engineering, May 2007, ISBN: 978-1-4244-1078-1, pp. 122-125

[27] T.C. Lueth, T. Laengle, “Task description, decomposition and allocation in a distributed autonomous multi-agent robot system”, IEEE/RSJ IROS, 1994, pp. 1516–1523

[28] A. Bonarini, P. Aliverti, M. Lucioni, “An omni-directional vision sensor for fast tracking for mobile robots”, IEEE Transactions on Instrumentation and Measurement, Jun 2000, Vol. 49, Issue: 3, ISSN: 0018-9456, pp. 509-512

[29] B.K.P. Horn, Robot Vision (MIT Electrical Engineering and Computer Science), MIT Press, 1986.

[30] M. Jamzad, B.S. Sadjad, V.S. Mirrokni, M. Kazemi, H. Chitsaz, A. Heydarnoori, M.T. Hajiaghai, E. Chiniforoosh, “A Fast Vision System for Middle Size Robots in RoboCup”, A. Birk, S. Coradeschi, S. Tadokoro (Eds.): RoboCup 2001: Robot Soccer World Cup V, Spinger Verlag, 2002, pp. 71 – 80

[31] J. Bruce, T. Balch, M. Veloso, “Fast and Inexpensive Color Image Segmentation for Interactive Robots”, IROS 2000, San Francisco, 2000, pp. 2061 – 2066

[32] F. Ercal, M. Allen, and F. Hao, "A Systolic Image Difference Algorithm for RLE-Compressed Images”, IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 5, May 2000, pp. 433 – 443

[33] Q. Chen, H. Xie, P. Woo, “Vision-based fast object recognition and distances calculation of robots”, 31st Annual Conference of IEEE Industrial Electronics Society, 2005, ISBN: 0-7803-9252-3, pp. 363-368

[34] J. Baltes, “Practical camera and colour calibration for large rooms”, in M. Veloso, E. Pagello, H. Kitano (Eds.), RoboCup-99: Robot Soccer World Cup III, New York, 2000, Springer, pp. 148 – 161

[35] M. Ramesh Jain, R. Kasturi, B.G. Schunck, Machine Vision, McGraw-Hill International Editions, Computer Science Series, 1995

[36] S. Park, J.K. Aggarwal, “Segmentation and tracking of interacting human body parts under occlusion and shadowing”, Proceedings of the workshop on Motion and Video Computing, Dec. 2002, ISBN: 0-7695-1860-5, pp. 105-111

[37] C. Garcia, X. Apostolidis, “Text detection and segmentation in complex color images”, Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 00), Istanbul, Turkey, 2000, ISBN: 0-7803-6293-4, pp. 2326-2329

[38] M. Shand, "Flexible image acquisition using reconfigurable hardware", IEEE Symposium on FPGAs for Custom Computing Machines (FCCM '95), 1995, pp. 125-134

[39] C.H. Messom, G. Sen Gupta and H.L. Sng, “Distributed Real-time Image Processing for a Dual Camera System”, CIRAS 2001, Singapore, 2001, pp. 53-59

[40] A. Chakravarthy, D. Ghose, "Obstacle avoidance in a dynamic environment: a collision cone approach", IEEE Transactions on Systems, Man and Cybernetics, Part A, Vol. 28, Issue: 5, Sep 1998, ISSN: 1083-4427, pp. 562-574

[41] M. Hariti, Y. Ruichek, A. Koukam, "A fast stereo matching method for real time vehicle front perception with linear cameras", IEEE Proceedings Intelligent Vehicles Symposium, 2003, pp. 247-252

[42] R.N. Chiou, C.H. Chen, K.C. Hung, J.Y. Lee, "The optimal camera geometry and performance analysis of a trinocular vision system", IEEE Transactions on Systems, Man and Cybernetics, 25:8, 1995, pp. 1207-1220

[43] M.A.H. Venter, J.J.D. van Schalkwyk, "Stereo imaging in low bitrate video coding", Southern African Conference Proceedings on Communications and Signal Processing. (COMSIG), 1989, pp. 115-118

[44] J. Fabrizio, J.P. Tare, R. Benosman, "Calibration of Panoramic Catadioptric Sensors Made Easier", Proceedings of Third Workshop on Vision, 2002, pp. 45-52

[45] F. Huang, S.K. Wei, R. Klette, G. Gimelfarb, "Cylindrical Panoramic Cameras – From Basic Design to Applications", Proceedings of Image and Vision Computing New Zealand conference (IVCNZ 2002), 2002, pp. 101-106

[46] J. C. Russ, "The Image Processing Handbook", Fifth Edition, Published by CRC Press, 2006, ISBN: 0849372542, (Section 7: Stereology, Computer Vision, Stereoscopy), 817 pages

[47] R. Swaminathan, M.D. Grossberg, S.K. Nayar, "Caustics of catadioptric cameras", Proceedings of the Eighth IEEE International Conference on Computer Vision, (ICCV 2001), 2001, Vol. 2, pp. 2-9

[48] S. Derrien, K. Konolige, "Approximating a Single Viewpoint in Panoramic Imaging Devices", Proceedings of the IEEE International Conference on Robotics and Automation, 2000, pp. 3931-3938

[49] S.K. Nayar, "Catadioptric omnidirectional camera", Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1997, pp. 482-488

[50] M. Fiala, A. Basu, "Feature extraction and calibration for stereo reconstruction using non-SVP optics in a panoramic stereo-vision sensor", Proceedings of the Third Workshop on Omnidirectional Vision, 2002, pp. 79-86

[51] J. Gluckman, S.K. Nayar, "Rectified Catadioptric Stereo Sensors", IEEE Transactions on Pattern Analysis and Machine Intelligence, 24:2, 2002, pp. 224-236

[52] J. Gluckman, S.K. Nayar, "Planar catadioptric stereo: geometry and calibration" IEEE Computer Society Conference on Computer Vision and Pattern Recognition. Vol. 1, 1999, pp. 22-28

[53] J. Gluckman, S.K. Nayar, “Catadioptric stereo using planar mirrors”, International Journal of Computer Vision, Springer Netherlands, Vol. 44, No. 1, August 2001, pp. 65-79

[54] D.G. Bailey, “Sub-pixel estimation of local extrema”, Proceeding of Image and Vision Computing New Zealand, 2003, pp. 414-419

[55] K. Konolige, K. Myers, ”The saphira architecture for autonomous mobile robots”, In David Kortenkamp, R. Peter Bonasso, Robin Murphy (Eds.), Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems, Chapter 9, MIT Press, Cambridge, MA, 1998, pp. 211–242

[56] M. Mataric, “Minimizing complexity in controlling a mobile robot population,” Proceedings of IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 830-835

[57] P. Riley, P. Stone, M. Veloso, "Layered Disclosure: Revealing Agents' Internals", Seventh International Workshop on Agent Theories, Architectures and Languages, 2000, pp. 344-351

[58] L. Parker L, “Current state of the art in distributed robot systems”, In L. Parker, G. Bekey, J. Barhen (Eds.), Distributed Autonomous Robotic Systems 4, 2000, Springer, pp. 3-12

[59] P. Thomas, G. Springer, R.J. Stonier, P. Wolfs, "Boundary-avoidance and attack strategy in robot soccer", Proceedings of the FIRA Robot World Cup France '98, pp. 27-32

[60] M. Butler, M. Prokopenko, T. Howard, “Flexible Synchronisation within RoboCup Environment: A Comparative Analysis”, RoboCup 2000: Robot Soccer World Cup IV, 2000, pp. 119-128

[61] J. Minguez, L. Montano., “Nearness Diagram Navigation (ND): Collision Avoidance in Troublesome Scenarios”, IEEE Transactions on Robotics and Automation, Vol. 20, No. 1, Feb 2004, pp. 45-59

[62] J. Baltes, N. Hildreth, “Adaptive Path Planner for Highly Dynamic Environments”, In P. Stone, T. Balch, G. Kraetzschmar (Eds.): RoboCup 2000: Robot Soccer World Cup IV, Spinger Verlag, 2001, pp. 76 - 85

[63] H.H. Shahri, “Towards Autonomous Decision Making in Multi-agent Environments Using Fuzzy Logic”, Lecture Notes in Artificial Intelligence, Vol. 2691, Springer Verlag, 2003, pp. 247 – 257

[64] Z. Su, B. Zeng, G. Liu, F. Ye, M. Xu, “Application of Fuzzy Neural Network in Parameter Optimization of Mobile Robot Path Planning Using Potential Field”, IEEE International Symposium on Industrial Electronics (ISIE 2007), June 2007, pp. 2125 - 2128

[65] S.J. Johansson, A. Saffiotti, “Using the Electric Field Approach in the RoboCup Domain”, Lecture Notes in Artificial Intelligence, Vol. 2377, Springer Verlag, 2002, pp. 399 - 404

[66] P. Jonker, J. Caarls, W. Bokhove, “Fast and Accurate Robot Vision for Vision Based Motion”, In P. Stone, T. Balch, G. Kraetzschmar (Eds.): RoboCup 2000: Robot Soccer World Cup IV, Spinger Verlag, 2001, pp. 149 -158

[67] M. Simon, S. Behnke, R. Rojas, “Robust Real Time Color Tracking”, In P. Stone, T. Balch, G. Kraetzschmar (Eds.): RoboCup 2000: Robot Soccer World Cup IV, Spinger Verlag, 2001, pp. 239 – 248

[68] F. Stolzenburg, O. Obst, J. Murray, “Qualitative Velocity and Ball Interception”, Lecture Notes in Artificial Intelligence, Vol. 2479, Springer Verlag, 2002, pp. 283 – 298

[69] J. Gutmann, T. Weigel, B. Nebel, “Fast, Accurate, and Robust Self-Localization in the RoboCup Environment”, In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano (Eds.): RoboCup-99, Robot Soccer World Cup III, Springer Verlag, 2000, pp. 304 – 317

[70] H.M. Barberá, A.G. Skarmeta, "A framework for defining and learning fuzzy behaviours for autonomous mobile robots", International Journal of Intelligent Systems, Wiley InterScience, Vol. 17, Issue 1, 2002, pp. 1-20

[71] N. Vlassis, B. Terwijn, B. Kr, “Auxiliary particle filter robot localization from high-dimensional sensor observations", Proceedings of the IEEE International Conference on Robotics and Automation, Washington D.C., 2002, pp. 7-12

[72] D. Schulz, W. Burgard, D. Fox, A. B. Cremers, “Tracking Multiple Moving Objects with a Mobile Robot”, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), 2001, Vol. 1, pp. 371-377

[73] S.I. Roumeliotis, G.A. Bekey, “Collective Localization: A distributed Kalman filter approach to localization of groups of mobile robots”, Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, California, 2000, pp. 2958-2965

[74] Y.X. Su, C.H. Zheng, Dong Sun, B.Y. Duan, “A simple nonlinear velocity estimator for high-performance motion control”, IEEE Transactions on Industrial Electronics, Vol. 52, No. 4, 2005, pp. 1161-1169

[75] K. Ogata, “Modern Control Engineering”, 4th Edition, Prentice Hall, 2003.

[76] L. Huang, W. Yu, S. K. Jhajharia, “Speed Control of Differentially Driven Wheeled Mobile Robots – Tracking and Synchronization”, Proceedings of the IEEE Instrumentation and Measurement Technology Conference (IMTC 2003), Vail, USA, 2003, pp. 1407 – 1412

[77] G. Sen Gupta, C.H. Messom, S. Demidenko, "Global Vision Based Optimization of Control Functions and Kalman Filtering for Robotic Applications", Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS 2003, Singapore, ISSN: 0219-6131

[78] G. Sen Gupta, C. H. Messom, S. Demidenko, "Vision Assisted Measurement for Optimization of Robot Motion and Position Control Functions", Proceedings of

the IEEE Instrumentation and Measurement Technology Conference, IMTC 2004, Como, Italy, ISBN: 0-7803-8249-8, pp. 297-302

[79] W.L. Xu, S.K. Tso, Y.H. Fung, "Fuzzy reactive control of a mobile robot incorporating a real/virtual target switching strategy", Robotics and Autonomous Systems, Vol. 23, No. 3, 1998 , pp. 171-186

[80] A. C. Shultz, J.J. Grefenstette, W. Adams, "Robo-shepherd: Learning complex robotic behaviours", In M. Jamshidi, F. Pin, P. Dauchez (Eds.), Robotics and manufacturing: Recent Trends in Research and Applications, Vol. 6, ASMEPress, 1996, pp. 763-768

[81] M. Mataric, D. Cliff, "Challenges in evolving controllers for physical robots", Robotics and Autonomous Systems, Vol. 19, 1996, pp. 67-83

[82] L. A. Meeden, D. Kumar, "Trends in evolutionary robotics", In L. C. Jain, T. Fukuda (Eds.), Soft Computing for Intelligent Robotic Systems, Physica-Verlag, New York, NY, 1998, pp. 215-233

[83] M. Potter, L. A. Meeden, A. Schultz, "Heterogeneity in the coevolved behaviours of mobile robots: The emergence of specialists", Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, USA, 2001, pp. 1337-1343

[84] P. Stone, M. VELOSO, "Towards collaborative and adversarial learning: a case study in robotic soccer", International Journal of Human-Computer Studies, Vol. 48, No. 1, January 1998 , pp. 83-104

[85] M. Balch, "Complete Digital Design: A Comprehensive Guide to Digital Electronics", Chapter 10: Logic Design and Finite State Machines, MacGraw Hill Professional, 2003, pp.221 – 247

[86] P. Payeur, C. Pasca, A. M. Cretu, E. M. Petriu, "Intelligent haptic sensor system for robotic manipulation**",** IEEE Transactions on Instrumentation and Measurement, Vol. 54, Issue 4, Aug. 2005, pp. 1583 – 1592

9 List of Selected Relevant Publications by the Author

Book Chapters

[01] **Gourab Sen Gupta**, Chris Messom and Serge Demidenko, "***Triangular Targeting Algorithm (TTA) for Accurate Motion Control of Fast Mobile Robots***", In Liu, D.; Wang, L.; Tan, K.C. (Eds.), Design and Control of Intelligent Robotic Systems, Series: Studies in Computational Intelligence, Springer-Verlag, 2008 (to appear)

[02] **Gourab Sen Gupta**, Donald Bailey, "***Fast Image Capture and Vision Processing for Robotic Applications***", In Mukhopadhyay, S.C.; Huang, R.Y.M. (Eds.), Sensors - Advancements in Modeling, Design Issues, Fabrication and Practical Applications, Series: Lecture Notes in Electrical Engineering, Vol. 21, ISBN: 978-3-540-69030-6, Springer-Verlag, 2008, pp. 329-354

Journal and refereed conference papers

[03] **G. Sen Gupta**, C .H. Messom and S. Demidenko, "***Real-Time Identification and Predictive Control of Fast Mobile Robots using Global Vision Sensing***", IEEE Transactions on Instrumentation and Measurement, Vol. 54, No. 1, February 2005, ISSN 0018-9456, pp. 200-214

[04] **G. Sen Gupta**, S.C. Mukhopadhyay, S. Demidenko and C.H. Messom, "***Master-slave Control of a Teleoperated Anthropomorphic Robotic Arm with Gripping Force Sensing***", IEEE Transactions on Instrumentation and Measurement, Vol. 55, No. 6, December 2006, pp. 2136-2145

[05] **G. Sen Gupta** and S. C. Mukhopadhyay, "***A Triangular Targeting Algorithm (TTA) for Motion Control of Wheeled Mobile Robots***", Proceedings of the International Conference on Emerging Mechanical Technology – Macro to Nano (EMTM2N 2007), February 16-18, 2007 at Pilani, India, pp. 204-209.

[06] **Gourab Sen Gupta**, R. Paddison, C.H. Messom, S. Demidenko, "***Wireless Master-Slave Embedded Controller for a Teleoperated Anthropomorphic Robotic Arm with Gripping Force Sensing***", Proceedings of the IEEE International Conference on Systems, ICONS'06, April 27-29, 2006, Mauritius, ISBN 0-7695-2552-0, Library of Congress Number 2006920236.

[07] **Gourab Sen Gupta**, Jay Kidd, Chris Messom and Serge Demidenko, "***Advanced Motion Control Algorithm for Fast Mobile Robots***", Proceedings of the 3rd International Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS 2005, Singapore, Dec 14-16, 2005, ISBN: 981-05-4674-2.

[08] **G. Sen Gupta**, S.C. Mukhopadhyay, C. H. Messom and S. Demidenko, "***Master-Slave Control of a Teleoperated Anthropomorphic Robotic Arm with Gripping Force Sensing***", Proceedings of the IEEE Instrumentation and Measurement Technology Conference, IMTC 05, May 17-19, 2005, Ottawa, Canada, ISBN 0-7803-8880-1, pp. 2203-2208.

[09] **G. Sen Gupta**, C. H. Messom and S. Demidenko, "***Vision Assisted Measurement for Optimization of Robot Motion and Position Control Functions***", Proceedings of 21st IEEE Instrumentation and Measurement Technology Conference, IMTC 2004, Como, Italy, May 18-20, 2004, ISBN: 0-7803-8249-8, pp. 297-302

[10] **G. Sen Gupta** and D. G. Bailey, "***A New Colour Space for Efficient and Robust Segmentation***", Proceedings of Image and Vision Computing New Zealand,, IVCNZ 2004, Akaroa, New Zealand, Nov 21-23, 2004, ISBN 0-478-09366-7, pp. 315-320

[11] **G. Sen Gupta**, C. H. Messom and S. Demidenko, "***State Transition Based (STB) Role Assignment and Behaviour Programming in Collaborative Robotics***", Proceedings of the 2nd International Conference on Autonomous Robots and Agents, ICARA 2004, Palmerston North, New Zealand, Dec 13-15, 2004, ISBN 0-476-00994-4, pp. 385-390

[12] **G. Sen Gupta**, Lim Y.S., Messom C. H., S. Demidenko, S. C. Mukhopadhyay, and D. N. Pinder, "***Robot Soccer: Integrated Framework for Multidisciplinary Hi-Tech Education***" Proc. of 6th international conference on Computer Based Learning In Sciences, CBLIS, July 6-10, 2003, Cyprus, pp. 322-333.

[13] **G. Sen Gupta**, C. H. Messom, S. Demidenko and Lim Yuen Siong, "***Identification and Prediction of a Moving Object Using Real-Time Global Vision Sensing***", Proceedings of 20th IEEE Instrumentation and Measurement Technology Conference, IMTC 2003, Vail, USA, May 20-22, 2003, ISBN: 0-7803-7706-0, pp 1402-1406

[14] **G. Sen Gupta**, Chris Messom and Serge Demidenko, "***Global Vision Based Optimization of Control Functions and Kalman Filtering for Robotic Applications***", Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS 2003, Dec 15-18, 2003, Singapore, ISSN: 0219-6131.

[15] **G. Sen Gupta**, C.H. Messom and Sng H.L, "***State Transition Based Supervisory Control for a Robot Soccer System***", DELTA2002 (IEEE International Workshop on Electronic Design, Test & Applications, 2002), Christchurch, New Zealand, Jan 29-31, 2002, ISBN 0-7695-1453-7, pp. 338 – 342

[16] **Gourab Sen Gupta**, Toh Ser Khoon, Sng Hong Lian and Chris Messom, "***State Transition Based Control of Synchronized Movements of a Contingent of Marching Robots***", Proceedings of the FIRA Robot World Congress 2002, Seoul, Korea, pp. 63 - 67.

[17] C.H. Messom, **G. Sen Gupta** and S. Demidenko, "***Hough Transform Run Length Encoding in Real-Time Image Processing***", Proceedings of the IEEE Instrumentation and Measurement Technology Conference, IMTC 05, May 17-19, 2005, Ottawa, Canada, ISBN 0-7803-8880-1, pp. 2198-2202.

[18] R. A. A. Paddison and **G. Sen Gupta**, "***The master-Slave Wireless Control of an Anthropomorphic Robotic Arm with Force Feedback***", Electronics New Zealand, ENZCon'05, Auckland, New Zealand, Nov 14-15, 2005, ISBN 0-473-10634-5, pp. 117-122

[19] Jonathan Seal, **Gourab Sen Gupta**, Geoffrey Barnes and Chris Messom, "***Post-Match Analysis Program for Robot Soccer***", Electronics New Zealand, ENZCon'04, Palmerston North, New Zealand, Nov 15-16, 2004, ISBN 0-476-01106-X, pp. 39-43

[20] Tim Brooks, **Gourab Sen Gupta** and Peter Xu, "***Teleoperated Anthropomorphic Robotic Arm with Force Feedback***", Electronics New Zealand, ENZCon'04, Palmerston North, New Zealand, Nov 15-16, 2004, ISBN 0-476-01106-X, pp. 259-264

[21] D. G. Bailey and **G. Sen Gupta**, "***Error estimation of Robot Soccer Imaging System***", Proceedings of Image and Vision Computing New Zealand, IVCNZ 2004, Akaroa, New Zealand, Nov 21-23, 2004, ISBN 0-478-09366-7, pp. 119-124

[22] C. H. Messom, **G. Sen Gupta**, S. Demidenko and Lim Yuen Siong, "***Improving Predictive Control of a Mobile Robot: Application of Image Processing and Kalman Filtering***", Proceedings of 20th IEEE Instrumentation and Measurement Technology Conference, IMTC 2003, Vail, USA, May 20-22, 2003, ISBN: 0-7803-7706-0, pp 1492-1496

[23] Toh Ser Khoon, **G. Sen Gupta**, Chris Messom, Serge Demidenko and Robert Craig, "***A contingent of autonomous marching robots: Intricacies of system design and motion control***", Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2003 , Sharjah, UAE, Dec 14-17, 2003, ISBN: 0-7803-8164-5, pp. 396-399

[24] Sng Hong Lian, **Gourab Sen Gupta** and Christopher H. Messom, "***Strategy for Collaboration in Robot Soccer***", DELTA 2002 (International Workshop on Electronic Design, Test & Applications, 2002), Christchurch, New Zealand, Jan 29-31, 2002, ISBN 0-7695-1453-7, pp. 347 – 351

[25] C.H. Messom, **G. Sen Gupta**, and Sng H.L., "***Distributed Real-Time Image Processing for a Dual-Camera System***", CIRAS 2001 (Computational Intelligence, Robotics and Autonomous Systems, 2001), Singapore, Nov 28-30, 2001, ISSN: 0219-6131, ISSN 0219-6131, pp. 53 – 59

Appendix A

Listing of OOP implementation of STBC-based robot behaviour (goalkeeper)

```
class CGoalieAction //----- To defend the goal from attack
{
//  Has four states S0, S1, S2, S3
//      S0      initialisation
//      S1      sweep ball
//      S2      track ball y and predict shot
//      S3      centre in goal
//      S4      Return back avoiding ball after sweeping

private:
        float MIDDLE;
        float NEARPOS;
        float XHYSTERESIS;
        float YHYSTERESIS;
        float HALF_GOALSIZE;
        float CLEARYOFFSET;
        int *pstate;
        int which; //---- which robot we're referring to
        float GOALSIZE;
        float GOALIESTANDX;
        float GoalieDistanceError;
        int *pcurrentSubStateG;
        //---- Set AVOID to true to enable Avoid state S4
        BOOL AVOID;
        BOOL ROBOTtooNEAR;
        floatPOINT nearRobotPos;
        floatPOINT BallPos;
        float Ballspeed;
        floatPOINT finalPos;
        floatPOINT GPos;
        float GAngle;
        BOOL BALLinFrontofGOAL;
        BOOL clearBall;
        BOOL BallBesideGoalie;
public:
        CGoalieAction(int w, int *ps)
        {
                floatPOINT nearpoint;
                floatPOINT homegoalbottom;
                ROBOTtooNEAR = FALSE;
                GOALSIZE = 40;
                homegoalbottom.x = (float)globaldata.homegoalbottom.x;
                homegoalbottom.y = (float)globaldata.homegoalbottom.y;
                pstate = ps ;
                which = w;
                mapxy(&homegoalbottom,&nearpoint,&globaldata);
                XHYSTERESIS = 3.0;
                YHYSTERESIS = 2.0;
                NEARPOS = nearpoint.x + XHYSTERESIS;
                HALF_GOALSIZE = (float)(GOALSIZE/2.0);
                MIDDLE = Physical_Xby2 + 30;
                float Balldx=globaldata.ballvelS.x;
                float Balldy=globaldata.ballvelS.y;

                BallPos = globaldata.GballposS;
                Ballspeed = (float)sqrt(Balldx*Balldx + Balldy*Balldy);

                switch (gCGoalieActionParameters.GBehaviour)
                {
                case 0://---- Conservative
                        CLEARYOFFSET = (float)fabs(Physical_Yby2 - nearpoint.y);
                        break;
                case 1://---- Normal
                        CLEARYOFFSET = (float)fabs(Physical_Yby2 - nearpoint.y/2);
                        break;
                case 2://---- Aggressive
```

```
        CLEARYOFFSET = Physical_Yby2;
        break;
}
GAngle = globaldata.goalieangleS;

int BALLclearCount = (int) (60*gBallClearTime);
GPos = globaldata.goalieposS;
if ( (BallPos.y <= GPos.y + ROBOTWIDTHby2) &&
         (BallPos.y >= GPos.y - ROBOTWIDTHby2) &&
         (BallPos.x >= GPos.x + ROBOTWIDTHby2)
        )
        BallBesideGoalie = TRUE;
else
        BallBesideGoalie = FALSE;

if ( BallBesideGoalie )
        BALLclearCount = 30;   //-- clear in 0.5 sec

float BALLspeedBound = (float)1.5; //-- 1.5 cms/frame = approx 0.9 m/s
static int slowBallcount = 0;
if (Ballspeed < BALLspeedBound && ownArea(BallPos))
        slowBallcount++;
else
        slowBallcount=0;
if (slowBallcount > BALLclearCount)
        clearBall = TRUE;
else
        clearBall = FALSE;

AVOID = FALSE;//---- false to switch off state S4
pcurrentSubStateG = &(globaldata.GDefenderSubState);

GOALIESTANDX = (NEARPOS-XHYSTERESIS)/2 -
                        gCGoalieActionParameters.GXcompensation;

finalPos.x = GOALIESTANDX;
finalPos.y = Physical_Yby2 + (BallPos.y-Physical_Yby2)*(float)0.5;
//---- Limit tracking y to penalty area boundary
if (finalPos.y > Physical_Yby2 + HALF_GOALSIZE)
        finalPos.y = Physical_Yby2 + HALF_GOALSIZE - 2;
else if (finalPos.y < Physical_Yby2 - HALF_GOALSIZE)
        finalPos.y = Physical_Yby2 - HALF_GOALSIZE + 2;

//---- safe condition calculation
float Gdx = finalPos.x - GPos.x;
float Gdy =    finalPos.y - GPos.y;
GoalieDistanceError = (float)sqrt(Gdx*Gdx+Gdy*Gdy);

//---- Check distance to other robots
//---- if less than 10cm then avoid that robot
floatPOINT     R1Pos = globaldata.robot1posS;
floatPOINT     R2Pos = globaldata.robot2posS;

float GR1dx = R1Pos.x - GPos.x;
float GR1dy = R1Pos.y - GPos.y;
float GR1Distance = (float)sqrt(GR1dx*GR1dx+GR1dy*GR1dy);

float GR2dx = R2Pos.x - GPos.x;
float GR2dy = R2Pos.y - GPos.y;
float GR2Distance = (float)sqrt(GR2dx*GR2dx+GR2dy*GR2dy);

float nearRobotDistance;

if (GR2Distance < GR1Distance)
{
        nearRobotDistance = GR2Distance;
        nearRobotPos = R2Pos;
}
else
{
        nearRobotDistance = GR1Distance;
        nearRobotPos = R1Pos;
}


if (nearRobotDistance < 10)
        ROBOTtooNEAR = TRUE;
```

```
                else
                        ROBOTtooNEAR = FALSE;

                BALLinFrontofGOAL = (BallPos.y < Physical_Yby2 + HALF_GOALSIZE - 2)
                                && (BallPos.y > Physical_Yby2 - HALF_GOALSIZE + 2);
        }
        ~CGoalieAction(){}

//--  Switching Conditions/ State Transitions -------------------------------------
//      S0S1, S0S2, S0S3, S0S4
//      S1S2, S1S4
//      S2S1, S2S3
//      S3S2
//      S4S2


        BOOL S0S1()
        { //---- ball in near
                if (clearBall && (BallPos.x <= NEARPOS)
                        && (BallPos.y < (Physical_Yby2 + CLEARYOFFSET))
                        && (BallPos.y > (Physical_Yby2 - CLEARYOFFSET)) )
                //-- ball in near
                        return true;
                else
                        return false;
        }

        BOOL S0S2()
        {//---- ball in middle
                if (BallPos.x > NEARPOS && BallPos.x < MIDDLE
                         ||( BallPos.x < NEARPOS &&
                         ((BallPos.y > Physical_Yby2 + CLEARYOFFSET)
                         || (BallPos.y < Physical_Yby2 - CLEARYOFFSET)) ) )
                //-- ball in middle
                        return true;
                else
                        return false;
        }       //-- S0S2()

        BOOL S0S3()
        {//---- ball in far
                if (BallPos.x >= MIDDLE)//-- ball moves from far to middle
                        return true;
                else
                        return false;
        }//-- S0S3()

        BOOL S0S4()
        {
                return false; //-- never start in this condition
        }//-- S0S4()

        BOOL S1S2()
        {//-- ball moves from near to middle
                if (BallPos.x > NEARPOS + XHYSTERESIS ||
                  (BallPos.x < NEARPOS + XHYSTERESIS &&(
                  (BallPos.y > Physical_Yby2 + CLEARYOFFSET + YHYSTERESIS)
                    || (BallPos.y < Physical_Yby2 - CLEARYOFFSET - YHYSTERESIS)) ))
                //-- ball moves from near to middle
                        return true;
                else
                        return false;
        }//-- S1S2()

        BOOL S1S4()//-- just finished clearing
        {
                BOOL DANGEROUS=(BALLinFrontofGOAL && GPos.x > BallPos.x + ROBOTWIDTH);
                return ( S1S2() || DANGEROUS );
                //-- same conditions as moving to middle
                //-- plus the ball is in a dangerous position

        }

        BOOL S2S1()
        {//---- ball moves from middle to near

                if ( clearBall && (BallPos.x < NEARPOS - XHYSTERESIS) &&
```

```
                (BallPos.y < (Physical_Yby2 + CLEARYOFFSET - YHYSTERESIS))
                && (BallPos.y > (Physical_Yby2 - CLEARYOFFSET + YHYSTERESIS)))
        //-- ball moves from middle to near
                return true;
        else
                return false;
}//--S2S1()

BOOL S2S3()
{//---- ball moves from middle to far

        if (BallPos.x >= MIDDLE + XHYSTERESIS)// ball moves from middle to far
                return true;
        else
                return false;
}//--S2S3()

BOOL S3S2()
{//---- ball moves from far to middle

        if (BallPos.x <= MIDDLE - XHYSTERESIS)// ball moves from far to middle
                return true;
        else
                return false;
}//--S3S2

BOOL S4S2()//-- safe condition
{
        if (GoalieDistanceError < 4)
                return true;
        else
                return false;
}

//-- Action States ------------------------------------------------------------
// S0 initialisation, void S0() {}; // No Action for Initialisation State

        void S1()       //--    S1 sweep/clear ball
        {
                float FINALVELOCITY = (float)0.88;

                if (BallBesideGoalie)
                {
                        FINALVELOCITY = (float)0.88;
                        positionG(BallPos, 90, FINALVELOCITY);
                        avoidBound(which,BallPos);
                        escapeGoal(which);
                }
                else    //-- ball is not beside the robot
                {
                        if (GAngle < -90)       //-- switch robot's front direction
                                GAngle += 180;
                        else
                        if (GAngle > 90)
                                GAngle -= 180;

                        //-- Action that is required
                        if ( (GAngle < 75 && GAngle > -75) ||
                                 (GAngle > 105 && GAngle < -105)
                           )
                        {
                                angleG(90);
                        }
                        else
                        {
                                positionG(BallPos, 90, FINALVELOCITY);
                                avoidBound(which,BallPos);
                                escapeGoal(which);
                        }
                }
        }

        void S2()       //-- S2  track ball y and predict shot
        {
                floatPOINT finalPos;
                float GOALIEyTRACKINGFACTOR;
                float VELOCITYFRAMECORRECTION;
```

```
floatPOINT Ballvel;
float ballYEstimate;
float m, C;

Ballvel = globaldata.ballvelS;

if (Ballspeed < 0.15)//-- ball is moving very slow or it is stationary
       m = 0;
else
       m = Ballvel.y/Ballvel.x;  //-- gradient of line of ball motion

C = BallPos.y - m * BallPos.x;
//-- calculation of constant in y1 = mx1 + C
//-- (x1, y1) is the ball position in this case
ballYEstimate = (float)(m*0.0 + C);
//-- estimating the position where ball crosses goal line
//-- in this case x1 is 0.0, the x coordinate of the goal line

if (gStartingPosition.StartPhase == PENALTY &&
        gStartingPosition.whosePossession == OPPONENT)
{
       if (gGoalieFollowClick)//-- goalie to follow the clicked point
       {
              floatPOINT clickPoint, clickPointS;

              clickPoint.x = (float)globaldata.capPoint.x;
              clickPoint.y = (float)globaldata.capPoint.y;

       //-- convert the screen coordinates to physical coordinates
       //-- physical coordinates are in the variable clickPointS
              mapxy(&clickPoint, &clickPointS, &globaldata);

              finalPos.y = clickPointS.y;
              //-- finalPos.x = GPos.x;
       }
       else    //-- goalie NOT to follow the clicked point
       {
              VELOCITYFRAMECORRECTION = 4;  // looking 4 frames ahead
              if (Ballspeed < 0.25)
                     finalPos.y = BallPos.y;
              else
                     finalPos.y = BallPos.y + Ballvel.y *
                                   VELOCITYFRAMECORRECTION;
        //-- track ball y position
       }
       //---- Limit tracking y to penalty area boundary
       float correction=(float)2.0;  //-- 2 cms
       if (finalPos.y > Physical_Yby2 + HALF_GOALSIZE)
              finalPos.y = Physical_Yby2 + HALF_GOALSIZE -
                                                  correction;
       else if (finalPos.y < Physical_Yby2 - HALF_GOALSIZE)
              finalPos.y = Physical_Yby2 - HALF_GOALSIZE +
                                                  correction;

       finalPos.x = GPos.x;
       //-- goalie must move straight maintaining X position
}
else    //-- it is not a penalty by opponent
{
       if ( (Ballvel.x < 0 && //-- ball travelling towards our goal
              //-- ball going to enter goal +/- 8 cms
              ballYEstimate > (Physical_Yby2 - HALF_GOALSIZE - 8) &&
              ballYEstimate < (Physical_Yby2 + HALF_GOALSIZE + 8))
         )
       {
              finalPos.y = (float) ( m*GOALIESTANDX + C );

       }

       else    //-- Ball is not estimated to enter goal
       {
              GOALIEyTRACKINGFACTOR = (float)0.4;
              if (BallPos.x < (NEARPOS+25))
                     GOALIEyTRACKINGFACTOR = (float)0.75;
              if (BallPos.x < (NEARPOS+15))
                     GOALIEyTRACKINGFACTOR = (float)1.0;
```

```
                        VELOCITYFRAMECORRECTION = 4;  // looking 4 frames ahead
                        if (Ballspeed < 0.25)
                                finalPos.y = BallPos.y;
                        else
                                finalPos.y = BallPos.y + Ballvel.y *
                                                        VELOCITYFRAMECORRECTION;
                         // track ball y position
                        finalPos.y = Physical_Yby2 + (finalPos.y -
                                        Physical_Yby2)*GOALIEyTRACKINGFACTOR;
                }

                //---- Limit tracking y to penalty area boundary
                float correction=(float)2.0;  //-- 2 cms
                if ( BallPos.x < NEARPOS) correction = -4;
                if ( BallPos.x < ROBOTWIDTH) correction = -7;
                if (finalPos.y > Physical_Yby2 + HALF_GOALSIZE)
                   finalPos.y = Physical_Yby2 + HALF_GOALSIZE - correction;
                else if (finalPos.y < Physical_Yby2 - HALF_GOALSIZE)
                   finalPos.y = Physical_Yby2 - HALF_GOALSIZE + correction;

                finalPos.x = GOALIESTANDX;
        }

        //---- Action that is required
        if(ROBOTtooNEAR)
        {
                float desiredAngle = 90.0;
                CAnyPositionAvoidObstacle move(which, pcurrentSubStateG,
                                        &finalPos, &nearRobotPos, &desiredAngle);
                move.runAction();
        }
        else
        {
                if(BallPos.x < Physical_Xby2)
                {
                        //-- When ball very close don't adjust the position too
                        //-- much, use angle
                        CStatePosition move(which, pcurrentSubStateG,
                                                        &finalPos, 90.0, 0);
                        move.runAction();
                }
                else
                {
                        positionG(finalPos, 90, 0);
                }

                avoidBound(which,finalPos);
                escapeGoal(which);
        }

} //-- S2()

void S3()       //----  S3  centre in goal
{
        floatPOINT finalPos;
        finalPos.x = GOALIESTANDX;
        finalPos.y = Physical_Yby2 + (BallPos.y - Physical_Yby2)*(float)0.25;

        //---- Limit tracking y to penalty area boundary
        if (finalPos.y > Physical_Yby2 + HALF_GOALSIZE)
                finalPos.y = Physical_Yby2 + HALF_GOALSIZE - 2;
        else if (finalPos.y < Physical_Yby2 - HALF_GOALSIZE)

                finalPos.y = Physical_Yby2 - HALF_GOALSIZE + 2;

        positionG(finalPos, 90, 0);
        avoidBound(which, finalPos);
        escapeGoal(which);
}

void S4()       //----  S4  go back avoiding ball
{
        float desiredAngle = 90;
        CAnyPositionAvoidObstacle move(which, pcurrentSubStateG, &finalPos,
                                                &BallPos, &desiredAngle);
        move.runAction();
        avoidBound(which, finalPos);
```

```
                escapeGoal(which);
        }

        //--------- Run function -----------------------------------
        void runAction()
        {
                switch(*pstate)
                {
                case 0: //--    S0      initialisation
                        if( S0S1() ) { *pcurrentSubStateG = 0; S1(); *pstate = 1;}
                        else if( S0S2() ){ *pcurrentSubStateG = 0; S2(); *pstate = 2;}
                        else if( S0S3() ){ *pcurrentSubStateG = 0; S3(); *pstate = 3;}
                        else
                        {
                        //-- Give error alert message;
                        }
                        break;
                case 1://--    S1      sweep ball
                        if( S1S2()&&!AVOID ){ *pcurrentSubStateG=0; S2();*pstate = 2;}
                        else if( S1S4()&&AVOID )
                        { *pcurrentSubStateG = 0; S4(); *pstate = 4;}
                        //-- Move to avoiding state
                        else S1();
                        break;
                case 2://--    S2      track ball y and predict shot
                        if( S2S3() ) { *pcurrentSubStateG = 0; S3(); *pstate = 3;}
                        else if( S2S1() ){ *pcurrentSubStateG = 0; S1(); *pstate = 1;}
                        else S2();
                        break;
                case 3://--    S3      centre in goal
                        if( S3S2() ) { *pcurrentSubStateG = 0; S2(); *pstate = 2;}
                        else S3();
                        break;
                case 4://--    S4      Avoid ball on return to centre
                        if( S4S2() ) { *pcurrentSubStateG = 0; S2(); *pstate = 2;}
                        else S4();
                        break;

                default: //-- Give error alert message
                        break;
                }
                return;
        }

};      //-- CGoalieAction class definition ends here
```

Appendix B

Listing of OOP implementation of g-h filter

```
#define SAMPLETIME 0.01667

//---- Filter Class
class CFilter
{
protected:
        float g, h, k; //-- Filter constants
        float T; //-- sample time
        float   sensorPositionNOW,
                sensorPositionPAST,
                filteredPositionNOW,
                filteredPositionPAST,
                expectedPositionNOW,
                filteredVelocityNOW,
                filteredVelocityPAST,
                expectedVelocityNOW,
                accelerationNOW,
                accelerationPAST,
                predictedPosition,
                predictedVelocity;
public:

        //-- Constructor
        CFilter(float g, float h, float k,float T)
        {
                this->g=g;
                this->h=h;
                this->k=k;
                this->T=T;

                sensorPositionNOW = 0;
                sensorPositionPAST = 0;
                filteredPositionNOW = 0;
                filteredPositionPAST = 0;
                expectedPositionNOW = 0;
                filteredVelocityNOW = 0;
                filteredVelocityPAST = 0;
                expectedVelocityNOW = 0;
                accelerationNOW = 0;
                accelerationPAST = 0;
                predictedPosition = 0;
                predictedVelocity = 0;

                return;
        }

        void updateData(float newSensorPosition)
        {
                //-- swap all old values
                sensorPositionPAST = sensorPositionNOW;
                filteredPositionPAST = filteredPositionNOW;
                filteredVelocityPAST = filteredVelocityNOW;
                accelerationPAST = accelerationNOW;

                //-- update sensor values and calculate new values
                sensorPositionNOW = newSensorPosition;

                expectedPositionNOW = filteredPositionPAST + filteredVelocityPAST*T;
                expectedVelocityNOW = filteredVelocityPAST;
                filteredPositionNOW = expectedPositionNOW + g*(sensorPositionNOW -
                                                        expectedPositionNOW);
                filteredVelocityNOW = expectedVelocityNOW + h*(sensorPositionNOW -
                                                        expectedPositionNOW)/T;

                accelerationNOW = (filteredVelocityNOW - filteredVelocityPAST)/T;

                predictedVelocity = expectedVelocityNOW - h*(sensorPositionNOW -
                                                        expectedPositionNOW)/T;
                predictedPosition = expectedPositionNOW + T*predictedVelocity
```

```
                                        + g*(sensorPositionNOW - expectedPositionNOW);

                return;
        }

        float getFilteredValue()
        {
                return filteredPositionNOW;
        }

        float getPredictedValue()
        {
                return predictedPosition;
        }

        void setG(float g){ this->g = g;}
        void setH(float h){ this->h = h;}
        void setK(float k){ this->k = k;}
        float getG(){ return g;}
        float getH(){ return h;}
        float getK(){ return k;}

};      //-- class CFilter


//-- CAngleFilter class
class CAngleFilter : public CFilter
{

public:
        CAngleFilter(float g, float h, float k, float T):CFilter(g,h,k,T)
        {
        }
        float normalisedDifference;


public:
        void updateData(float newSensorPosition)
        {
                //-- swap all old values
                sensorPositionPAST = sensorPositionNOW;
                filteredPositionPAST = filteredPositionNOW;
                filteredVelocityPAST = filteredVelocityNOW;
                accelerationPAST = accelerationNOW;
                //-- update sensor values and calculate new values
                sensorPositionNOW = newSensorPosition;

                expectedPositionNOW = filteredPositionPAST + filteredVelocityPAST*T;
                expectedVelocityNOW = filteredVelocityPAST;
                normalisedDifference = (sensorPositionNOW - expectedPositionNOW);
                while (normalisedDifference >180)normalisedDifference-=360;
                while (normalisedDifference <-180)normalisedDifference+=360;

                filteredPositionNOW = expectedPositionNOW + g*normalisedDifference;
                filteredVelocityNOW = expectedVelocityNOW + h*normalisedDifference/T;

                //-- normalise the filtered value
                while (filteredPositionNOW >180)filteredPositionNOW-=360;
                while (filteredPositionNOW <-180)filteredPositionNOW+=360;


                accelerationNOW = (filteredVelocityNOW - filteredVelocityPAST)/T;
                return;
        }

};      //-- class CAngleFilter
```

Appendix C

System for Real-Time Measurement of Robot Movements and Analysis of Collaborative Tasks

In any collaborative task involving the participation of several robots, there is a strategy involved to accomplish the task. The strategy employed greatly influences the agents' behaviour and performance. In the case of a highly dynamic and collaborative game of robot soccer, the strategy used makes a huge difference to the outcome of the game. There is a need to analyse the game so that the team dynamics and performance can be improved. The analysis often involves scrutinising the agent movements, calculating the work rate and several other parameters such as speed, distance travelled, etc. In recent years, significant research and development effort has been invested in developing an automated system for post-match analysis. While a post-match analysis is certainly useful for future performance improvements, some match statistics, if available in real-time during the game, is very effective in fine-tuning the strategy while the game is in progress. It is common place these days for TV broadcast stations and coaches to use such software. However, most of these applications rely, to varying extent, on some form of human inputs.

There is great demand for fully automated analysis software for analysing collaborative behaviours. A big hindrance to developing such a system is the availability of a flexible test platform which may be used to authenticate the usefulness of the software. The robot soccer platform has been employed to test the analysis software.

Majority of what applies to human version of soccer also holds true in a team game with robots even though it is in a different form - the players being robots and the coaches being programmers. The team performance is improved through the off-line modification of the strategy code or through the tweaking of certain parameters of the strategy during the game.

The collaborative task performance analysis system described in this chapter has many functions. Firstly, it can be operated in two modes – *off-line* and *real-time*. In the *off-line mode*, while the task is in progress, the data output from the image processing layer, namely the positions and angles of the robots and other objects (such as the ball in the case of a robot soccer game), is stored in a file which is later used to reconstruct and simulate the task. This eliminates the need to video record the performance. Additionally, the system can calculate the velocity of the robots in each frame, the total distance travelled and work done in a time window, the top speeds achieved and average accelerations. A very important feature of the program is that it also

automatically produces graphical outputs, so that the robots' performance can be visualized easily and quickly. The software program helps the coach make decisions such as replacing an agent, changing batteries of an 'over-worked' robot, tweaking behaviour and control parameters, and adjusting the strategy.

In the *real-time mode*, the captured data are processed and depicted immediately, with updates in each frame. This helps to make quick decisions while the task is in progress.

Section C.1 gives an overview of the system architecture for stand-alone and WAN (wide area network) operations. Section C.2 details the various capabilities and features of the system, namely the raw-data viewer, task replay simulator, match analysis program and the graph generation respectively. The chapter concludes with a summary in section C.3.

C.1 System Architecture

The hardware architecture of the system is shown in Figure C.1. Each team has a *strategy server* running on a PC. In the *stand-alone mode*, the analysis program can be run on this PC. In the *networked mode* the server passes the vision data to a remote controller (the client) using the Internet connections.

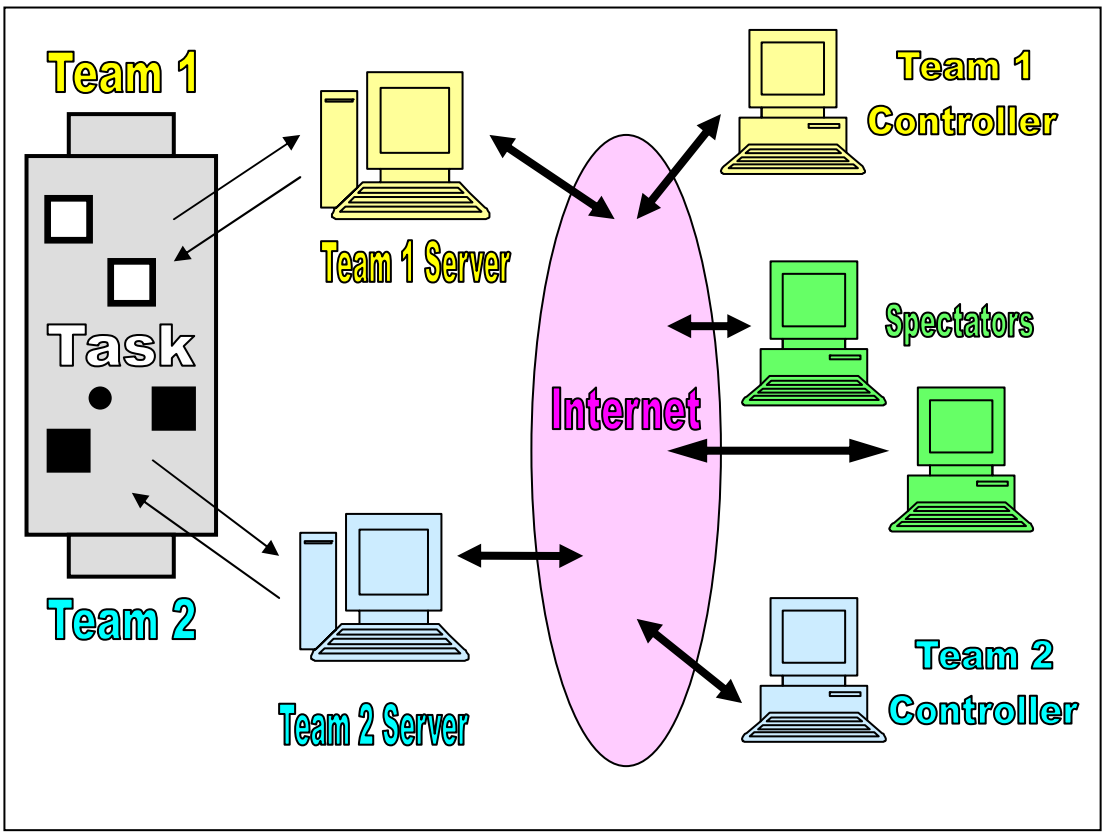


Figure C.1 System hardware architecture for analysis and control of collaborative tasks

In the networked mode, the standard Windows Sockets API (Winsock) is used to establish a connection between the server and the controller. Only the agents' posture (position and orientation) is sent to the controller which is stored in a file. The game is re-constructed and analysed by the controller either in *off-line* or *real-time* mode.

The advantages of the networked mode are many folds. It allows the collaborative task to be analysed (and also controlled) from a remote station which is often far away from the scene of the physical actions. The controller can manipulate the parameters and alter the strategy for the collaborative task. As shown in Figure C.1, there can be several other remote stations connected to the server just to observe the game in progress. These can be referred to as '*spectator terminals*'. These spectator terminals, however, do not have the right to manipulate the game. These can only receive the data from the server to reconstruct the game and watch it live.

C.2 Program Description

The image is captured and processed in two separate, odd and even, frames. In each frame the image is analysed and the data generated by the vision software (namely, the robot position and orientation) are logged into a file, which is subsequently used for various purposes. The four main features of the program are the raw-data viewer, the game replay simulator, the game statistics calculator and the graph generator. These are explained in detail in the following sub-sections.

Raw-Data Viewer

The raw-data viewer provides the 'coach' with a comprehensive list of the data that are logged into the file. Figure C.2 shows a screenshot of the raw data viewer dialog box.

The data that are presented in the dialog box is the robot position (X and Y coordinates) in the current and previous frame, robot angle in degrees in the current and previous frame, and current angular velocities of the robots. The robot position is shown in screen coordinates (pixels) as well as the real-world coordinates (cm).

The user can scroll through the data by using the UP and DN command buttons. The number of frames through which the data is scrolled, the *step by* size, can be specified; the default is one. A user can also randomly seek to a particular frame data by specifying the frame number. While scrolling, the current frame number is also depicted. Also shown on the screen are the coordinates of the field, home goal and opponent goal boundaries. This data, however, does not change with each frame.

For a complete game, the amount of data logged in is very large. From the screen and real-world coordinates, a user really can not infer much about the game state and the robot positions. It was thus necessary to create software that could graphically recreate

the game from the raw data. This game replay simulator is explained in the following sub-section.

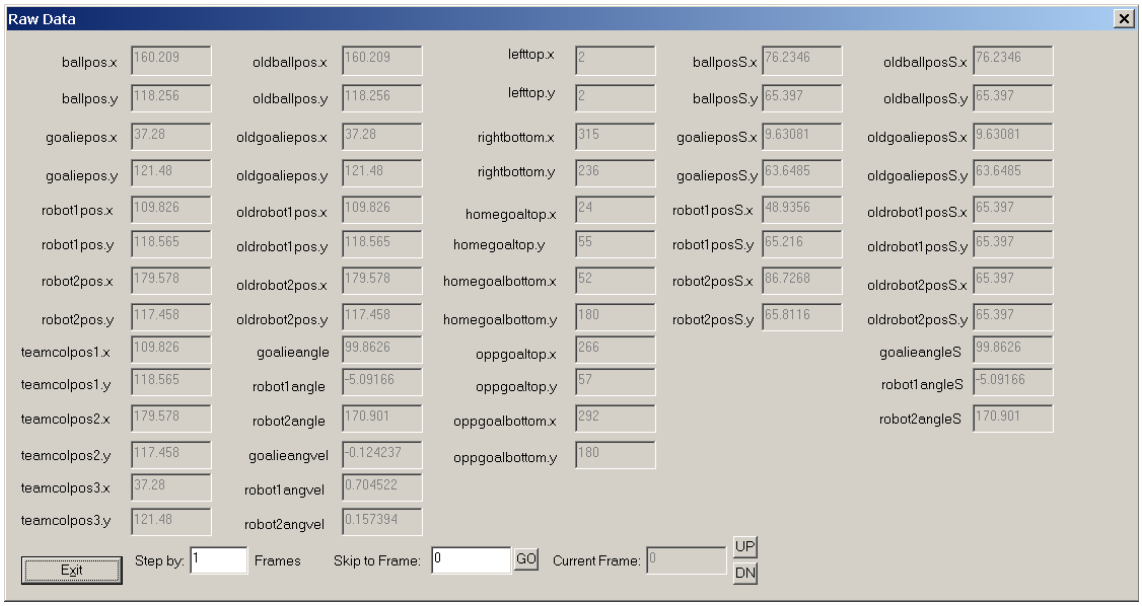


Figure C.2 A screenshot of the raw data viewer

Task Replay Simulator

Coaches traditionally analyse a game by watching the video recordings. In the robot soccer system a camera is already in place which overlooks the field. It is very convenient to feed the video signal to a recording unit and record the game for viewing later. While this is a possibility, it is not entirely necessary, if the sole purpose of the recording is to analyse the game.

While the game is in progress, the vision software generates data pertaining to the robot (and ball) position and robot orientation. This data, which are stored in a file, are utilized by the game replay simulator to 'reconstruct' the game. Bit-mapped graphic images, representing the robots and ball, are generated and depicted on the computer screen. The location and orientation of the bit-mapped images is determined from the data for each frame. The field, home goal and opponent goal boundaries are also drawn using the data stored in the file.

The simulator has the advantage of not requiring additional video recording hardware. Other features of the simulator are:

- It can step through the data, frame by frame, and construct the scenes.
- The step size can be changed. To look at the game situation every one second, the step size should be set to 60.
- It is able to advance as well as retract frames.

- It allows random seeking to any frame and construct the scene.
- It implements free running game reconstruction in which the game is replayed in real time at the exact speed at which it was recorded (60 frames per second).
- It allows stopping the simulator at any frame.

Figure C.3 shows a screen shot of the reconstructed scene for frame number 2115. The GUI control elements are also visible.

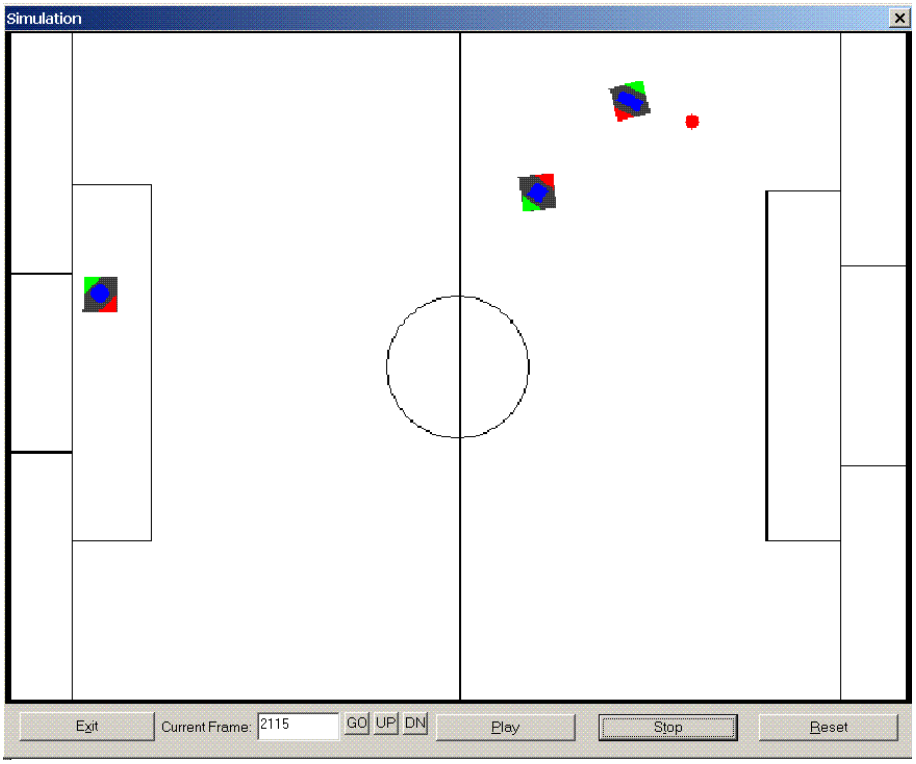


Figure C.3 A screenshot of the game replay simulator

From a coach's perspective, the most useful feature of the simulator is the ability to step through the scenes frame by frame. This enables him to explain to his players the strategy and to point out where it can be improved.

Task Statistics Calculator

One of the objectives of the program is to provide team members with statistical information that would normally not be available from traditional video recording or would be very complex to calculate. For this reason a statistics calculator has been developed to perform as many of these calculations as possible and to present these in a text format that is easy to comprehend. This data are extremely useful for the coach to make critical decisions about player selection, role assignments and, in general,

selecting a strategy for the team. Figure C.4 shows a screenshot of the GUI for the game statistics calculator.

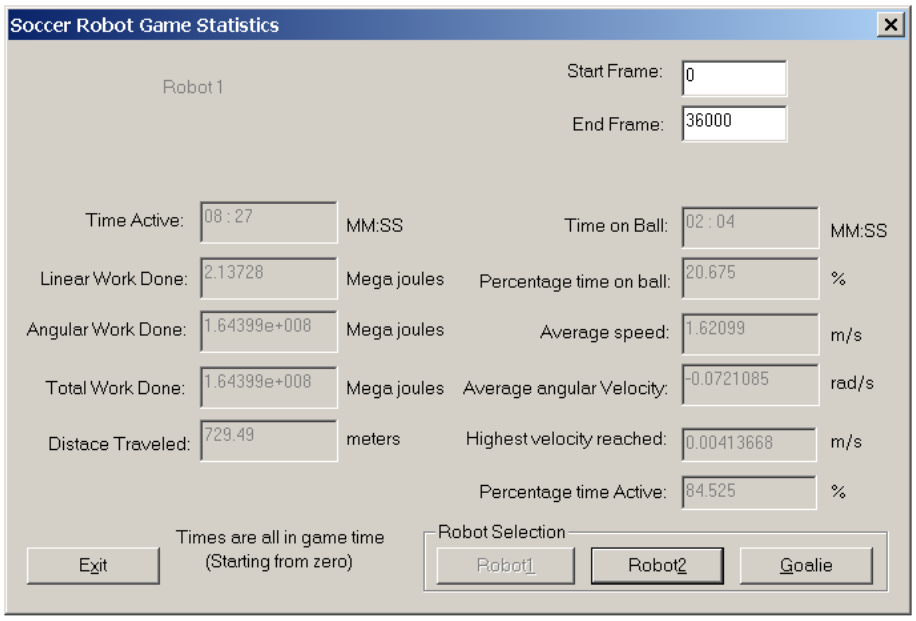


Figure C.4 A screenshot of the game statistics screen

The analysis software produces the following statistics for each player:

- the amount of time that the robot is 'active' in MM:SS and as a percentage of the monitoring time.
- the amount of time that the robot is within an arbitrarily set distance from the ball (called time on ball) in MM:SS and as a percentage of the monitoring time.
- the mount of rotational (angular) work done
- the amount of linear work done
- the total work done in Mega Joules
- the total distance travelled in meters
- the maximum speed that the robot attained during the game in m/s
- the average speed (angular and linear) of the robot player during the game.

The game statistics can be calculated for the entire game or over a set of contiguous frames with an arbitrarily set start point.

In the game statistics calculator, the values are calculated using the standard physical formulae.

Work done = force x distance

Force = mass x acceleration

Acceleration = Δspeed / Δtime

Speed = Δdistance / Δtime

Graph Generation

A very useful tool to use in the analysis of statistical data is graphing. To this end the analysis program produces comparison graphs for all the game statistics so that one can compare the performance of the team members in all the areas that are being computed. These graphs are produced from the data that are logged into the file during the game. Graphs can also be custom produced; any of the variables may be chosen to be depicted in a graph. In fact, any combination of variables, within the boundaries of normal restrictions on graph type classification, is possible. This gives the coach tremendous scope to analyse every aspect of the game.

There are several different types of graphs that can be produced. Figure C.5 shows the work done by the three robot players of a team in four equal time intervals of a game. Figure C.6 shows the total work done, as a percentage, by the robot players and Figure C.7 shows the total distance travelled by the robot players.

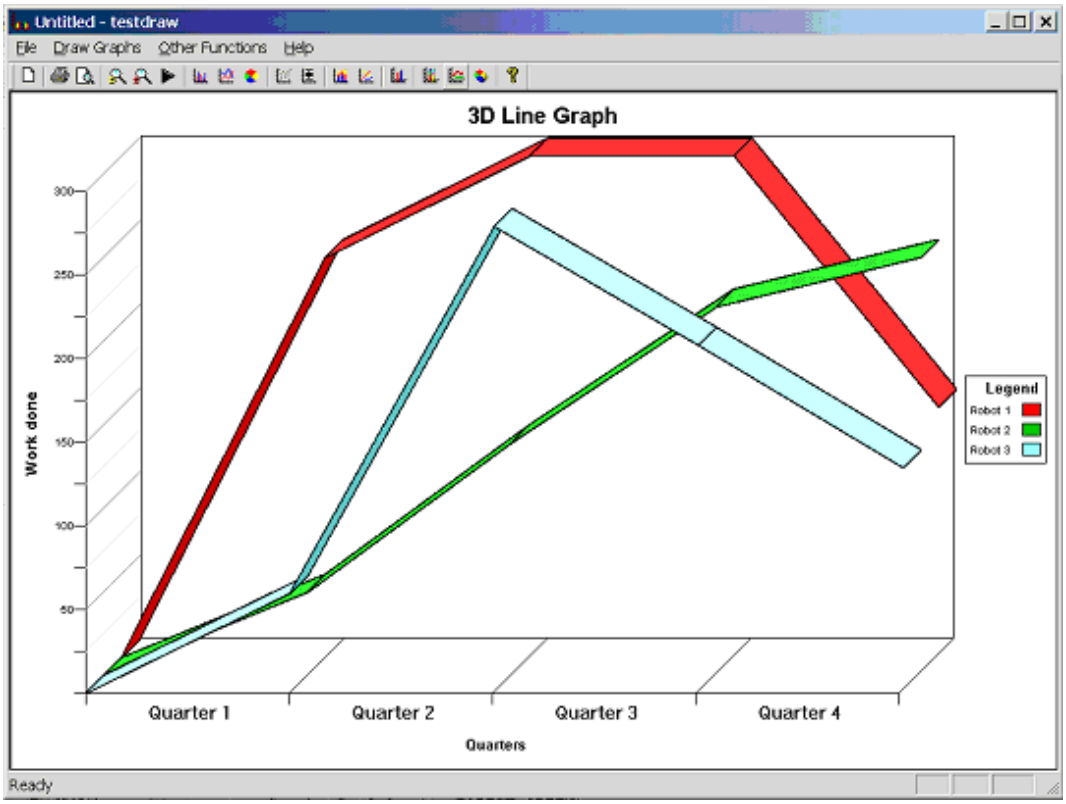


Figure C.5 Work done by the robots, shown using 3D line graph

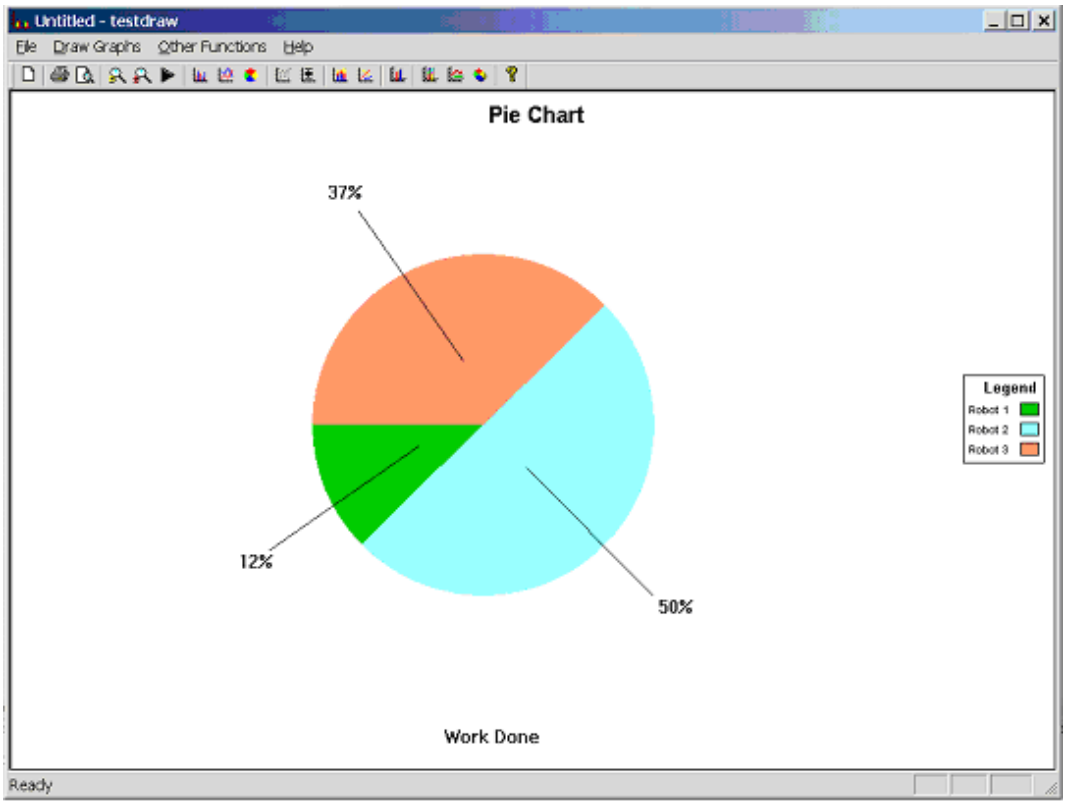


Figure C.6 Total work done, as a percentage, by the robots in a game, shown in a pie chart

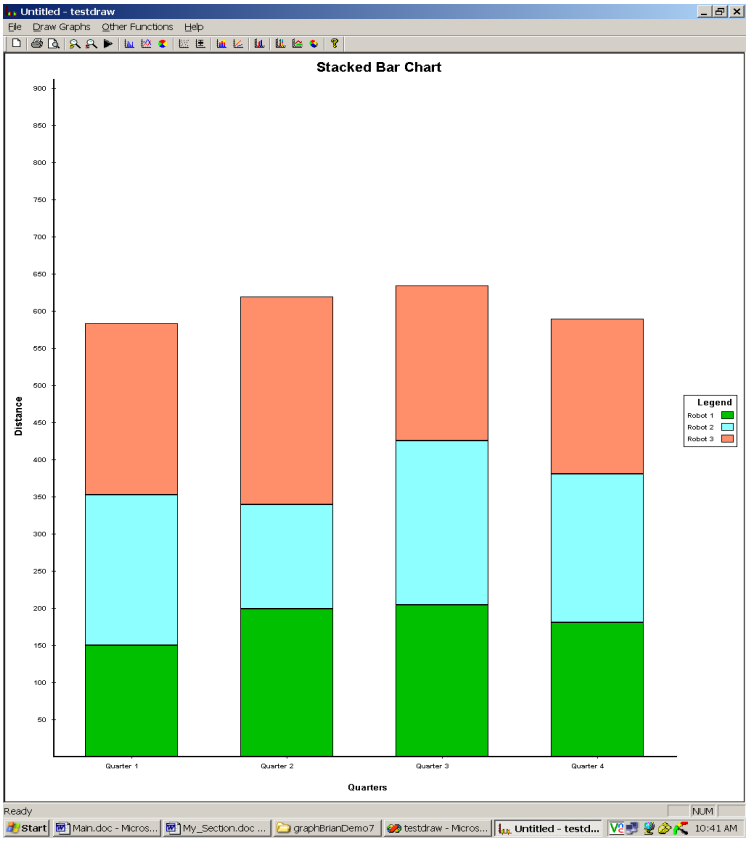


Figure C.7 Distance travelled by the robots, shown in a stacked bar chart

C.3 Summary

Robot Soccer system has been a useful test bed to develop systems for game analysis involving a team of players. This chapter has described the match analysis program's functionality, namely the raw-data viewer, game replay simulator, match analysis program and the graph generation respectively. The system has been fully tested on a system with 3 robots in a team. It can be adapted for other team games like rugby and applied to the human version of the game. The biggest challenge in adapting the system for human version of the game will be the vision processing software.

The program opens the door to the world of game statistics and analysis of cooperative tasks. The data can be analysed *after* the game (or the collaborative effort) is over to improve the strategy for future games (or tasks) or it can be analysed in *real-time*.

The system, in general, can be used to analyse the behaviour of *any* multi-agent cooperative task. It is a handy tool to evaluate the efficiency of the algorithms employed for the collaborative tasks. Such a tool is not readily available at the moment. What makes the tool even more versatile is its capability to analyse and update the performance data in real-time. The real-time analysis feature is useful, not only to the coaches and players but also to the spectators.

Another useful extension that has been done to the system is making the game data (robot/ball position and orientation) available on a remote station over the Internet. This enables the game to be reconstructed in real-time on any remote computer connected to the World Wide Web and the match statistics to be generated on remote local stations. From the analysis data, a coach, stationed on a remote computer, is able to tweak the game parameters and alter the strategy.