



University of Bradford eThesis

This thesis is hosted in [Bradford Scholars](#) – The University of Bradford Open Access repository. Visit the repository for full metadata or to contact the repository team



© University of Bradford. This work is licenced for reuse under a [Creative Commons Licence](#).

WIRELESS MOSAIC EYES BASED ROBOT

PATH PLANNING AND CONTROL

**Autonomous Robot Navigation using Environment
Intelligence with Distributed Vision Sensors**

Yongqiang CHENG

BSc, MSc

**Submitted for the degree
of Doctor of Philosophy**

School of Engineering Design and Technology

University of Bradford

2010

ABSTRACT

As an attempt to steer away from developing an autonomous robot with complex centralised intelligence, this thesis proposes an intelligent environment infrastructure where intelligences are distributed in the environment through collaborative vision sensors mounted in a physical architecture, forming a wireless sensor network, to enable the navigation of unintelligent robots within that physical architecture. The aim is to avoid the bottleneck of centralised robot intelligence that hinders the application and exploitation of autonomous robot. A bio-mimetic snake algorithm is proposed to coordinate the distributed vision sensors for the generation of a collision free Reference-snake (R-snake) path during the path planning process. By following the R-snake path, a novel Accompanied snake (A-snake) method that complies with the robot's nonholonomic constraints for trajectory generation and motion control is introduced to generate real time robot motion commands to navigate the robot from its current position to the target position. A rolling window optimisation mechanism subject to control input saturation constraints is carried out for time-optimal control along the A-snake. A comprehensive simulation software and a practical distributed intelligent environment with vision sensors mounted on a building ceiling are developed. All the algorithms proposed in this thesis are first verified by the simulation and then implemented in the practical intelligent environment. A model car with less on-board intelligence is successfully controlled by the distributed vision sensors and demonstrated superior mobility.

Keywords: Distributed vision sensors, Wireless sensor network, Robot control, Path planning, Trajectory generation, Accompanied snake.

ACKNOWLEDGEMENT

So many people, so many things I would like to take this opportunity to show appreciations during these three years of PhD study. This thesis would not been possible without them.

I would like to extend the deepest gratitude to my two supervisors, the wisest people I have ever met. Professor Yim Fun Hu, whose wisdom and commitment to the highest stands has inspired me all the time. I am thankful to her guidance as well as the financial support given throughout my PhD study. She has given me a peaceful mind to concentrate on my research. Doctor Ping Jiang is not only a good mentor but also a helpful friend. He has helped me to settle down in this foreign country and to tackle all the difficulties of the studies. I am heartily grateful for their great efforts on revising my thesis to make this possible.

I would also like to thank all my colleagues, especially the SANDRA team. They have shown complete understanding, especially when my thesis writing up took priority over the project.

In addition, I thank all my friends who have supported me and encouraged me in any respect during the completion of this research. A special thank to Miss Helen Hoy for her invaluable effort on correcting my English language grammars making the thesis more presentable.

Last but not least, I would like to thank my families, especially my wife whom providing me with unquestioning support, patience, understanding and love.

Thank you all my beloved people!

TABLE OF CONTENTS

ABSTRACT.....	II
ACKNOWLEDGEMEN.....	III
CHAPTER 1 : INTRODUCTION	1
1.1 Introduction.....	1
1.2 Background	1
1.3 Aims and Objectives of the Thesis.....	5
<i>1.3.1 Scientific Objectives</i>	<i>6</i>
<i>1.3.2 Technological Objectives</i>	<i>7</i>
1.4 Contributions of the Thesis	8
1.5 Structure of the Thesis.....	10
1.6 Research Publications	11
CHAPTER 2 : LITERATURE REVIEW	13
2.1 Introduction.....	13
2.2 Robot Navigation	15
<i>2.2.1 Reactive Control.....</i>	<i>16</i>
<i>2.2.2 Global Path Planning.....</i>	<i>21</i>
2.2.2.1 Configuration Space	21
2.2.2.2 Graph Based Planning	22
2.2.2.3 Wavefront Base Planning: Potential Field.....	25
<i>2.2.3 Active Contour Model.....</i>	<i>29</i>
2.3 Kinematic Constraints	31
2.4 Trajectory Generating	32
2.5 WSN Platforms	33
2.6 Bloom Filter	36
2.7 Summary	39

CHAPTER 3 : SYSTEM ARCHITECTURE OF AN INTELLIGENT ENVIRONMENT WITH WIRELESS MOSAIC EYES 41

3.1 Introduction 41

3.2 Mosaic Eyes and Intelligent Environment Architecture..... 43

3.3 System Components 48

 3.3.1 *The Remote Console* 48

 3.3.1.1 Visual Content Configuration Module.....52

 3.3.1.2 Data Download Module.....53

 3.3.1.3 Vision Sensor Control Module53

 3.3.1.4 Vision Sensor Status Remote Monitoring Module53

 3.3.1.5 Camera Calibrating Module54

 3.3.1.6 Goal Input Module55

 3.3.2 *Vision Sensors* 55

 3.3.2.1 Image Acquisition and Processing Module57

 3.3.2.2 Path Initialisation Module.....58

 3.3.2.3 Trajectory Generating Module.....59

 3.3.2.4 Predictive Controller Module59

 3.3.2.5 Control Robot Module.....60

 3.3.2.6 Position Estimation Module61

 3.3.2.7 Path Adjustment Module61

 3.3.2.8 Path Re-initialisation Module62

 3.3.3 *Wireless Controlled Robots* 62

 3.3.3.1 Motion Commands Analyse and Execution Module63

 3.3.3.2 Motor Servo Module63

 3.3.4 *Communication Interfaces*..... 63

3.4 Conclusion 65

CHAPTER 4 : MAP BUILDING AND OBJECTS LOCALISATION 66

4.1 Introduction 66

4.2 Multiple Bloom Filters for Map Building and Distributed Routing 69

 4.2.1 *Problem Statement*..... 69

4.2.2 Bloom Filters for Navigation Routing	72
4.2.3 Design of Multiple Bloom Filters Based on Error Expectation.....	74
4.2.4 Memory Usage of the Error Expectation Based Design.....	80
4.2.5 Multiple Bloom Filters for Routing	83
4.3 Objects Identification and Fast Colour Blobs Tracking	86
4.3.1 Overview.....	86
4.3.1.1 Colour Space Transformation.....	87
4.3.1.1.1 RGB Colour Format	87
4.3.1.1.2 HSL/HSV/HSI Colour Space	88
4.3.1.1.3 YUV Colour Space.....	90
4.3.1.2 Colour Recognition and Blobs Tracking	91
4.3.2 Overall Structure of Image Processing	93
4.3.3 Offline Algorithm.....	94
4.3.3.1 Colour Sampling and Filtering	95
4.3.3.2 Lookup Table Generating.....	97
4.3.4 Online Algorithm.....	98
4.3.4.1 Foreground Image Obtaining.....	98
4.3.4.2 Binary Image Generating and Segmenting	100
4.3.4.3 Object Identification and Tracking	102
4.4 Robot Locating.....	104
4.5 Obstacles Extraction.....	105
4.5.1 Occupancy Map Building.....	106
4.5.2 Multiple Cameras Based Obstacle Extraction.....	108
4.5.2.1 Problem Statement.....	109
4.5.2.2 Multiple Vision Fusion for Obstacles Extraction.....	111
4.6 Conclusion	115

**CHAPTER 5 : A DISTRIBUTED SNAKE ALGORITHM FOR MOBILE
ROBOTS PATH PLANNING WITH CURVATURE CONSTRAINTS116**

5.1 Introduction.....	116
5.2 Bio-inspired Snake Algorithm	118

5.2.1 Problem Statement.....	119
5.2.2 The Internal Force.....	121
5.2.3 The External Force.....	124
5.3 Three State Snake under Curvature Constraints.....	126
5.3.1 Snake Algorithm in a Distributed Environment.....	127
5.3.2 States Transfer.....	131
5.3.2.1 Flexible State to Rigid State	132
5.3.2.2 Rigid State to Flexible State	132
5.3.2.3 Flexible State to Broken State	132
5.3.2.4 Rigid State to Broken State.....	133
5.3.2.5 Broken State to Flexible State	133
5.4 Simulation	134
5.5 Conclusion	140

CHAPTER 6 : PREDICTIVE TRACKING CONTROL WITH DYNAMIC CONSTRAINTS 142

6.1 Introduction.....	142
6.2 A-snake Spatial Position Planning	144
6.2.1 Desired Direction to Approach the R-snake.....	144
6.2.2 Dynamic Constraints and Maximum Allowable Uniform Velocity.....	146
6.2.3 Convergence of A-snake Approaching R-snake.....	148
6.2.4 A-snake Growth in Discrete Time Domain.....	153
6.3 A-snake Time Series Trajectory Generation and Motion Control.....	156
6.3.1 Problem Statements	156
6.3.2 Optimal Control in the Rolling Window	158
6.4 Simulation and Experiment Results	162
6.4.1 A-snake Trajectory and Velocity	162
6.4.2 Path Planning and Predictive Control Experiment.....	164
6.5 Conclusion	168

CHAPTER 7 : IMPLEMENTATION 169

7.1 Introduction	169
7.2 Hardware Platform	170
7.2.1 <i>Hardware Architecture Overview</i>	170
7.2.2 <i>Vision Sensor Hardware Integration</i>	172
7.2.2.1 iMote2 Main Board Introduction	172
7.2.2.2 OV7620 Vision Module Introduction.....	175
7.2.2.3 Sensor Board Design and Implementation.....	177
7.2.2.4 Power Board Introduction.....	179
7.2.3 <i>Mobile Robot Hardware Design and Implementation</i>	179
7.2.3.1 Overview of the Robot Hardware Platform	179
7.2.3.2 The Mobile Robot Hardware Modules Introduction.....	180
7.2.3.2.1 Car Model Chassis	180
7.2.3.2.2 MCU Minimum System	182
7.2.3.2.3 DC Driving Motor	183
7.2.3.2.4 Steering Servo Motor	184
7.2.3.2.5 The Wireless Peripheral Module	184
7.2.3.3 Mobile Robot Hardware Design and Implementation	185
7.2.3.3.1 DC Motor Driving Module Design	185
7.2.3.3.2 Steering Module Design	188
7.2.3.3.3 Speed Detection Module Design	189
7.2.3.3.4 Mobile Robot System Implementation	190
7.2.4 <i>Remote Console</i>	191
7.3 Communication Protocol.....	192
7.3.1 <i>Overview</i>	192
7.3.1.1 Generic Packet Format	194
7.3.1.2 Commands List.....	195
7.3.2 <i>Communication between Vision Sensors</i>	196
7.3.2.1 CMD4: Obstacles Command.....	196
7.3.2.2 CMD7: Control Points Commands.....	197
7.3.2.3 CMD14: Handover Token Commands	198
7.3.2.3.1 Token Handover Signal Flows	199
7.3.3 <i>Communication between Vision Sensor and Robot</i>	203

7.3.4 <i>Communications between Vision Sensors and the Remote Console</i>	205
7.3.4.1 CMD1: Restart Client Program	207
7.3.4.2 CMD2: Acquire Image	208
7.3.4.3 CMD3: Change Background Image.....	208
7.3.4.4 CMD5: Send Data Request.....	209
7.3.4.5 CMD6: Reliable Data Packets	209
7.3.4.6 CMD7: Control Points.....	210
7.3.4.7 CMD8: Image Data	211
7.3.4.8 CMD9: ACK	211
7.3.4.9 CMD10: iMote2 System Status Information	212
7.3.4.10 CMD12: A-snake Control Points.....	213
7.3.4.11 CMD13: Robot Coordinate.....	213
7.3.4.12 CMD15: Control Points Display Switch.....	214
7.3.4.13 CMD16: Robot Control Mode.....	214
7.4 Software Implementation	215
7.4.1 <i>Overview of the Software Structure</i>	215
7.4.2 <i>iMote2 Vision Sensor Software Implementation (App 4 and App 5)</i>	218
7.4.2.1 Application 4 Software Implementation.....	218
7.4.2.2 Application 5 Software Implementation.....	220
7.4.2.2.1 Image Acquisition and Processing	226
7.4.2.2.2 R-snake Process.....	227
7.4.2.2.3 Trajectory Generating Process.....	228
7.4.2.2.4 Token Control Implementation	232
7.4.2.2.4.1 Four Timers Implementation.....	233
7.4.2.2.4.2 Token Control Process	234
7.4.2.2.4.3 CMD14 Message Response Functions Implementation.....	236
7.4.3 <i>Remote Console Implement (App 2)</i>	240
7.4.3.1 Remote Console GUI Interfaces	241
7.4.3.1.1 Commands Area.....	241
7.4.3.1.2 Display Area.....	241
7.4.3.1.3 Status Area	242
7.4.3.1.4 The Implementation	242
7.4.3.2 Remote Console Communication Module Implementation.....	243

7.4.3.3 Profile Files Generating	244
7.4.4 Remote Console Wireless Peripheral Software Implementation (App 3)	248
7.4.5 Gainz Software Implementation (App 6)	250
7.4.6 Mobile Robot Software Implementation (App 7)	251
7.4.7 Simulation Environment Implementation (App 1)	252
7.5 Discussion and Evaluation	257
7.5.1 Processing Time for Software Modules	257
7.5.2 The Impact of the Differences in Camera Physical Characteristics	259
7.6 Conclusion	261
CHAPTER 8 : CONCLUSION AND FUTURE WORK	263
8.1 Conclusion	263
8.1.1 A Distributed Intelligent Environment Architecture	265
8.1.2 The Distributed Snake Algorithm	266
8.1.3 A-snake Based Trajectory Tracking	266
8.1.4 A Control-Oriented Communication Protocol	267
8.1.5 Applying Multiple Bloom filters in Topological Routing	267
8.1.6 Occupancy Map Building and Geometry Obstacles Extraction	268
8.1.7 The Intelligent Environment Test Bed Development	268
8.1.8 Creation of a Simulation Environment for Snake Algorithm Evaluation	269
8.2 Future Work	269
8.2.1 Image Based Distributed Visual Servoing	270
8.2.2 Self-Calibration of a Wireless Vision Sensor Network	271
8.2.3 Automatic Background Image Adaptation	272
8.2.4 Bio-Mimetic Technique Researches	273
REFERENCE	275

LIST OF FIGURES

Figure 2.1 One potential field example in a configuration space. Initial and Goal configuration are indicated in the figure. Light colour indicate high potential, dark colour indicate low potential. The path starts from the initial configuration to the goal configuration is generated based on the potential gradient in the configuration.	25
Figure 2.2 An example of a simple Bloom filter. The filter begins as an array of all 0s. Each item Jerry, Linda and John’s room in the set is hashed a certain times, with each hash yielding a bit location; these bits are set to 1s. To check if an element y is in the set, hash it the same times and check the corresponding bits. The element y_1 is either in the set or the filter has yielded a false positive; the element y_2 cannot be in the set, since a 0 is found at one of the bits.	37
Figure 3.1 Mosaic eyes inspired intelligent environment	44
Figure 3.2 Overview of system components and functional modules	47
Figure 3.3 Snapshot of the remote console GUI interface	49
Figure 3.4 Robot on field	62
Figure 4.1 A simple topological map	71
Figure 4.2 Classification of random strings to discover the associated routing semantics	71
Figure 4.3 Length m vs. a given k	75
Figure 4.4: Error expectation comparison with two methods.....	85
Figure 4.5 RGB space example	88
Figure 4.6 HIS space example	89
Figure 4.7 Overall structure of image processing	94
Figure 4.8 Colour clustering cube	95
Figure 4.9 Colour sampling interface.....	95
Figure 4.10 Colour cubes with noise.....	96
Figure 4.11 Sample colour filtering	97
Figure 4.12 Obstacles extracting procedure flow chart.....	100
Figure 4.13 One encoded example by run-length.....	102
Figure 4.14 One occupancy map example from [155].....	106

Figure 4.15 Camera space divided into smaller areas	107
Figure 4.16 Projective image of obstacle.....	109
Figure 4.17 The angle of depression	110
Figure 4.18 Probability of obstacle existence when number of cameras seeing it increases.	113
Figure 4.19 Bold red circles denotes the real shadow cast.....	114
Figure 4.20 Planned path after fusion of two camera information.....	114
Figure 5.1 Curvature and curvature energy.....	122
Figure 5.2 Form of curvature energy function.....	123
Figure 5.3 Information exchange flow between control points. (x_{i-1}, y_{i-1}) , (x_i, y_i) and (x_{i+1}, y_{i+1}) are coordinates of control points p_{i-1} , p_i and p_{i+1} respectively, f^{back} , M^{back} are the average equivalent force and torque needed to send back to related control point, (x_{i-1}^c, y_{i-1}^c) , (x_i^c, y_i^c) are centroid coordinates of control points p_{i-1} and p_i , hop indicates the state of curvature at a control point, f_{i-1}^c, M_{i-1}^c and f_i^c, M_i^c are calculated average centroid force couple and average centroid torque.....	128
Figure 5.4 Control points state exchange chart.....	134
Figure 5.5 Initiated snake	135
Figure 5.6 Flexible state with obstacles	136
Figure 5.7 Some rigid state control points	137
Figure 5.8 More obstacles approach snake.....	138
Figure 5.9 Reaching broken state	138
Figure 5.10 Re-initiated path	139
Figure 6.1. Coordinates definitions.....	144
Figure 6.2. Control flow chart.....	149
Figure 6.3. Phase track of e and ℓ	151
Figure 6.4. Sliding mode happens.....	152
Figure 6.5 A-snake	155
Figure 6.6. Robot, A-snake and rolling window	157
Figure 6.7. Rolling window optimisation for trajectory generation (assume $v_{r0} = 0$).....	161

Figure 6.8. Deformable snake with A-snake	163
Figure 6.9. Allowed velocity, planned velocity and acceleration.....	163
Figure 6.10. Planned curvature derivative, curvature and orientation.....	164
Figure 6.11. Robot moving from eye-30 to obstacles free eye-60	165
Figure 6.12. Obstacles appear in eye-60.....	165
Figure 6.13. Robot passing obstacle area in eye-60.....	166
Figure 6.14. Robot passed obstacles area in eye-60 and move to eye-50	166
Figure 6.15. Robot control and velocity (driving force F^d : dot; steering torque τ :dashed; velocity v : solid).....	167
Figure 7.1 Intelligent environment	171
Figure 7.2 One wireless mosaic eye	172
Figure 7.3 iMote2 main board	173
Figure 7.4 Simplified block diagram of CC2420	175
Figure 7.5 Sensor board schematic circuit connecting iMote2 and OV7620 vision module.	177
Figure 7.6 Sensor board PCB layout	178
Figure 7.7 Power board and its specifications	179
Figure 7.8 Robot hardware modules	180
Figure 7.9 Car model	181
Figure 7.10 MC9S12DT128B MCU minimum system.....	183
Figure 7.11 DC driving motor.....	183
Figure 7.12 Servo motor	184
Figure 7.13 Gainz 2.4GHz IEEE 802.15.4 node.....	185
Figure 7.14 H-bridge circuit.....	186
Figure 7.15 Driving motor circuit.....	188
Figure 7.16 Pulse width – rotational angle relation, clockwise(+), anti-clockwise(-)	189
Figure 7.17 Tachogenerator is mounted on the mobile car	190
Figure 7.18 System PCB design	190
Figure 7.19 Remote console with attached iMote2 main board	191
Figure 7.20 Communication protocols overview	192

Figure 7.21 Sending border control points from preceding vision sensor to succeeding ones	197
Figure 7.22 Exchange border control points signal flow	198
Figure 7.23 Mobile robot in observation zones in vision sensor	199
Figure 7.24 Control token init signal flow, case 1	200
Figure 7.25 Control token init signal flow, case 2	200
Figure 7.26 Control token init signal flow, case 3	201
Figure 7.27 Control token handover signal flow - successful	202
Figure 7.28 Control token handover signal flow - failure	202
Figure 7.29 Robot control signal flow	204
Figure 7.30 Signal flow between remote console and vision sensor - Scenario 1	206
Figure 7.31 Signal flow between remote console and vision sensor - Scenario 2	206
Figure 7.32 Signal flow between remote console and vision sensor - Scenario 3	207
Figure 7.33 Image data structure	211
Figure 7.34 Software structures and the contributions	216
Figure 7.35 Interactions between App 4 and App 5	220
Figure 7.36 Token command concurrent threads	221
Figure 7.37 Classes and structures defined in App 5	223
Figure 7.38 Main program flow chart	225
Figure 7.39 Snake with 6 control points	226
Figure 7.40 Image acquisition and processing	226
Figure 7.41 R-snake process	228
Figure 7.42 Trajectory generating process	229
Figure 7.43 Command sent to robot	232
Figure 7.44 Four timers flow charts	233
Figure 7.45 Main token process thread	235
Figure 7.46 Process upon occupy token message receipt	236
Figure 7.47 Process upon token request message receipt	237
Figure 7.48 Process upon token reply message receipt	238
Figure 7.49 Process upon handover request message receipt	239

Figure 7.50 Process upon handover reply message receipt	239
Figure 7.51 Process upon handover confirmation message receipt	240
Figure 7.52 Application 2 GUI interface.....	242
Figure 7.53 Messages classifier	244
Figure 7.54 System profile meta-data.....	245
Figure 7.55 Camera calibration grid.....	246
Figure 7.56 Overlap areas in two cameras.....	246
Figure 7.57 Mapping calibration step 1 of 4	247
Figure 7.58 Mapping calibration step 2 of 4	247
Figure 7.59 Mapping calibration step 3 of 4	248
Figure 7.60 Mapping calibration step 4 of 4	248
Figure 7.61 Remote console wireless peripheral software processing flow	249
Figure 7.62 Command execution flowchart in mobile robot.....	252
Figure 7.63 Simulation software main interfaces	253
Figure 7.64 Simulation software classes structure	255
Figure 7.65 Software modules processing time: not sending display control points, no robot in view.....	257
Figure 7.66 Software modules processing time with robot in view.....	258
Figure 7.67 Running time comparison	259
Figure 7.68 Image taken by different cameras - 1.....	260
Figure 7.69 Image taken by different cameras - 2.....	261
Figure 7.70 Image taken by different cameras - 3.....	261

LIST OF TABLES

Table 2.1 List of available sensor nodes	34
Table 4.1 False positive probability with number of hops.....	74
Table 4.2 : Errors using the false positive based design	84
Table 4.3: Errors using the error expectation based design.....	84
Table 4.4: Overall table length using proposed method	85
Table 4.5 Binary image.....	101
Table 4.6 RLE image	101
Table 5.1 Relation between hops and T_h	130
Table 5.2 Parameters used in the simulation	135
Table 7.1 OV7620 registers	175
Table 7.2 Car model specifications.....	181
Table 7.3 RS-380SH-4045 motor technical data.....	183
Table 7.4 Generic packet format without source address	194
Table 7.5 Generic packet format with source address.....	195
Table 7.6 Packet command lists.....	196
Table 7.7 Obstacle packet format	197
Table 7.8 Control point packet format.....	198
Table 7.9 Token packet format	203
Table 7.10 Token messages	203
Table 7.11 Robot control commands packet format	204
Table 7.12 Robot control command fields	204
Table 7.13 Restart client program packet format	207
Table 7.14 Acquire image packet format	208
Table 7.15 Image format	208
Table 7.16 Change background image packet format	209
Table 7.17 Send data request packet format	209
Table 7.18 Data packet format.....	210
Table 7.19 Control point packet format.....	210

Table 7.20 Control point payload	210
Table 7.21 Image data packet format.....	211
Table 7.22 ACK packet format	212
Table 7.23 System status information packet format	212
Table 7.24 iMote2 system information	212
Table 7.25 A-snake control point packet format	213
Table 7.26 A-snake control point payload	213
Table 7.27 Robot coordinate packet format	213
Table 7.28 Control points display switch packet format	214
Table 7.29 Robot control mode packet format	214
Table 7.30 One generated trajectory	232
Table 7.31 Commands list of remote console GUI	241
Table 7.32 Get functions provided by communication class for GUI class.....	242
Table 7.33 Set functions provided by communication class for GUI class.....	243
Table 7.34 Gainz packet format.....	250
Table 7.35 Simulation environment function list	253

LIST OF ACRONYMS

ADC	Analog-to-Digital Converter
AI	Artificial Intelligence
A-snake	Accompanied snake
BDLC	Byte Data Link Controller
CAN	Controller-Area Network
CHK	Checksum
CMD	Command
CMOS	Complementary Metal Oxide Semiconductor
CP	Control Point
CRC	Cyclic Redundancy Check
Cspace	Configuration space
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
FIFO	First-In-First-Out
GPIOs	General Purpose I/Os
GUI	Graphic user interface
HIS	Hue Intensity Saturation
HLS	Hue Lightness Saturation
HSV	Hue Saturation Value
I2C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IF	Intermediate Frequency
iMote2	Intel Mote 2
IPC	Inter Process Communication
JTAG	Joint Test Action Group, IEEE 1149.1
LNA	Low Noise Amplifier
MCU	Microcontroller Unit
NSFC	National Natural Science Foundation of China
OTS	Off-The-Shelf
PA	Power Amplifier
PBP	Peripheral-Bus Peripheral
PCB	Printed Circuit Board
PCLK	Pixel Clock
PLL	Phased Locked Loop
PPC	Polar Polynomial Curve
PRM	Probabilistic Roadmap
PSO	Particle Swarm Optimisation
PWM	Pulse-Width Modulator
RF	Radio Frequency
RGB	Red Green Blue

RS-232	Recommended Standard 232
R-snake	Reference-snake
SCCB	Serial Camera Control Bus
SCI	Serial Communications Interface
SDIO	Secure Digital Input Output
SI	Swarm Intelligence
SPI	Serial Peripheral Interfaces
SSPs	Synchronous Serial Ports
TCP	Transmission Control Protocol
IP	Internet Protocol
UART	Universal Asynchronous Receiver/Transmitter
WiME	Wireless Mosaic Eyes

Chapter 1 : INTRODUCTION

1.1 Introduction

To date intelligent robotics researchers have mainly focused on the development of a large and smart “brain” to enable robot autonomy [1, 2]. They are, however, facing a bottleneck of complexity due to the dynamics of the unstructured environments. Steering away from this smart brain approach, the thesis investigates the use of low level intelligence, such as insect mosaic eyes, and combines them to produce a highly intelligent environment for autonomous robot navigation and control, which is named as wireless mosaic eyes.

Extensive research and development work has been carried out to achieve the wireless mosaic eyes based robot navigation in this PhD study, such as wireless sensor network, distributed mobile robot path planning, model based predictive trajectory generating, multiple view geometry, mobile motion control, and image processing etc.. This chapter will give the background of this research, raise both scientific and technical objectives, summarise the author’s contributions, give the structure of the thesis and finally the publications generated so far as a result of this PhD study.

1.2 Background

In order to achieve autonomous navigation, considerable efforts have been paid to cognitive aspects and knowledge representation that answer

some fundamental questions of “Where am I?”, “How do I get there?” and “Where have I been?”, corresponding to localisation, path planning and map-making, respectively [1]. Though human can answer these questions effortlessly, based on proper information resources and prior knowledge, massive computation and memory space are necessary for a robot. To achieve a human-mimetic brain power (100 billion neurons), a 100 million MIPS computer is required, which is far beyond today’s technology.

Moreover, if a “supercomputer” is used to implement the full intelligent system hierarchy [3] in a centralised manner for robot navigation, it should be able to deal with image processing, movement control, path planning, behaviour coordination and mission level task planning. In a dynamic and unstructured environment, the centralised approach will face difficulties in the following aspects:

- In terms of image processing and scene interpretation, the projection of a 3D world onto a 2D image plane through a pin-hole camera (camera eyes used by vertebrates) makes the motion analysis inherently ill-posed, such as the rotation and translation commingle in an inseparable way [4]. This causes difficulties for achieving efficient and reliable robot navigation;
- In terms of movement control, eye-in-head configuration introduces more uncertainties from the changing views due to movement of eyes. This makes the changes of intensity of environment and robot motion commingle in optical flow [5], which is usually the key information for robot motion control;

- In terms of behaviour coordination, current approaches need to maintain a centralised behaviour network for reactive behaviours using the subsumption architecture [2]. The difficulty appears to be that various behaviours commingle together so that modification of the network and prediction of the resulted behaviours are both difficult;
- In terms of planning, current approaches usually abstract a geometric environment into a topological map based on visual landmarks then a planning on this topological map is carried out [1]. After that, the obtained route on the topological map has to be converted back to the geometric space for real-time control. It introduces complexity and errors for linkage between planning and control;
- In terms of mission level planning, discovery of semantics from vision sensors are complicated for real-time applications [6]. A mission level planning requires enormous computation and memory space for semantics representation and interpretation.

The task often becomes unfeasible when such heavy computational burden is migrated to those light-weight commercial off-the-shelf robots. Example include intelligent wheelchairs to aid aging people, lower-level intelligent Roomba Vacuum cleaner, Sony Aibo dog and etc..

Why not release the massive computational requirement of a robot into the environment using distributed vision sensors instead of centralised and on-board vision? This thought leads us to find a distributed solution for robot navigation with pervasive intelligence, especially looking into the low level biological mechanisms. Ideally, biologically inspired autonomy should

not just mimic human intelligence. For example, in contrast to human eyes, insect eyes with smaller nervous systems are extremely creative and diverse [7]. They transmit information through the retina to the insect's brain where it is integrated to form a usable picture of the insect's environment and of activities in response to the changes in the environment. Some natural insects' brain can work in a highly efficient way that no computer can contest with today. This has attracted researchers' interests to answer how they work and what they can provide as a blueprint to build a neuromorphic system [8-10]. In bio-mimetic engineering, current works mostly focus on front-end sensing, for example, artificial compound eyes [11, 12] and neuromorphic vision systems [13, 14], and intend to integrate neuromorphic networks onto a single chip. As a profound intelligence phenomenon, sensorimotor control has also gone beyond the traditional control theory to be inspired from natural insects, such as recent reports on flight control based on artificial fly eyes [15] and robot control based on arthropod nervous systems [16]. They intend to answer how the brain, which transmits chemical signals between neurons in a relatively sluggish thousandth of a second, ends up performing some tasks faster and more efficiently than the most powerful digital processors. The secret appears to reside in how the brain organises its slow-acting electrical components [14].

The small nervous systems of insects and other invertebrates seem to be hardwired from birth. Each neuron has its own special predetermined links and functions. Even though the computation mechanisms developed in current research appear to be distributed via nervous networks, their

end products, such as a neuromorphic eye, are still implemented in a single and monolithic processor, like their bio-counterpart. Can the highly efficient neuron organisation be applied to a large scale and complex environment? In addition, current bio-mimetic navigation is still on low-level reactive behaviour control. Can the goal-driven and planning behaviours be implemented? These motivated us to develop mosaic eyes distributed in an intelligent environment to support navigation of wheelchairs for aging people. With distributed sensors and associated distributed information, the massive robot intelligence can be released to its living environment, which consists of a wireless mosaic eye network that detects robot and environment information and sends instructions to robot. It is a more realistic solution for domestic robot applications because it shifts most uncertainties involved in the decision making process of an autonomous robot to its environment and the uncertainties from a robot perspective can often be unambiguously represented in a local area by a cluster of eyes with designated topology. As a result, an unintelligent robot can receive interpretable and understandable information from the environment instead of inferring autonomously.

1.3 Aims and Objectives of the Thesis

This thesis aims to develop techniques for robot navigation under the control of wireless mosaic eyes. In the thesis, distributed path planning methodology in an intelligent environment will be investigated; a new sensor system, Wireless Mosaic Eyes, will be developed for providing ambient intelligence to coordinate a robot with its working environment. A

new communication protocol will be developed for managing topological and routing relations of the mosaic eyes. The proposed solution is expected to be a generic approach applicable to dynamic physical-system control using wireless sensor networks.

1.3.1 Scientific Objectives

- To develop bio-mimetic technologies and algorithms for achieving intelligent information processing using wireless mosaic eyes, from path planning, sensing fusion to routing

Compare to traditional monocular or binocular vision used by autonomous robot, the mosaic eyes approach appears to bring more complexity. If we refer to biological discovery of some insects, they may have tens of thousands of eyes even with a small nervous system. Therefore, the mechanism for information processing of mosaic eyes could be highly efficient relying on distributed role allocation and proper sensor routing. In this thesis, a snake mimetic path planning algorithm that makes use of the fused visions from wireless mosaic eyes will be intensively investigated in terms of system convergence and parameter optimisation. The idea is to exploit the topology and connection of snake control points to form an obstacle-free path for robots. Since autonomous intelligence is achieved by the environment rather than the robot itself, uncertainties from perception and complexity of localisation are significantly reduced. This method will bring a new attempt over the traditional path planning technologies and new research areas.

- To deploy distributed techniques and algorithms for robot visual servoing

The whole navigation process will be implemented in the intelligent environment. Each vision sensor will hold a portion of the whole planned path containing several control points. This will have the effect of combining different local views by wireless communication to achieve a global goal where vision sensors organisation protocol, topology management and vision information fusion all contribute to such a planning process.

1.3.2 Technological Objectives

- To develop wireless mosaic eyes via CMOS visual sensors and iMote2 hardware platform to achieve real time image capturing and processing;
- To develop high efficient topological map compression algorithms for distributed eyes in order to provide goal driven navigational and reactive actions for a robot;
- To construct a pilot test bed with a scenario of a wheelchair or model car moving inside a building autonomously;
- To develop a simulation software platform to test all algorithms and parameters.

1.4 Contributions of the Thesis

The mosaic eyes based intelligent environment is an international joint project sponsored by the Royal Society of the UK and the Natural Science Foundation of China. The partners are the University of Bradford, Xi'an Jiaotong University, Tongji University. With three years hard work, the key contributions claimed by the thesis as original and novel work are listed in but not limited to the following:

1. A novel snake algorithm is proposed for collision-free path planning carried out by collaboration of distributed wireless vision sensors;
2. A three-state transfer mechanism is proposed to overcome the limitation of local reaction of a snake. It provides a snake with global reset capability once the snake can no longer satisfy the curvature constraint;
3. A complete solution is proposed for integrating communication with global path planning, trajectory generation and motion control of mobile robots. The scheme taking a snake as a coordination mechanism can be applied to general motion control problems using wireless sensor networks. To the best knowledge of the author, this scheme is the first work to have a unified strategy that integrates communication with control. Previous works in navigation by a wireless sensor network either separate network route planning from robot motion control or conduct merely snake based path planning while ignoring the low level motion control;

4. An Accompanied snake (A-snake) method is originally proposed as a control scheme to guide a mobile robot approaching a reference snake. The convergence of the A-snake to the reference snake is proved, considering nonholonomic and curvature constraints;
5. Working with the A-snake together, a distributed predictive control is further developed to achieve time-optimal tracking, compliant to both kinematic and dynamic constraints;
6. A complete software simulation platform developed in Java and C++ for snake algorithm simulation, performance evaluation, intelligent environment parameter setting, system monitoring and management is developed. The software codes altogether are more than 10,000 lines exclusively written by the author;
7. A wireless sensor network intelligent navigation test bed prototype is built. A small car hardware as well as software platform are developed, including driving, steering, speed feedback and communicating modules with the collaborative work with Tongji University; a collaborative work with Xi'An Jiaotong University colleagues on integrating camera module with iMote2 main board, including hardware circuit design/realisation and device driver development; a communication protocol is proposed in the thesis to deal with robot existence discovering, control token negotiation and handover between mosaic eyes. The protocol framework also includes mosaic eyes remote control, configuration and version management data exchange etc..

1.5 Structure of the Thesis

A brief outline of the contents in the thesis is as follows:

In Chapter 2, a thorough review of the previous work on robot navigation, wireless sensor networks and Bloom filters are presented. In-depth background research is conducted in this chapter. Advantages and limitations of different techniques are described and summarised.

In Chapter 3, the system architecture of the intelligent environment supported by wireless mosaic eyes is introduced. The modules and functions required to build up the system are proposed. Information flow, communication protocols and the operation mechanisms of the system are also given. This chapter serves as a basis for the algorithms proposed in later chapters and also provides a blueprint for the system implementation chapter.

In Chapter 4, the global map building, image processing and obstacle extraction are discussed in detail, including multiple Bloom filters for map building and routing, background discrimination and colour blobs tracking for objects identification and localisation and multiple view geometry for more precise obstacles extraction.

In Chapter 5, the bio-inspired snake model, distributed snake path planning as well as the three states exchange approach is depicted to solve the distributed curvature constraint problem. Simulation results are also given to evaluate the algorithms.

In Chapter 6, an accompanied snake algorithm and an optimal rolling window mechanism are proposed for robot predictive control subject to

kinematic and dynamic constraints. Simulation and experiment results are provided to verify the proposed methods.

In Chapter 7, the hardware components selection and integration, communication protocol signalling flows and packet formats, and the software application processing flows and implementations are presented. A discussion is carried out to addressing the merits and limitations of the system.

In Chapter 8, we conclude with a summary of the contributions of the thesis and discuss about the future work.

1.6 Research Publications

[1] Y. Cheng, P. Jiang, Y. F. Hu and Z. R. Feng, "Snake Based Distributed Coordination and Control for Mobile Robot Navigation by Wireless Mosaic Eyes", Transactions of the Institute of Measurement and Control, Special Issue on "Bio-inspired Computation in Robotics", accepted.

[2] P. Jiang, Y. Q. Cheng, "Adaptive Iterative learning control for a class of Nonlinear Systems with Unknown Control Directions", The Open Automation and Control Systems Journal, Vol.1, pp.20-26, 2008.

[3] Y. Cheng, P. Jiang, Y. F. Hu, S. Brown and A. Metcalfe, "Design and Realisation of Linux Based Wireless Data Acquisition System", Communications of SIWN, Vol. 4, pp. 51-57, June 2008.

[4] P. Jiang, Y. Cheng , Y. Ji , Z. Feng , J. Zhu, "Navigation of Mobile Robots in an Intelligent Environment with Wireless Visual Sensors", 2010

IEEE International Conference on Networking, Sensing and Control, April 2010.

[5] Y. Cheng, P.Jiang and Y.F.Hu, "A-snake: Integration of Path Planning with Control for Mobile Robots with Dynamic Constraints", 2nd International Conference on Computer and Automation Engineering, vol.2, pp. 127-134, Feb, 2010.

[6] Y.Cheng, Y.F.Hu and P.Jiang, "A distributed snake algorithm for mobile robots path planning with curvature constraints", IEEE Int. Conf. on SMC, pp.2056-2062, Oct, 2008.

[7] Y. Cheng, P. Jiang and Y. F. Hu, "A snake based approach for robot path planning in an intelligent environment with distributed vision," Proc. Int'l. Conf. Automation and Computing, pp.264-269, Stafford, UK, Sep. 2007.

[8] D. Xue, P. Jiang, J. Zhu and Y. Cheng: "Self-organising diffusion protocol for robot navigation in a dynamic environment with wireless sensors", Proc. Int'l Conf. Automation and Control, Stafford, UK, Sep 2007.

Chapter 2 : LITERATURE REVIEW

2.1 Introduction

The last decade marks a remarkable progress in autonomous mobile robot navigation control. Nowadays, the use of car-like autonomous mobile robots in industrial compounds, ports, agricultural fields, etc. are quite rare. To enable autonomous mobile robot navigation, precise localisation techniques need to be applied. During the navigation process, not only does the mobile robot need to deal with predefined structure in its surrounding environment, but it also has to avoid unexpected obstacles that may block its path. Thus, it is important that the autonomous mobile robot can perceive the environment and react dynamically to unforeseen situations.

In [17], three subtasks have been defined for mobile robot motion control to enable low level path following along a relatively short distance, namely, path planning, position estimation and trajectory tracking. In this thesis, a large scale mobile robotic navigation to enable a robot moving freely in a building compound is considered. As such, a full hierarchical mobile robot control that involves path planning, trajectory generation, localisation and motion control has been investigated.

To date, research carried out in mobile robot control has mainly concentrated on centralised intelligence through the development of large and smart processing functionalities on-board the robot itself to enable autonomy[1, 2, 17-20]. While such an approach offers mention

advantages, the sole implementation of a large and smart “brain” on the mobile robot to accomplish complex tasks has several difficulties and creates a bottleneck of complexity, especially when the mobile robot has to maintain its autonomy in a dynamic and unstructured environment.

With the advent of wireless sensor network, new paradigms can be envisaged to tackle the challenges faced by traditional intelligent robotics research. As one attempt to this new paradigm, this thesis proposes an intelligent environment supported robot navigation scheme based on a distributed wireless vision sensor network that mimics mosaic eyes. The scope of the research covers path planning, trajectory generation, mobile robot motion control, wireless sensor network as well as global map building and storage.

This review starts with a survey of the path planning research methods and their merits and limitations. As the thesis will propose a biological snake inspired algorithm to perform the path planning with distributed vision sensors and coordinate them to achieve the global goal, the related works are addressed thereafter. In order to make a complete contribution on the path planning as well as time parameterised trajectory and mobile robot motion control, dynamic constraints and trajectory generation are reviewed too for a better understanding of the research background. Followed by this is a collection of the available wireless platforms and applications.

Unlike the traditional centralised robot control, the wireless mosaic eyes architecture distributes the smart processing functionalities to the wireless vision sensing nodes to provide environment intelligence to an

unintelligent mobile robot. Although the overall processing ability and storage can be vastly benefited from the scalable structure, the storage space in each individual node is limited. Given a goal, the mosaic eyes should be able to originate a route search in a building to realise the sequence of abstracted locations that can lead the robot to the goal with the shortest routes. Thus Bloom filters [21-23] are considered for encoding the physical location semantics so as to compress the topological map for global routing. Therefore, a review on different Bloom filters and their applications is given at the end of this review to assess their suitability for use in the wireless mosaic eyes architecture.

2.2 Robot Navigation

Mobile robot navigation can be achieved using either local or global path planning techniques. Local path planning can adopt reactive control techniques [1, 2, 4, 24-29] for a robot to respond and adapt to events under real-time constraints and is most suited for fast obstacle avoidance. Reactive control, also referred to as behaviour based control in some cases, basically exploits sensor-motor pair mechanism. Schema-based control and fuzzy logic behaviour fusion are two typical reactive control approaches. More complex reactive control such as topological path planning depends on the presence of landmarks. A landmark is one or more perceptually distinctive features of interest on an object or locale of interest. A landmark is not necessarily a single, self-contained object but can be a grouping of objects. Kortenkamp [30] popularized a particularly interesting special case of landmarks: gateways. A gateway is an

opportunity for a robot to change its overall direction of navigation. For example, an intersection of two hallways is a gateway; the robot can choose to go straight or turn left or right.

Global path planning techniques [5-8] is to produce a continuous motion that connects a start configuration and a goal configuration while avoiding collision with known obstacles. Global information about the environment is obtained through regular updates of sensory inputs. The robot and obstacles are described in 2D or 3D space while the path is represented in the Configuration space (Cspace). All global path planning techniques can fall into either one of two main categories: Graph Based Planning and Wavefront Based Planning. Graph based planning has many variants, including Grid-Based Search, Cell Decomposition, and Sampling Based Algorithms while Wavefront Based Planning is mainly based on the Potential Field method.

In Section 2.2.1 , Section 2.2.2 and Section 2.2.3 , more detailed review on reactive control techniques and global path planning techniques are provided.

2.2.1 Reactive Control

Reactive robotic systems originate from the cybernetic movement of the 1940s. Walter [31] developed an electronic “tortoise” capable of moving around, avoiding perceived threats but being attracted to target goals. Of special interest was the inclusion of changing goals regarding the robot’s recharging station. When the battery power was low, the tortoise was attracted to and docked with the recharger. When sufficient

energy was acquired, it lost its “appetite” (charge attraction) and was repelled by it. There was no use of abstract representational constructs as those found in traditional Artificial Intelligence (AI) techniques or perception directly controlled motor action. Only simple behaviours were created such as head towards weak light, back away from strong light, and turn-and-push to avoid obstacles.

Braitenberg [32] revived interests in this category of creatures. He demonstrated using simple analog circuitry that “creatures” manifesting behaviours comparable to those found in animals, e.g., cowardice, aggression, love, exploration, and logic could be built. The thought experiments in “synthetic psychology” showed that seemingly complex behaviour could result from a collection of simple sensor-motor transformations.

Departing from classical AI and breaking away from the sense-plan-act paradigm that dominated the AI field in the 1970-1980s, Brooks [3] pioneered the purely reactive robotic paradigm with the development of the subsumption architecture. The subsumption architecture is a layered control system decomposing complicated intelligent behaviour into simple layered behaviour modules. The goal of high layer relies on the behaviour its lower. For example, the decision to move forward by the eat-food layer takes into account the decision of the lowest obstacle-avoidance layer. The subsumption architecture was biologically motivated only in the behaviourist sense as it produced results that resembled certain insect systems but was unconcerned for the underlying biological mechanisms that produced them. Arkin [33] exploited these models using interacting

schemas as a basis for reactive robotic control system design. Beer [34], alternatively, used robotic systems to demonstrate the fidelity of neuroscientific models.

Reactive systems grew out of the dissatisfaction with existing methods for producing intelligent robotic response and a growing awareness of the importance of looking at biological system as a basis for constructing intelligent behaviour. The robots are instructed by a collection of low-level primitive behaviours. Complex physical behaviour emerges through the behavioural set defined by different technologies such as schema-based algorithm, fuzzy logic method, etc.. This approach provides more rapid and flexible response than deliberate planning. The features of reactive control can be classified as follows:

- Behaviours are basic primitives

A behaviour in these systems is usually a simple sensor-motor pair, where sensory activity consists of providing necessary information to support low-level reactive motor response, such as avoiding obstacles, escaping from predators, being attracted to goals, etc..

- Abstract representational knowledge is avoided

Creating and maintaining accurate representations of the environment is a time-consuming error-prone process. Pure reactive systems do not maintain world models. Instead they react directly to the stimuli that the world presents. This is particularly useful in highly dynamic and hazardous worlds, where the environment is unpredictable and potentially hostile.

- Animal models of behaviour are often used as a basis for these systems

Models from neuroscience, cognitive psychology, and ethologic are used to capture the nature of behaviours that are necessary for a robot to safely interact with a hostile world.

- Demonstrable robotic results have been achieved

Reactive control techniques have been applied to a wide range of robots including six-legged walking robots, pipe-crawling robots, robots for indoor/outdoor activities, mobile manipulators, dexterous hands, and entire herds of mobile robots. As these systems are highly modular, they can be constructed incrementally from bottom up by adding new behaviours to an existing repertoire. From an engineering perspective this is quite desirable as it facilitates the growth and application of existing software and hardware systems to new domains.

Many reactive systems designers look to biology as a source of models for applications in robots. A few exemplars that have affected reactive and hybrid system design are listed as below:

- Action-oriented perception

One example provided by neuroscientists and psychologists for relationships between perceptual activities and behaviours required for a particular task is presented in 1972 by Arbib [35]. His action-oriented perception model shows that an agent needs to perceive is based on its needs to act. This is a primary guiding principle in the design of reactive robots. Arkin [33] has further developed these ideas in the context of cognitive psychology.

- Ethological studies

A pressing question for reactive robotic system designers is what behaviours are necessary or sufficient for a particular task and environment. Many researchers turned to ethological studies as a source for behaviours that are relevant in certain circumstances, including bird flocking, cockroach escape and fish schooling. One example involving toad detour behaviour [36] provided motivation and justification for the use of vector fields in reactive schema-based robot navigation [33].

- Co-existence of parallel planning and execution systems

Norman and Shallice [37] modelled the co-existence of two distinct systems concerning with controlling human behaviour. One system models reactive systems and another handles deliberate conscious control and express an interface with reactive action system. Most hybrid robotic systems [3, 38, 39] incorporate both deliberative and reactive components as this model.

Hybrid architectures permit reconfiguration of reactive control systems based on available world knowledge, adding considerable flexibility over purely reactive systems. Dynamically reconfiguring the control system based on deliberation is an important addition to the overall competence of general-purpose robots. Arkin was among the first to advocate the use of both deliberative (hierarchical) and reactive (schema-based) control systems within the autonomous robot architecture. Incorporating a traditional planner that could reason over a flexible and modular reactive control system, specific robotic configurations could be constructed to integrate behavioural, perceptual and a priori environment knowledge [38]. Gat [39] proposed a three level hybrid system incorporating a lisp-based

deliberator, a sequencer that handled failures of the reactive system, and a reactive controller. This system was fielded and tested successfully on Mars rover prototypes.

2.2.2 Global Path Planning

2.2.2.1 Configuration Space

Planning a path based on a model is a problem that is fundamental to intelligent control of robot arms as well as mobile robots. Lozano-Perez [40] was the first to develop a formal version of the general path planning problem. This formalization is referred to as the “find-path” problem [41]. In its most general form, the goal of find-path is to determine a continuous path for an object from an initial location to a goal location without colliding with an obstacle.

Lozano-Perez provided a mathematical treatment of the find-path problem using the configuration space approach. The idea is to find those parts of free space which the object at particular orientations may occupy without colliding with an obstacle. Obstacles are “expanded” by the shape of an object at a set of orientations, while the object to be moved is shrunk to a point. The shortest path for the object, including rotations, is computed as the shortest connected path through the expanded obstacles.

The shortest path through obstacles generally leads through a sequence of points that are adjacent to the expanded obstacles. If there is a position error in the control of the path execution, such points can possibly result in a collision. Brooks [42] proposed an approach to the find-path problem based on modelling the free space. Brooks’ solution,

developed in a two-dimensional plane, involved fitting two dimensional “generalized cylinders” to the space between obstacles to obtain pathways in which the object may freely travel on a plane. The technique was extended to the third dimension by stacking planes.

A configuration describes the pose of the robot, and the Configuration space is the set of all possible configurations. For example:

- If the robot is a single point (zero-sized) translating in a 2-dimensional plane (the workspace), Cspace is a plane, and a configuration can be represented using two parameters (x, y) ;
- If the robot is a 2D shape that can translate and rotate, the workspace is still 2-dimensional, and a configuration can be represented using 3 parameters (x, y, θ) ;
- If the robot is solid 3D shape that can translate and rotate, the workspace is 3-dimensional, and a configuration requires 6 parameters: (x, y, z) for translation, and Euler angles (α, β, γ) ;

The set of configurations that avoids collision with obstacles is called the free space. The complement of free space is called the obstacle or forbidden region. Often, it is prohibitively difficult to explicitly compute the shape of free space. However, using collision detection to test whether a given configuration is in free space is efficient.

2.2.2.2 Graph Based Planning

Graph search algorithms appear in networks and routing problems, so they form a class of algorithms well understood by computer scientists.

However many of these algorithms require the program to visit each node on the graph in order to determine the shortest path between the initial and goal nodes. Visiting every node may be computationally tractable for a sparsely connected graph such as that derived from a Voronoi diagram [43], but rapidly becomes computationally expensive for a highly connected graph such as a regular grid. The A* search algorithm [1] is the classic method for computing optimal paths for holonomic robots where the heart of the method is the formula for measuring the plausibility of a node connecting to a goal.

Grid-based search [44] is considered by many as the most straightforward form of path planning. Once configuration space representations are introduced [40], it becomes clear that each component of Cspace is quantised, and a d -dimensional bitmap representation can be pre-computed by iterating a collision detector over all quantised configuration values. An example of this approach appears in [45], in which a bitmap is constructed by rasterising the Minkowski sums [46] of robot-obstacle pairs at discrete rotation values. A neighbourhood structure must be defined, such as the set of $2d$ neighbours for each interior element of the bitmap ("up", "down", "left", "right" in the case of $d = 2$). Once a given query is quantised, the bitmap can then be treated as a graph, which is searched using algorithms such as dynamic programming, A* algorithm, best-first, or bidirectional search, to connect the initial and goal configurations. In fact, the bitmap could also be searched using path planning methods that are based on incremental search, such as randomised potential fields [47], multiple heuristics [48], Ariadne's clew [49,

50], or RRTs [51]. In some of these works [47, 48], a lazy bitmap is actually used; collision checking is only performed as needed during the search. It is well-known that only resolution completeness can be obtained, and that for a fixed resolution, the number of samples (bitmap size) increases exponentially in dimensions.

The Probabilistic Roadmap (PRM) was introduced in [52] as a way to overcome the well-known inefficiency of dimensionality that exists in grid search; it is similar to earlier work by Glavina [53]. The primary philosophy behind the PRM was to perform substantial preprocessing so that multiple queries for the same environment could be handled efficiently. This is analogous to the bitmap precomputation in classical grid based search. First, a roadmap encoded as an undirected graph, G , is constructed in a preprocessing phase. In a query phase, G is used to solve a particular path planning question for a given initial and goal configuration. Each vertex in G represents an element of free space, and each edge represents a collision-free path between two configurations.

A PRM variant called the Lazy PRM has been proposed for the problem of answering single planning queries efficiently, as opposed to building an extensive roadmap prior to the consideration of a planning query [54]. The resulting planner is sometimes very efficient in comparison to the original PRM. This represents a shift from the multiple query philosophy of the original PRM [52] to the single query philosophy which was used in some earlier planners [45, 50].

2.2.2.3 Wavefront Base Planning: Potential Field

Wave front propagation styles of planning are well suited for grid types of representations. The basic principle is that a wavefront considers the Cspace to be a conductive material with heat radiating out from the initial node to the goal node. Eventually the heat will spread and reach the goal, if there is a way. Another analogy for wavefront planning is region colouring in graphics, where the colour spreads out to neighbouring pixels. The result is a map which looks like a Potential Field. One example of the potential field is shown in Figure 2.1.

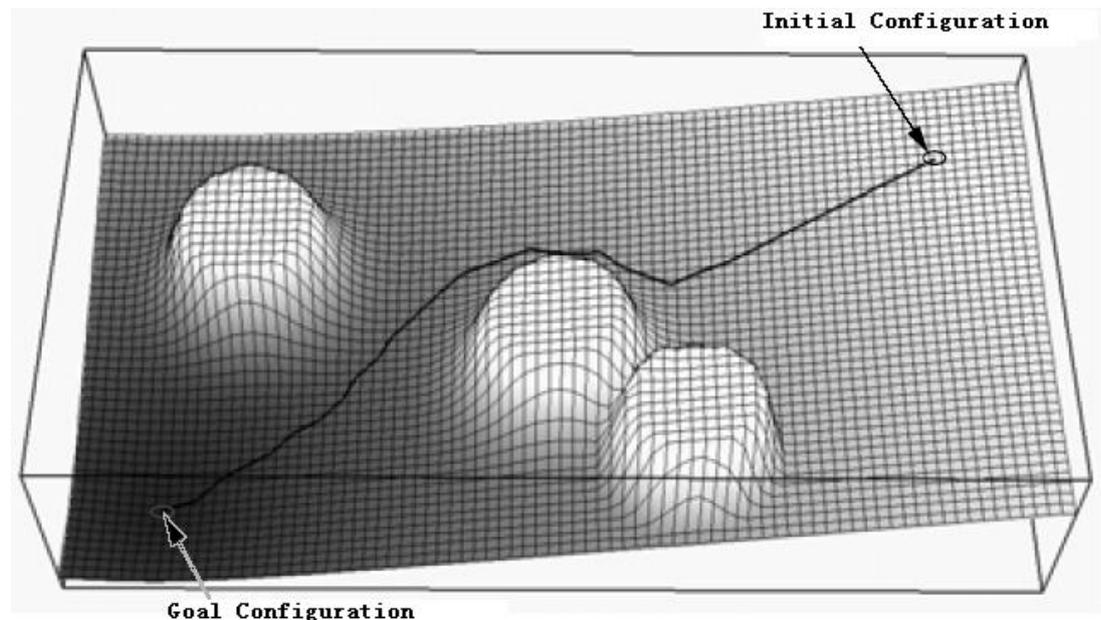


Figure 2.1 One potential field example in a configuration space. Initial and Goal configuration are indicated in the figure. Light colour indicate high potential, dark colour indicate low potential. The path starts from the initial configuration to the goal configuration is generated based on the potential gradient in the configuration.

A variety of potential field methods have been presented in the past few years. These methods have a common theme to use artificial potential fields applied to the obstacles and goal positions and use the resulting

field to influence the path of the robot which is subject to this potential. Although not as thorough as the geometric graph searching techniques, the speed of the algorithms and the easy extension to higher dimensions make them an excellent alternative to the graph searching techniques.

An early researcher who used artificial potential fields is Khatib [55]. His strategy involves the use of potential fields in real space as opposed to Cspace. Positive (repulsive) potential fields are placed around the obstacles and points subject to this potential are placed on the manipulator links. Also, negative (attractive) potential is placed at the goal position which acts on points subject to this potential located at the manipulator's end actuator. The result is that the position to be reached is an attractive pole for the end actuator and the obstacles are repulsive surfaces for the manipulator parts. The sum of the potentials is then placed into a Lagrangian formulation along with the manipulator kinetic energy to determine the end actuator equations of motion. These can be solved to determine a free path to the goal position. Since the mapping from the real space to the configuration space is not performed, the total solution time for the algorithm is greatly reduced. Also, the dimensionality of the problem is no greater than three dimensions, since it uses real space instead of Cspace. This algorithm is susceptible to local minima in the potential field, which limits its usefulness.

Other early researchers in the application of potential fields to path planning include Andrews and Hogan [56], Hogan [57], and Newman and Hogan [58]. Andrews and Hogan's work and Hogan's later works were in impedance control, which is a control scheme that considers the

interaction of the manipulator and its environment, including external and inertial forces when planning trajectories. The artificial potential fields were used to impede the movement of the manipulator in the direction of obstacles. This work differs from most others in that it is a complete manipulator controller and not just a path planner. Newman and Hogan [58] extended this to include revolute manipulators by using the Cspace of the manipulator.

However, potential field path planning has some inherent limitations. A systematic criticism of the inherent problems based on mathematical analysis was presented in [59], which includes the following: 1) trap situations due to local minima; 2) no passage between closely spaced obstacles; 3) oscillations in the presence of obstacles; and 4) oscillations in narrow passages. Besides the four problems mentioned above, there exists an additional problem: Goals NonReachable with Obstacles Nearby (GNRON). In most of the previous studies, the goal position is set relatively far away from obstacles. In these cases, when the robot is near its goal position, the repulsive force due to obstacles is negligible, and the robot will be attracted to the goal position by the attractive force. However, in many real-life implementations, the goal position needs to be quite close to an obstacle. In such cases, when the robot approaches its goal, it also approaches the obstacle nearby. If the attractive and repulsive potentials are defined as commonly used [59-61], the repulsive force will be much larger than the attractive force, and the goal position is not the global minimum of the total potential. Therefore, the robot cannot reach its goal due to the obstacle nearby.

Okutomi and Mori [62] attempted to overcome the problem of local minima in the potential field in path planning. The potential fields selected were different from those previously mentioned in that they were oval in nature and not as susceptible to local minima. Khosla and Volpe,[63], used a similar approach with the use of super quadric potential fields. The problem, however, was not eliminated. Koditschek [64] presents a rigorous description of the topological considerations of the potential fields. He also introduces potential fields that have only one global minimum and no local minima, but only for very simple geometric shapes in two dimensions. This was extended to include robots moving amidst spherically bounded obstacles in three dimensions [65].

It has been accepted that the past potential field methods are only reliable in local planning situations. Two attempts at using the best features of both graph searching and potential fields were presented by Krogh and Thorpe [66] and by Tournassoud [67]. They used the geometrical solutions for global planning and potential fields for local planning. These benefit from the global planning ability of the graph searching methods, but suffer the same shortcomings as well.

A more recent paper by Kanayama [68] presents the idea of modifying the entire path under the influence of “costs”. A combination of Voronoi diagrams in the configuration space [69] and the calculus of variation are used to select safe paths that balance the length of the path and the safety of the path.

2.2.3 Active Contour Model

The Active Contour Model, also termed “snake” because of the nature of its evolution, is a sophisticated approach to contour extraction and image interpretation. The determination of the presence of an object contour depends not only on the local image force at a specific point, but also on the properties of a contour’s shape. An active contour model, introduced by Kass et al. [70], has been used considerably and studied in the last decade. In this approach, an energy-minimising contour is controlled by a combination of the following two components: one controls the smoothness of the contour; the other attracts the contour toward the object boundary. Although the implementation of this approach is sensitive to its initial position and is vulnerable to image noise, it provides a powerful interactive tool for image segmentation and has been investigated extensively among the model-based techniques [71-74].

Caselles et al. [73] proposed a geodesic active contour model based on a level set method developed by Osher and Sethian [75]. They have proven that a particular case of the classical energy snake model is equivalent to finding a geodesic or minimal distance path in a Riemannian space with a metric derived from the image content. This means that under a specific framework, boundary detection can be considered equivalent to finding a path of minimal weighted length via an active contour model based on geodesic or local minimal distance computation.

Recently, Cohen et al. [76] proposed a global minimal path approach, based on fast marching method [74], for their active contour models. It

detects the global minimum energy path between two end points. Although this approach requires user intervention to mark some initial end points on the true boundary, it does permit a better handling of the noise than the other active contour models [76, 77]. In contrast, Gerger et al. [71] took further advantage of a priori knowledge of initial end points to design a potential window as the search constraints for their dynamic programming active contour model.

Similar concepts have been applied to path planning of centralised robot navigation with onboard sensors, such as elastic bands and bubbles [78, 79] and connected splines [80]. In general, a snake is defined as a series of control points connected one by one to represent a collision free path. At initialisation, the start and target (or the end) control points of the snake will be specified by the operator manually. Other control points between the start and target points are generated by a global searching algorithm, such as Dijkstra's search algorithm [81], to indicate an optimal path to the target location. Then the snake shape can be dynamically deformed by two kinds of forces exerted on all the control points in response to the changes in the environment, i.e. the internal forces and external forces. Internal forces are forces exerted by other control points within the snake body, such as the attraction force used to reduce the distance between two control points or the bending force used to reduce the bending of the snake body. External forces are actions generated from the environment rather than the control points. For example, the repulsive force from obstacles is an external force. The deformation of a snake body is only caused by interactions between adjacent joints and reactions to

surrounding obstacles in a certain range. Our recent work to adopt active contour model in distributed environment is reported in [82]. Each of the distributed vision sensors maintains one segment of the whole snake, and adjusts the control points in response to the detected dynamic obstacles. This pioneer work shows the suitability of the active contour model to be used in distributed path planning.

2.3 Kinematic Constraints

Obviously, the output of path planning will be the input of mobile agents which always has kinematic constraints [83]. How to satisfy mobile robot kinematic constraints when applying planned path to robot motion is a key factor to be considered during the planning phase [79]. Liang [84] presented a nonholonomic path planning method considering curvature constraint and length minimisation for a car-like robot based on cubic spirals. Winston [85] proposed two types of continuous steering functions to generate continuous curvature curves: Cartesian quintics for lane changes and polar splines for symmetric turns of arbitrary-angle. Based on the latter, Sam [83] investigated the use of this polar polynomial curve (PPC) to produce a path changing continuously in curvature and satisfying dynamic constraints. In [86], Khatib introduced an elastic band method to reduce curvature during the dynamic process of path modification, but the curvature constraint may be possibly violated.

2.4 Trajectory Generating

Various problems related to motion control of autonomous vehicles (including air, land, and marine robots) have been studied extensively in recent years. The problems addressed in the literature can be roughly classified into three groups [87]:

- Point stabilisation: the goal is to stabilise the vehicle at a given target point, with a desired orientation;
- Trajectory tracking: the vehicle is required to track a time parameterised reference;
- Path following: the vehicle is required to converge to and follow a path, without any temporal specifications.

Point stabilisation presents a true challenge to control system designers when the vehicle has nonholonomic constraints, since that goal cannot be achieved with smooth (or even continuous) state-feedback control laws, as pointed out in [88]. To overcome this difficulty, two main approaches have been proposed: smooth time-varying control laws [89, 90] and discontinuous and hybrid feedback laws[91-93].

The trajectory tracking problem for fully actuated systems is now well understood and satisfactory solutions can be found in advanced nonlinear control textbooks. However, in the case of under actuated vehicles, that is, when the vehicles have less actuators than state variables to be tracked, the problem is still a very active topic of research. Linearisation and feedback linearisation methods [94, 95] as well as Lyapunov based control laws [89, 96] have been proposed.

Path following control has received relatively less attention in the literature. See [97, 98] for the pioneering work in this area as well as [89, 99] and the references therein. Path following systems for marine vehicles have been reported in [100, 101]. The underlying assumption in path following control is that the vehicle's forward speed tracks a desired speed profile, while the controller acts on the vehicle orientation to drive it to the path. Typically, smoother convergence to a path is achieved (when compared to the behaviour obtained with trajectory tracking control laws) and the control signals are less likely pushed to saturation.

2.5 WSN Platforms

Wireless sensor network has been widely used in sensing and monitoring area, there are a wide range of successful applications, such as monitoring vital signs of patients [102] in hospital, monitoring plant growth [103] in grape garden by measuring soil moisture, light and temperature; creating virtual fencing by acoustic stimulus [104]; observing behaviour of wild animals within a spacious habitat [105] by sensing position and speed of their movements; observing the breeding behaviour of birds [106] by measuring humidity, pressure, temperature and ambient light level; monitoring wind farm surrounding environment [107]; monitoring sub-glacier environment to better understand the Earth's climate [108] by sensing pressure and temperature in at Briksdalsbreen; assisting rescue teams in saving people buried in avalanches [109] with equipped oxygen level sensor; monitoring possible break in cold chain of production [110]; measuring sniper localisation by acoustic sensors [111];

monitoring ocean water temperature, salinity and current profile [112]; tracking military vehicles [113]; monitoring power consumption in large and dispersed office buildings [114] etc.. The scope of sensors varies from temperature, sound, vibration, pressure, motion, to location. The basic components of a sensor node are microcontroller, radio transceiver, memory and battery life. A list of sensor nodes is shown in Table 2.1 [115]. From this table, iMote2 node has the potential to process real time images with its 32MB RAM, 32MB Flash and PXA271 processor.

Table 2.1 List of available sensor nodes

Sensor Node Name	Microcontroller	Transceiver	Program Data Memory	External Memory	Program language	Remarks
COOKIES	ADUC841	ETRX2 TELEGENESIS	4 Kbytes + 62 Kbytes	4 Mbit	C	Platform with hardware reconfigurability (Spartan 3FPGA based)
BEAN	MSP430F169	CC1000 (300-1000 MHz) with 78.6 kbit/s		4 Mbit		YATOS Support
BTnode	Atmel ATmega 128L (8 MHz @ 8 MIPS)	Chipcon CC1000 (433-915 MHz) and Bluetooth (2.4 GHz)	64+180 K RAM	128K FLASH ROM, 4K EEPROM	C and nesC Programming	BTnut and TinyOS support
COTS	ATMEL Microcontroller 916 MHz					
Dot	ATMEGA163		1K RAM	8-16K Flash	weC	
EPIC Mote	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		TinyOS
Eyes	MSP430F149	TR1001		8 Mbit		PeerOS Support
EyesIFX v1	MSP430F149	TDA5250 (868 MHz) FSK		8 Mbit		TinyOS Support
EyesIFX v2	MSP430F1611	TDA5250 (868 MHz) FSK		8 Mbit		TinyOS Support
FlatMesh FM1	16MHz	802.15.4-compliant		660 sensor readings	Over-air control	Commercial system, for digital sensors
FlatMesh FM2	16MHz	802.15.4-compliant		660 sensor readings	Over-air control	Commercial system, built-in tilt sensor
GWnode	PIC18LF8722	BiM (173 MHz) FSK	64k RAM	128k flash	C	Custom OS
IMote	ARM core 12 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash		TinyOS Support
IMote 1.0	ARM 7TDMI 12-48 MHz	Bluetooth with the range of 30 m	64K SRAM	512K Flash		TinyOS Support

Sensor Node Name	Microcontroller	Transceiver	Program Data Memory	External Memory	Program language	Remarks
IMote 2.0	Marvell PXA271 ARM 11-400 MHz	TI CC2420 802.15.4/ZigBee compliant radio	32MB SRAM	32MB Flash		Microsoft .NET Micro, Linux, TinyOS Support
Iris Mote	ATmega 128	Atmel AT86RF230 802.15.4/ZigBee compliant radio	8K RAM	128K Flash	nesC	TinyOS, MoteWorks Support
KMote	TI MSP430	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		TinyOS and SOS Support
Mica	ATmega 103 4 MHz 8-bit CPU	RFM TR1000 radio 50 kbit/s	128+4K RAM	512K Flash	nesC	TinyOS Support
Mica2	ATMEGA 128L	Chipcon 868/916 MHz	4K RAM	128K Flash		TinyOS, SOS and MantisOS Support
Mica2Dot	ATMEGA 128		4K RAM	128K Flash		
MicaZ	ATMEGA 128	TI CC2420 802.15.4/ZigBee compliant radio	4K RAM	128K Flash	nesC	TinyOS, SOS, MantisOS and Nano-RK Support
Mulle	Renesas M16C	Atmel AT86RF230 802.15.4 / Bluetooth 2.0	31K RAM	384K+4 K Flash, 2 MB EEPROM	nesC, C	Contiki, TinyOS, lwIP: TCP/IP and Bluetooth Profiles: LAP, DUN, PAN and SPP Support
NeoMote	ATmega 128L	TI CC2420 802.15.4/ZigBee compliant radio	4K RAM	128K Flash	nesC	TinyOS, SOS, MantisOS, Nano-RK and Xmesh Support, Industrial end-use product.
Nymph	ATMEGA128L	CC1000		64 kB EEPROM		MantisOS Support
Redbee	MC13224V	2.4 GHz 802.15.4	96 KB RAM + 120KB Flash		GCC (see mc1322x.dev1.org), IAR	Contiki; standalone
Rene	ATMEL8535	916 MHz radio with bandwidth of 10 kbit/s	512 bytes RAM	8K Flash		TinyOS Support
SenseNode	MSP430F1611	Chipcon CC2420	10K RAM	48K Flash	C and NesC	GenOS and TinyOS Support
SunSPOT	ARM 920T	802.15.4	512K RAM	4 MB Flash	Java	Squawk Java ME Virtual Machine
Telos	MSP430		2K RAM			
TelosB	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		Contiki, TinyOS, SOS and MantisOS Support
Tinynode	Texas Instruments MSP430 microcontroller	Semtech SX1211	8K RAM	512K Flash	C	TinyOS
T-Mote Sky	Texas Instruments MSP430 microcontroller	250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver	10k RAM	48k Flash		Contiki, TinyOS, SOS and MantisOS Support
weC	Atmel AVR AT90S2313	RFM TR1000 RF				

Sensor Node Name	Microcontroller	Transceiver	Program Data Memory	External Memory	Program language	Remarks
Wireless RS485	Atmega 128L	Chipcon CC2420 + Amplifier 250 kbit/s 2.4 GHz IEEE 802.15.4	4k RAM	128k Flash		Xmesh, TinyOS
XYZ	ML67 series ARM/THUMB microcontroller	CC2420 Zigbee compliant radio from Chipcon	32K RAM	256K Flash	C	SOS Operating System Support
FireFly	Atmel ATmega 1281	Chipcon CC2420	8K RAM	128K FLASH ROM, 4K EEPROM	C	Nano-RK RTOS Support

2.6 Bloom Filter

A Bloom filter is a simple space-efficient randomised data structure for representing a set in order to support membership queries. Bloom filters allow false positives¹ but the space savings often outweigh this drawback when the probability of an error is made sufficiently low. False negative is not possible in Bloom filter. Burton Bloom introduced Bloom filters in the 1970s [116] with an application to hyphenation programs. They are an efficient yet lossy way of describing membership of elements belonging to a set. They are broadly used for compressed presentation of sets in many fields, such as database applications, network applications, packet routing and etc.. One simple example of Bloom filter is shown in Figure 2.2.

¹ False positive, known as a Type I error in statistics, the error of rejecting a null hypothesis when it is actually true, an example of this would be if a test shows that a woman is pregnant when in reality she is not; false negative, also known as a Type II error, the error of failing to reject a null hypothesis when it is in fact not true, an example of this would be if a test shows that a woman is not pregnant when in reality she is.

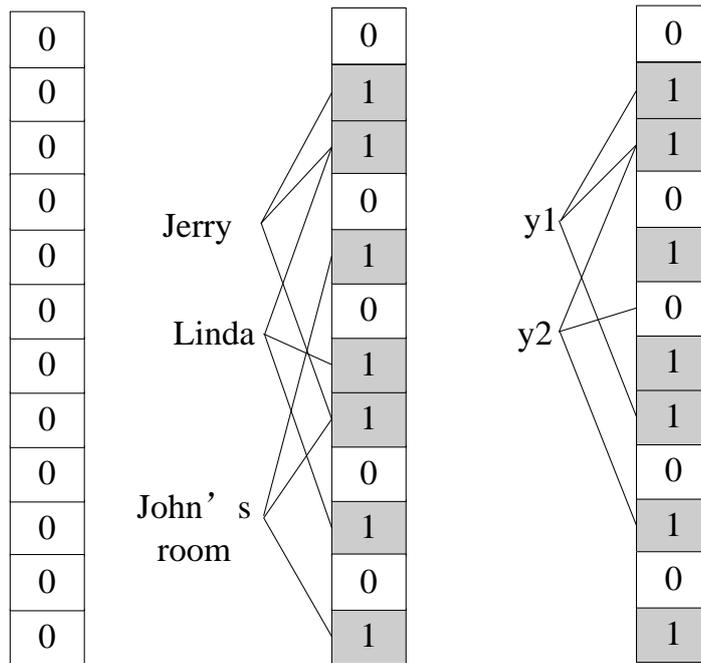


Figure 2.2 An example of a simple Bloom filter. The filter begins as an array of all 0s. Each item Jerry, Linda and John's room in the set is hashed a certain times, with each hash yielding a bit location; these bits are set to 1s. To check if an element y is in the set, hash it the same times and check the corresponding bits. The element y1 is either in the set or the filter has yielded a false positive; the element y2 cannot be in the set, since a 0 is found at one of the bits.

The most important area of Bloom filters applications is for dictionaries storing. Rather than storing and searching a dictionary, a Bloom filter representation of the dictionary was stored for spell-checkers such as [117, 118]. In this application, the Bloom filter offered significant performance advantages in space savings which are required by wireless sensors due to limited on board memory.

Bloom filters also found very useful to reduce the communication traffic. One example in its database application is to speed up semi-join operations [119-121]. For example, suppose that one wanted to determine the employees of a business that live in cities where the cost of living is

greater than 50,000 pounds. In a distributed database, one host might hold the information regarding the cost of living, while another might hold the information regarding where employees live. Rather than having the first database sending a list of cities to the second, the first could send a Bloom filter of this list. The second host can then send a list of potential employee/city pairs back to the first database, where false positives can be removed. This has the potential to reduce overall communication between the two hosts. Another example for the potential to reduce message traffic is the Summary Cache proposed by Fan, Cao, Almeida, and Broder [122], which uses Bloom filters for Web cache sharing. In their setup, proxies cooperate in the following way: on a cache miss, a proxy attempts to determine if another proxy cache holds the desired Web page; if so, a request is made to that proxy rather than trying to obtain that page from the Web. In Summary Cache, to reduce message traffic, proxies do not transfer URL lists corresponding to the exact contents of their caches, but instead periodically broadcast Bloom filters that represent the contents of their cache. If a proxy wishes to determine if another proxy has a page in its cache, it checks the appropriate Bloom filter. By using the Bloom filter, right web service requests are made based on periodically maintained network resource information which reduces messages traffic significantly.

Another application inspired the thesis to adopt Bloom filter to the global map building is their resource routing applications. A general framework that highlights the main idea of resource routing protocols was described by Czerwinski et al. as part of their architecture for a resource discovery service [123]. Suppose that there is a network in the form of a

rooted tree, with nodes (the leaves) holding resources. Resource requests starting inside the tree head toward the root. Each node keeps a unified list of resources that it holds or that are reachable through any of its branches, as well as individual lists of resources for it and each branch. When a node receives a request for a resource, it checks its unified list to make sure that it has a way of routing that request to the resource. If it does, it then checks the individual lists to find whether it holds the resource or how to route the request toward the proper node via an appropriate child node; otherwise, it passes the request further up the tree toward the root.

This rather straightforward routing protocol becomes more interesting if the resource lists are represented by Bloom filters. The property that a union of Bloom filters can be obtained by bitwise OR operation on the corresponding individual Bloom filters, allows easy creation of unified resource lists. False positives in this situation may cause a routing request to go down an incorrect path. In such a case backtracking up the tree may be necessary, or a slower but safer routing mechanism may be used as a back-up. Several recent papers utilise a resource routing mechanism of this form.

Thus Bloom filters are considered for encoding the physical location semantics so as to compress the topological map for global routing in the intelligent environment.

2.7 Summary

This chapter reviewed related work on distributed robot navigation.

Traditional path planning techniques such as graph searching, artificial potential fields and behaviour based reactive control assume a centralised implementation. However, navigation techniques assisted by an environment with distributed information intelligence bring new opportunities and challenges for robot navigation and control, which need to be reconsidered and developed.

From the review above, new techniques should be employed instead of traditional path planning approaches for path planning in a distributed vision sensor network. A bio-inspired snake algorithm could be a suitable technique for distributed implementation since the planning uses only local information exchange between two adjacent control points.

In association with snake based path planning, the trajectory generation and robot motion control also need to be reconsidered due the unsuitability of the current algorithms for distributed implementation, thus an A-snake algorithm will be proposed in this thesis to produce a complete solution for robot navigation in a distributed intelligent environment.

A Bloom filter is a simple space-efficient representation of a set or a list that handles membership queries. As can be seen from the survey, Bloom filters have been successfully applied in resource routing, storing space savings and communication traffic reducing areas. Thus the multiple Bloom filters using error expectation as the design criterion to store the global map with high compression rate are presented in this thesis.

Chapter 3 : SYSTEM ARCHITECTURE OF AN INTELLIGENT ENVIRONMENT WITH WIRELESS MOSAIC EYES

3.1 Introduction

In the classical-architecture of autonomous mobile robots with on-board centralised intelligence, localisation has always been one of the difficult tasks. A mobile robot, as its name implies, does not work in a structured environment, like an industrial manipulator does. As a premise for robot navigation, a robot has to identify its location in an unstructured and dynamic environment, using more than one type of sensors to localise itself [124], such as ultrasonic sound sensors, laser scanner, vision sensors along with a compass. Wheel slip, parameter shift, and collisions etc. are making localisation extremely uncertain and complicated. Therefore, robot must possess immense computation power to carry out the localisation tasks by means of sensor fusion and inference. Control of a mobile robot in such an environment also encounters similar implementation difficulties. Even if the subsumption approach [3] is adopted to decompose complicated intelligent behaviour into many simple behaviour modules and organised layers, computational complexity does not decrease significantly.

Moreover, if different types of mobile robots coexist in a shared environment, different software for each robot has to be carefully designed to consider issues of collaboration. Mutual interference between active

sensors and signals from different robots also create difficulties for robot control, especially when ultrasonic sensors are used [125].

Besides, if sensors are directly connected to a centralised computer unit which processes all data from sensors on-board, the incrementally learned spatial knowledge can be outdated due to the changes, e.g. maps, cannot reflect the changes if the target environment's shapes and size vary [126]. Therefore, an intelligent environment with distributed wireless vision sensors is proposed for robot navigation control.

With a static camera configuration mounted in the environment rather than on a mobile robot, the intelligent environment with distributed processing ability will facilitate both localisation and control. This architecture not only reduces the sensing complexity, but also releases the massive computation into distributed sensors, with low processing speed and less memory [127]. Each vision sensor covers one area of the whole space. Vision sensors are connected wirelessly forming a Wireless Sensor Network (WSN), which is power consumption sensitive [128]. As a result, the low power IEEE 802.15.4 protocol is selected to exchange information between vision sensors to share their observations in the environment.

Different from conventional centralised computer, distributed Intelligent sensors, however, with the ability to process data close to the source and the flexibility to be placed anywhere with bi-directional communications connecting each other, can adapt themselves easily to work in an unstructured and dynamic environment. Since each distributed sensor only processes the information under its coverage, computation demand of individual sensors will not increase even if the environment space

shapes change or sizes grow. In addition, mutual interference among sensors implemented in a centralised multi-sensor platform, such as mutual interference between acoustic sensors, will disappear since only vision sensors are chosen. Furthermore, better localisation result can be achieved by fusing more than one vision sensors input in overlapping sensor coverages. As the robot itself is not involved in the process of navigation or localisation, wheel slippage problem in localisation processing no longer exist.

This chapter provides a system level introduction of the wireless mosaic eyes infrastructure and how it is used to support robot navigation method. An outline of the proposed intelligent environment architecture, descriptions of each and every system functions and modules, organisation and connection of the system components, the general idea of the proposed algorithms are described. This chapter focuses on proposing the distributed system architecture, which tries to solve the heavy computation bottle neck faced by the traditional centralised method.

3.2 Mosaic Eyes and Intelligent Environment Architecture

The nervous systems of low level insects and invertebrates are hardwired from birth. Each neuron has its own special predetermined links and functions. This might be one reason why lower insect with smaller nervous systems are so creative and diverse [7]. Inspired by the biological insect nervous systems, the proposed mosaic eyes intelligent infrastructure is a large-scale camera network with wireless vision sensors

deployed in the environment, resembling that of insect eyes connected by the nervous system, which observe the environment and control the motion of robots directly, as shown in Figure 3.1. Localisation and control functions are shifted from the robot processor into the distributed vision sensors.

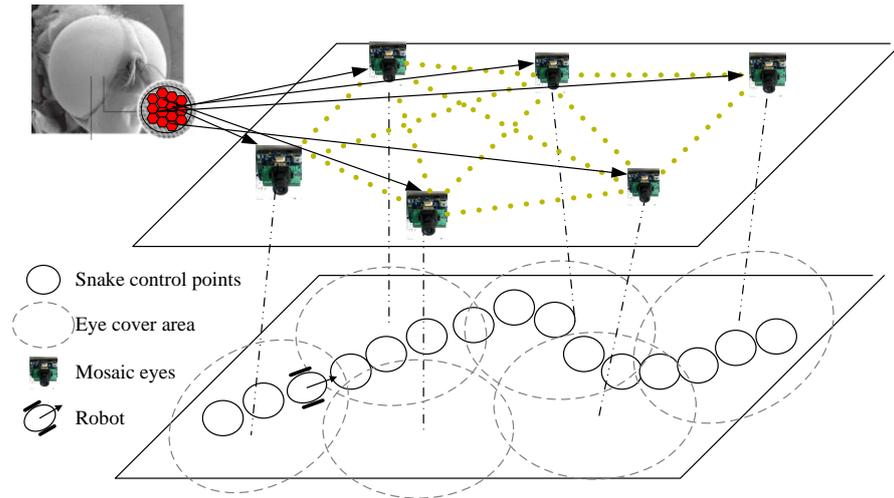


Figure 3.1 Mosaic eyes inspired intelligent environment

In order to guide a robot, a snake algorithm utilising the information captured by the distributed wireless vision sensors is developed for path planning. The advantage of adopting a snake scheme is that once the start and target positions of the robot are specified, a path can be initialised globally with multiple path segments connected one by one in a serial manner through the vision sensors. A path segment, which is made up of a sequence of connected Control Points (CPs), forms a collision free path within the coverage of a vision sensor node. By connecting the different path segments, a reference collision free optimum path, hereafter referred to as the Reference-snake (R-snake) path is created and maintained by the networked vision sensors. When obstacles appear or

disappear or the positions of the obstacles change in the view of vision sensors, the R-snake will deform its shape accordingly and contract to its minimum length in order to keep all the control points away from the obstacles. Due to sensor sensing errors, position calculation errors and robot mechanic movement errors, the robot's position may diverge from the R-snake path, causing a deviation of the robot navigation path from the optimum R-snake path. To compensate for such position deviation errors, robot trajectory tracking is performed using an Accompanied-snake (A-snake) that starts from the current robot position and follows as closely as possible the R-snake path for local trajectory tracking and mobile robot motion control.

Since overlapping areas between vision sensors exist, a negotiation procedure has to be activated to elect a dominant vision sensor and that sensor will send its planned motion commands to the robot for actuating. The exchange of information among mosaic eyes is carried out between neighbouring eyes through wireless communications. At any specific time, there is one and only one dominant mosaic eye to collect and fuse the information from its neighbouring eyes. The fused information will be used to deform the segment of the R-snake in this mosaic eye view and generate an A-snake to accompany the robot and to send motion instructions to control the robot movement. Details of the R-snake and A-snake algorithms are further described in Chapter 5 and Chapter 6 respectively.

The mosaic eyes supported robot navigation architecture involves three physical system components:

- the **remote console** responsible for all offline configurations, system status monitoring, mosaic eyes controlling and files transferring service;
- the **networked wireless vision sensors** (or mosaic eyes) are the main bodies to implement algorithms and functions including image processing, localisation and robot navigation;
- the **mobile robot** itself which can receive motion control signals from the intelligent environment via its IEEE 802.15.4 communication peripheral, drive and steer in response to these commands.

Figure 3.2 shows the interactions between these three physical components, their functional modules, processing flows and connections. The link between the remote console and vision sensors works in offline mode for transmitting system configuration data; the links among vision sensors and between vision sensors and the mobile robot are online mode for real time information exchanges. There is no direct communication link between the mobile robot and the remote console is involved.

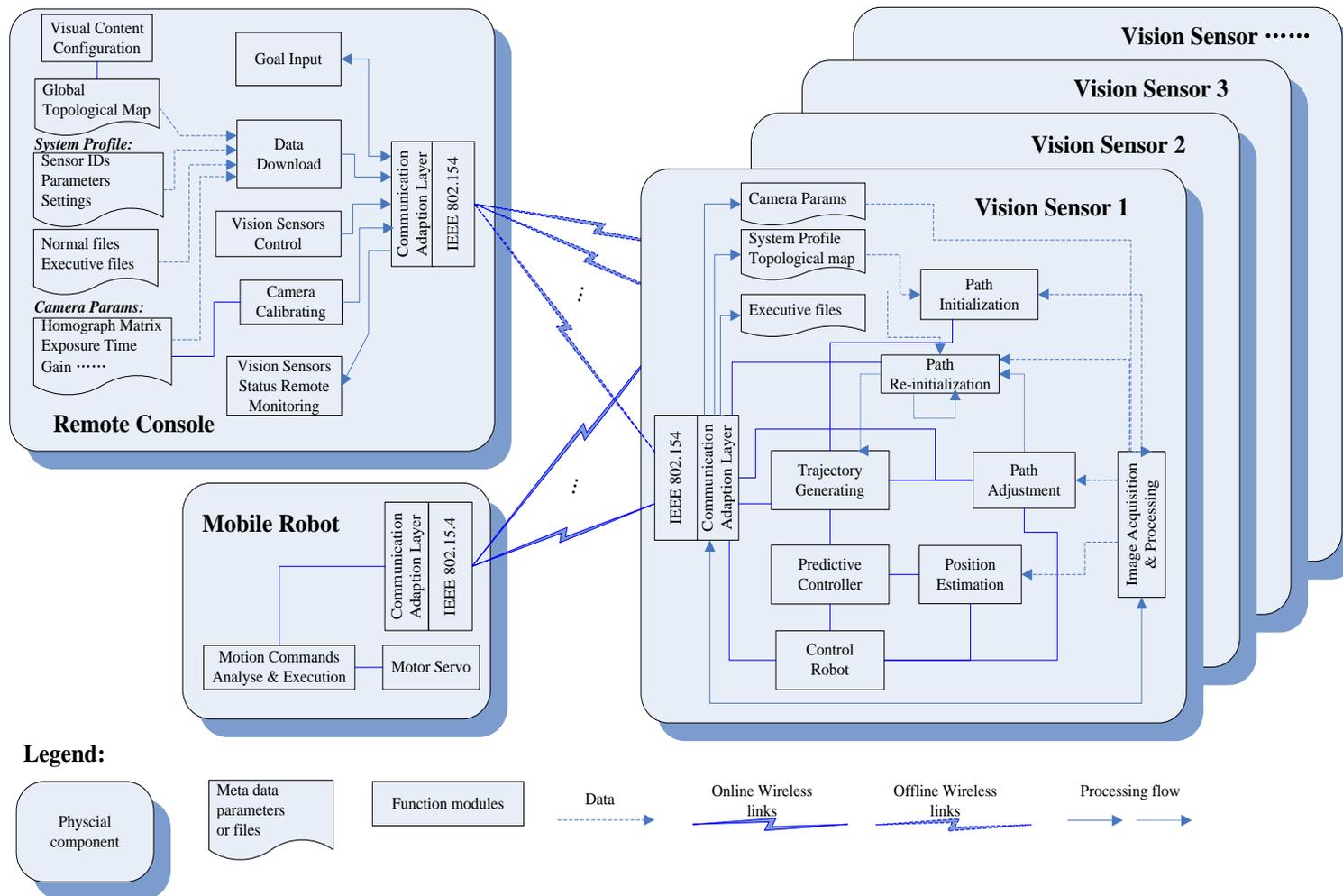


Figure 3.2 Overview of system components and functional modules

An area of consideration of this approach is its complexity in comparison with the traditional onboard navigation. This is analogous to comparing the biological phenomena of insects with very small brains but having many eyes with that of human with binocular vision but a big brain for processing information. The information processing functions of an insect compound eye could be highly efficient subject to proper role allocation and proper information routing mechanism. Therefore, the success of such a distributed vision sensor architecture relies on providing each sensor with unambiguous semantics, predetermined roles and coordinating links with other sensors. Vehicles with little intelligence are expected to perform superior intelligent mobility under the control of the mosaic eyes. This PhD study forms a part of the Wireless Mosaic Eyes (WiME) project, which is supported by the Royal Society and National Natural Science Foundation of China (NSFC). Original contributions in this thesis include the following:

- localisation using mosaic eyes;
- distributed snake path planning and motion control;
- global map building and routing;
- vision sensors and mobile robot hardware platform integration.

3.3 System Components

3.3.1 The Remote Console

The Remote Console can be either a Windows based or Linux based PC plus an IEEE 802.15.4 hardware module which is attached with the PC

via USB to enable its wireless communication ability. The remote console is responsible for system configuration, global topological map configuration, data download, vision sensors control, mosaic eyes calibration and human-machine interaction. In order to carry out these functions, a suite of software modules are implemented in the remote console. In addition, an application layer protocol stack, Communication Adaptation Layer, is designed on top of the IEEE 802.15.4 MAC&PHY layer to exchange control signals and user specific data etc. with other components.

A screen snapshot generated by the console software suite is shown in Figure 3.3, on which data from four vision sensors are displayed simultaneously. The drop down menu shows the available functions, such as re-sample background frames and update blob information, etc..

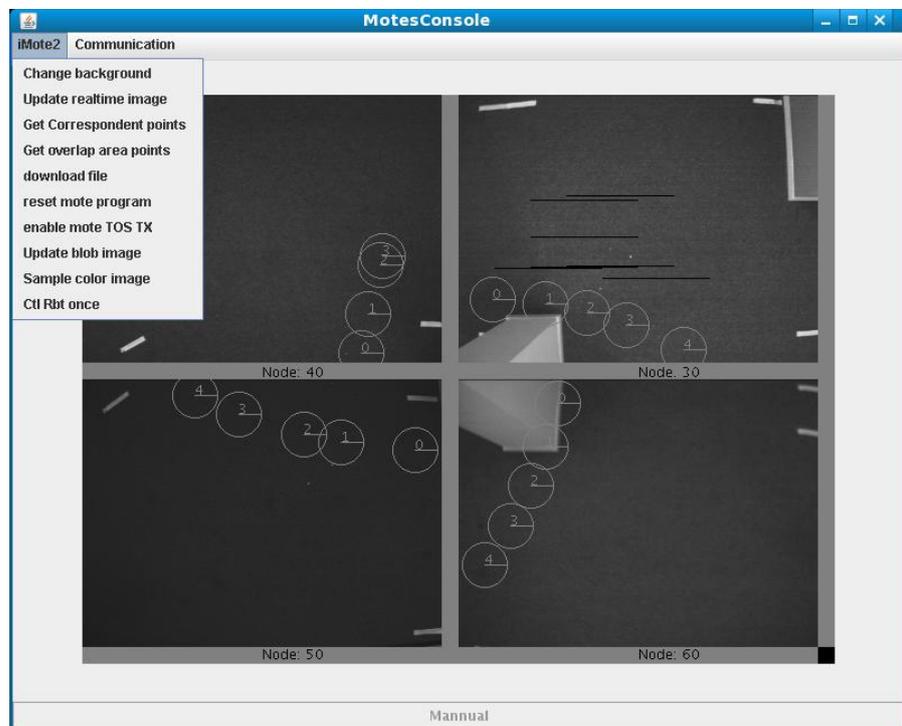


Figure 3.3 Snapshot of the remote console GUI interface

The console software features as a configuration tool but is also responsible for mapping the global information onto local data via calculated routing tables and acts as a repository centre for storing and distributing profile files. Once initial calibration and configuration functions are carried out, profile files that contain all necessary parameters, routing tables and data will be generated for and launched onto individual vision sensors. Specifically, the files fall into following categories:

- The **Global Topological Map** is generated by Vision Content Configuration module. It consists of many hashed binary tables. Each table represents a set of routes connecting one vision sensor to other reachable ones. In another word, the map is split into smaller portions according to the different node IDs which are short addresses (0~65535) to identify vision sensors. Each portion forms a part of the global map but is not a complete one. Portions will be downloaded to different vision sensors based on the node ID by Data Download Module;
- The **System Profile** describes all necessary settings and parameters for vision sensors running. Sensors will need this system profile to set default variables value in the program and algorithms, e.g. coefficients, thresholds, sampling intervals, expected dimension of robot, whether obstacles are captured automatically or set by profile file, maximum allowed car speed, acceleration limitation, maximum steering torques etc.. These profile files have exactly the same entries but the item values may be different to achieve different performance. Each profile file is dedicated to one vision sensor distinguished by vision sensor

node ID and should not be mixed with others because it also contains some hardware dependent information such as MAC address and radio frequency;

- **Normal files** and **executive files** in the vision sensors are also managed by remote console. The remote console acts as a repository centre of all software libraries, upgrades, and drivers which are running in the vision sensors. During the system integrating phase, the camera driver (OV7620.ko), the IEEE 802.15.4 driver module (CC2420.ko), the general video module (VideoDev.ko) are modified and downloaded to the vision sensors. Rather than taking vision sensors off the building and connecting them to a debugging PC by cables for file transferring, downloading them by remote console PC wirelessly increases efficiency significantly. Similarly, during the algorithms implementation phase, the user programs are first compiled and linked as executive files in the remote console and then downloaded to vision sensors for launch;
- The **Camera Parameters** are cameras related data in the vision sensors. They are marked with node ID and maintained separately for each of the vision sensors. All these camera hardware or position dependent parameters are generated by Camera Calibrating module and kept here. Such parameters include camera exposure time, saturation gain, blue and red gain, colour thresholds in YUV colour space (details are in Chapter 4) for object identifying and tracking; the projection mapping matrix from one camera to another one, which will be used to convert coordinates from one vision sensor's view to

another by geometry homograph [129]; overlapping areas between the vision sensor and others; the ratio of how many meters on the floor (work plane), on which robot moves around, equal to one pixel in the image (image plane).

Details of the function modules are shown in Figure 3.2 and depicted in the following subsections.

3.3.1.1 Visual Content Configuration Module

The visual content configuration module provides the tools for creating a topological map with wireless sensor geographic locations. It links a geometric map with a sensor topological map. Each wireless sensor is provided with a node ID and clear semantics, such as “it is John’s office connecting to the D wing corridor”. Optimal routes in the geographic space are obtained by offline searching in the constructed topological map. It is a complete set of all possible connections from one vision sensor to others with semantics. The routes with semantics representations are then converted into hash tables by a set of hash functions in order to compress the geographic routes by multiple Bloom filters [22, 130] global route searching algorithm which will be discussed in detail in Chapter 4. The routing tables are split into smaller portions based on vision sensors such that each vision sensor has all the routing tables containing all reachable destinations start from it.

Finally, all smaller portions of the global topological map are stored in remote console along with specific vision sensor node IDs as input to Data Download module.

3.3.1.2 Data Download Module

The Data Download module is responsible for reading local files and data in the remote console disk and transferred them to respective vision sensors by the request of operators. All four categories of data are downloaded during sensors' run time and they will exist in vision sensors permanently. The communications between remote console and vision sensors are IEEE 802.15.4 unconnected connections. Before transferring files, they are fragmented into small packets and are marked by serial numbers. These packets are further fed to the Communication Adaptation Layer for data format conversion and transmission. After that, all formatted data are sent out through wireless connections. With the help of GUI interface shown in Figure 3.3, the operator is able to choose which file to download to which vision sensor.

3.3.1.3 Vision Sensor Control Module

This module is designed to send control signals to vision sensors in case the operator needs to interfere with the system during running time. This module enables the operator to re-sample the background image, to fetch the planned path, to inspect the foreground image, to restart the vision sensor user program or to manually send robot control commands. All these signals are distinguished by the type field in the header of Communication Adaptation Layer packet.

3.3.1.4 Vision Sensor Status Remote Monitoring Module

As a response to control signals sent by vision sensors control module, vision sensors send related information back to the remote console. In

addition, it is important to know the working status and algorithm outputs especially when software debugging is required. This module is designed to monitor the status of individual vision sensors through console display. Each vision sensor has its dedicated display area in the console.

The data are received and categorized into individual display area by the Control Adaptation module which is a proposed application layer protocol on top of the IEEE 802.15.4 stacks. By default, when the robot enters one mosaic eye's coverage, the real time robot location and its A-snake coordinates are displayed. Other information such as real time R-snake coordinate, background image or foreground image can be received and displayed by sending responsive commands.

3.3.1.5 Camera Calibrating Module

Due to the manufacturing process whereby the Complementary Metal Oxide Semiconductor (CMOS) sensors in the camera experience random asymmetries during the fabrication process and maybe in the packaging process, the pixel colours of image captured by different cameras on the same object may be interpreted differently even the exposure time and gains are identical. One function of this module is to sample targets and filter the desired colour thresholds for each individual camera by controlling the vision sensors.

According to multiple geometry homography theory, projection matrix from one view to another view can be decided by four correspondent points in the overlapping area as long as the four points are not in a line. All vision sensors are mounted in the building in the manner that they have

overlapping areas with their neighbouring sensors. By sending control commands to sensors via the Vision Sensors Control module, frames from neighbouring vision sensors are captured. Four correspondent points in each of the cameras are selected to calculate the homography matrix. Meanwhile, overlapping areas are recorded in the Camera Parameters files.

The last function of this module is to calculate the “meter per pixel” ratio. This calibration is done with Vision Sensors Control module and the mobile robot. In vision sensors, a certain kind of movement patterns is set in advance, such as go one meter ahead, steer 45 degree and etc.. This can be triggered by the Vision Sensors module from the remote console. With this preset commands and the associated robot movements measured in the image plane, the “meter per pixel” is then calculated.

3.3.1.6 Goal Input Module

This module accepts navigation goals from either remote console itself or IEEE 802.15.4 compatible terminals. The goal is analysed and sent automatically to the nearest vision sensor for route querying.

3.3.2 Vision Sensors

Vision sensors are the brains of the intelligent environment. Each sensor covers a certain area of the building and provides service within that region. They are fixed and “hardwired” by communicating with each other thus form an infrastructure mimicking mosaic eyes and nerves. Although a single vision sensor only equips with a maximum 416MHz processor, 256KB SRAM, 32MB SDRAM and 32MB flash, the “brain”

formed by multiple vision sensors can compete the most powerful computer available today or even more. That is the main attraction point of the proposed idea.

As the main component in the proposed architecture, vision sensors have the largest range of functional modules, shown in Figure 3.2, from single view image acquisition and processing to multiple views projections and mapping; from path initialisation, adjustment, trajectory generating to eventually robot motion control; from wireless packets receiving, sending and analysing to all kinds of interested data reporting. The process requirement and time consumed also vary. To meet the real time performance as well as messages waiting and responding requirement, the functions are fitted into three program threads in the developed software which is running on the Linux based Kernel in IMote2 main board [127]. They are communication, image processing and path planning threads. Communication thread is a blocked thread waiting for data. It is unblocked on any available data received. Image processing has the longest processing cycle. Once a cycle is completed, results are stored in a shared memory for other threads usages. Path planning thread needs to maintain its real time performance to control robot while keeping its knowledge about the dynamic environment up to date, therefore, two processing loops are designed within the thread, one generates control commands by prediction, the other updates R-snake once real time information becomes available.

As discussed in Section 3.3.1 , all generated files including executive files, system profile file, global topological map and camera parameters

are downloaded and stored in vision sensors. They are organised in the vision sensors as following,

- **Executive files** like driver modules (VideoDev.ko, OV7620.ko and CC2420.ko) are loaded when the Linux kernel starts up. The main programs (ClntStart.o and mClnt.o) which realise all functional modules, start after all drivers are initialised properly;
- **System profile files** and **global topological map** are stored in flash as files and loaded into program RAM or program variables during main programs initialisation;
- **Camera Parameters** are read by Image Acquisition and Processing module in main programs and used to update the values of program variables.

Functional modules and their input & output relationships are discussed as below,

3.3.2.1 Image Acquisition and Processing Module

The Image Acquisition and Processing module takes input from the Camera Parameters file, which is received from remote console. It senses the environment, looks after the real time information such as robot location and obstacles coordinates. Then all these data are shared with the path planning modules such as the Path Initialisation module, the Path Adjustment modules and the Position Estimation modules. The background and foreground images in the memory are also ready for fetching by remote console via the Communication Adaptation Layer.

There are two classes of algorithms involved in the overall image processing: online and offline algorithms. The offline algorithm will sample the colour thresholds of the targets and generate colour lookup tables. The Visual Content Configuration module in remote console is taking care of this. On the other hand, the entire image processing functions mentioned here are within the scope of online algorithm, including capturing real time images, sampling background frame, subtracting foreground image to background image to get dynamic information, segmentation of interested colours, identifying and tracking objects.

The data obtained in this module are real time obstacles coordinates and robot location and direction. They are put in shared buffers when processing is completed and completion indications are sent to notice other modules such as Path Adjustment module.

3.3.2.2 Path Initialisation Module

When the program starts with default navigation goal set in the system profile file or a goal of the navigation is passed to vision sensor via remote console, the R-snake algorithm is initialised. The first vision sensor will check its own global topological routing table, and send route queries to the nodes connected to it to confirm or find the destination, then a searching within its coverage for an optimal R-snake segment is carried out to plan a set of control points connected in series. Real time obstacles information is taken into account during the searching algorithm.

Snake control points are represented by Euclidian coordinates in a Configuration space (Cspace), as $(x_i, y_i) \in R^2$, where i is the sequence number of the i^{th} control point.

3.3.2.3 Trajectory Generating Module

This module plans the space position of A-snake based on the R-snake which comes from the Path Initialisation module or the Path Adjustment module or the Path Re-initialisation module. The start point of R-snake does not reflect the actual position of the robot; hence A-snake is generated whenever a new robot position is available to deal with this situation. A key issue in this step is whether the A-snake can converge to the R-snake path without violating the steering torque and driving force limitations. One implication here is that this module will do nothing if robot is not appeared in view.

Once the planned result is ready, it is written to the shared buffer for the Predictive Controller module described below and to Communication Adaptation Layer for the remote console to display.

3.3.2.4 Predictive Controller Module

As the name shows, a predictive technology is employed in this module to achieve a high performance tracking speed up to robot driving limit.

The Communication Adaptation Layer always keeps an eye on the information in the air. This includes the control points from neighbouring vision sensors. By combing the information from neighbouring nodes,

vision sensor is enabled with the knowledge out of its view and hence a possible better prediction. A rolling window is set up based on the combine A-snake, and a time optimisation problem is tackled to generate series of robot control commands including driving force and steering torque with specific associated time.

This module processes both information from Trajectory Generating module and Position Estimation. The former has larger time interval than the latter since Trajectory Generating only processes the observed robot position while Position Estimation makes forecast locations for robot. It is a balance of precision and dense control.

3.3.2.5 Control Robot Module

There is a robot dominant token negotiation procedure involved in this module. All vision sensors are trying to control the robot if it is presented. If an area is only covered by one eye, the sensor has the dominant control right straight away; if the area is covered by more than one sensor, they all compete for a token to become the dominant eye to ensure that there is one and only one vision sensor controls the robot. Competition is initiated in random, and once the token is with a sensor, it will send out signals regularly to indicate its ownership of the token. When a robot approaches the edge of its coverage, a token handover request message will be sent out. A new owner of the token will be yielded as a result.

Once the ownership of the token is clear, the vision sensor will send the planned series of commands to Communication Adaptation Layer for wireless transmission.

3.3.2.6 Position Estimation Module

Both obstacles' and the robot's positions are estimated based on the passed time and the calculated moving speeds.

In this estimation -> predictive -> control -> estimation processing loop, the space positions of the R-snake and A-snake are assumed to be unchanged and the robot is following the desired trajectory blindly. During this time the module predicts the movement of the real robot without feedback from the visual observations. By skipping the image processing, this processing loop can generate control commands rapidly. However, the robot may deviate from the desired direction by executing these blind control commands. Thus a very short duration should be set for running this loop.

3.3.2.7 Path Adjustment Module

Compared with the Position Estimation Module, the processing interval of this module is much longer. It reads the shared buffer for all available real time information, abstracts obstacles and the distance between control points into forces exerting on the control points and check whether any violation happens, e.g. the snake collides with obstacles. As a result, the resultant force will push or pull the control points to a new balance position thus forming a new R-snake. If violation happens, the processing will discard all results and try to re-initiate a snake in the Path Re-initialisation module.

Once the planned result is ready, it is written to the shared buffer for Trajectory Generating module and may be sent to the Communication

Adaptation Layer for remote console displaying if requested by the operator.

3.3.2.8 Path Re-initialisation Module

In case the Path Adjustment processing fails, a re-initialisation procedure is triggered to perform the process as described in Section 3.3.2.2.

If the re-initialisation process fails again, it keeps on trying once new information from the Image Acquisition and Processing is available. If it still fails after a certain times, an alarm will be sent to the remote console.

3.3.3 Wireless Controlled Robots

In contrast to highly intelligent robot, the mobile robot used here is developed by refitting an off-the-shelf model car, which has very limited on-board computational power but can access the wireless sensor network by integrating an IEEE 802.15.4 node, as shown in Figure 3.4.

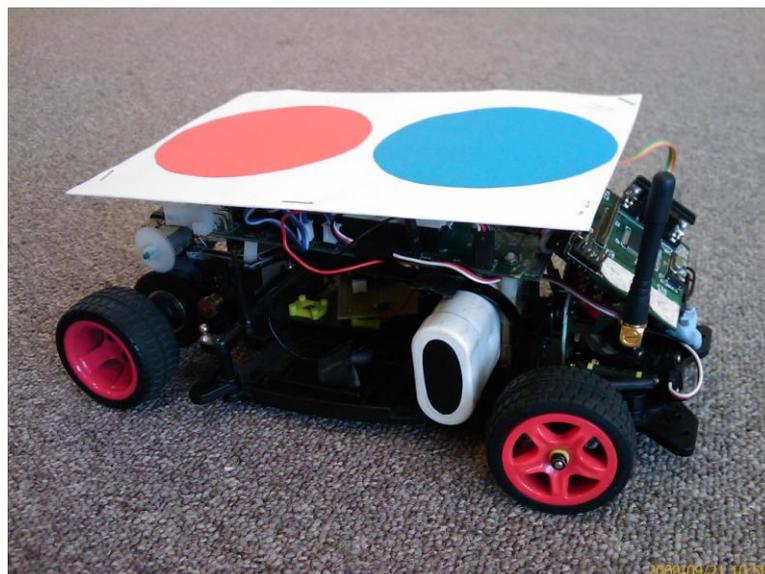


Figure 3.4 Robot on field

Two different colour blobs, blue and red, are placed on top of the robot for vision sensors to distinguish it from obstacles and to indicate the robot motion direction: the blue one represents the head and red one back.

3.3.3.1 Motion Commands Analyse and Execution Module

Commands received from vision sensors have time tags with commands series. Due to transmission delay or image processing delay in vision sensors, some of the commands in the series may overlap with the previous received ones. The robot has simple functions to compare them and execute the correct ones.

3.3.3.2 Motor Servo Module

Both driving and steering motors are Pulse-Width Modulator (PWM) driving motors. Motor Servo is realised by outputting PWM pulse to the respective channels. There are totally eight PWM channels available and can be controlled by the Microcontroller Unit (MCU) of the mobile robot. Specifically, PWM01 is used to control steering and PWM02 and PWM03 are used to control driving.

3.3.4 Communication Interfaces

As shown in Figure 3.2, one Communication Adaptation Layer is added on top of the IEEE 802.15.4 MAC&PHY layer in all three system components. The communication protocol defined in this layer consists of a uniform packet format and a collection of signal flow scenarios. A set of physical independent function calls are specified for the system modules.

Apart from centralised intelligent control system, this distributed architecture requires a harmonic collaboration between all distributed vision sensors to achieve a desired goal. Scenarios and packet formats are designed to serve for robot navigation such as path planning and motion control. They provide a data exchange mechanism suitable for distributed control purpose.

By designing kinds of error tolerance mechanisms, some data loss without re-transmission will not affect the operation of the system. For example, a procedure for re-competing for the robot control token is invoked in case no one has the control token due to a handover process failure caused by a handover related message loss. But message loss is not allowed during a file downloading. However the IEEE 802.15.4 standard intends to offer the fundamental lower network layers of a type of Wireless Personal Area Network (WPAN) which focuses on low-cost, low-speed ubiquitous communication between devices. Packets loss or network nodes disappearances are inevitable. Therefore, some mechanisms are proposed in this layer to deal with the package loss and package delay. Specifically,

- Kalman filter [131] is used to filter the received information such as coordinates of the snake control points and predict their new positions. If the communication connection is not available at a specific time, the predicted value will be used as new position; if there is no new data coming from a node for a certain time span, the node will be marked as nonexistent;

- Data are tagged with serial numbers to make sure they can be re-assembled in the right sequence if they arrive in an unexpected sequence due to delay. The sequence number also provides a mechanism for retransmission if some packets are missing;
- Error checksum, existence periodical checking for communication nodes and etc. are also considered.

3.4 Conclusion

This chapter gives an overview of the proposed mosaic eyes intelligent environment architecture including Remote Console, Vision sensors and Mobile Robot. Its functional modules and descriptions for all the modules are given accordingly. The proposed distributed intelligent environment with vision sensors mounted in the building releases the massive computation into smaller sensors, simplifies the localisation problem, promises a scalable environment variation and makes different mobile robots sharing the same software possible.

The main contribution of the chapter is the proposed distributed intelligent architecture, definitions of the system modules. All these form a basis to understand the mechanisms in the research works in later chapters.

Chapter 4 : MAP BUILDING AND OBJECTS LOCALISATION

4.1 Introduction

One of the key problems in mobile robotics is to estimate a robot's location and orientation relative to the environment [132-135]. Thus localisation has sometime been regarded as “the most fundamental problem to providing a mobile robot with autonomous capabilities” [136]. According to the pre-conditions given to a robot, the localisation problems can be classified into three types [137], namely, position tracking [44, 135, 138], global localisation problem [139-141] and kidnapped robot problem [142, 143]. The simplest one is position tracking where the robot's initial position is known and the odometer error and robot's uncertainty are restricted. The more challenging one is the global localisation problem, where the initial position is unknown and the position errors cannot be assumed to be negligible. The most difficult one is the kidnapped robot problem, in which the robot is placed in a location unknown to the robot itself. Furthermore, all these problems are become extremely hard when dynamics are introduced in the environment where the robot's sensor measurements may be corrupted by people.

Wireless Sensor Networks are providing the IT infrastructure to support distributed motion control, which can lead to a new paradigm for autonomous navigation, leveraging the complexity of centralised intelligence that hinders applications of autonomous robots in our daily life

and supporting high mobility with very limited onboard computational power. First, a distributed sensor network can provide a topological map for the unknown environment. The route planning problem can be converted into queries to the sensor network. Second, distributed sensors also provide active landmarks for localisation, which is more reliable and efficient than traditional localisation. Third, the wireless mosaic eyes are static relevant to the environment, perception of environment and control of vehicles will face less uncertainty with higher reliability.

However, individual sensors in a wireless sensor network have very limited view of observation and rely on ad hoc communication with less memory space, computational capacity, as well as power on board in order to fulfil battery powered applications [144, 145]. How to efficiently coordinate the distributed eyes to cope with global navigation becomes a big issue for this type of applications.

The architecture proposed in the thesis is an attempt to provide new paradigms for navigation. Through support from vision sensors mounted in the buildings, the environment has the intelligence to send navigation instructions to robots to move around. In this new architecture, navigation problems also need to be reconsidered. Firstly, global static topology map and local dynamic map need to co-exist harmoniously. Relations between vision sensors are fixed, their positions intuitively provide the global topology map landmarks if they are associated with explicit semantics. Vision sensors only need to focus on processing their information and to build a real time local map with dynamic obstacles and robots. This layered structure map not only separate global topology information from

local real time information; it also makes creating global map offline in advance possible. Thus more processing time can be saved for other tasks. Furthermore, vision sensors processes the local information can lead to achieve a global target. Secondly, part of the path planning in the global topology level planning process, is converted to vision sensors querying. With a certain coordination mechanisms proposed in this thesis, complex dynamic global path planning can be realised by local navigation techniques

This chapter concentrates on the contributions of two kinds of map building. As discussed above, global topology path planning is converted into vision sensor level routing, which is more related to the global map building than path planning techniques. Hence, the global routing algorithm described in this chapter is complementary to global map building. Firstly, a multiple Bloom-filters approach to compress the global topological map into distributed vision sensors is used to achieve efficient routing. Each wireless sensor is provided with unambiguous semantics for navigation by maintaining a multiple Bloom filter to indicate where branch paths can lead to. A method for multiple Bloom-filter design using error expectation instead of conventional false positive rate is proposed, which can increase compression rate and produce a uniform relative error expectation for all branch paths. Secondly, benefiting from the distributed statically mounted vision sensors, obstacles are detected and robots are localised by foreground-background discriminating and colour blobs matching. In addition, a statistical model is proposed to fuse different views of vision sensors to have a more precise localisation.

4.2 Multiple Bloom Filters for Map Building and Distributed Routing

For navigation, global routing is the first faced task. It locates at the top of a tiered navigation architecture [146]. Given a goal, a robot needs to know a sequence of abstracted locations which can lead it to the goal with the least cost. It is traditionally implemented by a centralised search approach in terms of a topological map. For distributed applications, an intuitive approach is to adopt the routing techniques from computer networks, e.g. flooding a query to a destination on the basis of a routing table [147, 148]. The drawback could be on the high memory usage and heavy communication load. This section proposes a multiple Bloom filter approach to overcome these two drawbacks.

4.2.1 Problem Statement

By the configuration software in the remote console PC, the vision sensor network has been configured as a topological map in accordance with the geographic layout. The topological map can be represented as a graph of $S = \{s_1, s_2, \dots, s_c\}$ consisting of a collection of sensor nodes, s_j , in the network; $E = \{e_{i,j} : i \in [1\dots c]; j \in [1\dots c]; i \neq j\}$ is a collection of edges representing the geographic passages, which connect pairs of nodes. For each node $s_i = \{I_i, E_i\}$, I_i is the identity of the i^{th} node and E_i is the set of edges connecting s_i with other nodes. The routing problem is to find a sequence of edges from the current location to a goal with the shortest

distance. When a vehicle is under the control of node s_i , it will query for the next edge $e_{i,j}$ between s_i and s_j to follow in order to reach the desired destination s_g . Instead of using an artificial code, semantic queries using a human friendly identity I_g such as “Fun’s office”, “Meeting room”, is adopted to simplify the development of human-machine interface, for example, for user querying through a PDA.

Due to the limited computation power of each wireless node, an on-line shortest route search should be avoided. Therefore, a routing table approach is adopted, where routing tables are generated for each node off-line by the Dijkstra’s search algorithm. Each node s_i has the same number of routing table entries as its edges. Each routing table entry $T_{i,j}$ corresponding to edge $e_{i,j}$ records the set of nodes that can be reached by s_i with the minimum routing cost by following edge $e_{i,j}$. Taking a map shown in Figure 4.1 as an example, two routing table entries for nodes A and D, three routing table entries for B and C are generated as bellow:

Node A : $T_{AC} = \{\text{Kai}\}$, $T_{AB} = \{\text{Jay, Meeting room}\}$;
Node B : $T_{BA} = \{\text{Fun's office}\}$, $T_{BC} = \{\text{Kai}\}$, $T_{BD} = \{\text{Meeting room}\}$;
Node C : $T_{CA} = \{\text{Fun's office}\}$, $T_{CB} = \{\text{Jay}\}$, $T_{CD} = \{\text{Meeting room}\}$;
Node D : $T_{DC} = \{\text{Kai}\}$, $T_{DB} = \{\text{Jay, Fun's office}\}$;

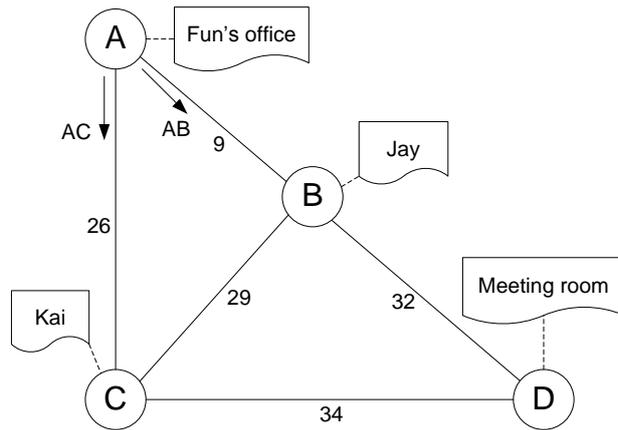


Figure 4.1 A simple topological map

With the routing table saved in each node, a robot at a location covered by a node can query which passage it should take in order to go to a goal, e.g. “Meeting room”, with the lowest routing cost. The node can answer the query by determining the string that represents the goal belongs to which routing table entry. The problem is equivalent to determining the group membership of the random strings as shown in Figure 4.2. This approach can eliminate the time taken for on-line search but at the expense of increased memory space.

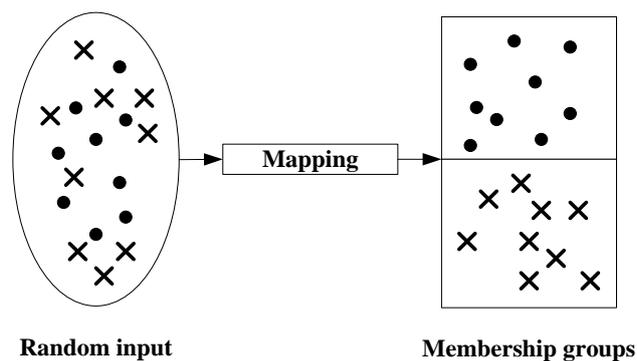


Figure 4.2 Classification of random strings to discover the associated routing semantics

For a big map, the required memory for storing routing tables is huge, which is often not feasible for sensor nodes with small on-board memory space. As such, a Bloom filter is exploited. A Bloom filter is a compressing

technique to determine memberships of random elements. Multiple Bloom filters with a short table length are developed to reduce the memory space required for saving the routing tables. The following section describes the application of Bloom filters for navigation routing.

4.2.2 Bloom Filters for Navigation Routing

Bloom filters are an efficient yet lossy way of describing membership of elements belonging to a set [116]. They are broadly used for compressed presentation of sets in many fields. In order to store n elements belonging to a branch path, a routing table T can be implemented by a Bloom filter that is a bit-vector of length m . Each element can be a human friendly string with underpinning semantics, such as "Meeting room". To represent n random strings in the entries of table T , k independent hash functions are used to generate digital fingerprints in T , which map elements belonging to the path onto k integers, where $k \in [0, m]$. The corresponding bits in bit-vector T are set as shown in Figure 2.2. If there are L branch paths for a node, L Bloom filters are needed for routing. These L Bloom filters are collectively referred to as multiple Bloom filters in this paper.

A vehicle can query multiple Bloom filters in a sensor node to determine which path to take. If any of the bits in a Bloom filter are not set to 1, the corresponding path definitely should not be taken. If all of the bits for a path are set, the vehicle *may* take that path. The advantage of using Bloom filters to represent routing tables is its ability to compress n elements with any string length to m bits, and its high query efficiency to

enable n elements search to k hash functions check. However, there is a probability that a wrong decision is made because a bit could be set by other elements when they are hashed. It is known as *false positive*. Thus there is a trade-off between the compression rate and false positive rate, which is relevant to number of elements, n , hash function number k and vector length m [21] as given in (4.1):

$$P_{fp} \approx (1 - e^{-nk/m})^k \quad (4.1)$$

The minimum false positive probability can be obtained as

$$P_{fp} = 1/2^k, \text{ when } k = (m/n) \cdot \ln 2 \quad (4.2)$$

As an example, for a path with 1000 locations ($n = 1000$) and 4 hash functions ($k = 4$), the optimised table length $m = 5771$ bits (721 bytes) is obtained by (4.2), with a false positive rate of $P_{fp} = 6.25\%$. The required memory space is still reasonable for wireless sensor network applications in terms of a medium map. The probability of a false positive can be further reduced by following a multi-hop approach, where a query is forwarded to the next node in the path for double checking of its membership. Suppose a vehicle is under the control of node s_i , it queries multiple Bloom filters $T_{i,j}$, $j = 1 \dots L$, about which edge $e_{i,j}$ to follow in order to reach goal I_g . Followings are the steps:

Step 1. If there is only one path, the j^{th} path, passing the hash check, e.g. $T_{i,j}(H_i(I_g)) = 1$, where H_i is the set of hash functions in n_i , the j^{th} path is taken;

Step 2. If there are more than one path passing the hash check, e.g. $T_{i,j}(H_i(I_g)) = 1$, there are multiple choices for j , so a query is sent to their one hop successor s_j . The multiple Bloom filters in node s_j are checked and go to Step 1;

Step 3. Backtrack all the nodes passing the check and confirm the corresponding edge is the path to follow.

If independent hash functions are used, the probability of a false positive P_{fp} decreases as a power of hops, P^{hops} . Taking the previous example with $n = 1000, k = 4, P = 6.25%$, this gives Table 4.1,

Table 4.1 False positive probability with number of hops

Hops	1	2	3
P_{fp}	0.38%	0.024%	0.0015%

The false positive can be reduced to a very low level with only few hops and the correct branch path can be determined.

4.2.3 Design of Multiple Bloom Filters Based on Error Expectation

In Section 4.2.2, the routing scheme using multiple Bloom filters is presented. An immediate question is how to determine the number of hash functions and the table length of each Bloom filter. From Equation (4.2), the optimal number of hash functions should be selected based on a desired P_{fp} . However, the number of hash functions in multiple Bloom filters is fixed for several tables and often specified beforehand to simplify implementation such as reducing on-line computation spent on the hash

check. Therefore, in order to ensure queries with a false positive less than P_{fp} , an optimal design by simply adjusting the number of hash functions often cannot be achieved. Instead, the length m of each Bloom filter is changed for a given P_{fp} to carry out a sub-optimal design. From equation (4.1), the length, m , of each Bloom filter can be expressed as follows:

$$m = \frac{-nk}{\ln(1 - \sqrt[k]{P_{fp}})} \quad (4.3)$$

From equation (4.2), m can be related to $k_0 = -\log_2 P_{fp}$ and $m_0 = nk_0 / \ln 2$ as shown in equation (4.4) and Figure 4.3 below, where k_0 and m_0 are optimal values of m and k respectively.

$$m = \frac{k}{k_0} \cdot \frac{\ln(1/2)}{\ln(1 - (1/2)^{k_0/k})} \cdot m_0 \quad (4.4)$$

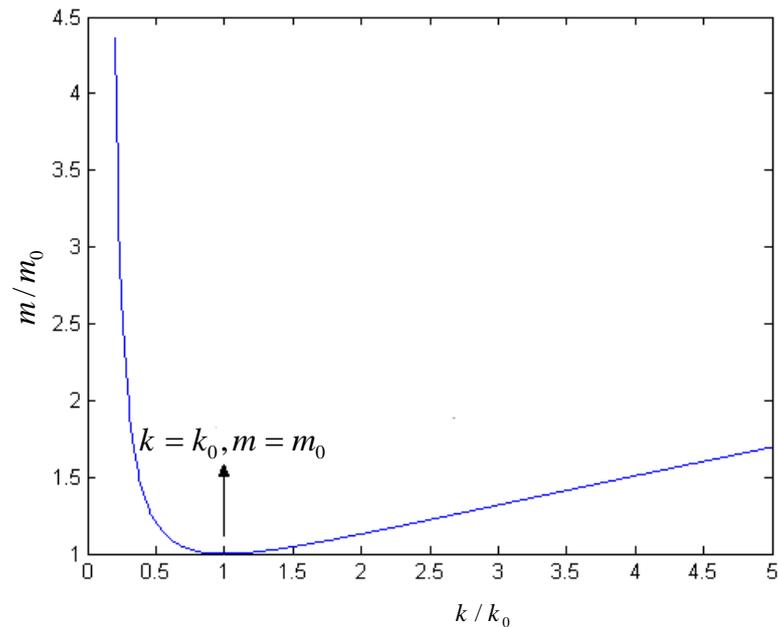


Figure 4.3 Length m vs. a given k

From Figure 4.3, it can be observed that the table length of a Bloom filter is always bigger than the optimal m_0 in order to reach the same P_{fp} if k is different from k_0 .

Consider that node s_i has L edges and subsequently L Bloom filters. Each Bloom filter is expected to encode $n(i)$, $i=1..L$, nodes with altogether $n = \sum_{i=1}^L n(i)$ encoded in multiple Bloom filters. In order to design the table lengths $m(i)$ for L Bloom filters, an intuitive thought is to select $m(i)$ to meet an identical false positive rate P_{fp} based on equation (4.3) for all branch paths. However, if both legal queries and illegal queries¹ are considered, this may not be a sensible approach. In a multiple Bloom filters system, the encoded numbers of nodes $n(i)$, $i=1..L$ in individual Bloom filters could be quite different. For example, some edges may lead to very few destination nodes but others may lead to many. However, they will be used to check the same amount of queries, which could be legal queries or illegal queries, such as a query belonging to another table entry. Equal P_{fp} means equal number of errors for all paths, no matter how many nodes are present in a path. If a false positive happens at a path due to an illegal query, the path with fewer nodes will have a higher chance to become a wrong path than a path with denser nodes. This can be verified by evaluating the error expectation. Considering that there are

¹ Queries which have not been compressed in a Bloom filter

altogether N possible queries, a possible query consisting of 6 decimal digits could have N reaching 10^6 , we expect any query not belonging to an edge will not be answered “yes” because of the false positive. The relative error expectation for edge i due to N queries is

$$E_{EPN}(i) = \frac{(N - n(i)) \cdot P_{fp}}{n(i)} \quad (4.5)$$

It can be seen from equation (4.5) that an equal rate of false positive, P_{fp} , will result in more errors for a Bloom filter with a smaller $n(i)$ than one with bigger $n(i)$. The overall error expectation for checking L tables in multiple Bloom filters can be calculated as,

$$E_d = \sum_{i=1}^L (N - n(i)) \cdot P_{fp} = (L \cdot N - n) \cdot P_{fp} \quad (4.6)$$

Therefore, it would be a more reasonable approach to take the error expectation as the design criterion instead of the false positive rate such that a uniform relative error expectation to all paths can be obtained. This is important for navigation applications, where driving to any route has equal probability of error.

In order to ensure a uniform relative error expectation for all edges, we have to take a biased approach such that a different false positive rate, $P_{fp}(i)$, $i=1...L$, will be assigned to a different edge i , for example, an edge leading to fewer destination nodes will be given a lower possibility. Let

$$P_{fp}(i) = P_{fp} / t \text{ with } t = \frac{N - n(i)}{N - \bar{n}} \cdot \frac{\bar{n}}{n(i)} \quad (4.7)$$

where $\bar{n} = n / L$ represents the average number of nodes to be encoded for each branch.

From (4.7), if $n(i) \leq \bar{n}$, $t \geq 1$ and $P_{fp}(i) \leq P_{fp}$, a lower false rate is set ;
if $n(i) > \bar{n}$, $t < 1$ and $P_{fp}(i) > P_{fp}$, a higher false rate is set. It is expected that this modification can make all edges exhibit an equal relative error expectation in spite of how many nodes an edge can lead to.

Theorem 1: If $n(i) \leq N / \left(P_{fp} \cdot \left(\frac{N - \bar{n}}{\bar{n}} \right) + 1 \right)$, $P_{fp}(i)$ defined in (4.7) can

be used as a false positive probability for the Bloom filter design of edge i . Then all edges will have equal relative error expectation and the overall error expectation will maintain to be unchanged as (4.6) for equal P_{fp} design.

Proof: If the $P_{fp}(i)$ in (4.7) can be used as a probability, $P_{fp} \leq t$ must be true. From (4.7), it can be obtained that,

$$n(i) \leq N / \left(P_{fp} \cdot \left(\frac{N - \bar{n}}{\bar{n}} \right) + 1 \right) \quad (4.8)$$

The relative error expectation of edge i due to N queries can be calculated by using the $P_{fp}(i)$:

$$\begin{aligned} E'_{EPN}(i) &= \frac{(N - n(i)) \cdot P_{fp}(i)}{n(i)} = \frac{(N - n(i))}{n(i)} \cdot \frac{P_{fp}}{t} \\ &= \frac{N - \bar{n}}{\bar{n}} \cdot P_{fp} \end{aligned} \quad (4.9)$$

Equation (4.9) shows that $E'_{EPN}(i)$ is a constant independent of $n(i)$.

The overall error expectation for all edges in a multiple Bloom filter can be obtained as:

$$\begin{aligned}
 E'_d &= \sum_{i=1}^L (N - n(i)) \cdot P_{fp}(i) = \sum_{i=1}^L (N - n(i)) \cdot \frac{P_{fp}}{t} = \sum_{i=1}^L \frac{(N - \bar{n})}{\bar{n}} \cdot n(i) \cdot P_{fp} \\
 &= \frac{(N - \bar{n})}{\bar{n}} \cdot L \cdot \bar{n} \cdot P_{fp} = (L \cdot N - n) \cdot P_{fp}
 \end{aligned} \tag{4.10}$$

which is identical to the expectation in (4.6) for the equal P_{fp} based design.

Therefore the false positive rate, $P_{fp}(i)$, in (4.7) for multiple Bloom filters design balances the relative error expectations of all edges without losing the overall error expectation. Due to $N \gg n$ and $P_{fp} \ll 1$ in most cases, the condition $n(i) \leq N / \left(P_{fp} \cdot \left(\frac{N - \bar{n}}{\bar{n}} \right) + 1 \right)$ is often true. However, it would be difficult to estimate the number of all possible queries, N , when applying (4.7) for a given design. One reasonable approximation of (4.7) can be obtained by considering a very large N such that:

$$P_{fp}(i) = P_{fp} \cdot \frac{n(i)}{\bar{n}} \tag{4.11}$$

Therefore, the false positive possibility of each edge can be easily specified by the number of resident nodes on the edge.

4.2.4 Memory Usage of the Error Expectation Based Design

Wireless sensor nodes often have less memory capacity. A key factor to be concerned about when developing a wireless sensor network application is the amount of memory usage.

Assume that there are L Bloom filters in a sensor node and each Bloom filter encodes $n(i)$, $i = 1 \dots L$, members. Following the conventional design, we specify the table lengths such that all tables will achieve the same rate of false positive, P_{fp} . From (4.3), the table length for the i^{th} bloom filter can be expressed as,

$$m'(i) = \frac{-n(i) \cdot k}{\ln(1 - \sqrt[k]{P_{fp}})} \quad (4.12)$$

The overall table length for the multiple Bloom filters becomes:

$$m_{FPB} = \sum_{i=1}^L \frac{-n(i) \cdot k}{\ln(1 - \sqrt[k]{P_{fp}})} \quad (4.13)$$

If the proposed expectation based design is followed, the $P_{fp}(i)$ will be adjusted according to (4.11). The overall table length becomes,

$$m_{EEB} = \sum_{i=1}^L \frac{-n(i) \cdot k}{\ln(1 - \sqrt[k]{P_{fp} \cdot [n(i) / \bar{n}]})} \quad (4.14)$$

Theorem 2 will state the memory usage of the proposed method in comparison with the conventional one shown in (4.13).

Theorem 2: *The table length of a multiple bloom-filter designed by (4.11) will not be longer than that of the false positive based design. The maximum table length will be as (4.13) when $n_1 = n_2 = \dots = n_L = \bar{n}$.*

Proof: Considering optimisation of (4.14) with respect to variables of $n(i)$, $i=1..L$, subject to the constraint of $\sum_{i=1}^L n(i) = n$, the following

associated Lagrange function can be obtained as:

$$U(n(i), i=1 \dots L) = \sum_{i=1}^L \frac{-n(i) \cdot k}{\ln\left(1 - \sqrt[k]{P_{fp}} \cdot \left[\frac{n(i)}{\bar{n}}\right]\right)} + \lambda \left(\sum_{i=1}^L n(i) - n \right) \quad (4.15)$$

where λ is a Lagrange multiplier.

Let $g(n(i)) = 1 - \sqrt[k]{P_{fp}} \cdot \left[\frac{n(i)}{\bar{n}}\right]$, $i=1..L$, the optimisation can be solved by the first partial derivatives of U with respect to $n(i)$ and λ :

$$\begin{cases} \frac{\partial U}{\partial n(i)} = -\frac{k \cdot g(n(i)) \cdot \ln(g(n(i)) + 1 - g(n(i)))}{g(n(i)) \cdot \ln^2(g(n(i)))} + \lambda = 0, i=1 \dots L. \\ \sum_{i=1}^L n(i) - n = 0 \end{cases} \quad (4.16)$$

From (4.16), there is a critical point at $n_1 = n_2 = \dots = n_L = \bar{n}$. Substitute it into (4.14), the extremum of U can be obtained,

$$m_{EEB}(n(i) = \bar{n}) = \sum_{i=1}^L \frac{-n(i) \cdot k}{\ln\left(1 - \sqrt[k]{P_{fp}}\right)} = -\frac{n \cdot k}{\ln\left(1 - \sqrt[k]{P_{fp}}\right)} \quad (4.17)$$

It has the same length as the conventional design in (4.13). We need to further prove that it is the maximum for all $n(i)$. The second derivative of $m_{EEB}(n(i))$ in (4.14) with respect to $n(i)$ can be obtained as:

$$\frac{\partial^2 m_{EEB}}{\partial n(i)^2} = \frac{\frac{\partial g}{\partial n(i)} \cdot f(g)}{g^2 \cdot \ln^2(g)} \quad (4.18)$$

where,

$$f(g(n(i))) = 1 + k \cdot g(n(i)) + 2 \frac{1 - g(n(i))}{\ln(g(n(i)))} \quad (4.19)$$

Because $\frac{\partial g}{\partial n(i)} = -\left[P_{fp} \cdot \frac{n(i)}{\bar{n}}\right]^{\frac{1-k}{k}} \cdot \frac{P_{fp}}{\bar{n}} \leq 0$, the extremum must be the

maximum if $f(g) \geq 0$.

Because of $g \geq 0$ and $k \geq 1$, we have

$$f(g(n(i))) \geq 1 + g(n(i)) + 2 \cdot \frac{1 - g(n(i))}{\ln(g(n(i)))} \equiv f_1(g) \quad (4.20)$$

Taking its derivative with respect to g , we have,

$$\begin{aligned} \frac{\partial f_1}{\partial g} &= 1 + 2 \cdot \frac{-\ln(g) - (1-g)/g}{\ln^2(g)} \\ &= \frac{f_2(g)}{\ln^2(g)} \end{aligned} \quad (4.21)$$

where $f_2 = \ln^2(g) - 2 \cdot \ln(g) + 2 - 2/g$, further,

$$\frac{\partial f_2}{\partial g} = \frac{f_3}{g^2} \quad (4.22)$$

$$\frac{\partial f_3}{\partial g} = 2 \cdot \ln(g) \quad (4.23)$$

where, $f_3 = 2 \cdot g \cdot \ln(g) - 2g + 2$.

Now we will track back to prove $\frac{\partial^2 m_{EEB}}{\partial n(i)^2} \leq 0$ for $0 \leq g(n(i)) \leq 1$,

because of $g(n(i)) = 1 - \sqrt[k]{P_{fp}(i)}$.

From (4.23), we have $\frac{\partial f_3}{\partial g} \leq 0$ for $0 \leq g \leq 1$. Hence, f_3 is non-

increasing. From $f_3(1) = 0$, we know $f_3(g) \geq 0$ and therefore $\frac{\partial f_2}{\partial g} \geq 0$ for

$0 \leq g \leq 1$. It indicates that f_2 is non-decreasing for g from 0 to 1. From

$f_2(1) = 0$, we have $f_2(g) \leq 0$ from 0 to 1, hence $\frac{\partial f_1}{\partial g} \leq 0$ for $0 \leq g \leq 1$.

f_1 is non-increasing for g from 0 to 1.

Because $\lim_{g \rightarrow 1} f_1(g) = 2 + 2 \cdot \lim_{g \rightarrow 1} \frac{-1}{1/g} = 0$, we have $f_1(g) \geq 0$ for g

from 0 to 1, and thus $f(g) \geq 0$ in (4.20). Substituting it into (4.18), we

know $\frac{\partial^2 m_{EEB}}{\partial n(i)^2} \leq 0$ for $0 \leq g \leq 1$. The critical point at $n_1 = n_2 = \dots = n_L = \bar{n}$

reaches the maximum value of m_{EEB} , which is the table length of the conventional design.

From Theorem 2, a multiple Bloom filters using expectation based design will not only achieve uniform relative error expectation but also save memory space used for implementation in comparison with the conventional Bloom filter design, where false positive rate is the design criterion.

4.2.5 Multiple Bloom Filters for Routing

In order to evaluate the performance of the multiple Bloom filters proposed in Section 4.2.3, we randomly generate topological maps a total of $n = 1000$ nodes. For $k = 4$ hash functions and a desired false positive rate $P_{fp} = 0.01$, we design two Bloom filters for a node with 4 branches (Path1, Path2, Path3 and Path4), which have 23, 193, 332 and 452 nodes, respectively. Both conventional false positive based and the new error expectation based algorithms are performed. Then we generate

4 groups of 10^6 random strings to query the two Bloom filters for the purpose of evaluation. Table 4.2 and Table 4.3 show the experimental results of the two Bloom filters.

Table 4.2 : Errors using the false positive based design

<i>Group</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>Average Errors</i>	<i>Relative Errors</i>
Path 1	9987	10013	9979	10025	10001	434.83
Path 2	9976	9984	9981	10012	9988	51.75
Path 3	10014	9985	9998	10002	9998	30.11
Path 4	9982	9986	10011	9993	9993	22.11

Table 4.3: Errors using the error expectation based design

<i>Group</i>	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>	<i>Average Errors</i>	<i>Relative Errors</i>
Path 1	937	945	914	909	926	40.26
Path 2	7702	7734	7724	7727	7722	40.00
Path 3	13178	13297	13206	13308	13247	39.90
Path 4	17882	18123	18105	17934	18011	39.84

From Table 4.2, the conventional false positive based design results in much higher relative errors for a path with fewer nodes than a path with more nodes. From the working process of a multiple Bloom filter proposed in Section 4.2.2 , any query needs to be checked by all Bloom filters. Therefore, a query to a path with more nodes is more likely to be wrongly guided to a path with fewer nodes. The mistake will result in more than one Bloom filters passing their hash check, and therefore needs to communicate with the next node for double confirmation as described in Section 4.2.2 , which implies more communication expenses in a wireless network. However, the proposed expectation based design guarantees uniformly distributed relative errors. In Table 4.3, all paths show similar amount of relative errors of about 40. This means that there is an equal chance of a query in one path is wrongly assigned to another. We can expect much less amount of communication burden for double check in

comparison with Table 4.2, as an example, which is about a scale of $(452 - 23)^2 / (452 + 23) \cdot 0.01 \cdot 1000 = 3875$ less for 1000 scanned quires.

In terms of the overall memory usage, we randomly generate 1000 topological maps with 1000 nodes each. The table lengths by using the false positive based design and the error expectation based design are shown in Figure 4.4.

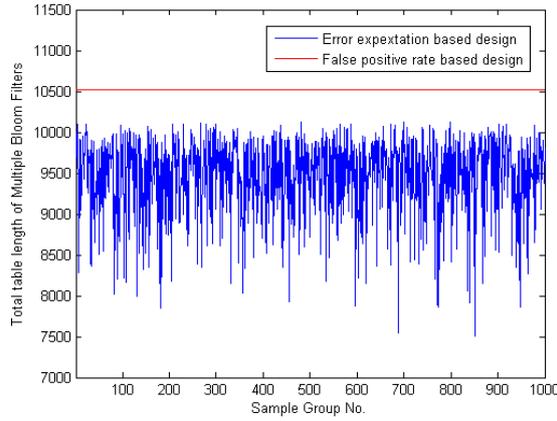


Figure 4.4: Error expectation comparison with two methods

It is clear that the proposed method outperforms traditional design, which verifies Theorem 2. In order to have a quantitative view, we list the memory saving rates of 4 groups of results into Table 4.4, which is defined to be $1 - (m_{EEB} / m_{FPB})$.

Table 4.4: Overall table length using proposed method

Group	1	2	3	4	Average
Overall table length	9449	9452	9483	9462	9449
Memory saving rate	0.102	0.102	0.099	0.101	0.102

From Table 4.4, the average memory savings rate is about 10.2%, which means that 10% of the memory space in sensor nodes can be reduced.

4.3 Objects Identification and Fast Colour Blobs Tracking

This section proposes a fast object identification and tracking algorithm based on the YUV colour space and background-foreground discrimination. Some preparation jobs should be done before the whole image processing loop starts, namely, offline tasks. Offline tasks include sampling images with target objects in view to select and learn colour classes thresholds of different objects, and then creating a colour lookup table based on this data. During the system initialisation, a background image is obtained and stored by sampling and averaging a certain number of the environment images. After that, foreground images are obtained by subtracting real time image with the stored background image. Then a decomposing procedure is carried out by bitwise AND the colour lookup table and the foreground image into a binary image for object identification and tracking. One of the most significant advantages of this approach is that it can determine a pixel's membership in multiple colour classes simultaneously.

4.3.1 Overview

For a complete colour tracking system, one needs to consider the following questions:

Which colour space to choose? There are many ways to describe a colour in a digital system, so called colour model or colour space, such as YUV, HIS and RGB. Different choices of colour space have a direct impact on image processing efficiency and algorithms.

How to recognize a colour? One colour is normally represented by 3D values. However, colour recognition can be done by comparing one, two or three dimensions based on a certain colour space.

What method to identify and track the target object? Obviously, better method means more robustness, less noise and higher processing efficiency.

How to adapt the algorithms to different illumination conditions?

The discussions below are based on these four questions.

4.3.1.1 Colour Space Transformation

The most widely used colour spaces in image process are RGB (CMY is a variation from this model designated for printing), YUV model and HSI model. The following will briefly describe the characteristics of these models, as well as conversion formula.

4.3.1.1.1 RGB Colour Format

The RGB colour format is based on the three colour components Red (R), Green (G) and Blue (B). The colour space [149] is shown in Figure 4.5. The RGB colour model describes a colour space with additive colour composition as it is used in CRT or LCD monitors. Thus, the RGB colour format is very popular in computer graphics and image processing.

The three colour components can be seen as three linear independent vectors, describing a three dimensional colour space or a colour cube. The origin of the colour cube represents black colour with $RGB(0, 0, 0)$ and the end of the cubes' diagonal represents white colour with $RGB(1, 1, 1)$. The diagonal itself covers all gray values from black to white.

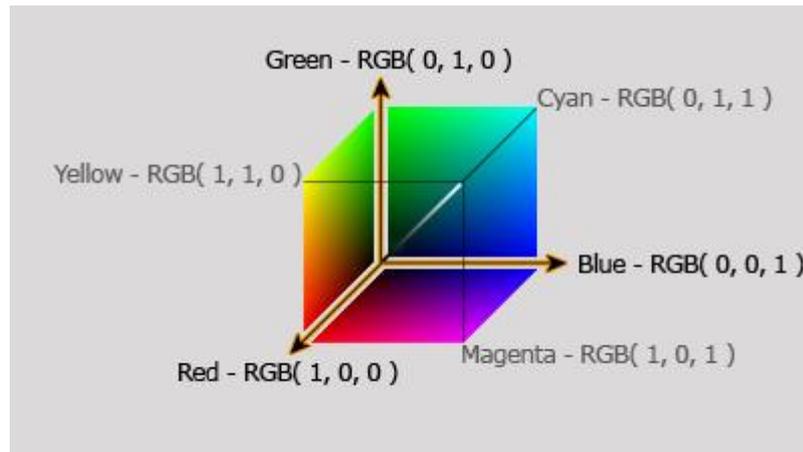


Figure 4.5 RGB space example

Although RGB is the most popular colour space used by image processing, it has some inherent disadvantage. Each of the RGB colour need to present not only the chrominance but also the luminance. In the RGB space, the space distance between two similar colours may be very large while that of two totally different colours may be very close. A small change in lightness may cause dramatic variance of all RGB colour components. This disadvantage makes colour identification extremely difficult.

4.3.1.1.2 HSL/HSV/HSI Colour Space

HSL and HSV are the two most common cylindrical coordinate representations of point in an RGB colour model, which rearrange the

geometry of RGB in an attempt to be more perceptually relevant than the Cartesian representation. HSL stands for hue, saturation and lightness and is also called HLS, or HIS (I for intensity). HSV stands for hue, saturation and value, and is also called HSB (B for brightness). This colour model has the advantage that chrominance is coded in two of the dimensions (H and S) while intensity is coded in the third.

The HSL colour model can be seen as a double-cone [149]. The angle around the central axis represents the hue, the distance to the axis represents the saturation and position along the central axis represents the luminance, as shown in Figure 4.6. The advantage of this colour model comes from the independence of luminance and hue. Luminance variant will not change the value of hue or saturation, thus brings a vast range of applications in colour image segmentation.

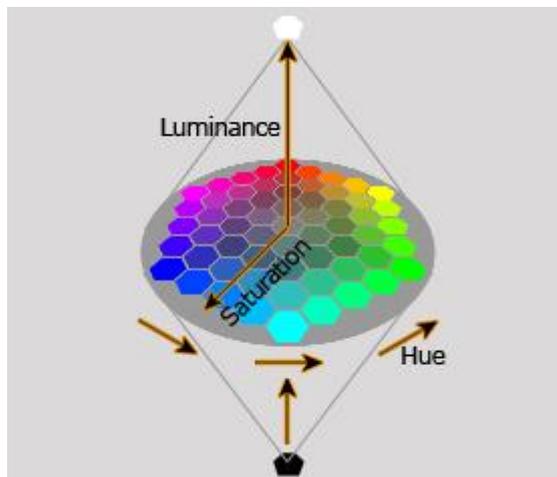


Figure 4.6 HIS space example

The transformation from RGB to HSI is as follows,

$$I = \frac{R + G + B}{3} \quad (4.24)$$

$$S = 255 \cdot \left(1 - \frac{3 \cdot \min(R, G, B)}{R + G + B}\right) \quad (4.25)$$

$$H = \begin{cases} \theta & \text{if } G \geq B \\ 360 - \theta & \text{if } G < B \end{cases} \quad (4.26)$$

where,

$$\theta = \cos^{-1} \left[\frac{2R - G - B}{\sqrt{2 \times [(R - G)^2 + (G - B)^2 + (B - R)^2]}} \right] \quad (4.27)$$

The disadvantage of HSI colour space is due to the conversion complexity, in which a lot of Floating-point operations are involved. This can be seen from equations (4.27).

4.3.1.1.3 YUV Colour Space

The YUV colour format describes a colour by using the colour components luminance and chrominance. The luminance component (Y) represents the brightness information of a colour and the chrominance components (U and V) contain the colour differences.

The YUV colour format was developed for analogy TV transmissions to provide compatibility between black-and-white television and colour television: The luminance component is sufficient for black-and-white TV sets, whereas colour TV sets need the additional chrominance information. Another advantage of the YUV colour format is based on the characteristics of the human visual perception: Since the human eye is much more sensitive for brightness information compared to colour information, the chrominance information can be reduced within video streams without losing significant quality. This method is called chroma sub-sampling.

The term YUV can be seen as a representative for all luminance / chrominance colour formats, without defining exactly the coefficients for colour conversion. Historically, the YUV colour format is used to describe the colour format of analogy PAL and analogy NTSC video equipment. Today, the term YUV is often misused instead of YCbCr to describe the colour format of digital video or images. For component digital video the YCbCr colour format is used, for component analogy video the YPbPr colour format is used.

The transformation of RGB to YUV is shown in (4.28) below,

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.148 & -0.289 & 0.437 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (4.28)$$

(4.28) can be rewritten as,

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 77 & 150 & 37 \\ -38 & -73 & 112 \\ 157 & -132 & -26 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (4.29)$$

From the discussion above, the YUV space is chosen for colour identification and object tracking. The increased robustness and higher processing efficiency will benefit from the advantages of the independence of luminance Y and chrominance U and V, thus the hue value will remain unchanged even the light condition changes. To avoid colour information loss, three dimensional comparisons is employed.

4.3.1.2 Colour Recognition and Blobs Tracking

For colour target recognition, colour is the best way to distinguish the target from other objects, such as the threshold algorithm [150, 151], the probability models algorithm [152], the self-organising clustering algorithm

[153] and etc.. In the YUV colour space, colours are clustered if they have the same hue value but under different luminance conditions. So a colour cube is set up to contain and present the colour cluster. During image segmentation, interested pixels which are inside the cube are marked with 1, and uninterested pixels are cleared to 0. As a result, one binary image is generated with the target indicated by white colour and background indicated by black. This is a very useful approach if the processing ability is powerful enough. The drawback of this solution is that each pixel has to be compared six times to determine whether it is in the given colour cube, which can be seen from the pseudo codes:

```
IF (Y >= Y_Lower) AND (Y <= Y_Upper) AND (U >= U_Lower)
AND (U <= U_Upper) AND (V >= V_Lower) AND (V <= V_Upper)
Pixel colour is set to 1
ELSE
Pixel colour is cleared to 0
```

Especially, when there are multiple targets to be tracked, the identification efficiency is very low.

To improve the colour identification efficiency, [154] proposed one colour lookup table which is constructed with an encoded target colour. With this lookup table, bitwise AND operation is employed other than comparison.

Benefitted from the mosaic eyes architecture, vision sensors are mounted in the building and they will always have a relative static background image. By subtracting captured image with a background one, most of the unrelated pixels are eliminated.

After the target pixels are being extracted from the image, blob tracking becomes the primary problem. In general, according to the search scope in the image, applications can be divided into two categories: whole image search and predicted region search. As the name indicated, whole image searching can find a target if it is disappeared in the view for some reasons but it takes more time to process one frame since it analyses the whole image pixels. However, predicted region search relies on the previous process result. It has high processing efficiency since it reduces the searching area dramatically, but failing to locate a target in one frame will cause failures in the succeeded frames. To have a good balance of efficiency and robustness, a combined solution is adopted in this thesis. One complete image searching is carried out within a certain period to find new appeared objects or to relocate the lost targets while still maintaining high processing efficiency by predicted region searching most of the time.

4.3.2 Overall Structure of Image Processing

A combination of offline and online algorithms are used to achieve the robustness and real time requirement of the system. Offline algorithm includes target colour sampling and filtering and colour lookup table creating, this is done before the real time tasks start. Online algorithm will deal with all real time processing i.e. image capturing, foreground calculating, binary image generating, objects identification and objects tracking. The overall structure of image processing is shown in Figure 4.7.

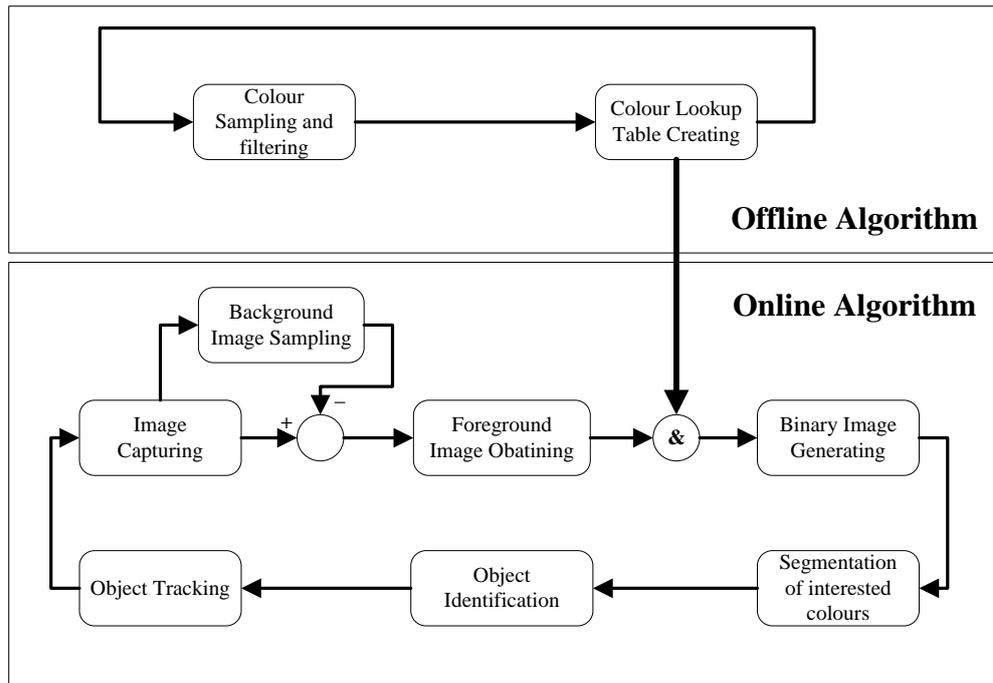


Figure 4.7 Overall structure of image processing

4.3.3 Offline Algorithm

Pixels of one identical colour object captured in one image can be clustered in the colour space. To describe this colour cluster, a cube as shown in red in Figure 4.8 is used to distinguish its colour pixels. The offline algorithm is to find such a cube and create a lookup table for use by the online algorithm.

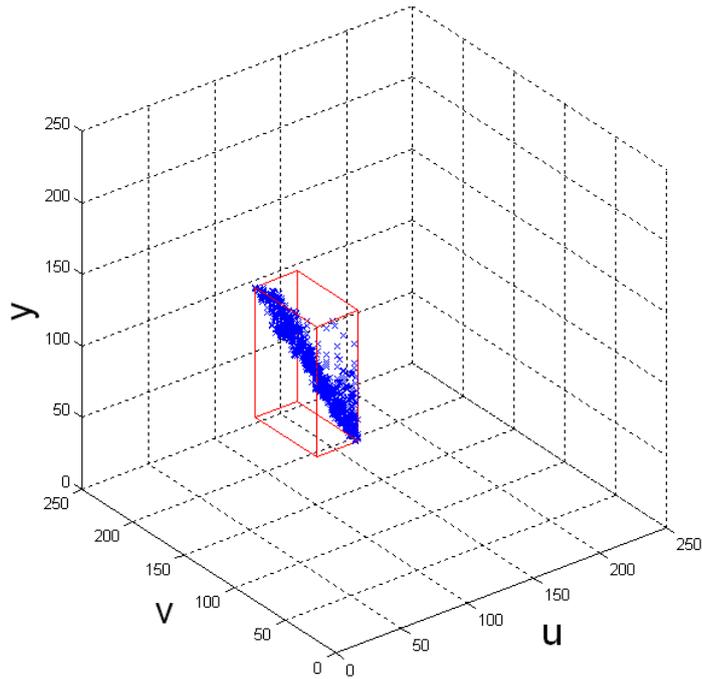


Figure 4.8 Colour clustering cube

4.3.3.1 Colour Sampling and Filtering

As discussed above, the thresholds of a desired colour should be input to generate the cluster cube. This is realised by specifying a certain area in the image, taking all colour pixel samples and then filtering out the abnormal pixels.

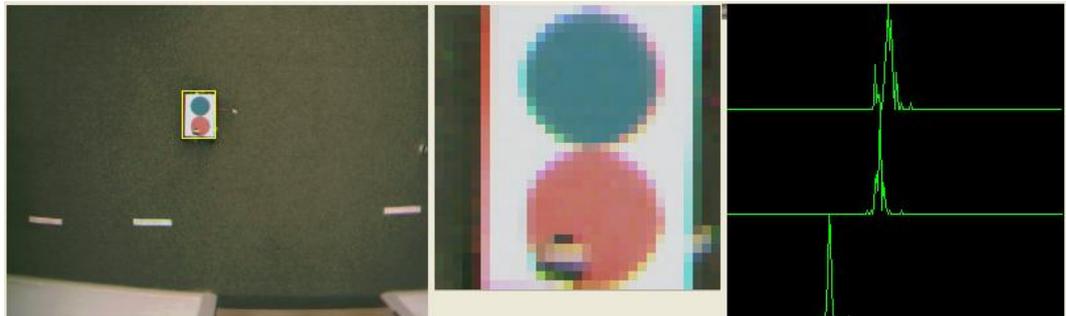


Figure 4.9 Colour sampling interface

This procedure is shown in Figure 4.9. The left side of the picture is one frame captured by a vision sensor. The target is highlighted by a

yellow rectangle, in which blue and red sample pixels are included. The zoomed out area is shown in the middle of Figure 4.9. The right side of Figure 4.9 shows the statistics values of the red area.

Because of the background noise in the sampled image, the colours are not evenly distributed in the image. This is illustrated in Figure 4.10. The desired cube is the red one rather than the expanded green one.

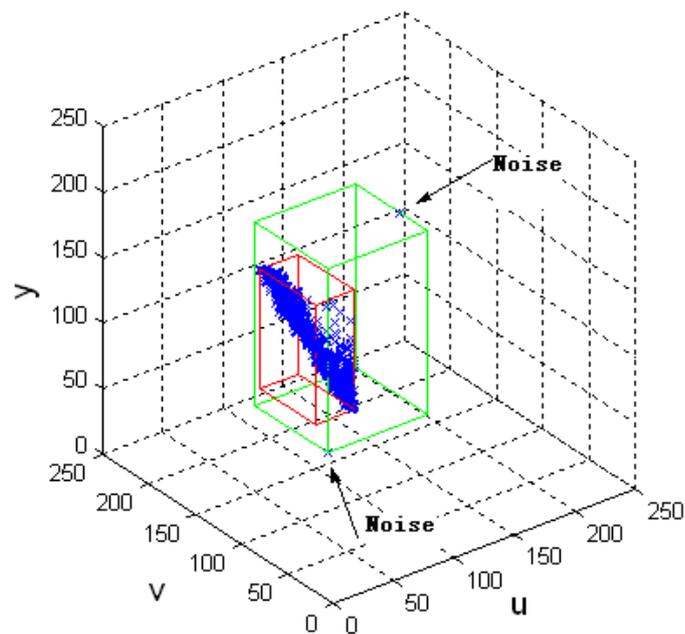


Figure 4.10 Colour cubes with noise

Extreme Value Theory (EVT) [155] is carried out to eliminate the abnormal maximum and minimum pixels. The numbers of pixels having identical U, V and Y values are counted separately. If one of the statistic numbers is less than a given threshold, the pixel will be removed from the colour samples, as shown in Figure 4.11.

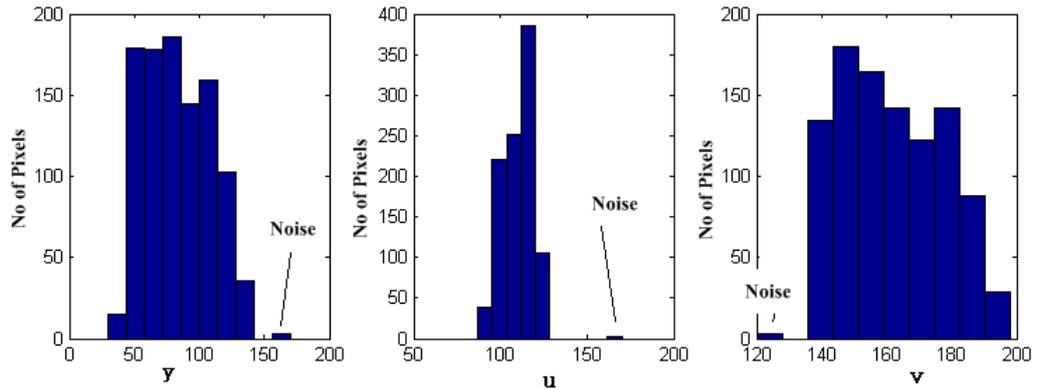


Figure 4.11 Sample colour filtering

4.3.3.2 Lookup Table Generating

Once the threshold cube is obtained, six times of comparisons are engaged traditionally to identify whether a pixel is inside the cube. This will dramatically increase the processing time especially when more than one target colours are involved.

Instead, this thesis uses three character arrays to decompose the multidimensional thresholds. Each represents one vector of the YUV colour. The dimension of the array is determined by the colour resolution. For a scale of 256, the arrays are defined as,

```

Char Yclass[256]
Char Uclass[256]
Char Vclass[256]

```

Each element of the char array has 8 bits. Each bit position represents one target. For a char array, a maximum of 8 object colours can be stored. For a target colour $blob_i$ ($0 \leq blob_i < 8$), the colour cube is projected into the Y, U and V axis. Set the j^{th} ($0 \leq j < 256$) bit of the array elements inside the cube to 1, and the ones outside the cube to 0. For example, one

YUV threshold cube obtained by sampling and filtering the first target is Y [50-100], U [100-142] and V [124-150], then the first bits from 50 to 100 elements of *Yclass* are set to 1 and the rest of *Yclass* are set to 0. Similarly, *Uclass* and *Vclass* are processed.

Once the lookup table is created, the procedure of checking whether one pixel belongs to a target colour cluster becomes a simple bitwise AND operation. Specifically, to check whether pixel (10, 30, 70) belongs to the first target becomes to check the value of the first bit of the expression (4.30). If the value of the first bit is 1, the answer is yes; otherwise no.

$$Yclass[10] \& Uclass[30] \& Vclass[70] \quad (4.30)$$

4.3.4 Online Algorithm

As shown in Figure 4.7, the online algorithms mainly deal with the image capturing, binary image generating, segmenting the image in colour space, extracting object features and providing their coordinates and tracking. The following sections provide descriptions on these individual algorithms.

4.3.4.1 Foreground Image Obtaining

The first thing needs to be done for image processing is to capture a digital image. Discrimination of the foreground image from the background image is always a choice to extract moving objects in the case of a fixed camera being used. In an image processed by this approach, the dynamic foreground will be extracted and analysed whilst the static background information is eliminated. The background image is taken off-line and

averaged over a certain frames of continuous sampled background images as follows:

$$\begin{aligned}
 Y_{bg}(img_i, img_j) &= \frac{1}{noofframes} \sum_{frame_i=1}^{noofframes} Y_{frame_i}(img_i, img_j) \\
 U_{bg}(img_i, img_j) &= \frac{1}{noofframes} \sum_{frame_i=1}^{noofframes} U_{frame_i}(img_i, img_j) \\
 V_{bg}(img_i, img_j) &= \frac{1}{noofframes} \sum_{frame_i=1}^{noofframes} V_{frame_i}(img_i, img_j)
 \end{aligned} \tag{4.31}$$

where, img_i, img_j are the row and column indexes of a pixel in an image; Y_{bg} , U_{bg} and V_{bg} are the luminance and chrominance values of the background image; the total number of images sampled for calculating the background is indicated by $noofframes$ with a serial number of the image $frame_i$; the variables Y_{frame_i} , U_{frame_i} and V_{frame_i} are the values of the luminance and chrominance components of $frame_i$.

The obtained background image is stored in the system memory. A background image re-sampling procedure may be invoked during system run-time by an operator due to environment changes.

Based on the background image, every captured real time image will be processed with the following steps in Figure 4.12,

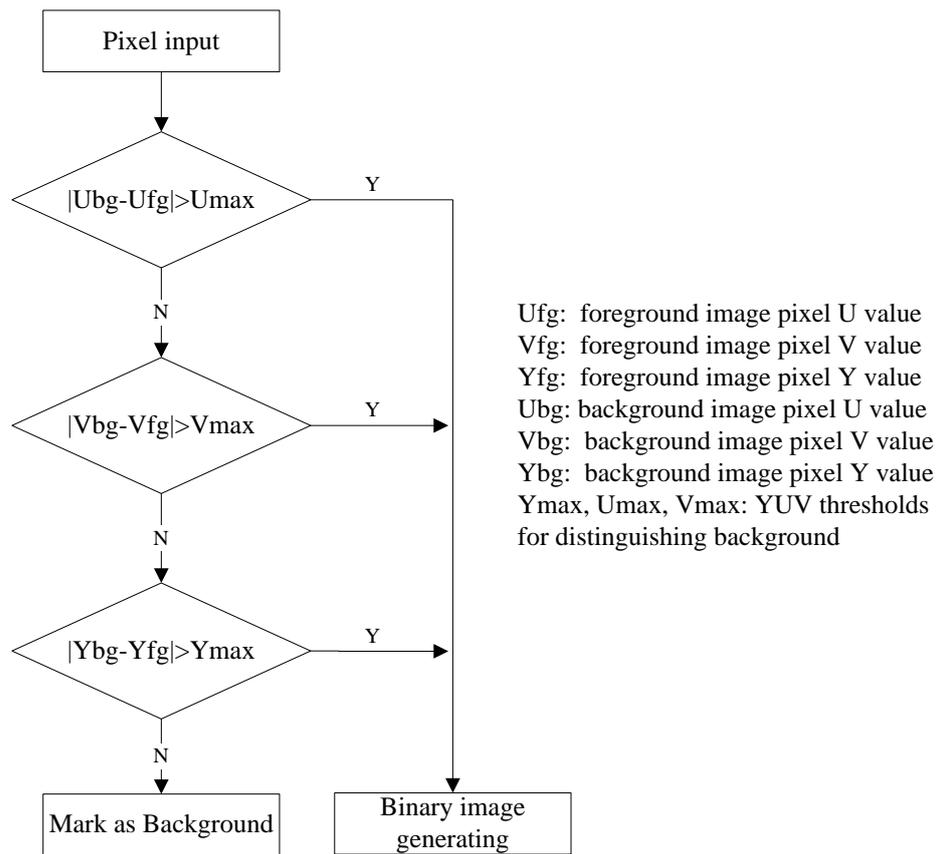


Figure 4.12 Obstacles extracting procedure flow chart

Notice that not all pixels need to complete the whole comparison procedures. This will save a lot of the processing time.

4.3.4.2 Binary Image Generating and Segmenting

When a foreground image is extracted, all the colour pixels are processed by bitwise AND operation with the associated elements in the lookup table. The result is a binary image with interested pixels being set to non zero and uninterested pixels to zero as shown in Table 4.5. In comparison with the traditional region growing algorithm for image segmentation [151], the bitwise lookup table approach needs much less computation and thus has better real-time performance. Furthermore, high efficiency can be achieved by tracking multiple objects simultaneously.

Table 4.5 Binary image

.....
1	1	1	1	0	0	0	0	2	2	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	2	2	2	2	2	2

The segmented image can be efficiently represented by run-length encoding. Each identified target in a row can be described by its colour index and the running length,

```

{
    unsigned char colour;
    int length;
}

```

For example, the image in Table 4.5 can be described by run-length encoding as Table 4.6. Specifically, the length of colour 1 (0b00000001) is 4, then length of colour 0 (background) is 4 and the length of colour 2 (0b00000010) is 2 and so on.

Table 4.6 RLE image

(1,4)	(0,4)	(2,2)	(0,2)	(0,12)	(0,3)	(1,3)	(2,6)
-------	-------	-------	-------	--------	-------	-------	-------	-------

The whole image can be compressed with a high ratio since the majority of the image is background. Figure 4.12 is an example of a run-length encoded image with red, blue and yellow targets.

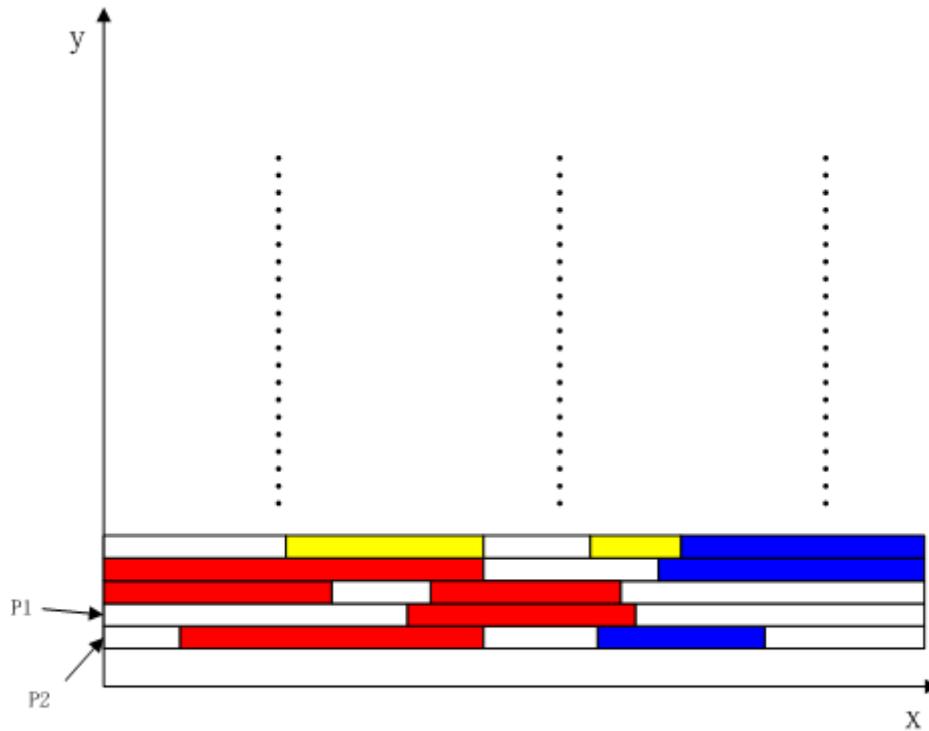


Figure 4.13 One encoded example by run-length

The following two steps are used to classify the different colour targets,

1. Two pointers, P1 and P2 as shown in Figure 4.12, are used to point to two neighbouring rows. The run-length of non zero colours is checked to see whether they overlap vertically. If overlapping occurs, then they are marked as an identical blob, otherwise they are marked as different ones;
2. Scan and merge the adjacent colour blobs.

4.3.4.3 Object Identification and Tracking

The merged blobs from the above discussions are chained arrays. Object features like blob area and blob centre coordinates can be calculated for further path planning.

The area can be easily obtained by adding all the run-lengths belonging to the same colour blob, while the centre of the blob coordinates can be obtained as,

$$\begin{aligned} x_{blob_j} &= \frac{1}{numpixels} \sum_{i=1}^{numpixels} x_{blob_j,i} \\ y_{blob_j} &= \frac{1}{numpixels} \sum_{i=1}^{numpixels} y_{blob_j,i} \end{aligned} \quad (4.32)$$

where (x_{blob_j}, y_{blob_j}) is the centre coordinates of $blob_j$ in the image, $(x_{blob_j,i}, y_{blob_j,i})$ is the coordinates of the pixels which belong to $blob_j$ and $numpixels$ is the number of pixels belong to the blob.

Due to the sensing uncertainty, two images captured in sequence by one vision sensor may be different. This will cause some offsets of the coordinates even though the scenes are actually still the same. In this case, it is necessary to smooth the coordinates by filtering. Another reason to employ a filter is that image processing takes time. During image processing, other processing modules, e.g. path planning, run in parallel and need the results obtained from the image processing module as inputs. This means that the coordinates should be ready for use even if they have not yet been updated.

Kalman filter is widely used to smooth and predict data. Define a gain coefficient $a, (0 \leq a \leq 1)$ to present the weight in percentage of the estimated coordinates when fusing them with the actual measured values.

$(\hat{x}_{blob_j,k}, \hat{y}_{blob_j,k})$ is the k step estimated coordinates of $blob_j$ given the knowledge of the process prior to step k and $(\hat{x}_{blob_j,k}^-, \hat{y}_{blob_j,k}^-)$ is the

k step posteriori estimated coordinate of $blob_j$ given measurement $(x_{blob_j,k} \ y_{blob_j,k})$ at step k .

If a measurement value is available at step k , then the following equation is used to calculate the coordinates at this step and predict the coordinates at the next step,

$$\begin{aligned}\widehat{x}_{blob_j,k}^- &= a \cdot \widehat{x}_{blob_j,k} + (1-a) \cdot x_{blob_j,k} \\ \widehat{x}_{blob_j,k+1}^- &= \widehat{x}_{blob_j,k}^- + v_{blob_j,k} \cdot \Delta t\end{aligned}\tag{4.33}$$

where,

$$\begin{aligned}v_{blob_j,k} &= (\widehat{x}_{blob_j,k}^- - \widehat{x}_{blob_j,k-1}^-) / \Delta t \\ \Delta t &= t_k - t_{k-1}\end{aligned}\tag{4.34}$$

However, if the measurement is not available at step k , the following equation is used instead of (4.33),

$$\begin{aligned}\widehat{x}_{blob_j,k}^- &= \widehat{x}_{blob_j,k} \\ \widehat{x}_{blob_j,k+1}^- &= \widehat{x}_{blob_j,k}^- + v_{blob_j,k} \cdot \Delta t\end{aligned}\tag{4.35}$$

Finally, $\widehat{x}_{blob_j,k}^-$ would be used as the actual x coordinate of blob $blob_j$ at step k . $\widehat{y}_{blob_j,k}^-$ is calculated in the similar way.

4.4 Robot Locating

In Section 4.3, a fast way to track an object is discussed. The procedure includes background and foreground discrimination, binary image generating, object identification and tracking. As introduced in Chapter 3, a robot is identified by two colour blobs on top of it. With the coordinates extracted in the last section, the robot's position and orientation can be calculated as following,

$$\begin{aligned}
\theta_{rbt} &= \arctan2(y_{blueblob} - y_{redblob}, x_{blueblob} - x_{redblob}) \\
x_{rbt} &= (x_{redblob} + x_{blueblob})/2 \\
y_{rbt} &= (y_{redblob} + y_{blueblob})/2
\end{aligned} \tag{4.36}$$

where $(x_{redblob}, y_{redblob})$ and $(x_{blueblob}, y_{blueblob})$ are the coordinates of the red and blue objects calculated by equation (4.32); the heading direction is obtained from the relation between red to blue blobs.

4.5 Obstacles Extraction

From the above discussion, colour blobs matching and discrimination of foreground-background image are used to scan dynamic obstacles. The proposed path planning algorithm is based on snake body adjustment responding to nearby obstacles. All obstacles are treated in the same way to generate forces to the snake body. So an occupancy map is used to represent obstacles for planning to simplify obstacles clustering. The earliest literature on occupancy map was [156], the map is shown in Figure 4.14. It depicts multiple cameras connected to a central computer to fuse different vision information. Exploiting the occupancy map and multiple-view observation, an obstacle detection algorithm using multiple cameras is also proposed in this section.

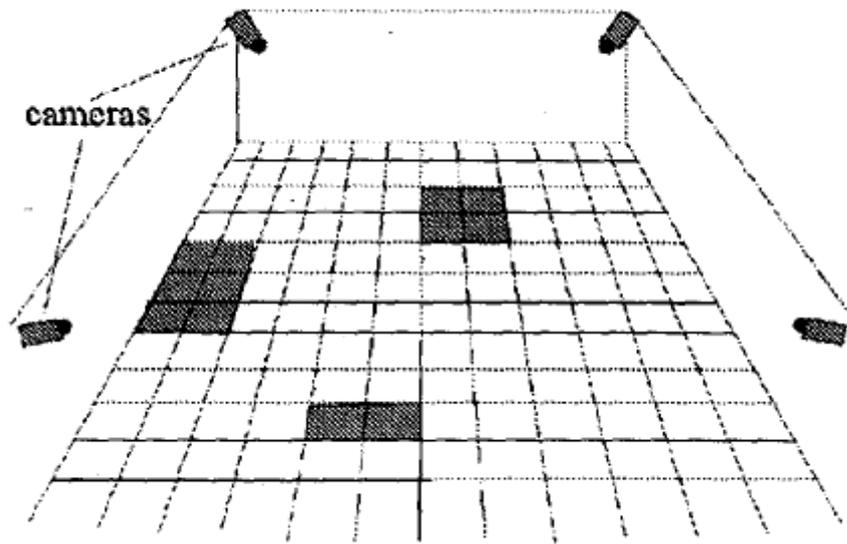


Figure 4.14 One occupancy map example from [156]

4.5.1 Occupancy Map Building

All moving objects detected by vision sensors should be treated as obstacles, except those identified as robot. Similar to the occupancy map in Figure 4.14, an image plane is divided into small overlapping regions. If an object other than a robot is detected in that region, the area is marked as an obstacle. Every obstacle, represented by regions in the image plane, has a fixed coordinate and behaves like an on-off switch with two states: active and inactive. The divided regions are shown in Figure 4.15.

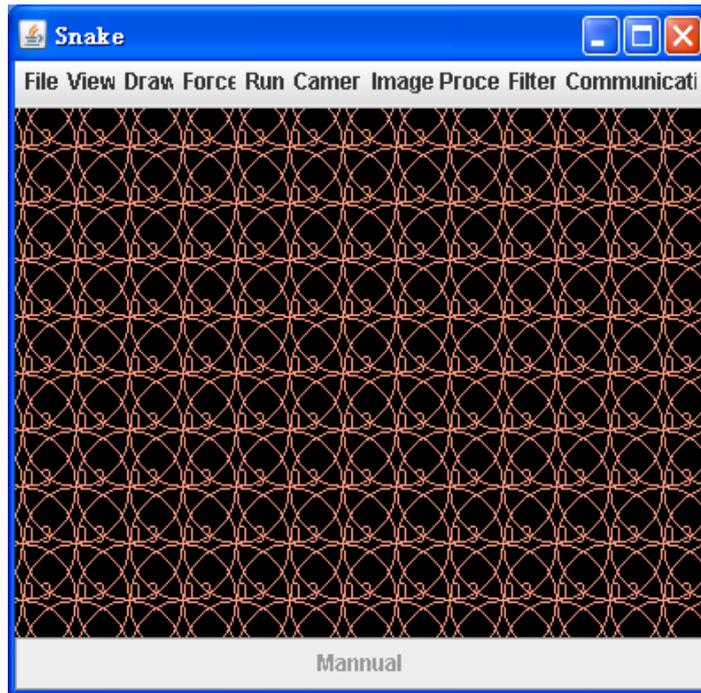


Figure 4.15 Camera space divided into smaller areas

A pseudo code is listed below to mark an image as an occupancy map, where *obr* is defined as the minimum radius of an obstacle region.

```

For ( row = obr ; row <= IMAGE_HEIGHT - obr ; row = row + obr)
For ( column = obr ; column <= IMAGE_WIDTH - obr ; column = column + obr){
  For ( prow = row - obr ; prow < row + obr ; prow = prow + 2 )
  For( pcolumn = column - obr ; pcolumn < column + obr ; pcolumn = pcolumn + 2 )
  {
    if pixel[ prow * IMAGEWIDTH + pcolumn] is occupied, increase counter;
  }
  if(counter > threshold ){
    if ( overlapping with robot )
      switch obstacle status to inactive;
    else
      switch this obstacle status to active;
  }
  clear counter;
}

```

In the above codes, image area with *IMAGE_HEIGHT* by *IMAGE_WIDTH* pixels is divided into regions indexed by (*row*, *column*) with radius *obr* and the numbers of pixels been marked as occupied within each of the regions are counted. By comparing the counting result with a given threshold and checking whether the region is overlapping with the robot, decision can be made to turn this region as obstacle or not.

When all the regions are processed completely, an occupied map with obstacles can be obtained accordingly.

4.5.2 Multiple Cameras Based Obstacle Extraction

An image captured by a vision sensor is a 2D projection of a 3D scene. In that case, reconstruction of the 3D world is not feasible from a single view. One example is shown in Figure 4.16. From the view of a vision sensor in the bottom-right area, the occluded white area in the top-left area has to be interpreted as an occupied area, even though the obstacle only occupies a small area, indicated in black in Figure 4.16, on the floor. The interpretation from a single view may enlarge the area with obstacles and affect path planning and motion control.

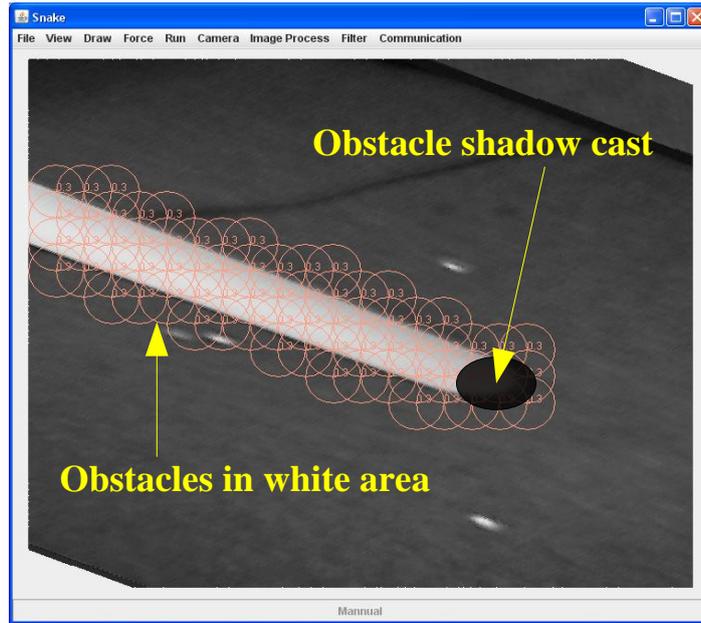


Figure 4.16 Projective image of obstacle

With multiple vision sensors provided in the proposed intelligent environment, a mapping from one vision sensor to another can be established and information can be exchanged between sensors. Thus, by employing multiple view geometry homograph [129] technique, a more accurate location of the object can be obtained.

Hence, a probability based approach by increasing the likelihood of the presence of an obstacle supported by multiple cameras is proposed to achieve more precise obstacles information in this section.

4.5.2.1 Problem Statement

A statistical model is proposed to fuse different views of vision sensors thus to increase or decrease the belief of the occupancy of one grid by an obstacle, i.e. whether a grid area is occupied. Hence the real occupancy of obstacles can be decided by using multiple sensors.

Suppose there are $numcam$ cameras mounted in the building represented by $Cam_1, Cam_2, \dots, Cam_{numcam}$ with overlapping areas in the workspace plane, which is the plane a robot will move in. The regions are indexed by j with total number of regions in the workspace plane $numreg$, $0 < j \leq numreg$. The depression angle of i^{th} camera is denoted by φ_i , with $0 < \varphi_i < \pi$ and $0 < i \leq numcam$. It is defined in Figure 4.17 as the angle formed by the horizontal direction which is in parallel with the workspace plane and the camera lens axis with the angle vertex in the camera centre.

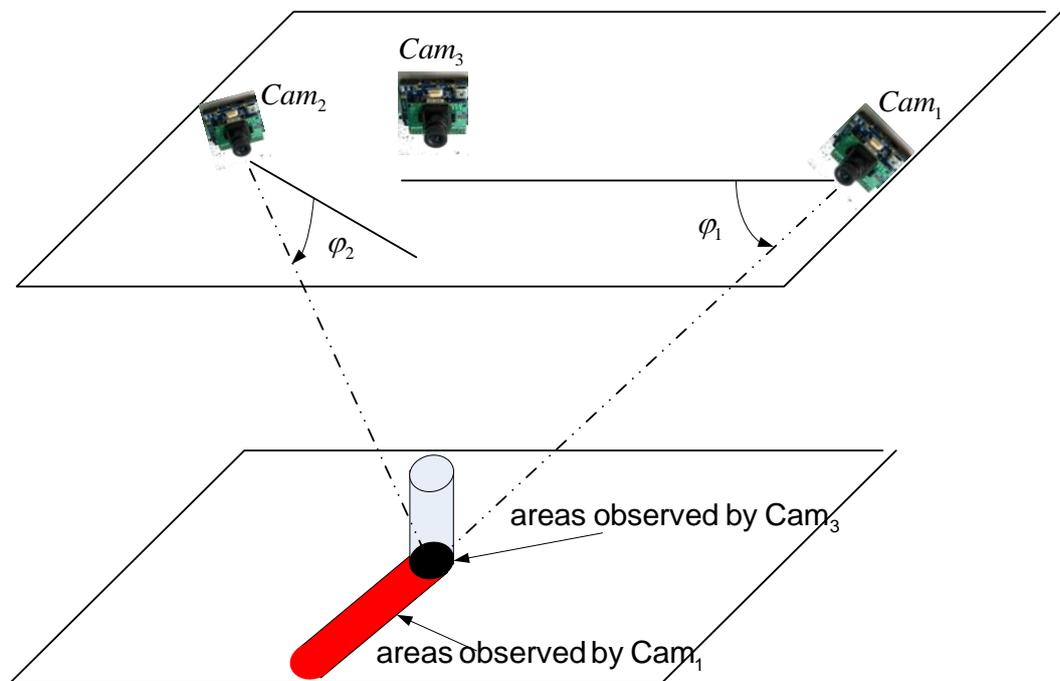


Figure 4.17 The angle of depression

Let $A_{i,j}$ be an event of objects in region j being observed by camera i ; B_j be a random variable to indicate a region j is an obstacle. So $P(A_{i,j})$ is the probability of a region j being observed as an obstacle by

camera i if an object presents and $P(B_j | A_{i,j})$ denotes the conditional probability of region j being an obstacle in the workspace given the region is observed by camera i as an obstacle. This conditional probability implies that some regions actually are not obstacles may be wrongly interpreted as obstacles by a camera due to information loss from 3D to 2D projection such as the example in Figure 4.16.

According to Bayesian theorem, the probability of region j being a real obstacle in the workspace that can be observed by camera i directly is given by,

$$P(A_{i,j} | B_j) = \frac{P(A_{i,j}) \cdot P(B_j | A_{i,j})}{\sum_{j=1}^{numreg} P(A_{i,j}) \cdot P(B_j | A_{i,j})} \quad (4.37)$$

4.5.2.2 Multiple Vision Fusion for Obstacles Extraction

To find out whether an obstacle is in existence formally, the value of $P(A_{i,j} | B_j)$, $P(A_{i,j})$ and $P(B_j | A_{i,j})$ should be constructed.

The probability $P(A_{i,j})$ is determined by the physical features of the camera such as sensor components, exposure time etc.. This probability could be very high if an object is present within the vision range of a camera. Without loss of generality, 0.9 is adopted.

$$P(A_{i,j}) = \psi \cdot 0.9 + (1 - \psi) \cdot 0.1 \quad (4.38)$$

where,

$$\psi = \begin{cases} 1, & \text{object is present} \\ 0, & \text{otherwise} \end{cases} \quad (4.39)$$

The simplest yet most important factors impacting the possibility of $P(B_j | A_{i,j})$ are the mounting positions. As shown in Figure 4.17, given a cylinder obstacle on the workspace, the obstacle area in image plane, indicated by black, observed by Cam_3 is the same as the area in workspace while the area seen by Cam_1 , marked as red, demonstrates inexact obstacles, thus the probability can be expressed as,

$$P(B_j | A_{i,j}) = \sin(\varphi_i) \quad (4.40)$$

When one region is observed by multiple cameras, each camera can see only a view of obstacles with uncertainty about the occluded areas. The real occupied area in the workspace can be detected as an obstacle or an occluded area from any view. Therefore, the joint probability of an occupied area detected by all $numcam$ cameras can be obtained by applying AND operation,

$$P(A_j | B_j) = \prod_{i=1}^{numcam} P(A_{i,j} | B_j) \quad (4.41)$$

which reports the probability of obstacles in the workspace by joint decision-making.

By substituting (4.37), (4.38) and (4.40) into (4.41), the probability of region j being an obstacle is expressed as,

$$\begin{aligned} P(A_j | B_j) &= \prod_{i=1}^{numcam} \frac{P(A_{i,j}) \cdot P(B_j | A_{i,j})}{\sum_{j=1}^{numreg} P(A_{i,j}) \cdot P(B_j | A_{i,j})} \\ &= \prod_{i=1}^{numcam} \frac{(\psi \cdot 0.9 + (1 - \psi) \cdot 0.1) \cdot \sin(\varphi_i)}{\sum_{j=1}^{numreg} (\psi \cdot 0.9 + (1 - \psi) \cdot 0.1) \cdot \sin(\varphi_i)} \end{aligned} \quad (4.42)$$

Figure 4.18 shows the probability of the existence of obstacles in regions in Figure 4.17. The depression angles of Cam_1 , Cam_2 and Cam_3 are $\pi/3$, $\pi/4$ and $\pi/2$ respectively. After fusing all three cameras, the higher probability of the existence of obstacles is gained by more observations from different cameras. The regions in the centre, highlighted in black in Figure 4.17, have the highest probability to be obstacles. This sees the same result as what been discussed above.

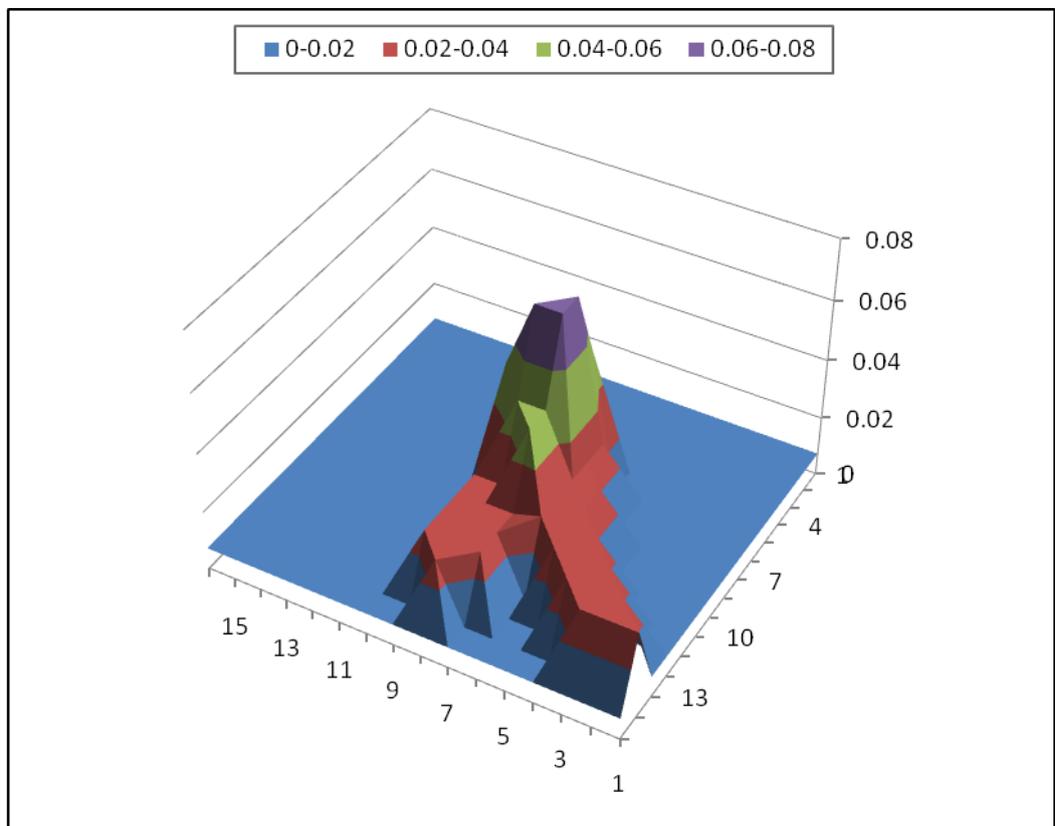


Figure 4.18 Probability of obstacle existence when number of cameras seeing it increases

One simulation result conducted in the simulation environment developed in this PhD work with two cameras is shown in Figure 4.19 and Figure 4.20, details of the software program is given in Chapter 7. Figure 4.19 depicts the obstacle projection on the workspace plane, in which the bold red circles represent the actual obstacle on the workspace plane after fusion of obstacle information from different camera information

using (4.42). Figure 4.20 illustrates the path generated from fusion of the two cameras. After the one camera combines the homography projection from the other, the probability of existence of obstacles in the overlap area becomes higher than that of others. Accordingly a path can be planned to pass the place which was blocked before applying the algorithm.

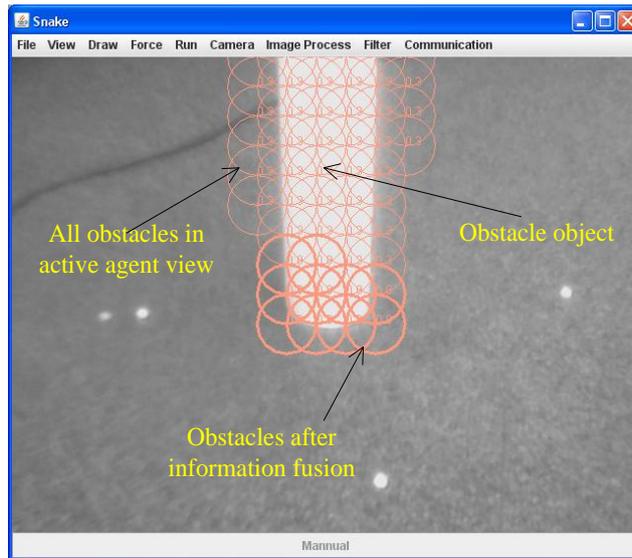


Figure 4.19 Bold red circles denotes the real shadow cast

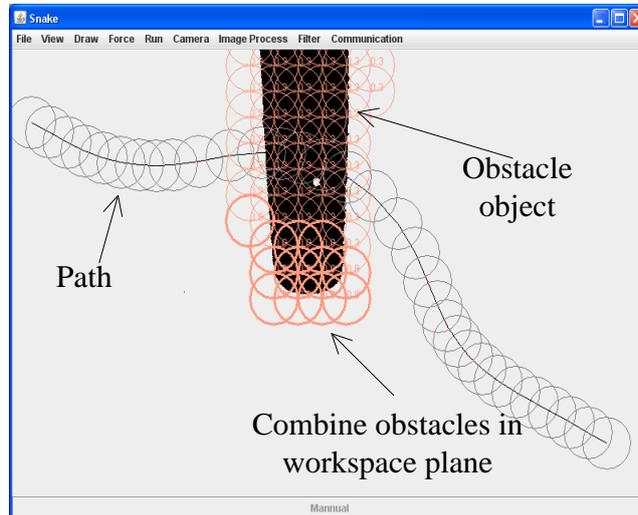


Figure 4.20 Planned path after fusion of two camera information

4.6 Conclusion

In a distributed intelligent environment with wireless vision sensors, a new paradigm can be considered for mobile robot navigation, in which the basic navigation problems such as localisation, map building, path planning etc. are significantly simplified in comparison with the traditional approach for autonomous navigation. Map building turns into two parts: static global topological map and local dynamic map; Path planning is divided into global topological routing and local path adjustment.

Hence in this chapter, Multiple Bloom filters are used as an effective mechanism for static global map building, compressing and storing, taken into account the fact that wireless sensors have limited memories and low rate communication bandwidth. By giving each of vision sensors unambiguous semantics, global topological path planning becomes on-line Bloom filters querying. After that, a fast multiple colour blob tracking mechanism is presented for robot localisation and obstacle identification. By employing a colour lookup table and bitwise AND operation, multiple target colours can be identified simultaneously. By run-length encoding, target colour blobs can be segmented very efficiently. Finally, a probabilistic model based fusion algorithm is proposed for determination of area occupancy from the distributed wireless vision sensors by cooperation of the multiple vision sensors.

The global topological routing and blob coordinates obtained in this chapter will be used as the input for dynamic path planning and trajectory generating in the next two chapters.

Chapter 5 : A DISTRIBUTED SNAKE ALGORITHM FOR MOBILE ROBOTS PATH PLANNING WITH CURVATURE CONSTRAINTS

5.1 Introduction

As discussed in previous chapters, the intelligent environment supported by a wireless sensor network provides an ideal solution to resolve the bottleneck of complexity and to cope with the dynamic change and ambiguity in the environment faced by centralised autonomous robots [1, 2]. How to coordinate the distributed sensors to generate a collision-free whilst constraint-compliant global path is one of the key challenges of this approach.

Inspired by the snake body structure and its kinetics, the contour snake model [70] is broadly used and plays an important role in computer vision for image segmentation and contour tracking. Similar concepts have been applied to path planning of centralised robot navigation with onboard sensors, such as elastic bands and bubbles [78]. In [86], a snake is defined as a flexible entity that can be deformed by applying internal and external forces on it. It is represented in the configuration space by a sequence of connected configurations. Since the deformation of a snake body is accomplished by the interactions between adjacent joints (called control points in this thesis), information exchange between distributed wireless vision sensors can be transmitted along the snake, which is an efficient mechanism to coordinate a series of sensors via communication.

Local information, such as obstacles appear, detected in one vision sensor will cause the snake segment in that vision sensor to reshape accordingly. The reshaping of that segment will affect the force balance of the neighbouring segments by exchanging the internal forces with its neighbouring vision sensors. As a result, the whole snake will be adjusted to reach a new balance to form a new path. Therefore, the snake-based approach is an ideal path planning method with distributed implementation and dynamic reaction capabilities for robot navigation supported by a wireless sensor network.

However, there are several challenges in path planning. How to satisfy mobile robot kinematic constraints when applying a planned path to the generation of robot movement instructions is a key factor to be considered during the planning phase [79]. Liang [84] presented a nonholonomic path planning method considering curvature constraint and length minimisation for a car-like robot based on cubic spirals. Winston [85] proposed two types of continuous steering functions to generate continuous curvature curves: cartesian quintics for lane changes and polar splines for symmetric turns of arbitrary-angle. Based on the latter, Sam [83] investigated the use of Polar Polynomial Curve (PPC) to produce a path that changes continuously in curvature and satisfies dynamic constraints. However, for distributed path planning especially in a dynamically changing environment lack of dedicated planning and centralised information makes it much harder to plan a path in advance subject to curvature constraints. In [86], Khatib introduced a special curvature force on the elastic band to reduce the curvature but without violation-free guarantee.

In this chapter, a snake algorithm is proposed for path planning in an environment with distributed sensors. The elastic dynamics is considered as a constrained process with three states: flexible state, rigid state and broken state. The flexible state corresponds to the conventional snake deformation. The rigid state of a snake segment indicates that the allowed curvature has been reached and the segment will move as a rigid body. Finally, extremely high bending forces exerted to a rigid segment caused by narrow passages may break the snake body and force the snake to enter into a re-planning process. The snake path planning with the state switching capability ensures that both local and global performance of each planned path segment will always be compliant with the curvature constraint locally and a path from a start position to any reachable goal is always obtainable. The proposed algorithm is suitable for distributed implementation in a wireless sensor network.

The rest of the chapter is organised as the following: Section 5.2 introduces the snake algorithm model; Section 5.3 focuses on the proposed distributed path planning by state switching; experimental results and conclusion are given in Section 5.4 and Section 5.5 respectively.

5.2 Bio-inspired Snake Algorithm

The snake algorithm [79, 80] is inspired by a snake body structure. It has ribs attached to a flexible backbone like a spring. When it moves, it sends compression/reshaping waves down the body and this force pushes the snake body to move. Snake-like movement is introduced into the robot path planning process for easy and flexible implementation in a distributed

sensor network. Suppose a snake consists of a set of control points, connected by straight lines. Each control point is moveable in the workspace, so a snake is entirely specified by the number and coordinates of its control points, which defines an admissible path for robot travelling. Adjustment of a path is made by moving the control points. All the control points are under the effect of both internal and external forces in order to reduce its energy to avoid collision with obstacles and to minimise its length. In response to changes, the snake is reshaped to minimise its energy value, which forms a safe path with lower control cost to a destination.

5.2.1 Problem Statement

Let p_i be a Euclidian coordinate, the Cartesian configuration (x_i^{cp}, y_i^{cp}) in space R^2 represents a snake control point, where $i: 0 \leq i < n$ is an integer, the superscript cp stands for control point and n is the total number of control points. A snake is defined by a collection of n configurations being connected in a sequential order and is denoted as $\{p_0, p_1, \dots, p_{n-1}\}$. An obstacle is defined as a circle with a radius d_o centred at (x_j^o, y_j^o) and is denoted as q_j , where $j: 0 \leq j < m$ is an integer, the superscript o stands for obstacle, m is the total number of obstacles. Then the objective of the snake algorithm is to adjust the n control points $\{p_0, p_1, \dots, p_{n-1}\}$ dynamically in order to keep the robot at a safe distance from the m obstacles $\{q_0, q_1, \dots, q_{m-1}\}$, satisfying the given curvature

constraint and maintaining the shortest path length from the start to the goal.

As discussed above, the safe path for a robot is represented as a snake maintained by the distributed mosaic eyes. Define the internal and external energy functions in space R^2 as the energy caused by attractive actions from adjacent control points and repulsive actions from obstacles respectively, then the total energy, E^{snake} , of the snake can be expressed as,

$$E^{snake} = E^{internal} + E^{external} \quad (5.1)$$

where the internal energy, $E^{internal}$, is only concerned with the intrinsic actions of the snake such as its shape and length while the external energy, $E^{external}$, is concerned with the effect from the environment, such as obstacles.

To minimise the energy of a snake, p_i should move along the negative energy gradient direction so that the total energy of the snake decreases. The total force, F^{snake} , exerted on the snake can be expressed in terms of $E^{internal}$ and $E^{external}$ as,

$$\begin{aligned} F^{snake} &= F^{internal} + F^{external} \\ &= -\nabla(\eta_{int} \cdot E^{internal}) - \nabla(\eta_{ext} \cdot E^{external}) \end{aligned} \quad (5.2)$$

where, η_{int} and η_{ext} are positive coefficients representing the force strength, ∇ is the gradient operator.

In (5.2), different forces serve different purposes. The target of path planning is to find an obstacle-free path satisfying the curvature constraints with the shortest length. Thus, the elastic contraction energy and curvature

bending energy are considered as internal energy and the obstacles repulsive energy is considered as external energy in the subsections that follows.

To define a suitable energy and then work out the forces accordingly, two rules should be applied:

- The forces should be normalised;
- Each force should have an action range.

These two rules are explained and applied to the internal and external forces in the subsections below.

5.2.2 The Internal Force

The first term in the internal force is defined as an elastic force. This force exerts attractions between control points, causing contraction of the snake and as a result the length of the snake is reduced. The energy of this elastic force is expressed as,

$$E^{elastic} = \kappa_1 \cdot \sum_{i=1}^{n-1} (\|p_i - p_{i-1}\|)^2 \quad (5.3)$$

where, $\|p_i - p_{i-1}\|$ is the Euclidian distance between p_i and p_{i-1} ; κ_1 is a constant factor. Consider that each control point is pulled by two neighbours, the energy is defined as,

$$E^{elastic} = \kappa_2 \cdot \sum_{i=1}^{n-2} (\|p_i - p_{i-1}\|^2 + \|p_{i+1} - p_i\|^2) \quad (5.4)$$

where, κ_2 is the constant factor of the elastic force. By substituting (5.4) into (5.2), the force on the x direction can be obtained as,

$$\begin{aligned}
f_{i,x}^e &= -\nabla(\eta_{\text{int}} \cdot E_x^{\text{elastic}}) = -\eta_{\text{int}} \cdot \kappa_2 \cdot \frac{\partial(\|p_i - p_{i-1}\|^2 + \|p_{i+1} - p_i\|^2)}{\partial x_i} \\
&= -\eta_{\text{int}} \cdot \kappa_2 \cdot \frac{\partial((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 + (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2)}{\partial x_i} \\
&= -\kappa_3 \cdot ((x_i - x_{i-1}) + (x_i - x_{i+1}))
\end{aligned} \tag{5.5}$$

where $\kappa_3 = 2 \cdot \kappa_2 \cdot \eta_{\text{int}}$. The normalized force in the x direction is,

$$f_{i,x}^e = -\kappa_3 \cdot \left(\frac{(x_i - x_{i-1})}{\|p_i - p_{i-1}\|} + \frac{(x_i - x_{i+1})}{\|p_i - p_{i+1}\|} \right) \tag{5.6}$$

Similarly, the normalized force in the y direction can be obtained as,

$$f_{i,y}^e = -\kappa_3 \cdot \left(\frac{(y_i - y_{i-1})}{\|p_i - p_{i-1}\|} + \frac{(y_i - y_{i+1})}{\|p_i - p_{i+1}\|} \right) \tag{5.7}$$

Another internal force is termed as curvature force which can be deduced from a curvature energy $E^{\text{curvature}}$. The curvature energy should be designed based on the shape of the snake. The more the snake bends, the bigger the energy it needs. Figure 5.1 shows how the curvature energy works.

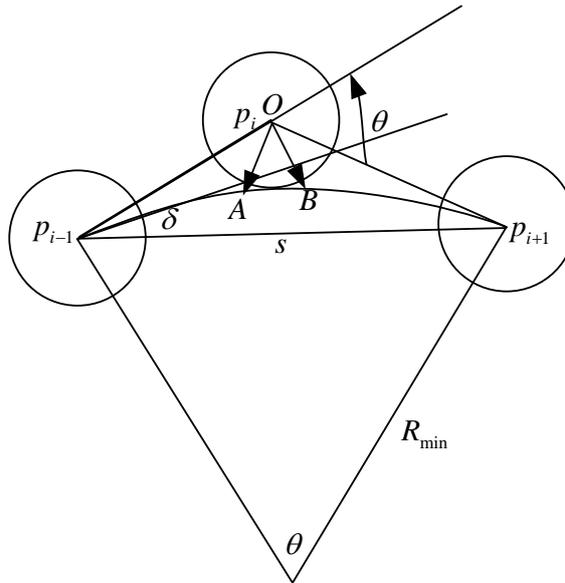


Figure 5.1 Curvature and curvature energy

The preceding and succeeding control points of control point p_i form a bending angle θ that reflects the value of the curvature. Define a threshold of the minimum allowed bending angle $\delta: \delta \in R^+$. When the bending angle is small enough such that $-\delta < \theta < \delta$, then the energy on this specific control point becomes zero. In the proportion of the bending angle, the curvature energy can be expressed as,

$$E^{curvature} = \begin{cases} \kappa_4 \cdot (|\theta| - \delta)^2, & |\theta| > \delta \\ 0, & \text{others} \end{cases} \quad (5.8)$$

where the bending angle θ belongs to the interval $(-\pi, \pi]$ and κ_4 is a positive coefficient for curvature energy. Figure 5.2 shows the profile of the curvature energy.

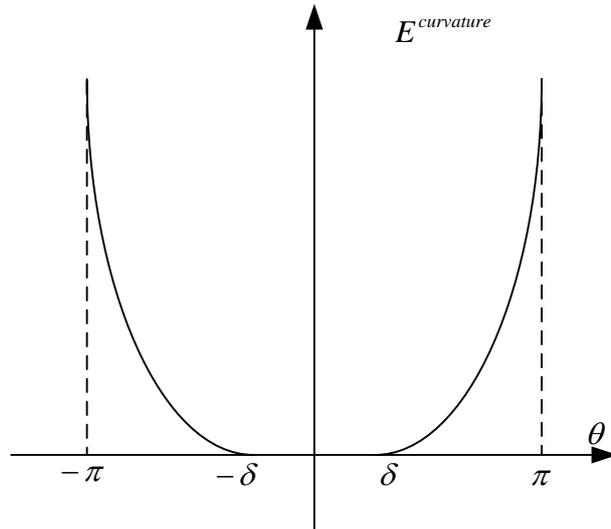


Figure 5.2 Form of curvature energy function

In (5.8), the threshold δ can be tuned by the maximum allowed curvature k_{\max} and the Euclidian distance s between the preceding and the succeeding control points of p_i as,

$$\sin(\delta) = s/2R_{\min} = s \cdot k_{\max} / 2 \quad (5.9)$$

For a small δ , it can be approximated as,

$$\delta = s/2R_{\min} = s \cdot k_{\max} / 2 \quad (5.10)$$

By substituting (5.8) into (5.2), the curvature forces can be derived as,

$$f_{i,y}^{curvature} = \begin{cases} -\kappa_5 \cdot \text{sign}(\theta)(|\theta| - \delta) \frac{\partial \theta}{\partial y_i}, & |\theta| > \delta \\ 0, & \text{others} \end{cases} \quad (5.11)$$

$$f_{i,x}^{curvature} = \begin{cases} -\kappa_5 \cdot \text{sign}(\theta)(|\theta| - \delta) \frac{\partial \theta}{\partial x_i}, & |\theta| > \delta \\ 0, & \text{others} \end{cases} \quad (5.12)$$

where, $\kappa_5 = 2 \cdot \kappa_4 \cdot \eta_{\text{int}}$, θ is the angle from direction $p_i \rightarrow p_{i+1}$ to direction

$p_{i-1} \rightarrow p_i$, $\text{sign}(\theta)$ is defined as,

$$\text{sign}(\theta) = \begin{cases} 1, & \theta \geq 0 \\ -1, & \theta < 0 \end{cases} \quad (5.13)$$

and,

$$\begin{aligned} \frac{\partial \theta}{\partial x_i} &= \frac{\partial(\arctan(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}) - \arctan(\frac{y_{i+1} - y_i}{x_{i+1} - x_i}))}{\partial x_i} \\ &= \frac{(-1)(y_i - y_{i-1})}{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} + \frac{(-1)(y_{i+1} - y_i)}{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \\ &= \frac{-\sin(\theta(p_i, p_{i-1}))}{d(p_i, p_{i-1})} + \frac{-\sin(\theta(p_i, p_{i+1}))}{d(p_i, p_{i+1})} \end{aligned} \quad (5.14)$$

$$\frac{\partial \theta}{\partial y_i} = \frac{\cos(\theta(p_i, p_{i-1}))}{d(p_i, p_{i-1})} + \frac{\cos(\theta(p_i, p_{i+1}))}{d(p_i, p_{i+1})} \quad (5.15)$$

5.2.3 The External Force

External forces are generated by obstacles. The obstacles energy is proportional to the sum of the Euclidian distances between obstacles and control points. When the distance is greater than the threshold d_0 , the

energy is reduced to zero. The repulsive external energy at p_i is defined as,

$$E^{obstacle} = \begin{cases} \kappa_6 \cdot \sum_{j=0}^{m-1} (\|p_i - q_j\| - d_0)^2, & d < d_0 \\ 0, & \text{others} \end{cases} \quad (5.16)$$

where, m is the number of obstacles with distance threshold d_0 .

Substitute (5.16) into (5.2), the obstacle force exerted on p_i is obtained as,

$$f_{i,x}^o = \begin{cases} \kappa_7 \cdot \sum_{j=0}^{m-1} (\|p_i - q_j\| - d_0) \cdot \frac{x_i - x_j}{\|p_i - q_j\|}, & \|p_i - q_j\| < d_0 \\ 0, & \text{others} \end{cases} \quad (5.17)$$

$$f_{i,y}^o = \begin{cases} \kappa_7 \cdot \sum_{j=0}^{m-1} (\|p_i - q_j\| - d_0) \cdot \frac{y_i - y_j}{\|p_i - q_j\|}, & \|p_i - q_j\| < d_0 \\ 0, & \text{others} \end{cases} \quad (5.18)$$

where, $\kappa_7 = 2 \cdot \kappa_6 \cdot \eta_{ext}$

By adding up all internal and external forces, one gets the resultant

force $\|f_i^r\|$ on a control point p_i where $\|f_i^r\| = \sqrt{f_{i,x}^r + f_{i,y}^r}$ and,

$$f_{i,x}^r = f_{i,x}^o + f_{i,x}^e + f_{i,x}^{curvature} \quad (5.19)$$

$$f_{i,y}^r = f_{i,y}^o + f_{i,y}^e + f_{i,y}^{curvature} \quad (5.20)$$

All control points will be moved by the exerted forces, following the iterative formula as,

$$x_{i+1} = f_{i,x}^r + x_i \quad (5.21)$$

$$y_{i+1} = f_{i,y}^r + y_i \quad (5.22)$$

Being exerted by the internal and external forces, the snake will deform accordingly to attenuate the total energy so as to achieve a new balance. As a result, an obstacle free, energy minimised snake path can be obtained.

5.3 Three State Snake under Curvature Constraints

In a distributed environment, the control points of a snake are adjusted and maintained by different vision sensors and are treated individually. The connections between them are provided by different forces. Although dedicated curvature forces are designed in (5.11) (5.12) to reduce the snake curvatures, they cannot guarantee that the curvature will not violate the curvature threshold. This is due to the lack of any curvature violation check throughout the snake control points, though this kind of check is easy to be carried out in centralised applications. Thanks to the inspiration from a snake body, the vertebrae of a snake set a limitation on how far the snake body can bend. Different from a rubber band, a snake body becomes rigid when it reaches a certain range and cannot be bent further. If one forces the snake body to bend more than its limitation, the vertebrae bones will be broken. Hence, a three state snake is proposed to solve the distributed snake path curvature constraint problem. The states during snake deformation due to the curvature are:

1. **Flexible state:** in which a control point can be moved freely by the forces from obstacles and neighbour nodes. This state happens when the curvature of a control point is less than the threshold or the resultant force is trying to reduce the curvature;
2. **Rigid state:** in which a control point and its immediate neighbours move as a rigid body under the forces from obstacles and other neighbour nodes. Their relative positions remain unchanged when moving in order not to violate the curvature constraint;

3. **Broken state:** in which a snake segment in the rigid state is broken under a big force from obstacles, for example due to a very narrow passage such that the curvature constraint has to be broken for the robot to traverse. Under this circumstance, a new snake will be re-initiated in order to find a safe path.

5.3.1 Snake Algorithm in a Distributed Environment

During the snake adjustment, all control points will be moved by the exerted internal and external forces from the neighbouring nodes and the environment respectively. If the whole snake is maintained by a centralised computer, all obstacles and control points coordinates are buffered and shared in the memory and are ready for use. Constraints violation can be checked by approaches reported in [83-85] considering all global information. However, in a distributed environment with sensors, the snake is segmented into small portions which are maintained and adjusted by different sensors. Each portion is a composite of several control points. In order to coordinate distributed control points, every control point is encapsulated into a software component which can exchange information with other control points through communication.

Intuitively, the interactions between neighbouring segments of a nature snake body or elastic band are through the inner force determined by their relative positions. This is the case under the flexible state, control points are adjusted freely without violating any constraints by exchanging coordinates or inner forces with other control points. However, more data exchanges are required when the control points are under the rigid state.

Because these control points are linked rigidly, their motions have to be synchronised and as a result signals need to be passed to all connected control points which are in the rigid state. The information exchange flow between control points p_i is shown in Figure 5.3.

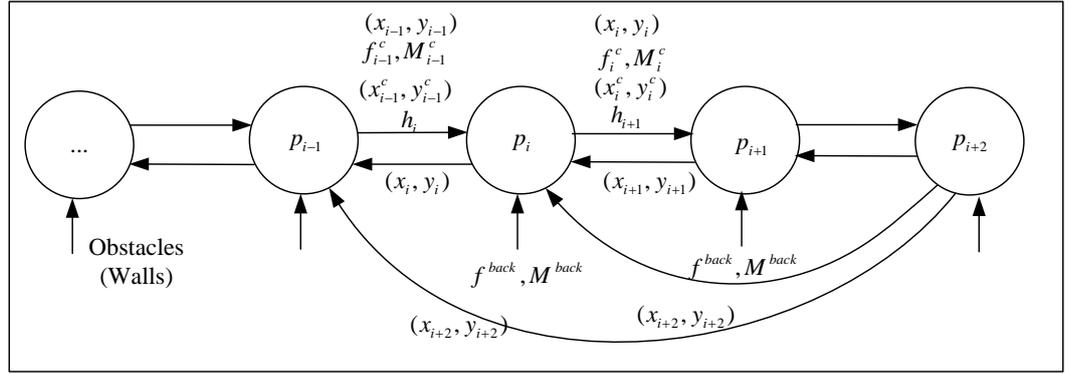


Figure 5.3 Information exchange flow between control points. (x_{i-1}, y_{i-1}) , (x_i, y_i) and (x_{i+1}, y_{i+1}) are coordinates of control points p_{i-1} , p_i and p_{i+1} respectively, f^{back} , M^{back} are the average equivalent force and torque needed to send back to related control point, (x_{i-1}^c, y_{i-1}^c) , (x_i^c, y_i^c) are centroid coordinates of control points p_{i-1} and p_i , hop indicates the state of curvature at a control point, f_{i-1}^c, M_{i-1}^c and f_i^c, M_i^c are calculated average centroid force couple and average centroid torque

Let (x_{i-1}, y_{i-1}) , (x_i, y_i) and (x_{i+1}, y_{i+1}) be the coordinates of control points p_{i-1} , p_i and p_{i+1} respectively. From the definition of force functions in (5.6), (5.7), (5.11), (5.12), (5.17) and (5.18), the coordinates are the only information required by control point p_i from p_{i-1} and p_{i+1} when it is in the flexible state. If only elastic force exists, the snake will shrink to its minimum length. If external obstacle forces exist, the snake will avoid obstacles and evolve to equilibrium according to the exerted forces. If there is a change in the environment, the snake will dynamically deform in such a way that each control point will exert a force that will affect its neighbours until the force flow pass through all nodes, reaching a new equilibrium again.

If p_{i-1} , p_i and p_{i+1} are in the rigid state, it implies that these three control points have reached or exceeded the curvature constraint and they will behave collectively as a whole rigid body and maintain their relative positions. p_{i-2} and p_{i+2} should also be parts of this rigid body in order to maintain their relative positions among all these five points in order to satisfy the curvature constraints when they switch to the rigid state. According to theoretical mechanics [157] for the plane motion of a rigid body, multiple forces acting on a rigid body can be replaced by a single force together with a force couple acting at a specified point. Thus for each control point p_i , it will calculate the coordinates of a specified point, the centroid (centre of mass), the resultant forces and force couple by adding up coordinates and forces received from its preceding node p_{i-1} and then passing them to its succeeding node p_{i+1} . When the last node of the rigid body is reached, it will pass back the final centroid coordinates, the resultant forces and force couple to preceding nodes in order to generate movements of the whole rigid body.

In Figure 5.3, h_i is the hop value at control point p_i , where $0 \leq h_i < n, h_i \in R$, indicating the state of curvature at the control point. Suppose that the node p_{i-1} is the first node reaching its curvature limitation, it will increase its hop value by 1¹. This hop value will determine whether the node is within its rigid body state, which is indicated by T_h (0 if the control point is in a rigid state or 1 otherwise). For example, if both the

¹ Note that the initialisation hop value of a control point is 0.

current control point p_i and the preceding control point p_{i-1} have not reached its curvature limitation ($h_i = 0$ and $h_{i-1} = 0$), then T_h is 1 and the control point p_i is not in a rigid state. Thus, T_h will only be equal to 1 if and only if both h_i and h_{i-1} are equal to 0, as shown in Table 2.1 where $\lambda > 0, \in \mathcal{N}$ indicating the hop value of control point p_i .

Table 5.1 Relation between hops and T_h

h_{i-1}	h_i	T_h
0	0	1
0	1	0
λ	0	0
λ	$\lambda + 1$	0

Taking the assumption that all control points have the same mass, the centroid is then the geometry centre of all nodes within the rigid body. The task is to calculate a) the rigid body centroid coordinates (x_i^c, y_i^c) formed by p_{i-1} and p_i ; b) the equivalent force on centroid f^c ; and c) the equivalent force couple M^c on the centroid. These can be obtained from equations (5.23) and (5.24) for (a), (5.25) and (5.26) for (b), and (5.27) and (5.28) for (c) respectively.

$$x_i^c = (1 - T_{h_i}) \cdot \frac{(h_{i-1} + 1) \cdot x_{i-1}^c + x_i}{h_{i-1} + 2} + T_{h_i} \cdot x_i \quad (5.23)$$

$$y_i^c = (1 - T_{h_i}) \cdot \frac{(h_{i-1} + 1) \cdot y_{i-1}^c + y_i}{h_{i-1} + 2} + T_{h_i} \cdot y_i \quad (5.24)$$

$$f_{i,x}^c = (1 - T_{h_i}) \cdot f_{i-1,x}^c + f_{i,x}^r \quad (5.25)$$

$$f_{i,y}^c = (1 - T_{h_i}) \cdot f_{i-1,y}^c + f_{i,y}^r \quad (5.26)$$

$$M_{i,x}^c = (1-T_{h_i}) \cdot (M_{i-1,x}^c + (y_i - y_i^c) \cdot f_{i,x}^r + (y_{i-1}^c - y_i^c) \cdot f_{i-1,x}^c) \quad (5.27)$$

$$M_{i,y}^c = (1-T_{h_i}) \cdot (M_{i-1,y}^c + (x_i - x_i^c) \cdot f_{i,y}^r + (x_{i-1}^c - x_i^c) \cdot f_{i-1,y}^c) \quad (5.28)$$

The average equivalent force, f^{back} , and the average force couple, M^{back} , being passed back from the last node of the rigid body to preceding nodes can then be calculated as,

$$f_{i,x}^{back} = \frac{f_{i,x}^c}{h_{i-1}+2}, f_{i,y}^{back} = \frac{f_{i,y}^c}{h_{i-1}+2} \quad (5.29)$$

$$M_i^{back} = \frac{M_i^c}{h_{i-1}+2} \quad (5.30)$$

As a result, all the nodes in the rigid body will move as a rigid body without violating the curvature constraints.

5.3.2 States Transfer

Environment change will cause a snake to deform. During snake deformation, how to determine in which state the control points should be? The criteria could be the distance between control points or could be external forces or could be internal forces. From the above discussion, the information exchanged between vision sensors is the internal force, thus this section will provide a uniform criteria of internal force thresholds to determine the states transfer. There are five state switchovers of a control point as shown in Figure 5.4 and these will be described in the following sections.

5.3.2.1 Flexible State to Rigid State

From (5.11) and (5.12), the curvature force is designed to reduce the curvature value. If a control point moves within a limited area, the curvature force is 0. If the curvature force is nonzero and the resultant force vector exerted on a control point p_i is in the opposite direction of the curvature vector, this control point will change from its current flexible state to the rigid state and the precondition can be expressed as

$$\| \vec{f}_i^{curvature} \| > 0, \text{ and, } \vec{f}_i^r \cdot \vec{f}_i^{curvature} \leq 0 \quad (5.31)$$

5.3.2.2 Rigid State to Flexible State

From the above statement, if the curvature force is zero or the resultant force vector exerted on the control point p_i is in the same direction of the curvature vector, the control point will be in the flexible state with the following precondition,

$$\| \vec{f}_i^{curvature} \| = 0, \text{ or, } \vec{f}_i^r \cdot \vec{f}_i^{curvature} > 0 \quad (5.32)$$

5.3.2.3 Flexible State to Broken State

Regardless of the reason, if the distance between an obstacle and a control point is less than the sum of the control point's radius and the obstacle's radius, the control point is at the risk of colliding with the obstacle. The snake path should enter into the broken state and a re-initialisation of the path is needed. The precondition for this change is as follow,

$$\| \vec{f}_i^o \| > \zeta_1 \cdot f^{\max} \quad (5.33)$$

where ζ_1 is a threshold constant, and,

$$f^{\max} = \zeta_2 \cdot (d_{cp} + d_o - \zeta_3) \quad (5.34)$$

where ζ_2 and ζ_3 are positive constants, d_{cp} is the radius of the control point and d_o is the radius of an obstacle.

5.3.2.4 Rigid State to Broken State

In the rigid state, the actual force exerted on an individual control point is passed by the successive node. The observed internal force is defined as below,

$$f_i^{internal} = f_i^{back} - f_i^o \quad (5.35)$$

When the observed internal force increases to a threshold, it means that the control point formally enters into the broken state such that,

$$\| \vec{f}_i^{internal} \| > \zeta_4 \cdot f^{\max} \quad (5.36)$$

where ζ_4 is a positive constant.

5.3.2.5 Broken State to Flexible State

In the broken state, the whole snake should be re-initiated by a global path search algorithm. If the re-initiation process is successful, a new path is obtained and all control points will return to the flexible state. Otherwise another attempt to re-initiate a new path will be made. There is no direct state transfer from the broken state to the rigid state because a re-initiated snake is always in its flexible state.

Figure 5.4 summarises the state criteria and state changes discussed in the previous paragraphs.

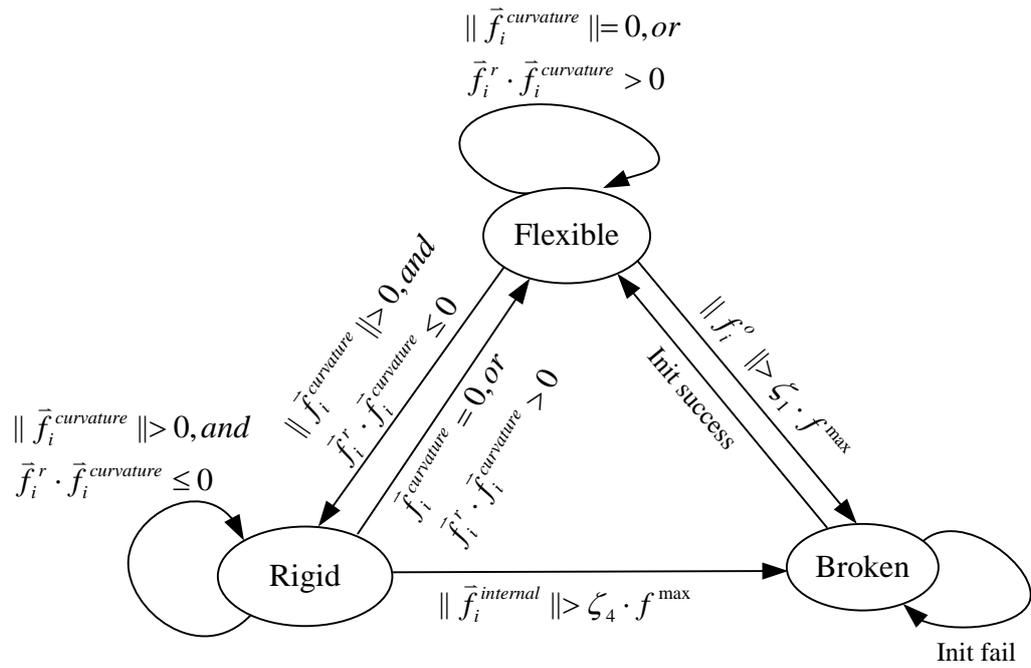


Figure 5.4 Control points state exchange chart

5.4 Simulation

This section aims to develop a distributed intelligent path planning simulation environment in which wireless connected cameras are mounted on the wall to provide live video of an environment. The start and destination points are given as inputs by a user. Initial snake control points are generated by the Dijkstra algorithm, which takes around 500ms to generate 17 control points in our simulation with 16x12 searching space. Once a snake is initiated, all control points except the start and destination control points are exerted by the resultant force from internal and external ones and will be able to adjust their position accordingly in real time. The connections of all control points thus make up a dynamic collision free path.

All control points are encapsulated in independent Java class modules for simulating the communication flow between control points. The simulations are carried out on a PIV Intel 1.6GHz PC with 1GB memory and communicating with two GP-723 2.4GHz wireless cameras. The curvature constraint of the path is 0.01, and other parameters used in this simulation are listed in Table 5.2,

Table 5.2 Parameters used in the simulation

Param	η_{int}	η_{ext}	κ_2	κ_6	κ_4	k_{max}	R_{min}
Value	1	1	0.05	0.05	20	0.01	100
Param	ζ_3	f^{max}	d_{cp}	d_o	ζ_4	ζ_1	ζ_2
Value	60	4	20	20	8	1	0.1

Figure 5.5 shows the initiated snake with $n = 17$ control points and $m = 0$ obstacles. The numbers attached to each control point denotes its sequence number and its curvature, separated by a comma. Since there are no obstacles around, the external force is 0. The resultant force from neighbours balances the path to be a straight line.

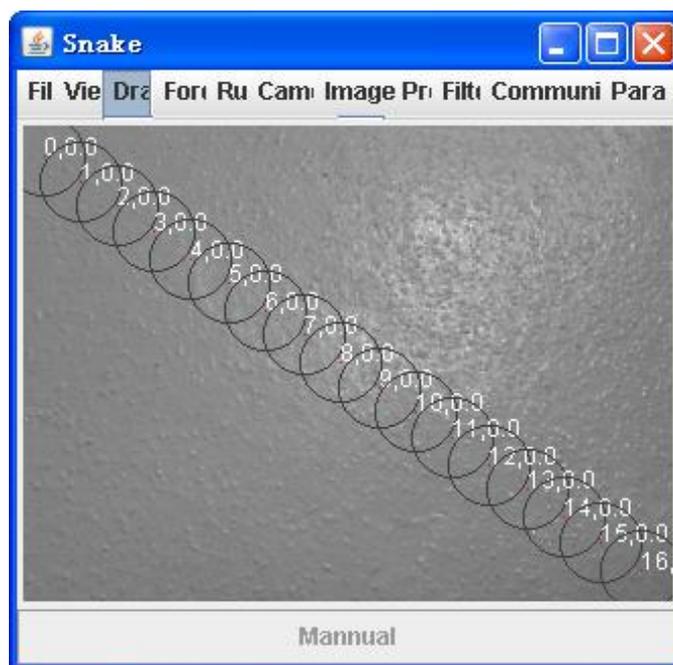


Figure 5.5 Initiated snake

When obstacles approach the snake control points within the threshold distance, control points nearest to the obstacles are pushed by the obstacle force and move to their new equilibrium positions. The yellow circles (numbered 9, 11) indicate that the curvatures of these control points almost reach their curvature constraint but they are still in the flexible state. To guarantee the connectivity of snake control points, some algorithms are developed to dynamically add or remove control points. After this, the number of control points increases to $n = 18$. The number of obstacles is $m = 17$ as shown in Figure 5.6.

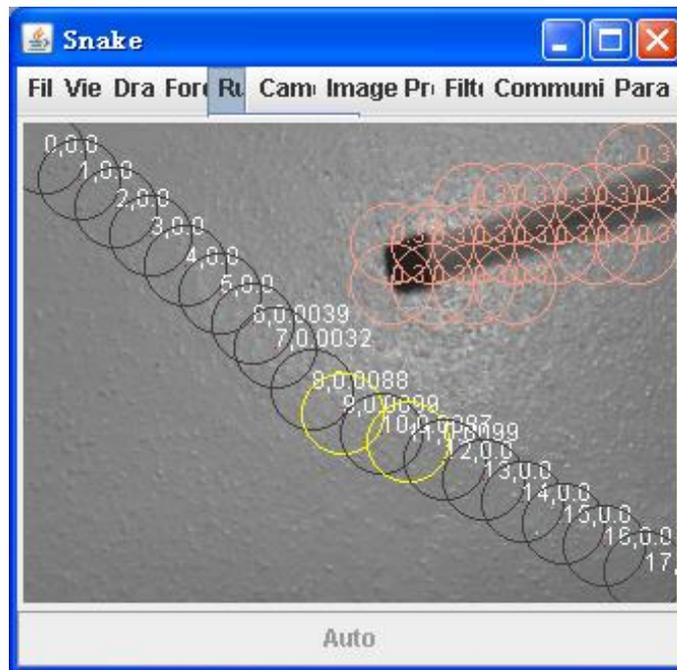


Figure 5.6 Flexible state with obstacles

When the obstacles move nearer to the snake, the states of the control points 7,8,9,10,11 are transferred to the rigid state, forming a rigid body to avoid any further increase in the curvature. As seen in Figure 5.7, the maximum curvature is 0.0099 which is less than the curvature constraint of 0.01. Control points are trying to deform in order to avoid the obstacles.

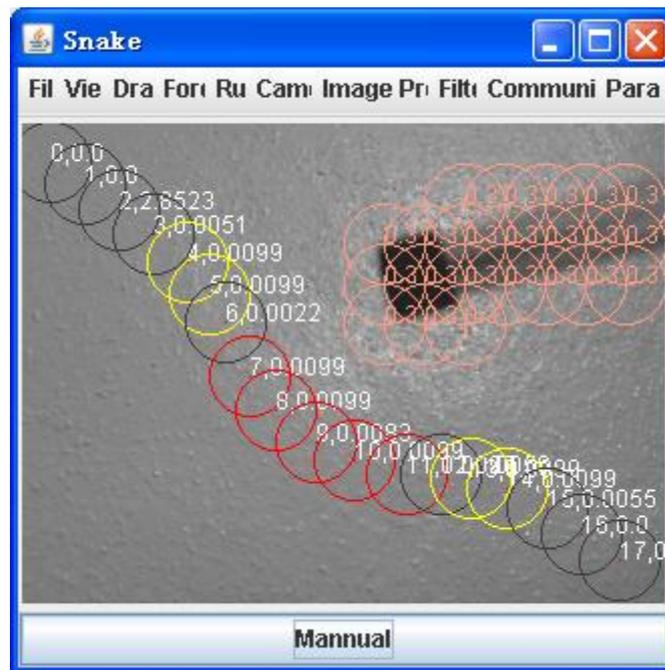


Figure 5.7 Some rigid state control points

When the internal force is greater than the threshold ($f_{\max} = 2.0$), the whole way is blocked and the control points will enter into the broken state. A new snake will be re-initiated as shown in Figure 5.8 and Figure 5.9.

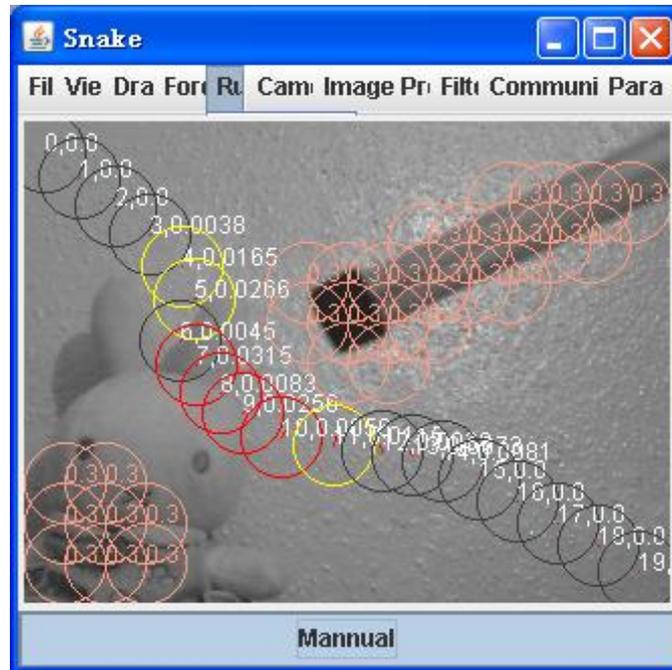


Figure 5.8 More obstacles approach snake

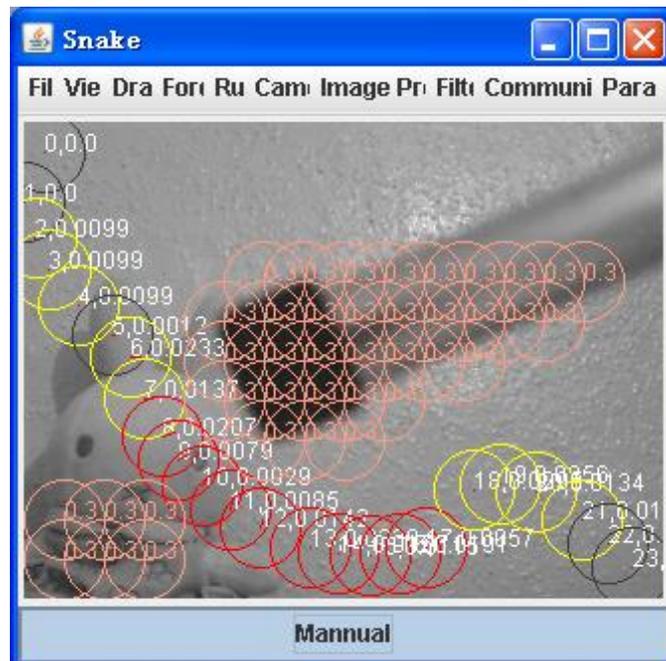


Figure 5.9 Reaching broken state

During the broken state, a global Dijkstra algorithm is carried out to search for a path from a start point to a destination point in order to re-initiate a new snake. The process keeps on trying to search for a new path until it finds a feasible path when the scene changes. The re-initialised snake is shown in Figure 5.10.

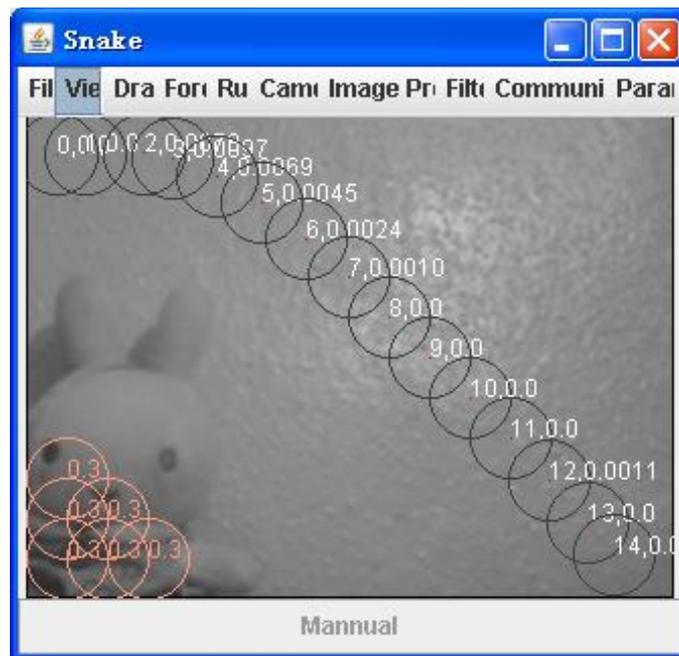


Figure 5.10 Re-initiated path

A snake path planning process experiencing flexible – rigid – broken – flexible states transfer is simulated. Although the initiated snake is planned by a global search algorithm, each of the control points is encapsulated in a separate class instances. The information exchange between control points is realised by accessing a virtual communication module. More details of the simulation environment will be discussed in Chapter 7. From the simulation result, the following facts can be concluded,

1. The snake is able to react to the dynamic obstacles to form an obstacle-free path;

2. By the exchange of internal forces and following the defined state transfer thresholds, distributed control points successfully achieve a global performance collaboratively subject to the curvature constraints by switching their states;
3. The algorithm can be applied in distributed environment for path planning.

5.5 Conclusion

This chapter proposes a snake path planning algorithm in a distributed intelligent environment. To solve the curvature constraint problem, a three state snake is proposed. One snake is split into smaller portions which can be maintained by different sensors; each portion contains several control points and each control point is treated individually. Control points can be adjusted according to the force exerted on them and can exchange information with neighbouring control points. A state machine was presented to control the snake control point movement such that the curvature constraints can always be satisfied. When control points are in different states, segments of the snake may become flexible, rigid or broken depending on the obstacles in the environment and the curvature constraints of robot kinematics. The general preconditions for state transfer were determined by using only local information and shown to be suitable for distributed applications. Simulations verified that the proposed snake based approach is an effective distributed solution for simultaneously conducting global and local path planning.

The snake obtained in this chapter will form a reference path for further trajectory generating and mobile robot motion control in Chapter 6.

Chapter 6 : PREDICTIVE TRACKING CONTROL WITH DYNAMIC CONSTRAINTS

6.1 Introduction

While the R-snake obtained in Chapter 5 provides a reference path for a robot to travel, controlling the robot to follow the path is another difficult task. This involves trajectory generation and motion control subject to non-holonomic constraints and saturated torques, slippage speed etc.. A possible approach to take into account such constraints is to develop a distributed control scheme to provide the environment with the ability to perceive information from a large area using distributed sensors to maintain the robot navigation performance up to its maximum driving speed limit. Traditional approaches exploiting centralised intelligence implemented in a robot to support autonomous navigation cannot implement such an optimal control technique due to the limited view of the robot on-board sensors.

In addition, due to sensor sensing errors, position calculation errors and robot mechanic movement errors, the robot's position may diverge from the R-snake path, causing deviation of the robot navigation path from the desired path.

To the authors' knowledge, the application of snake-based algorithm for robot trajectory planning and tracking utilising distributed environment intelligence and subjecting to robot non-holonomic constraints with input saturations has yet to be explored.

Based on the R-snake, an Accompanied snake (A-snake) is introduced in this chapter to cope with all these constraints as well as to plan the shortest travelling time trajectory. To compensate for position deviation, robot trajectory tracking is performed using the A-snake which is designed to start from the robot's current position and converges to the R-snake path for local trajectory tracking.

The basic idea of the A-snake is to decouple the time domain trajectory planning from the spatial path planning, alleviating the complexity of the trajectory tracking. The A-snake algorithm is divided into two phases: 1) A-snake spatial position planning to extend its trajectory from the robot's current position to approach the R-snake and at the same time to satisfy the constraints of saturated torques; and 2) A-snake tracking control by generating an optimal speed profile along the trajectory with minimum travelling time. The A-snake spatial path planning only deals with path geometry features without considering the time factor. The primary goal is to generate an associated path that converges to the reference snake path and at the same time complies with the dynamic and non-holonomic constraints and control saturation. Predictive control [158] is selected to optimise the forthcoming tracking taking into account the dynamic constraints of maximum friction, limited driving force and limited steering torque. The Model Predictive Control (MPC) [159-161] also alleviates the negative impacts due to the slow feedback from vision sensors. A rolling window optimisation [162] is carried out to generate the optimum velocity profile of the mobile robot up to a certain distance from its current position. As a result, the mobile robot can be driven towards the

global R-snake by following the A-snake whereas optimum control can be obtained within the rolling window.

6.2 A-snake Spatial Position Planning

Feedback control of distributed vision sensors is adopted for the dynamic generation of an A-snake spatial position. The A-snake extends from the robot's current position and orientation to approach the R-snake path by taking into account the limited steering torque under the non-holonomic constraints. This will correct the deviation of the robot's position from the reference path. A key issue in this phase is whether the A-snake can converge to the R-snake path without violating the steering torque and driving force limitations.

6.2.1 Desired Direction to Approach the R-snake

To describe how the A-snake converges to the R-snake when it extends iteratively, define a coordinate system as shown in Figure 5.2.

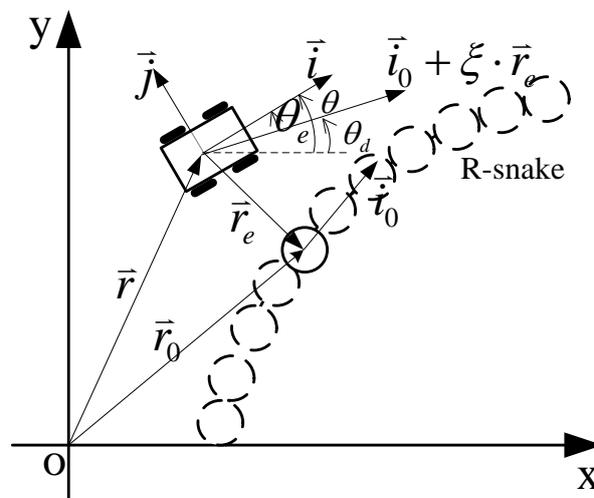


Figure 6.1. Coordinates definitions

The dotted circles represent the R-snake control points and the square box indicates the robot. θ_d is the desired direction which the robot should follow at each step; θ is the robot's current direction; \vec{i} and \vec{j} are the unit tangential vector and normal vector of the robot's local coordinates along its direction of movement; \vec{i}_0 is the tangential direction on the nearest reference control point; \vec{r} is the robot's location vector, \vec{r}_0 is the vector of the nearest reference control point of the snake; θ_e is the direction error between the desired direction and the robot's current direction; ξ is a positive coefficient. The A-snake will always start from the robot current position;

To determine a proper desired direction θ_d which can lead the A-snake to converge to the R-snake, assume the position error vector between the robot current position and the R-snake to be,

$$\vec{r}_e = \vec{r}_0 - \vec{r} \quad (6.1)$$

Construct a Lyapunov function [163] V such that

$$V = \vec{r}_e^T \cdot \vec{r}_e \geq 0 \quad (6.2)$$

where T indicates the transpose of a vector. The derivative of the above function with respect to the arc length s can be obtained as,

$$\frac{\partial V}{\partial s} = 2 \cdot \vec{r}_e^T \cdot \frac{\partial \vec{r}_e}{\partial s} \quad (6.3)$$

Substitute (6.1) into (6.3), one can get,

$$\begin{aligned} \frac{\partial V}{\partial s} &= 2 \cdot \vec{r}_e^T \cdot \left(\frac{\partial \vec{r}_0}{\partial s} - \frac{\partial \vec{r}}{\partial s} \right) \\ &= 2 \cdot \vec{r}_e^T \cdot (\vec{i}_0 - \vec{i}) \end{aligned} \quad (6.4)$$

If the robot's direction can be controlled according to,

$$\bar{i} = \bar{i}_0 + \xi_1 \cdot \bar{r}_e \quad (6.5)$$

then,

$$\frac{\partial V}{\partial s} = -2 \cdot \bar{r}_e^T \cdot \xi_1 \cdot \bar{r}_e \leq 0 \quad (6.6)$$

According to Lyapunov theorem and from (6.2) and (6.6), one can draw the conclusion that if the robot's heading direction is towards (6.5), the position error between the A-snake and the R-snake will be reduced to zero exponentially.

Assuming that the robot can only move forward, define the movement along direction \bar{i}_0 as $\Delta r_{\bar{i}}$ and direction \bar{j}_0 as $\Delta r_{\bar{j}}$ below,

$$\begin{cases} \Delta r_{\bar{i}} = 1 \\ \Delta r_{\bar{j}} = \xi_1 \cdot r_e \end{cases} \quad (6.7)$$

where ξ_1 is the positive gain defined in (6.5) and,

$$r_e = \text{sign}(\bar{r}_e) \cdot \|\bar{r}_0 - \bar{r}\|, \quad (6.8)$$

$$\text{sign}(\bar{r}_e) = \begin{cases} -1, & \text{robot is on the left side of the R-snake} \\ 1, & \text{robot is on the right side of the R-snake} \end{cases}$$

Then the A-snake should be expanded along the direction in (6.5) in order to converge to the R-snake such that,

$$\theta_d = \theta_0 + \arctan\left(\frac{\Delta r_{\bar{j}}}{\Delta r_{\bar{i}}}\right) \quad (6.9)$$

6.2.2 Dynamic Constraints and Maximum Allowable Uniform Velocity

Equation (6.9) gives the desired direction for the A-snake to follow so as to converge to the R-snake. However, the robot's moving direction cannot always reach the desired value due to the non-holonomic

constraints and dynamic constraints. As shown in Figure 5.2, the motion of a wheeled robot can be modelled as,

$$\begin{cases} \dot{\vec{r}} = v \cdot \vec{i} \\ \omega = v \cdot k \end{cases} \quad (6.10)$$

where $\vec{r} = (x, y)^T$ is the robot position vector, \vec{i} is the unit vector of its tangential direction, v is the robot velocity, ω is the robot angular velocity, and k is the trajectory curvature.

After differentiating the above equation, the acceleration and the angular accelerations of the robot are given as below,

$$\begin{cases} \ddot{\vec{r}} = \dot{v} \cdot \vec{i} + v\omega\vec{j} \\ \dot{\omega} = \dot{v}k + v\dot{k} \end{cases} \quad (6.11)$$

where \vec{j} is the unit normal vector of the robot.

According to Newton's law, the dynamic equation of the robot can be obtained as,

$$\begin{cases} F^d = M\dot{v} \\ f^{friction} = Mv\omega = Mv^2k \\ \tau = J(\dot{v}k + v\dot{k}) = J(\dot{v}k + v^2 \frac{\partial k}{\partial s}) \end{cases} \quad (6.12)$$

where $f^{friction}$ is the lateral friction on the wheels, s is the arc length along the snake, M and J are the mass and the inertia of the robot.

From equation (6.12), one can see that if the steering torque τ is given, there is a tradeoff between the driving velocity v and the derivative of curvature $\frac{\partial k}{\partial s}$. If $\frac{\partial k}{\partial s}$ is very large, the driving speed has to be very low

under a limited torque. This is equivalent to the situation when the robot makes a very sharp turn only when it is driving very slowly. On the other

hand, the driving speed can be very fast if $\frac{\partial k}{\partial s}$ is very small. Otherwise, wheel slippage will happen. Therefore, considering the saturated torque τ_{\max} of a robot, the limitation to $\frac{\partial k}{\partial s}$ is imposed as the following,

$$\left| J \cdot \frac{\partial k}{\partial s} \right| < \gamma \quad (6.13)$$

where J is the rotational inertia, γ is a positive constant.

If a trajectory is generated to satisfy (6.13), there exists a torque in $[-\tau_{\max}, \tau_{\max}]$ to follow that trajectory.

From (6.12), the robot is also subject to dynamic constraints of maximum driving force and frictions, $|F^d| < F_{\max}^d$ and $|f^{friction}| < (\mu N)_{\max}$. As a result, the maximum uniform velocity of the robot is limited by the geometry features of its trajectory. By substituting $\dot{v} = 0$ into (6.12), the maximum uniform velocity subject to the saturated torque and friction can be obtained as,

$$\begin{aligned} v_{\tau \max}^2 &= \frac{\tau_{\max}}{J \left| \frac{\partial k}{\partial s} \right|} \\ v_{f \max}^2 &= \frac{(\mu N)_{\max}}{M |k|} = \frac{g \mu_{\max}}{|k|} \end{aligned} \quad (6.14)$$

The maximum velocity is thus obtained as,

$$v_{\max} = \min(v_{\tau \max}, v_{f \max}) \quad (6.15)$$

6.2.3 Convergence of A-snake Approaching R-snake

To design a path converging to the R-snake without violating the limitation imposed by equation (6.13), a feedback control algorithm as

shown in Figure 6.2 is designed to generate the heading direction θ of the path to follow the desired θ_d with a limiting parameter u , such that $|u| < b$, where b is a positive constant, u is defined as,

$$u = \frac{\partial k}{\partial s} \quad (6.16)$$

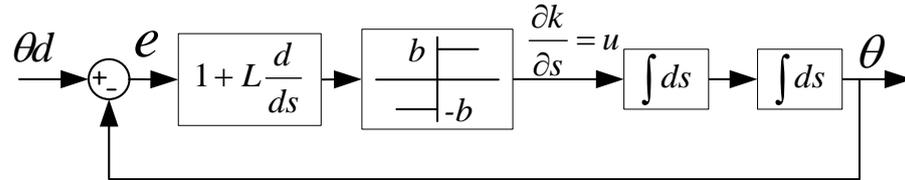


Figure 6.2. Control flow chart

Proposition 1: Let L be a constant coefficient factor, $L \in \mathbb{R}^+$, e be the orientation error between θ_d and the robot direction θ , b be the maximum curvature derivative a robot can follow. If the given desired direction θ_d is constant, then e and $\frac{\partial k}{\partial s}$ converge to 0 with oscillation by the control scheme as defined in Figure 6.2.

Proof: The orientation error between θ_d and the robot direction θ is,

$$e = \theta_d - \theta \quad (6.17)$$

where $\theta_d = \xi_2$, ξ_2 is a constant, then

$$\frac{\partial^2 e}{\partial s^2} = -\frac{\partial^2 \theta}{\partial s^2} = -\frac{\partial k_d}{\partial s} = -u \quad (6.18)$$

where k_d is the curvature on A-snake. From Figure 6.2, let

$$Q = L \cdot \frac{\partial e}{\partial s} + e \quad (6.19)$$

then the control torque u is given by

$$\frac{\partial^2 \theta}{\partial s^2} = u = b \cdot \text{sgn}(Q) \quad (6.20)$$

where,

$$\text{sign}(Q) = \begin{cases} -1, & Q < 0 \\ 1, & Q \geq 0 \end{cases} \quad (6.21)$$

Let,

$$\frac{\partial e}{\partial s} = \ell \quad (6.22)$$

One gets

$$\frac{\partial \ell}{\partial s} = -u \quad (6.23)$$

Then

$$\frac{d\ell}{de} = -\frac{u}{\ell} \quad (6.24)$$

Rewrite the above equation gives

$$\ell d\ell = -u de \quad (6.25)$$

Integrate both sides of equation (6.25) to obtain,

$$\frac{1}{2} \ell^2 = -u(e + \xi_3) \quad (6.26)$$

where ξ_3 is a constant. The phase track of ℓ and e shown in Figure 6.3 are parabola clusters intersected by the switch line Q , and the direction of the trajectory is clockwise. If $L > 0$, then e and $\frac{\partial e}{\partial s}$ converge to 0.

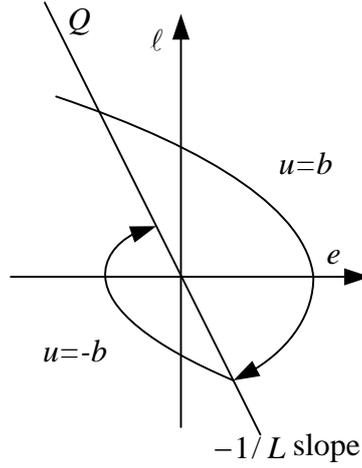


Figure 6.3. Phase track of e and l

If the values of L and b are limited within a certain range, the convergence can be in the sliding mode without oscillation.

Proposition 2: Let L be a constant coefficient factor, $L \in \mathbb{R}^+$, e be the orientation error between θ_d and the robot direction θ , b be the maximum curvature derivative the robot can follow. If a change of direction

θ_d satisfies the condition of $\left| \frac{\partial \theta_d}{\partial s} - \frac{\partial \theta}{\partial s} \right| < L \cdot b$, sliding mode happens, that

is e and $\frac{\partial e}{\partial s}$ converge to 0 along a sliding plane by the control scheme as

shown in Figure 6.2.

Proof: Let

$$Q = L \cdot \frac{\partial e}{\partial s} + e \quad (6.27)$$

Then the sliding mode condition [164] is defined by the following,

$$Q \cdot \frac{\partial Q}{\partial s} < 0 \quad (6.28)$$

Substitute (6.27) into (6.28), then

$$Q \cdot \frac{\partial Q}{\partial s} = Q \cdot \left(L \cdot \frac{\partial^2 e}{\partial s^2} + \frac{\partial e}{\partial s} \right) \quad (6.29)$$

Since,

$$\frac{\partial^2 e}{\partial s^2} = -b \cdot \text{sign}(Q) \quad (6.30)$$

One gets,

$$\begin{aligned} Q \cdot \frac{\partial Q}{\partial s} &= Q \cdot \left(-L \cdot b \cdot \text{sign}(Q) + \frac{\partial e}{\partial s} \right) \\ &= -L \cdot b |Q| + Q \cdot \frac{\partial e}{\partial s} \end{aligned} \quad (6.31)$$

Thus, if

$$\left| \frac{\partial e}{\partial s} \right| = \left| \frac{\partial \theta_d}{\partial s} - \frac{\partial \theta}{\partial s} \right| < L \cdot b \quad (6.32)$$

then (6.28) is satisfied. The sliding mode happens with a sliding plane $Q = 0$, which implies that e and $\frac{\partial e}{\partial s}$ converge to 0. Figure 6.4 shows the sliding mode tracks.

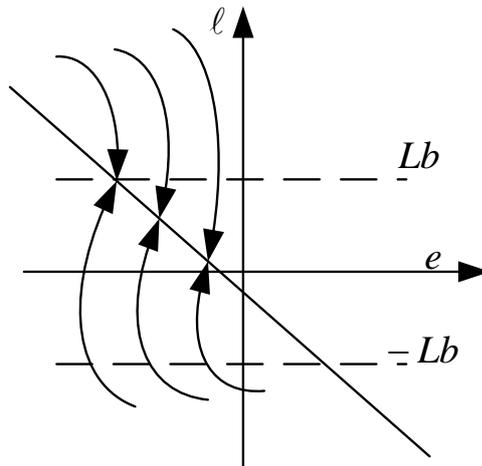


Figure 6.4. Sliding mode happens

The control points of A-snake can grow iteratively with the following equation (6.33) ,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \int \begin{pmatrix} \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial s} \end{pmatrix} ds + \begin{pmatrix} x \\ y \end{pmatrix} \quad (6.33)$$

where, $\begin{pmatrix} x \\ y \end{pmatrix}$ and $\begin{pmatrix} x' \\ y' \end{pmatrix}$ are the current and newly calculated control point positions respectively, and ,

$$\begin{pmatrix} \frac{\partial x}{\partial s} \\ \frac{\partial y}{\partial s} \end{pmatrix} = \begin{pmatrix} \cos\left(\int \frac{\partial \theta}{\partial s} ds\right) \\ \sin\left(\int \frac{\partial \theta}{\partial s} ds\right) \end{pmatrix} \quad (6.34)$$

6.2.4 A-snake Growth in Discrete Time Domain

To simplify the calculation, the robot rotational inertia is assumed to be $J=1$. The control points of A-snake are indexed by integer j , $0 \leq j < l$ where l is an integer constant. By applying integration on

$\frac{\partial k}{\partial s} = u$, one can get the orientation of the j^{th} control point of A-snake in

discrete form as,

$$\begin{aligned} \theta_j &= \theta_{j-1} + \int_{s_{i-1}}^{s_i} \left(k_{j-1} + \int_{s_{i-1}}^s u_{j-1} ds \right) \cdot ds \\ &= \theta_{j-1} + k_{j-1} \cdot (s_i - s_{i-1}) + \frac{1}{2} (s_i - s_{i-1})^2 \cdot u_{j-1} \end{aligned} \quad (6.35)$$

$$\begin{aligned} k_j &= k_{j-1} + \int_{s_{i-1}}^{s_i} u_{j-1} \cdot ds \\ &= k_{j-1} + u_{j-1} \cdot (s_i - s_{i-1}) \end{aligned} \quad (6.36)$$

where k_j is the curvature of the j^{th} control point on the A-snake; $i: 0 \leq i < n$ is the index of control points of the R-snake. $i = j + \hbar$, and \hbar is an integer denoted to be the offset between the A-snake and the R-snake corresponding control points, s_i is the arc length from the start point to another control point p_i on the R-snake.

Taking the assumption that $\theta_d = \xi_4$ in one step, ξ_4 is a constant.

$$\dot{e}_j = \dot{\theta}_{d_j} - \dot{\theta}_j \approx 0 - k_j = -k_j \quad (6.37)$$

Define $\dot{e}_j = \left(\frac{\partial e}{\partial s}\right)_j$ and $\dot{\theta}_j = \left(\frac{\partial \theta}{\partial s}\right)_j$, then $\left(\frac{\partial k}{\partial s}\right)_j = u_j$ can be

calculated by,

$$u_j = \begin{cases} -b, L \cdot \dot{e}_j + e_j < 0 \\ b, L \cdot \dot{e}_j + e_j \geq 0 \end{cases} \quad (6.38)$$

Then,

$$\Delta\theta_j = k_{j-1} \cdot \Delta s_i + \frac{1}{2} \Delta s_i^2 u_{j-1} \quad (6.39)$$

$$\theta_j = \theta_{j-1} + \Delta\theta_j \quad (6.40)$$

where,

$$\Delta s_i = s_i - s_{i-1} \quad (6.41)$$

By iteration,

$$\begin{pmatrix} x_j \\ y_j \end{pmatrix} = \begin{bmatrix} \cos(\theta_{j-1}) & -\sin(\theta_{j-1}) \\ \sin(\theta_{j-1}) & \cos(\theta_{j-1}) \end{bmatrix} \cdot \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} + \begin{pmatrix} x_{j-1} \\ y_{j-1} \end{pmatrix} \quad (6.42)$$

where,

$$\begin{cases} \Delta x = \Delta s_i \cdot \cos(\Delta\theta_j) \\ \Delta y = \Delta s_i \cdot \sin(\Delta\theta_j) \end{cases} \quad (6.43)$$

Equation (6.43) is the desired trajectory movement in one step from the preceding control point. Δs_i is the arc length from control point p_i to control point p_{i-1} in the R-snake, Δk_j is the curvature of control point p_i on the A-snake; θ_j is the desired direction of control point p_j of the A-snake.

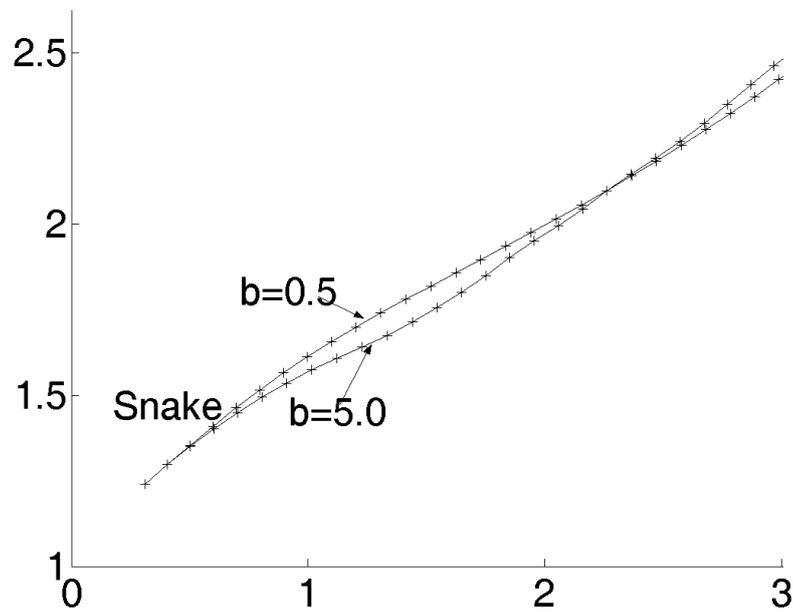


Figure 6.5 A-snake

Figure 6.5 shows an example of the A-snake. The robot's initial coordinates are (0.31, 1.24) and its orientation is 0.535 rad. For the robot to follow an R-snake, the robot has to "jump" to the R-snake path and then follow. This is unrealistic. Behaving as a run-up, an A-snake is introduced to guide the robot to "soft land" the R-snake and approach the target as well. When the parameter b , physically known as the maximum rotational torque, is increased from 0.5 to 5, the degree of convergence is also increased as a result.

6.3 A-snake Time Series Trajectory Generation and Motion Control

Although the A-snake spatial position provides a reference path for a robot to travel, due to the limited field of view of the onboard sensors, the tracking speed has to compromise with safety. In fact, the far sight provided by a large scale distributed sensor network is developing the foundation and potential for the optimal control of vehicles. A vehicle is envisaged to be able to respond to changes on its way ahead and be driven with optimal time or energy in a dynamic environment. Distributed control to achieve high performance tracking up to the driving limit becomes a promising technique.

6.3.1 Problem Statements

Model predictive control is widely used in industry processes, where system behaviours in future are predicted and an optimal current control action is calculated accordingly [159, 165, 166]. One advantage of model predictive control is its ability to cope with constraints [160]. It has also been used for mobile robot trajectory tracking with uncertain dynamics [161]. In terms of a dynamic environment, a rolling window optimisation was used for real-time trajectory planning of a mobile robot. In this section, a predictive control scheme combining trajectory planning and dynamic control is developed to achieve optimal time tracking, taking into account the future path that needs to be tracked, subject to non-holonomic constraints, robot kinematic and dynamic constraints, the maximum velocity without slippage, driving force and steering torque saturation.

Define a rolling window with length l along a snake, which could be distributed in several wireless sensors and evolved by individual vision sensors asynchronously. The rolling window is shown in Figure 6.6.

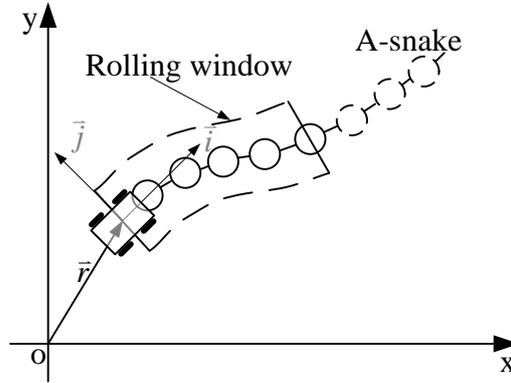


Figure 6.6. Robot, A-snake and rolling window

For every sampling period, the optimal driving force and steering torque are calculated taking into account the dynamic constraints and the geographic features of the snake in the window. The l -window will roll forward one step for the next step predictive control. Working in this way repeatedly, a vehicle can react to possible risks on its way in advance and use its driving capacity sufficiently.

In a rolling window l , the optimal control for a robot can be formulated as:

$$\min_{F^d, \tau} (\Gamma) = \min_{F^d, \tau} \left(\int_0^l 1/v(s) ds \Big|_{s \in A} \right) \quad (6.44)$$

with boundary conditions: $v(0) = v_{r0}, v(l) = 0$; subject to

$$\begin{cases} |f^{friction}| < (\mu N)_{\max}, & \text{for non - slippage} \\ |F^d| < F_{\max}^d, & \text{for limited driving force} \\ |\tau| < \tau_{\max}, & \text{for limited steering torque} \end{cases} \quad (6.45)$$

where A is the A-snake; $v(s)$ is the velocity profile of the robot; $v_{r,0}$ is the sampled velocity of the robot at any instant; f is the friction of tires with coefficient μ and the normal force N ; F^d and τ are the driving force and steering torque of the robot with upper bounds of F_{\max}^d and τ_{\max} respectively. The objective function in equation (6.44) for predictive control is to minimise the robot's travelling time Γ along the snake from its current location until the end of the l -window. However the robot should be able to stop at the end such that $v(l) = 0$ in order to respond to the worst possible circumstances which are not covered by the current rolling window.

In order to optimise (6.44), the area under the velocity profile $v(s)$ will be maximised so that the area of its reciprocal can be minimised following two calculation steps: first, the maximum uniform velocity is obtained, which is the maximum allowable velocity for the robot to travel along the A-snake without any effort for acceleration or deceleration; then, the optimal force and torque will be obtained to approach this maximum allowable velocity as much as possible, considering the dynamic constraints and satisfying the boundary conditions.

6.3.2 Optimal Control in the Rolling Window

In order to satisfy the two boundary conditions in (6.44), we need to design optimal control inputs such that the robot can safely travel a path and is able to stop at the end of the rolling window, considering the dynamic constraints. A sharp bend on the way requires the robot to

decelerate in advance due to the limited driving force and steering torque. The optimisation of (6.44) can be solved by maximising the area of $v(s)$, which can be achieved by squeezing a velocity profile from the two ends towards the middle using the maximum force and torque.

From (6.12), the maximum allowable positive acceleration can be obtained as

$$\begin{aligned} a_{F \max +} &= F_{\max}^d / M \\ a_{\tau \max +} &= \frac{\tau_{\max}}{J |k|} - \frac{v^2}{k} \frac{\partial k}{\partial s} \\ a_{\max +} &= \min(a_{F \max +}, a_{\tau \max +}) \end{aligned} \quad (6.46)$$

If $v \leq v_{\max}$, then $a_{\max +} \geq 0$, this implies that a positive acceleration exists. Therefore, the acceleration process has to be bounded by v_{\max} .

Similarly, the maximum allowed negative deceleration can be obtained as

$$\begin{aligned} a_{F \max -} &= -F_{\max}^d / M \\ a_{\tau \max -} &= -\frac{\tau_{\max}}{J |k|} - \frac{v^2}{k} \frac{\partial k}{\partial s} \\ a_{\max -} &= \max(a_{F \max -}, a_{\tau \max -}) \end{aligned} \quad (6.47)$$

A negative deceleration exists if the velocity is bounded by v_{\max} .

The squeezing process is approximated by segments of uniform acceleration/deceleration movements from two boundary velocities with an incremental step δ :

For the acceleration at s_+ , forward planning is carried out

$$v_+^2(s_+) = v_+^2(s_+ - \delta) + 2\alpha_{\max +} \delta \quad (6.48)$$

For the deceleration at s_- , backward planning is carried out

$$v_-^2(s_-) = v_-^2(s_- + \delta) + 2\alpha_{\max} \delta \quad (6.49)$$

During the squeezing process, we need to ensure that the values of $v_+^2(s_+)$ and $v_-^2(s_-)$ in equations (6.48) and (6.49) do not exceed v_{\max} for the segment in between. If this happens at any point $s_{\#}$, the velocity profile can be squeezed for the segment from $v_+(s_+)$ to $v_-(s_{\#}) = v_{\max}(s_{\#})$. The process continues until the acceleration segment and the deceleration segment encounter each other, as shown in Figure 6.4 as an example. The velocity profile will be obtained by repeating this squeezing process for the remaining segments. Working in this way, the area of $v(s)$ is maximised and therefore the travelling time is minimised. The generated velocity profile tells the robot when to accelerate or decelerate in advance in order to safely track the dynamic snake in a predictive l -window. The algorithm is summarised as below and is shown in Figure 6.7:

- 1) According to the snake, obtain the maximum allowable velocities v_{\max} from (6.15) in rolling window l ;
- 2) Initialise the squeezing process with boundary conditions: initial state $s_+ = 0, v_+(0) = v_{r0}$ and terminal state $s_- = l, v_-(l) = 0$;
- 3) Plan/increase v_+ and v_- in parallel. If $v_+ \leq v_-$, increase v_+ by (6.48) and $s_+ = s_+ + \delta$. If $v_+ > v_-$, increase v_- by (6.49) and $s_- = s_- - \delta$;
- 4) If $s_+ = s_-$ and there is any unplanned segment, set s_+ and s_- to be the unplanned segment and go to 3); if the maximum allowable velocity is found at $s_{\#}$ between $s_+ \sim s_-$, create two new segments, $s_+ \sim s_{\#}$ and $s_{\#} \sim s_-$ then go to 3), otherwise go to 5);

- 5) Calculate the driving force and steering torque for $s=0$ as control signals for time t :

$$F^d(t) = M a_{\max}(0)$$

$$\tau(t) = J a_{\max}(0) k(0) + J v^2(0) \left. \frac{\partial k}{\partial s} \right|_{s=0} \quad (6.50)$$

where $a_{\max} = a_{\max+}$ or $a_{\max-}$ obtained in step 3;

- 6) Send $F^d(t)$ and $\tau(t)$ to the robot for control and shift the rolling window one step forward;
- 7) For every servo period Γ , $F(t+n\Gamma)$ and $\tau(t+n\Gamma)$ will be continuously generated from the obtained velocity profile to control the vehicle until a new profile is received from a vision sensor;
- 8) For every image sampling period, the mosaic eyes update the A-snake and go to 1).

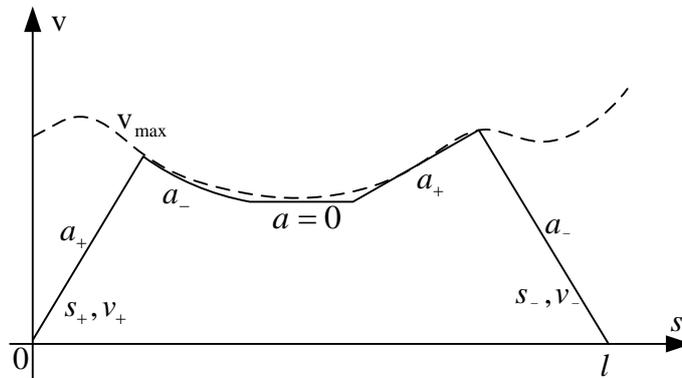


Figure 6.7. Rolling window optimisation for trajectory generation (assume $v_{r0} = 0$)

6.4 Simulation and Experiment Results

6.4.1 A-snake Trajectory and Velocity

As seen in (6.47), when given a certain rotational torque, a bigger curvature derivative on the path means a lower allowed velocity; when a certain friction factor and mass of robot are given, a bigger curvature indicates lower speed. Figure 6.8, Figure 6.9 and Figure 6.10 show the simulation result generated by Matlab. The solid line in Figure 6.9 shows the maximum velocity for the A-snake path in Figure 6.8. Because of the position error between the robot and the reference snake, the A-snake curvature derivative is saturated at the beginning trying to reduce the difference. The first part of the velocity (segment A) is dominated by the curvature derivative (around 0.9 m/s), although the highest speed that the robot can reach is 1.5 m/s .

When obstacles are observed (shown in Figure 6.8), the obstacle forces will force the snake to deform. This information then spreads out throughout the snake body by internal forces. When the rolling window (size=50) can "see" this situation from the reference snake, the snake starts to bend. As seen in part B of Figure 6.9, the curvature becomes big enough to dominate the velocity limitation and reduce the maximum allowed velocity in advance to make sure that the robot will not collide with the obstacles due to the limitation of deceleration. After the obstacles area reaching Part C in Figure 6.9, the A-snake almost converges to the reference snake, so the curvature derivative and the curvature itself are no

longer subject to limitation and the maximum allowed velocity goes high up to 1.5 m/s.

From Figure 6.9, one can see that the planned velocity has similar profile as the maximum allowed velocity. The difference comes from the start and the end of the planning window. The start speed is the same as robot's current speed and the end speed is zero to allow the robot to stop when it is out of the rolling window range. From the planned acceleration, one can see that the desired acceleration is always within the robot acceleration limit (0.2 m/s^2).

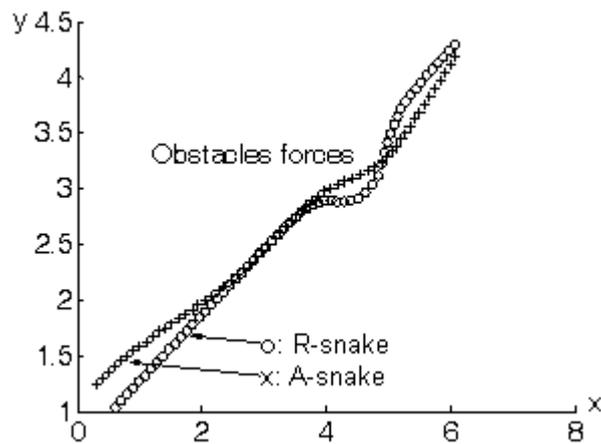


Figure 6.8. Deformable snake with A-snake

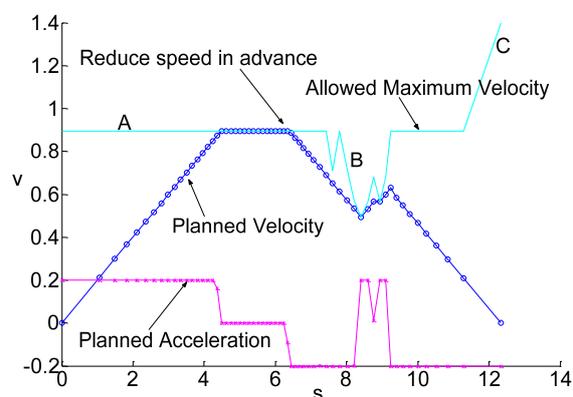


Figure 6.9. Allowed velocity, planned velocity and acceleration

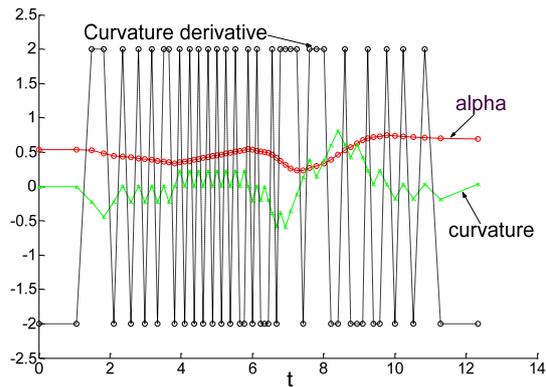


Figure 6.10. Planned curvature derivative, curvature and orientation

Figure 6.10 shows the planned curvature derivative, the curvature and the robot desired orientation. The curvature derivative is within the robot's constraints but it switches between -2 and 2 because the controller in Figure 6.2 is a bang-bang controller, which is a controller that switches abruptly between two states.

6.4.2 Path Planning and Predictive Control Experiment

Trajectory tracking of a car like robot using the mosaic eyes is experimented. Four vision sensors are mounted on the ceiling to form the mosaic eyes. They are placed in such a way that each eye has neighbouring eyes at both side, one on the left and another on the right. An independent remote console is setup to monitor the mosaic eyes working status on demand and to maintain the mosaic eyes when needed. The MCU of the car like robot is based on the Motorola MC9S12DT128B CPU to execute the received commands from mosaic eyes. The mosaic eyes, the remote console and the robot are all IEEE 802.15.4 communication enabled. The car like robot is marked by red and blue blobs on top of it which is used by the mosaic eyes to locate the robot's

position and to distinguish the robot from the obstacles, as shown in Figure 3.4.

The predictive control has a rolling window with $l = 20$ (control points). The maximum travelling speed is 0.8 m/s , the maximum driving force is $F_{\max}^d = 4.4(N)$ with a $0.56(kg)$ robot mass and the friction factor $\mu_{\max} = 0.6$ and $\tau_{\max} = 2.0(N \cdot m)$.



Figure 6.11. Robot moving from eye-30 to obstacles free eye-60

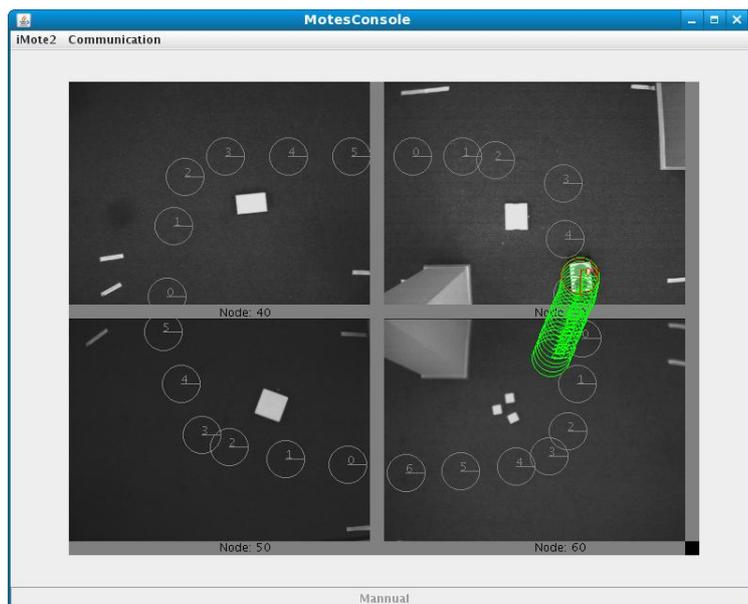


Figure 6.12. Obstacles appear in eye-60

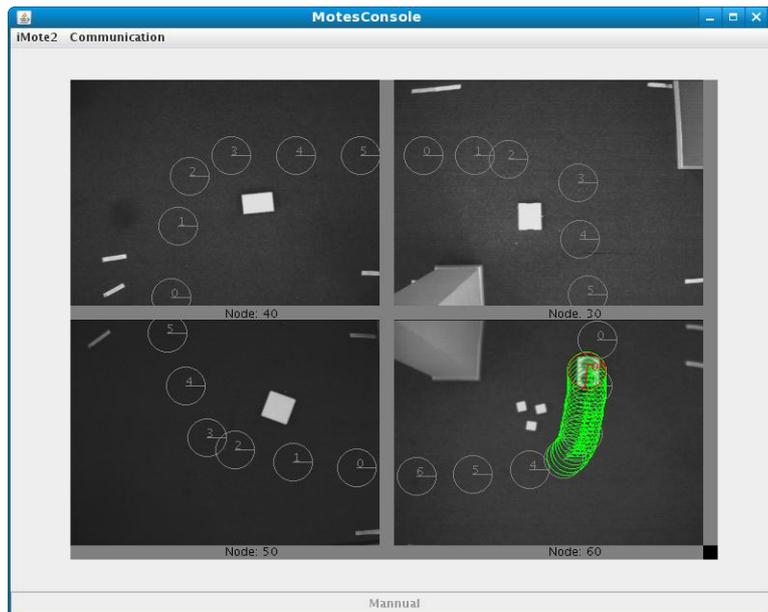


Figure 6.13. Robot passing obstacle area in eye-60

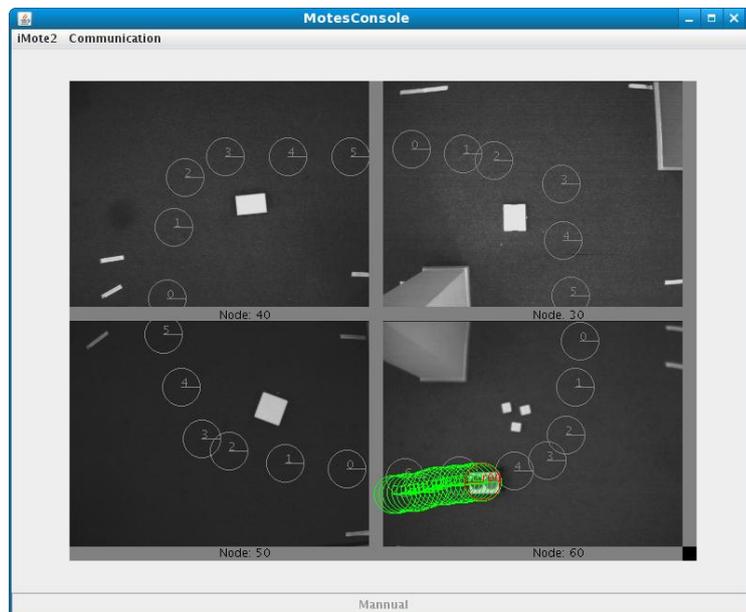


Figure 6.14. Robot passed obstacles area in eye-60 and move to eye-50

Figure 6.11, Figure 6.12, Figure 6.13 and Figure 6.14 show the real time experiments of robot control by the mosaic eyes. Each figure displays four views from each of the four eyes. Let eye-30, eye-40, eye-50 and eye-60 be the names of the mosaic eyes starting from top-right one and counting anti-clockwise. In Figure 6.11, the robot is controlled by eye-30 heading to the control area of eye-60. The sparse white circles with numbers in the centre represents the desired path that robot should follow.

The white rectangle blobs represent dynamic obstacles. As one can see in Figure 6.11, the dynamic obstacles are within the views of eyes-30, eye-40 and eye-50 but out of sight of eye-60. In Figure 6.12, an obstacle appears within the sight of eye-60. At this point, the robot is under the control of eye-30 and eye-30 does not know the existence of the new obstacle. With the information sent from eye-60 notifying eye-30 of the obstacle, the predictive path is updated to avoid the obstacle. In Figure 6.13, the robot control is handed over from eye-30 to eye-60. The figure shows that with the predictive path updated by eye-30 and with the control of eye-60, the robot has successfully avoided the obstacle and continued to move along the updated predictive path.

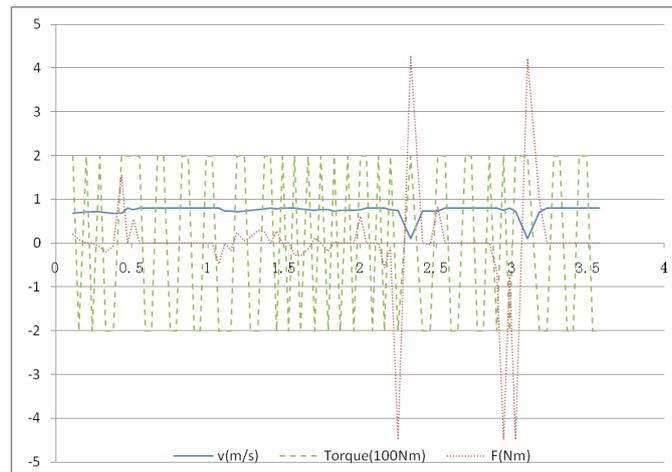


Figure 6.15. Robot control and velocity (driving force F^d : dot; steering torque τ :dashed; velocity v : solid)

The corresponding control and driving velocity are shown in Figure 6.15. It shows that the driving force and the steering torque are both kept within the allowable ranges. The sign changes in the steering torque indicate the feedback regulation of the predictive control in order to track the snake, although predictive control is designed in an open-loop

manner for each single prediction in the l -window. The robot speed is 0.76 m/s when data collection started as shown in Figure 6.11. There are no dynamic obstacles in view of eye-60, so the path is extracted to a straight line to minimise the distance. When the obstacle is detected by eye-60, a path change occurred in Figure 6.12 so that the robot can still maintain a high speed to go across the overlapping area of eye-30 and eye-60. The robot decelerates when it approaches the obstacle in order to avoid slippage (Figure 6.13) and accelerates again after passing the obstacle (Figure 6.14).

6.5 Conclusion

This chapter proposes a distributed predictive control scheme utilising a bio-inspired intelligent environment for robot navigation via communication links between vision sensors. The scheme takes into account robot dynamic constraints and the dynamic changes of the environment. A novel A-snake approach is proposed for robot motion control under the dynamic constraints. Details on how to generate the A-snake, the proof of A-snake convergence and the rolling window optimisation for time-optimal control are also presented. Both simulations and experiments have been carried out to verify that the proposed method offers an effective distributed solution for integrating robot navigation, path planning, trajectory generation and motion control into a unified snake based control mechanism.

Chapter 7 : IMPLEMENTATION

7.1 Introduction

“The merging of identification technologies, wireless communication and sensors allows the birth of a new vision: pervasive and unobtrusive intelligence embedded within the objects of the environment around us.”

Interview to Prof. Jianhua Ma by Wise Media, March 2005

Wireless sensor networks are providing the IT infrastructure to support distributed motion control, which could lead to a new paradigm for autonomous navigation. With the advent of wireless sensor networks, it is envisaged that the complexity of centralised intelligence, which hinders applications of autonomous robots in our daily life, can be greatly simplified and high mobility can be achieved with a very limited onboard computational power. This chapter describes the practical implementation of an intelligent environment test bed supported by vision sensors mounting on the building. The test bed as a prototype can provide a better understanding of the ubiquitous intelligence, tackling the desired communications mechanisms between wireless sensors, revealing new application opportunities as well as testing and evaluating the proposed distributed navigation algorithms derived in this PhD work.

The scope of this chapter covers the hardware platform selection and integration, software designs and implementation and communication protocols proposing and assessment.

The system architecture and modules are deployed based on the blueprint from Chapter 3. The path and motion instructions are planned by the distributed vision sensors collaborating with each other through the bio-inspired snake algorithms proposed in Chapter 4, Chapter 5 and Chapter 6. The results obtained by the experiments in this test bed and from the simulation software output provide strong evidence on the validity and credibility of the theories in return.

This chapter is organised as follows: Section 7.2 provides details of the hardware components in the intelligent environment, including the remote console, the vision sensors, the mobile robot as well as the IEEE 802.15.4 devices to enable the wireless capability of the remote console PC and the mobile robot. Communication protocols between different system entities are thoroughly discussed in Section 7.3 with the signal flows and packet formats. Software structures, modules and applications are given in Section 7.4 . Section 7.5 discusses some issues about the system performance and implementation. Finally, a conclusion is given in Section 7.6 .

7.2 Hardware Platform

7.2.1 Hardware Architecture Overview

The intelligent environment test bed is designed based on the proposed system architecture in Chapter 3. An overview of the platform is shown in Figure 7.1.



Figure 7.1 Intelligent environment

In the environment, Intel Mote 2 (iMote2) wireless vision sensors (also referred to as mosaic eyes) are mounted in the building ceiling. They are placed in such a way that each sensor covers an area on the floor and has overlapping area with the others. The robot is guided by the vision sensors and moves on the floor accordingly. A remote console is set up far from the field but within the communication range.

The connections between these components are carried out via IEEE 802.15.4 links, i.e. communications from one vision sensor to another, from the vision sensor to the robot, and between the vision sensor and the remote console.

7.2.2 Vision Sensor Hardware Integration

The wireless vision sensors are developed based on the iMote2 microcontroller board [167]. One vision sensor, shown in Figure 7.2, consists of one Off-The-Shelf (OTS) iMote2 main board, one OTS power board, one OTS OV7620 vision sensor module [168] and one self-developed sensor board to integrate the OV7620 with the iMote2 main board. The components are described in detail in the following subsections.

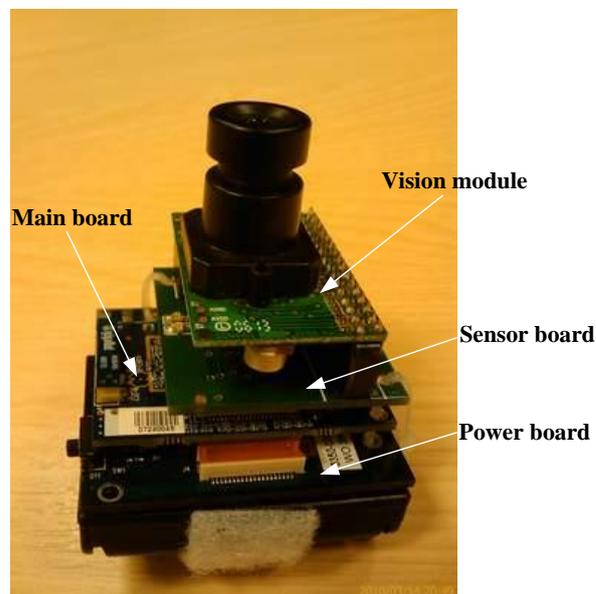


Figure 7.2 One wireless mosaic eye

7.2.2.1 iMote2 Main Board Introduction

The iMote2 is an advanced sensor network platform with 13-416MHz PXA271 XScale Processor, 256kB SRAM, 32MB Flash and 32MB SDRAM on-board. The top view and bottom view of the iMote2 main board are shown in Figure 7.3.

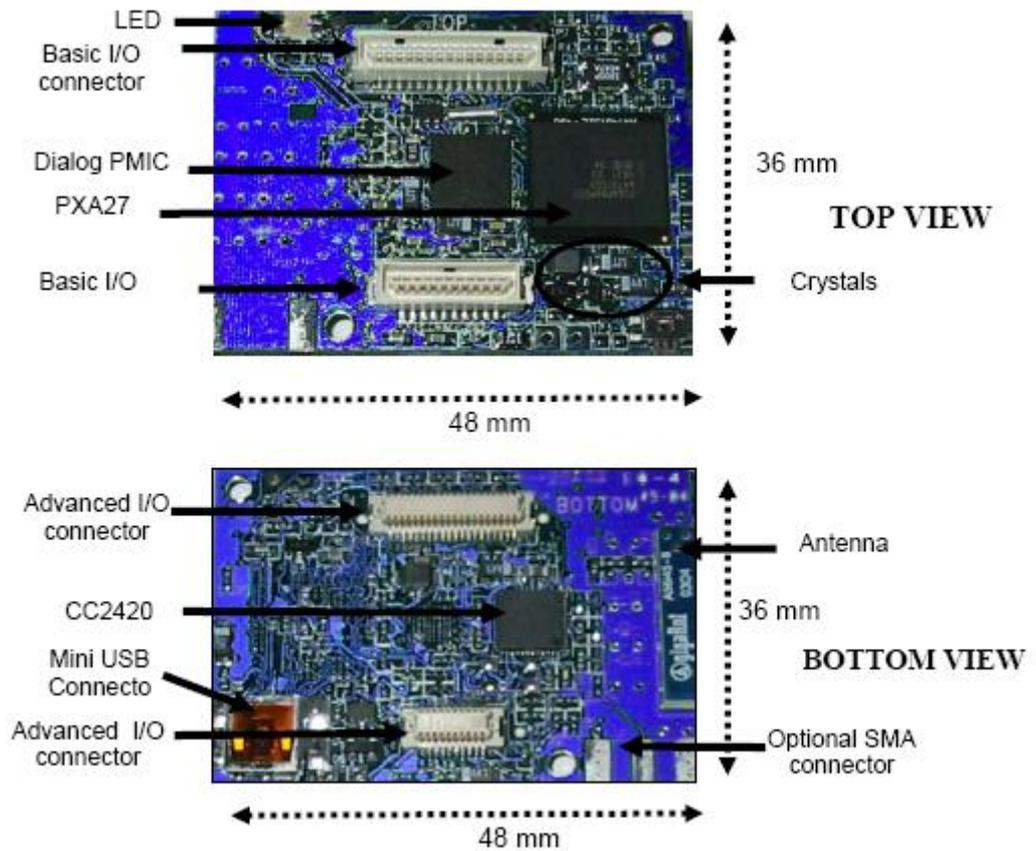


Figure 7.3 iMote2 main board

The iMote2 platform exposes two sets of connectors, the basic set (basic I/O connector shown in Figure 7.3 top view) and the advanced set (advanced I/O connector shown in Figure 7.3 bottom view). The pins on each connector set are split into two physical connectors (basic set: 31 pin and 21 pin connectors; advanced set: 40 pin and 20 pin connectors) to enhance the mechanical stability. The basic set is meant to enable low cost sensor boards (low density connectors were chosen) and support the most common sensor interfaces. This connector set is defined as the “architectural” set, and can be supported in future mote designs. One standard Universal Asynchronous Receiver/Transmitter (UART), two high speed UART ports, two Synchronous Serial Ports (SSPs), one Secure

Digital Input Output (SDIO) port, one Inter-Integrated Circuit (I2C) port and multiple General Purpose I/Os (GPIOs) are exposed in this connector set. The advanced connector set exposes some of the PXA271 advanced features, such as camera interface, high speed bus, audio interface, etc, and is assumed to be platform specific.

In addition, the iMote2 main board has one built in IEEE 802.15.4 radio transceiver CC2420 radio module [169] with onboard antenna which is shown in Figure 7.3 bottom view. It can provide a 250kbps communication data rate with 16 data channels in the 2.4 GHz band.

The simplified block diagram of CC2420 is shown in Figure 7.4. It has a separate 128-byte transmit data First-In-First-Out (FIFO) buffer and a 128-byte receive data FIFO buffer which enable its real bi-directional operation. CC2420 is a low Intermediate Frequency (IF) receiver. The received Radio Frequency (RF) signal is amplified by the Low Noise Amplifier (LNA) and down-converted in quadrature (I and Q) to the IF. Then the complex I/Q signal is filtered and amplified, and then digitised by the Analog-to-Digital Converters (ADCs). CC2420 buffers the received data in a 128-byte receive FIFO which can be read out through a serial peripheral interface. The Cyclic Redundancy Check (CRC) is verified in hardware. The transmitter is based on direct up-conversion. The data are buffered in a 128-byte transmit FIFO (separate from the receive FIFO), pre-processed in the chip and then output to the Digital-to-Analog Converters (DACs). An analogous low pass filter passes the signal to the quadrature (I and Q) up-conversion mixers. The RF signal is amplified in

the Power Amplifier (PA) and fed to the antenna. The preamble and start of frame delimiter are generated by hardware.

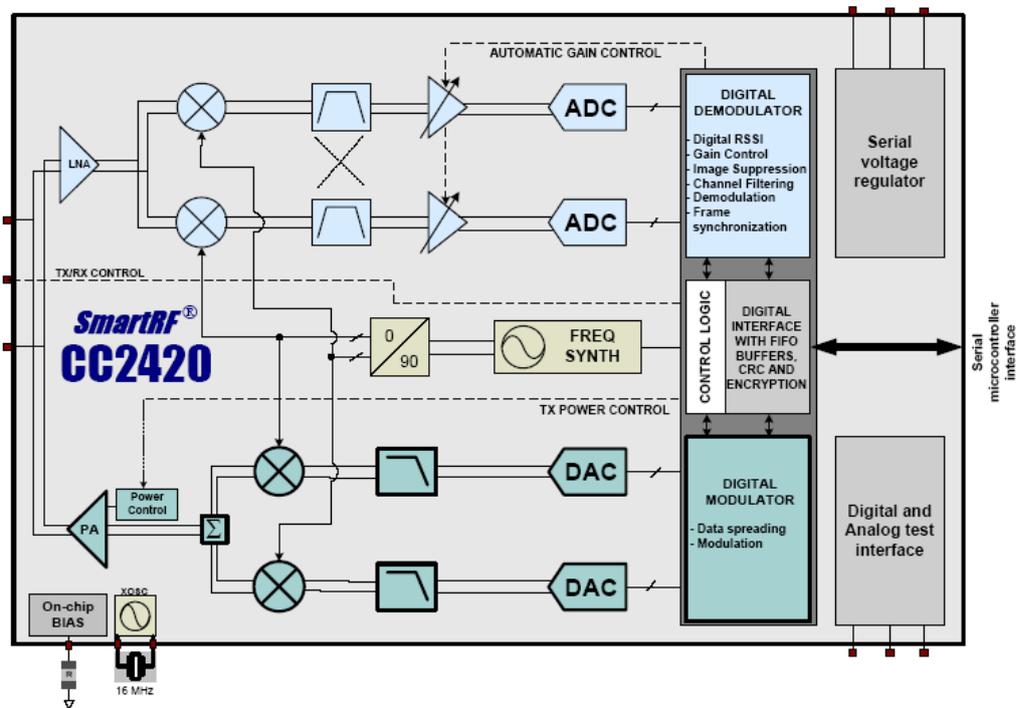


Figure 7.4 Simplified block diagram of CC2420

7.2.2.2 OV7620 Vision Module Introduction

The OV7620 vision module [168] is a highly integrated Interlaced/Progressive Scan CMOS digital colour video chip with a resolution of 640x480. The registers, which can be accessed through the Serial Camera Control Bus (SCCB), are listed in Table 2.1.

Table 7.1 OV7620 registers

Name	Address	Default Value	Notes
OV7620_GAIN	0x00	B0000,0000	Max:0x3f
OV7620_BLUE	0x01	B1000,0000	Blue gain
OV7620_RED	0x02	B1000,0000	Red gain
OV7620_SATU	0x03	B1000,0000	Saturation
OV7620_BRIGHT	0x06	B1000,0000	Lightness
OV7620_ANGALOG_SHAP	0x07	B1100,0011	
OV7620_WRITE_BRIGHT_BLUE	0x0c	B0010,0000	
OV7620_WRITE_BRIGHT_RED	0x0d	B0010,0000	

Name	Address	Default Value	Notes
OV7620_AUTO_EXPOSURE	0x10	interlace: B0111,1111 progressive: B1111,1111	ET
OV7620_CLK_RATE	0x11	B0000,0000	
OV7620_COMA	0x12	B0010,0100	
OV7620_COMB	0x13	B0000,0001	
OV7620_COMC	0x14	B0000,0100	
OV7620_COMD	0x15	B0000,0001	
OV7620_FRAME_DRO	0x16	B0000,0011	
OV7620_HORIZONTAL_START	0x17	B0010,1111	
OV7620_HORIZONTAL_END	0x18	B1100,1111	
OV7620_VERTICAL_START	0x19	B0000,0110	
OV7620_VERTICAL_END	0x1a	B1111,0101	
OV7620_PIXEL_SHIFT	0x1b	B0000,0000	
OV7620_ID_HIGH	0x1c	B0111,1111	
OV7620_ID_LOW	0x1d	B1010,0010	
OV7620_COME	0x20	B0000,0000	
OV7620_Y_OFFSET	0x21	B1000,0000	
OV7620_U_OFFSET	0x22	B1000,0000	
OV7620_CRYSTAL_CURRENT	0x23	B0000,0000	
OV7620_AEW	0x24	interlace: B0000,1000; progressive: B0001,0000	
OV7620_AEC	0x25	interlace: 0100,1010; progressive: 1000,1010	
OV7620_COMF	0x26	B1010,0010	
OV7620_COMG	0x27	B1110,0010	
OV7620_COMH	0x28	B0000,0000	
OV7620_COMI	0x29	B0000,0000	
OV7620_FRAME_RATE1	0x2a	B0000,0000	
OV7620_FRAME_RATE2	0x2b	B0000,0000	
OV7620_BLACK_EXPANDIND	0x2c	B1000,1000	
OV7620_COMJ	0x2d	B1000,0001	
OV7620_V_OFFSET	0x2e	B1000,0000	
OV7620_SIGNAL_A	0x60	B0010,0111	
OV7620_SIGNAL_B	0x61	B1000,0010	
OV7620_RGB_GAMMA	0x62	B0001,0010	
OV7620_Y_GAMMA	0x64	B0101,1001	
OV7620_SIGNAL_C	0x65	B0100,0010	
OV7620_AWB	0x66	B0101,0101	
OV7620_COLOUR_SPAC	0x67	B0001,1010	
OV7620_SIGNAL_D	0x68	B1100,1100	
OV7620_ANALOG_SHAPNESS	0x69	B0111,0010	
OV7620_VERTICAL_EDGE	0x6a	B0100,0010	
OV7620_NOICE_COMPENSATION	0x6f	B0011,1010	
OV7620_COMK	0x70	B1000,0001	
OV7620_COML	0x71	B0000,0000	
OV7620_HORIZONTAL_SYNC1	0x72	B0001,0100	

Name	Address	Default Value	Notes
OV7620_HORIZONTAL_SYNC2	0x73	B0101,0100	
OV7620_COMM	0x74	B0010,0000	
OV7620_COMN	0x75	B1000,0010	
OV7620_COMO	0x76	B0000,0000	Bit5 PWDN

7.2.2.3 Sensor Board Design and Implementation

The sensor board is designed to integrate OV7620 vision module with the iMote2 main board. The system schematic circuit is shown in Figure 7.5.

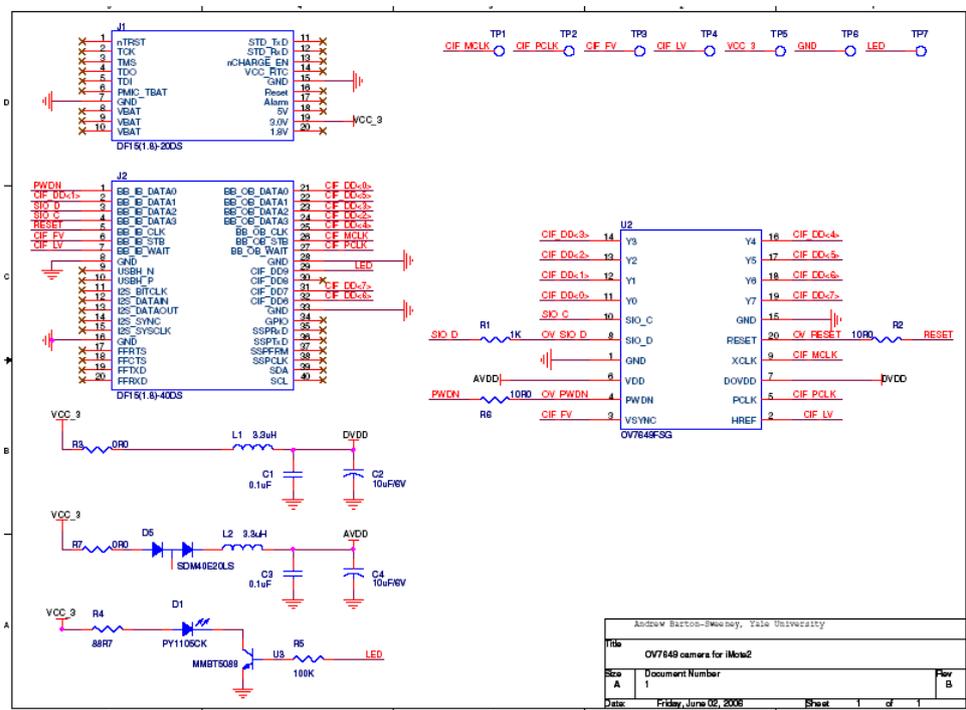


Figure 7.5 Sensor board schematic circuit connecting iMote2 and OV7620 vision module

The DF15-20S connector of the advanced pin set is used to get 3.0V power from the battery. In addition to the 8 bits data bus, the reset pin, the XCLK1 clock input, Pixel Clock (PCLK) output and SCCB Serial clock (SIO_C) and data (SIO_D) are also connected with iMote2 through the DF15(1.8)-40DS connector. Thanks to the design of the Intel PXA27x processor, the Direct Memory Access (DMA) flow-through and fly-by transfers are supported. As far as the Peripheral-Bus Peripheral (PBP) is

concerned, only flow-through mode is supported. Both sides of the corresponding Printed Circuit Board (PCB) layout are shown in Figure 7.6.

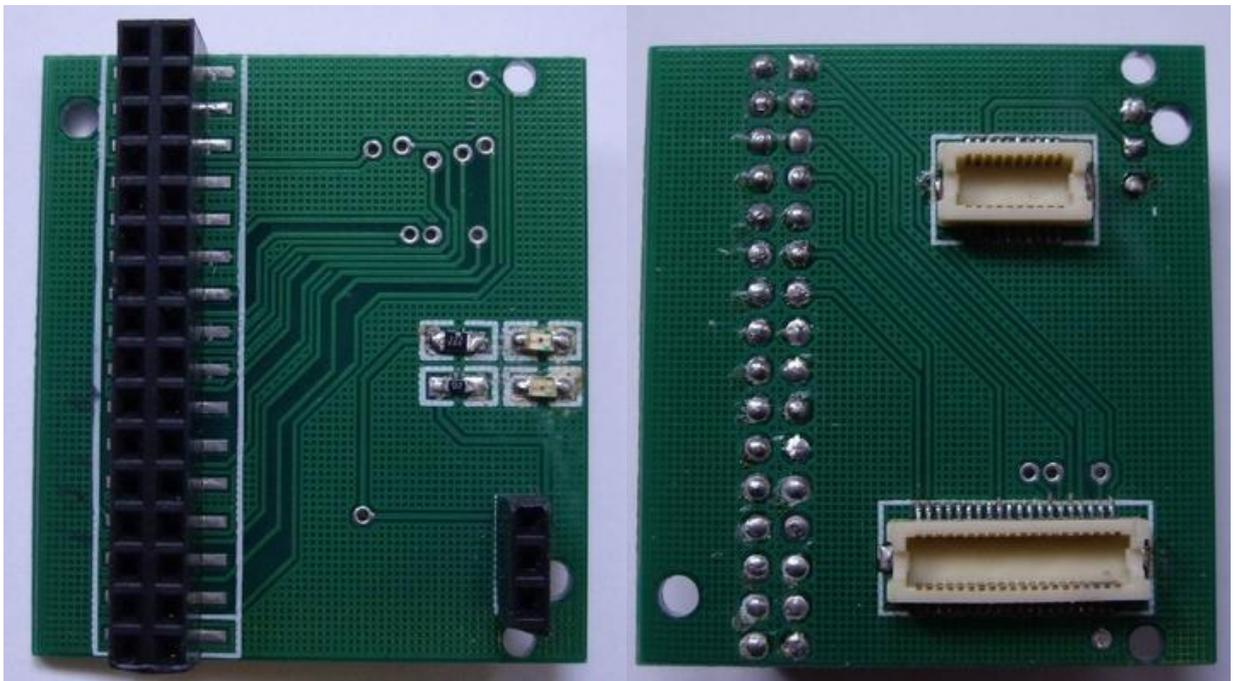


Figure 7.6 Sensor board PCB layout

In the design, parallel data output is chosen rather than serial port to enable DMA access with high speed data rate. It captures video in 30 fps continuously. DMA operation copies all image frames from OV7620 cache to the iMote2 kernel buffer which can hold a maximum of 4 frames. The buffer is filled in with frames one after another until the buffer is full of 4 frames. When the first frame in the buffer is cleared, a new frame will be written to the buffer. The procedure performs in a cyclic manner continuously. When the application needs to read the captured frames, it first sends reading command to the Linux kernel; after receiving the command, the kernel will lock the newest captured frame and copy it from the kernel level buffer to the user level buffer, which can be accessed by the application.

7.2.2.4 Power Board Introduction

The OTS power board, shown in Figure 7.7, is used to power the iMote2 using 3 primary AAA cells. As mentioned above, the battery board accommodates the plugged-in iMote2 via the advanced connectors. The battery board is fused for a 500mA maximum current with a mechanic switch on the side for manual power shut-off.



Batteries	3x AAA
Maximum Current	500mA Fused
Size	52mm x 43mm x 18mm
Weight with 3 AAA Batteries	51g
Weight without Batteries	14g

Figure 7.7 Power board and its specifications

7.2.3 Mobile Robot Hardware Design and Implementation

7.2.3.1 Overview of the Robot Hardware Platform

To fulfil the commands sent by intelligent environment, the mobile robot should be able to receive and analyse IEEE 802.15.4 packets, drive forward and steer accordingly. Hence a full functional mobile robot under the control of the environment intelligence should at least consist of five components in addition to the chassis: a *driving* module, a *steering* module, a *power* module, a *speed detection* module and a *wireless communication* module and a *Microcontroller Unit* (MCU). The hardware modules are shown in Figure 7.8. The reliable and stable power provided by the *power* module is the foundation of the system stability. The MCU is the brain of the robot to coordinate all other modules. The *speed detection* module measures and reports the speed of the motors to the MCU to form

a feedback control mechanism. The precision of speed feedback from the *speed detection* module determines the performance of the motor control. High power of the driving motor ensures a quick response of the movement, fast acceleration and deceleration while the *steering* module providing high steering torque guarantees a yare manoeuvre of the robot. The *reliable wireless communication* module enables the robot to interact with the environment. The completed robot hardware on the field is shown in Figure 3.4.

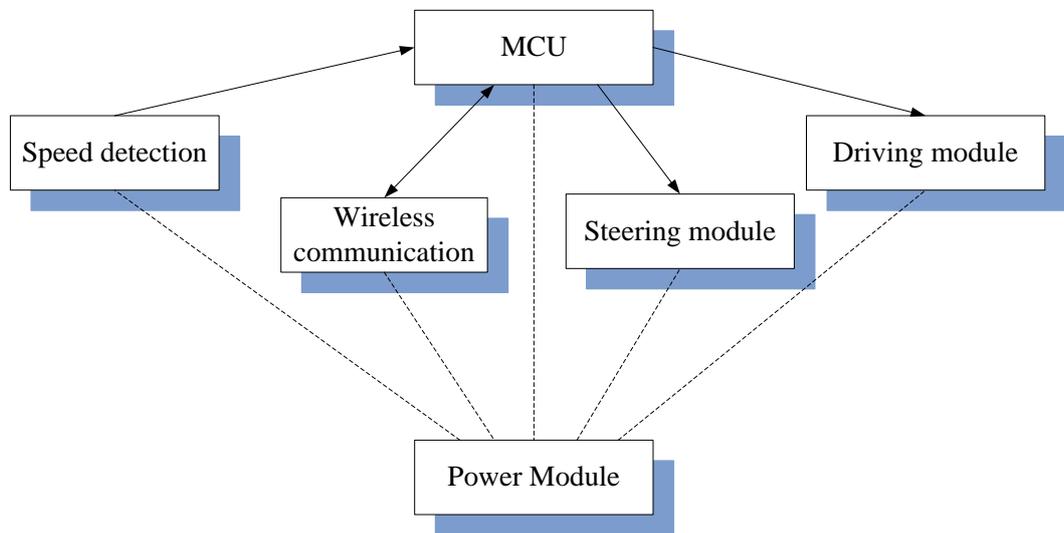


Figure 7.8 Robot hardware modules

7.2.3.2 The Mobile Robot Hardware Modules Introduction

This section introduces all the main parts of the mobile robot. The rationale on the choice of these OTS components is also given in the subsections.

7.2.3.2.1 Car Model Chassis

Under the philosophy behind this PhD work is that, the higher-level tasks in a general tiered architecture of an intelligent robot (autonomous

mobile robot), such as perception, localisation and path planning and etc., are processed by the distributed vision sensors; only the real-time motion control at the lowest level is implemented on-board so that a robot with very limited computational power can exhibit superior mobility in an intelligent environment. In order to focus on the proposed research, a car model as shown in Figure 7.9 from Tongji University is used as the robot in the following experimental study.



Figure 7.9 Car model

Table 7.2 lists the specifications of the car model, the rear wheel axis length of which is adjustable.

Table 7.2 Car model specifications

Item	Value
Wheelbase	198mm
Front axis length	137mm
Rear axis length	138mm/146mm
Wheel diameter	52mm
Transmission ratio	18/76

7.2.3.2.2 MCU Minimum System

With the advantage of high bus operation frequency up to 50MHz, rich on-chip resource and peripherals and on-chip error-correcting capability, the MC9S12DT128B Microcontroller Unit (MCU) is chosen as the brain of the robot. The MCU is a 16-bit device composed of standard on-chip peripherals including a 16-bit Central Processing Unit (HCS12 CPU), 128K bytes of Flash EEPROM, 8K bytes of RAM, 2K bytes of EEPROM, two asynchronous Serial Communications Interfaces (SCIs), two Serial Peripheral Interfaces (SPIs), an 8-channel IC/OC enhanced capture timer, two 8-channel 10-bit ADCs, one 8-channel Pulse-Width Modulator (PWM), one digital Byte Data Link Controller (BDLC), 29 discrete digital I/O channels (Port A, Port B, Port K and Port E), 20 discrete digital I/O lines with interrupt and wakeup capability, three Controller-Area Network (CAN) 2.0 A, B software compatible modules (MSCAN12) and an Inter-IC Bus. The MC9S12DT128B has full 16-bit data paths throughout. However, the external bus can operate in an 8-bit narrow mode, so single 8-bit wide memory can be interfaced for lower cost systems. The inclusion of a Phased Locked Loop (PLL) circuit allows power consumption and performance to be adjusted to suit operational requirements.

The MCU minimum system provides all the necessary peripherals for the MCU to operate, such as power supply, clock, I/O drivers, hardware reset circuit, RS-232 drive circuit, eight indication LEDs and etc.. Figure 7.10 shows the minimum system.

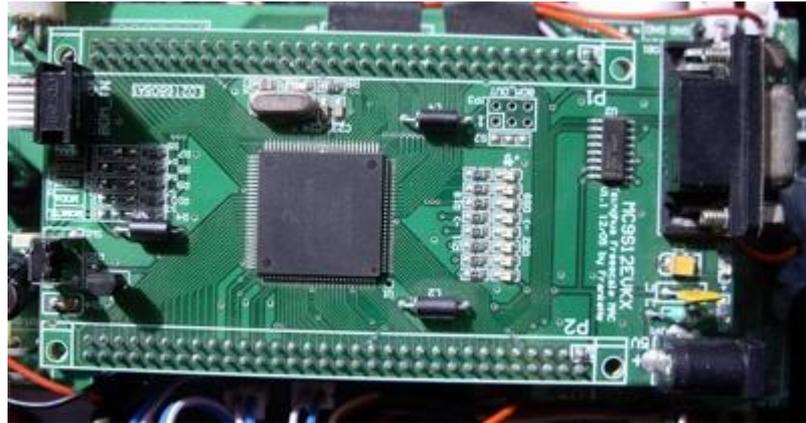


Figure 7.10 MC9S12DT128B MCU minimum system

7.2.3.2.3 DC Driving Motor

The driving motor is shown in Figure 7.11. The technical data is shown in Table 7.3. The driving motor is connected with the rear wheels to drive the mobile robot forward or backward.



Figure 7.11 DC driving motor

Table 7.3 RS-380SH-4045 motor technical data

Voltage		no load		at maximum efficiency				stall			
operating range	Nominal	speed	current	speed	current	torque		output	torque		current
		r/min	A	r/min	A	mN·m	g·cm	W	mN·m	g·cm	A
3~9	7.2V	16200	0.5	14060	3.29	10.9	111	16	82.3	839	21.6

7.2.3.2.4 Steering Servo Motor

As its name indicated, the servo motor for steering is used to steer the mobile robot to rotate. It is the essential component for mobile robot control and is shown in Figure 7.12.



Figure 7.12 Servo motor

The specifications are as following:

Dimension: 40.0 x 20.0 x 38.1mm

Speed: 0.2 sec/60° to 0.16 sec/60°

Torque: 5.2 kg·cm to 6.5 kg·cm

Weight: 41g

7.2.3.2.5 The Wireless Peripheral Module

The Gainz 2.4GHz IEEE 802.15.4 node is a low cost, low power consumption wireless sensor built with a low-power CMOS 8-bit microcontroller ATmega128L and a CC2420 chip. It can be programmed by the JTAG interface online. In addition, it exposes a standard RS-232 serial port which can be used to connect the SCI port of the MCU. Thus the Gainz wireless sensor node is chosen to act as a bridge converting serial port data to/from wireless 802.15.4 packets to enable the mobile

robot IEEE 802.15.4 wireless communication capability. It is shown in Figure 7.13.



Figure 7.13 Gainz 2.4GHz IEEE 802.15.4 node

7.2.3.3 Mobile Robot Hardware Design and Implementation

7.2.3.3.1 DC Motor Driving Module Design

The most common way of controlling DC motors is PWM which is typically controlled by an H-bridge circuit. Termed from the graphical representation of the circuit, H-bridge is built with four switches (solid-state or mechanical). When the switches S1 and S2 as shown in Figure 7.14 are closed (and S3 and S4 are open), a positive voltage will be applied across the motor. By opening the S1 and S2 switches and closing the S3 and S4 switches, this voltage is reversed, allowing reverse operation of the motor. Four flyback diodes are connected in parallel with the switches to avoid the heat generated by the motor when it works in electricity generating mode. However, the switches S1 and S3 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S2 and S4. This condition is known as shoot-through.

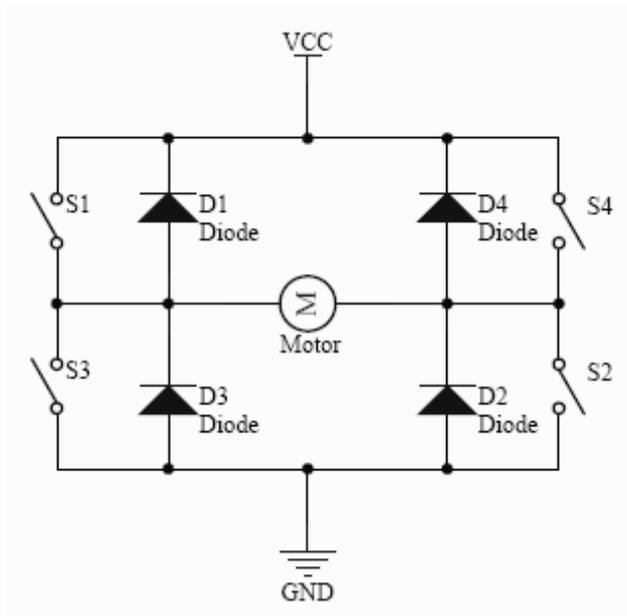


Figure 7.14 H-bridge circuit

Comparing with the H-bridge circuits built up by discrete components, Integrated Circuits (ICs) have following advantages:

1. ICs perform more stably and have better consistence for identical production processes;
2. Normally the shoot-through condition is prevented by the IC design making it more concise and reliable;
3. IC comes with multiple internal protection mechanisms, such as short-circuit protection, over-temperature protection, again making it more reliable;
4. IC has smaller footprints.

From the above discussion, IC is chosen to implement the PWM control. The H-bridge IC MC33886 is used because of the following features,

- 5.0 V to 30 V Operation

- 120 mΩ RDS(ON) H-Bridge Switches
- TTL /CMOS Compatible Inputs
- PWM Frequencies to 10 kHz
- Automatic PWM Overcurrent Limiting
- Output Short Circuit Protection
- Overtemperature Output Current Reduction with Shutdown
- Undervoltage Shutdown
- Fault Status Reporting

The following decisions are made in the design of the drive circuit:

1. Two MC33886s are used in parallel to double the current output to achieve better drive power (10A) and better heat dissipation;
2. D1 is grounded and D2 is connected to VCC to enable the IC in working condition;
3. Fault status report is not reported to the MCU;
4. IN1 and IN2 are connected to the MCU PWM output for control.

The driving circuit is shown in Figure 7.15.

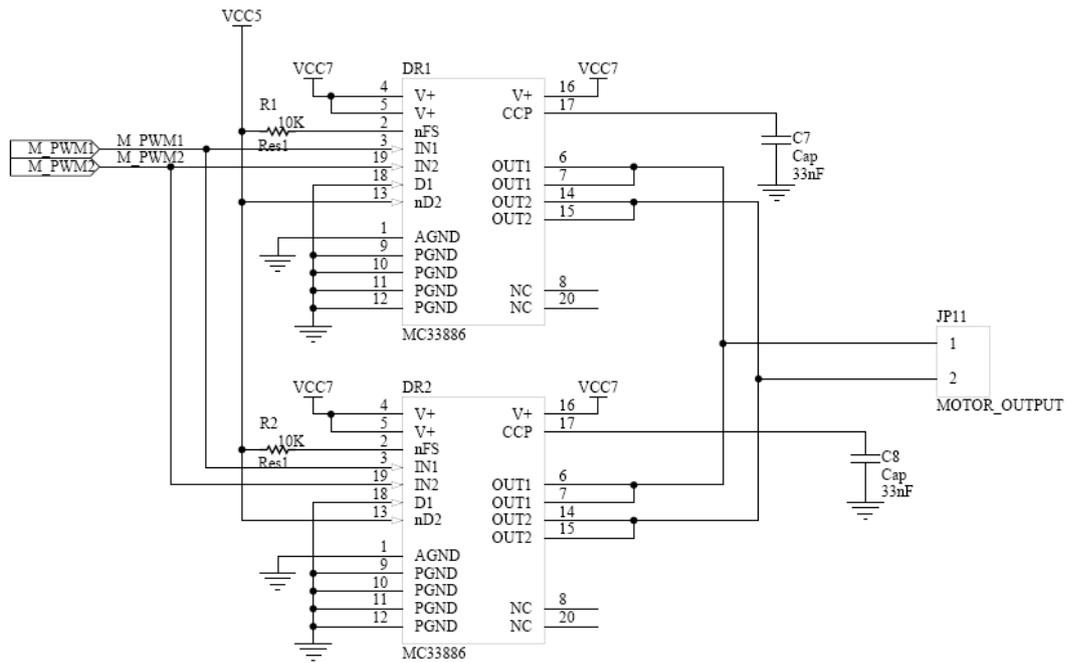


Figure 7.15 Driving motor circuit

7.2.3.3.2 Steering Module Design

The servo motor for steering consists of the cover case, control circuit, non-core motor, gears and position detector. When a control signal reaches the servo motor, the circuit will drive the motor accordingly by feedback control of rotational angle and then the driving torque is transmitted to its output shaft through the reduction gear pulse width signals are used to control the servo motor. The relation between rotational angle and the pulse width is shown in Figure 7.16.

The wiring for servo motor is straightforward: two power lines (red for positive, black for ground) and one control input (blue). Due to the dramatic change of the driving current, the power supply is separated from other modules in the design.

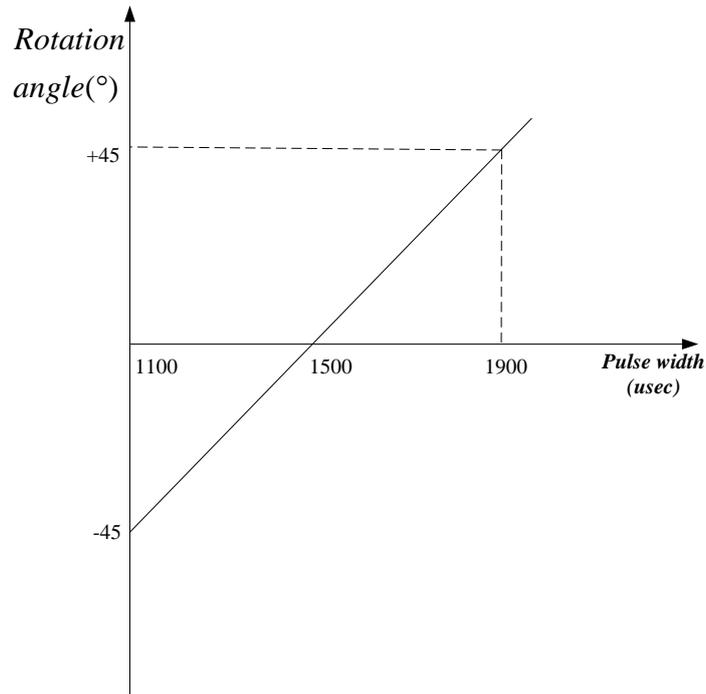


Figure 7.16 Pulse width – rotational angle relation, clockwise(+), anti-clockwise(-)

7.2.3.3.3 Speed Detection Module Design

Speed detection provides a speed feedback mechanism. It has a direct impact on the motor control performance. Two main mechanisms are categorised: contact and contactless. A tachogenerator is a typical contact method while optical encoder is a typical contactless approach. Optical encoder requires more CPU resource to count pulses in order to calculate the speed while a tachogenerator can achieve this function by utilising one ADC. Comparing with the tachogenerator, optical encoders are larger in volume and weight. As a result, the tachogenerator is chosen. The tachogenerator is geared with the rear wheel and mounted on the car as shown in Figure 7.17.

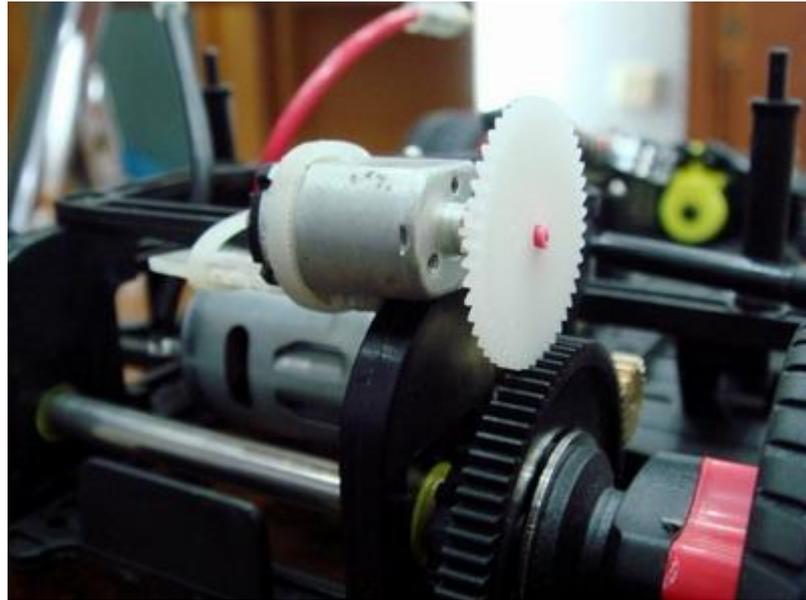


Figure 7.17 Tachogenerator is mounted on the mobile car

7.2.3.3.4 Mobile Robot System Implementation

The system PCB design is shown in Figure 7.18. Two MC33886 H-bridges, which are located in the right-centre of the board, are used to drive the DC motor. The 7V battery output is directly connected to the servo motor in the right bottom. Two 64pin-connectors are used to connect with the MCU minimum system board.

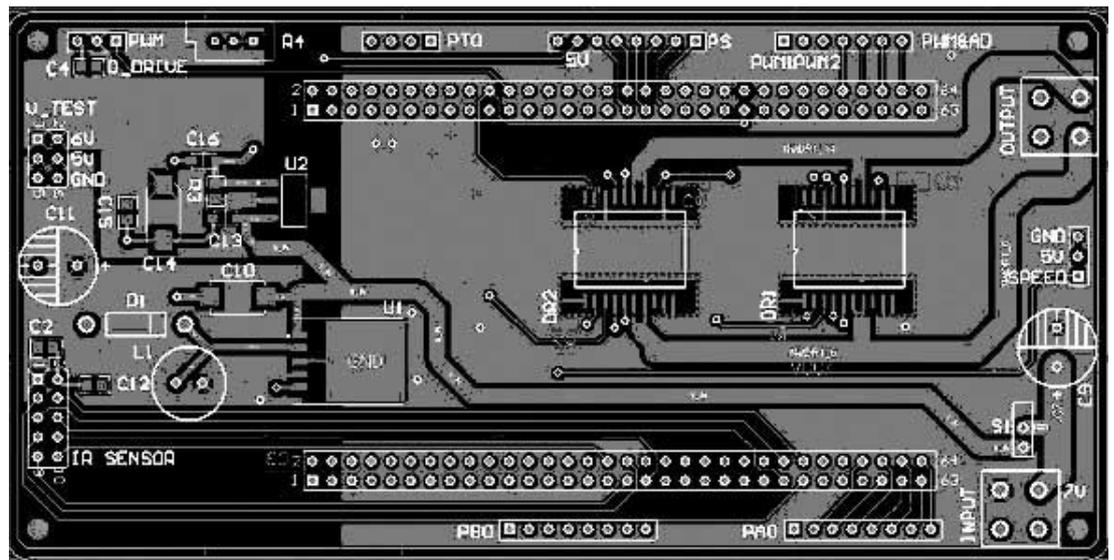


Figure 7.18 System PCB design

7.2.4 Remote Console

The remote console PC has a 2.0GHz duo-core AMD CPU, 1.5GB memory, an 80GB hard disk and four USB2.0 ports. The operation system running in the PC is Fedora 11.0 with 2.6.29 kernel. The same processing board used in the vision sensor, the iMote2 main board, is attached with the PC via a USB cable to enable the remote console IEEE 802.15.4 communication capability. The attached iMote2 is treated as an IP device by the PC after being assigned with an IP address. The Security Shell (SSH) network protocol is used to exchange files between the iMote2 and the PC over the USB cable. As a bridge, the attached iMote2 will exchange IP packets with PC over wired TCP/IP socket connections and send/receive datagram to/from the vision sensors by IEEE 802.15.4 protocol wirelessly.



Figure 7.19 Remote console with attached iMote2 main board

7.3 Communication Protocol

The packet formats and information exchange flows are defined in the communication protocol. The communications include vision sensor and vision sensor communications, vision sensor and mobile robot communication, and vision sensor and remote console communications. The protocol implementations form an adaptation layer in each of the components.

7.3.1 Overview

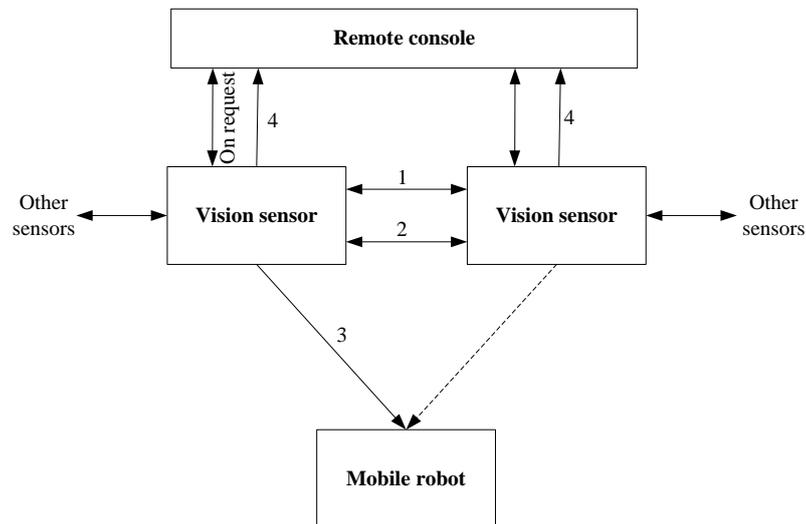


Figure 7.20 Communication protocols overview

Figure 7.20 is an outline of all the communication parties and the links between them. The purpose of the protocols is to establish a reliable and precise data exchange among the intelligent environment in order to support the path planning and motion control. Depending on different types of commands, communication can be invoked either by remote console or by vision sensors. As the key intelligent component, vision sensors are involved in all commands, either as a sender or as a receiver.

Except the on request commands originated by an operator in the remote console and the responses to these requests, commands are invoked periodically according to a certain sequence:

1. **Control points and/or obstacles exchange command:** Control points coordinates are used to calculate internal forces to deform a snake. Snake segments in two neighbouring vision sensors can interact with each other by adjusting their positions according to the forces generated by the received control points coordinates from others. Obstacles observed by one vision sensor will be sent to others if localisation fusion is required. After snake deformation, vision sensor will send the planned control points to its neighbouring sensors;
2. **Control token command:** At a specific time, one robot should be controlled by only one vision sensor. Only the vision sensor with control token can send control commands to robot and it will become the dominant vision sensor. If it has the token, it needs to broadcast its ownership of the token periodically or initiate a token handover procedure;
3. **Control commands:** The vision sensor with control token will send commands to the robot; the one without a control token will withdraw its planned commands;
4. **Monitoring purpose commands:** If a vision sensor is marked by an operator to send monitoring related information, such as control points, it will send out the corresponding information to the remote console.

The adaptation layer is built on top of the IEEE 802.15.4 short frame protocol. As discussed in 7.2.2.1, both the receiver and transmitter buffers

in the CC2420 are 128-byte buffers. To be compatible with TinyOS, the CC2420 driver adopts the following data structure,

```
typedef struct __TOS_Msg
{
    __u8 length;    // data length of payload
    __u8 fcfhi;    // Frame control field higher byte
    __u8 fcflo;    // Frame control field lower byte
    __u8 dsn;      // sequence number
    __u16 destpan; // destination PAN
    __u16 addr;    // destination Address
    __u8 type;     // type id for Active Message Model handler
    __u8 group;    // group id
    __s8 data[TOSH_DATA_LENGTH]; // payload
    __u8 strength; // signal strength
    __u8 lqi;
    __u8 crc;
    __u8 ack;
    __u16 time;
} TOS_Msg;
```

As seen in the TOS_Msg structure, 16 bytes are used as headers, the maximum payload length, TOSH_DATA_LENGTH, should be 112 bytes.

7.3.1.1 Generic Packet Format

Based on the TOS_Msg data structure, two kinds of generic packet are defined in this thesis: with or without source address. All packets discussed are the payload of the TOS_Msg.

The first type of generic packet without source address is shown in Table 7.4.

Table 7.4 Generic packet format without source address

CHK(B0)	CMD(B1)	User payload(B2~B111)
---------	---------	-----------------------

- B0: check sum
- B1: Command Type
- B2~B111: user payload, length varies.

The second type of generic packet with source address is shown in Table 7.5

Table 7.5 Generic packet format with source address

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
User payload(B8~B111)							

- B0: check sum
- B1: Command Type
- B2, B3: Sender short address from 1 to 65535 (0 is broadcast address)

$$address = SrcAddrH * 256 + SrcAddrL \quad (7.1)$$
- B4, B5: Packet sequence

$$SN = SNH * 256 + SNL \quad (7.2)$$
- B6, B7: Total number of packets in one transmission

$$Total = TotalH * 256 + TotalL \quad (7.3)$$
- B8~B111: user payload, length varies.

7.3.1.2 Commands List

There are totally 16 commands implemented to meet the data exchange and control requirements. Their descriptions are listed in Table 7.6.

Table 7.6 Packet command lists

CMD	Description	Direction
-1,0	Not used	N/A
1	Restart iMote2 user program	RC to VS
2	Request an image	RC to VS
3	Resample the background image	RC to VS
4	Obstacles	VS to VS
5	Send data request	RC to VS
6	Packet data	RC to VS
7	Control points	VS to RC, VS to VS
8	Image data	VS to RC
9	Ack	VS to RC
10	iMote2 system status Info	VS to RC
11	Robot control commands	VS to MR
12	A-Snake	VS to RC
13	Robot coordinate	VS to RC
14	Token	VS to VS
15	Control points display switch	RC to VS
16	Robot control mode	RC to VS

VS: Vision Sensor, RC: Remote Console, MR: Mobile Robot

7.3.2 Communication between Vision Sensors

Three types of commands between vision sensors are obstacles, control points and handover. The commands type fields are CMD4, CMD7 and CMD14 respectively.

7.3.2.1 CMD4: Obstacles Command

This command is created to provide information for multiple geometry obstacle localisation. If obstacles are observed by one vision sensor, and this vision sensor has overlapping areas with the coordinator, it will transmit the observed obstacles to the coordinator. This function can be disabled to reduce communication burden in the program.

The data format is shown in Table 7.7.

Table 7.7 Obstacle packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
NOB(B8)	X1H(B9)	X1L(B10)	Y1H(B11)	Y1L(B12)	X2H(B13)	X2L(B14)	Y2H(B15)
Y2L(B16)	Obstacle coordinates (B17 ~ B111)						

- CMD = 4
- B0 ~ B7 refer to 7.3.1.1
- B8: Total number of obstacles to be sent
- B9~B111: obstacles coordinates

Packet fields calculation formula and signal flow are similar to the control point commands in 7.3.2.2.

7.3.2.2 CMD7: Control Points Commands

The purpose of this command is to transmit the planned control points from one vision sensor to another. To reduce the communication burden and save frequency resource, only the preceding vision sensors send border control points to the succeeding ones, as shown in Figure 7.21.

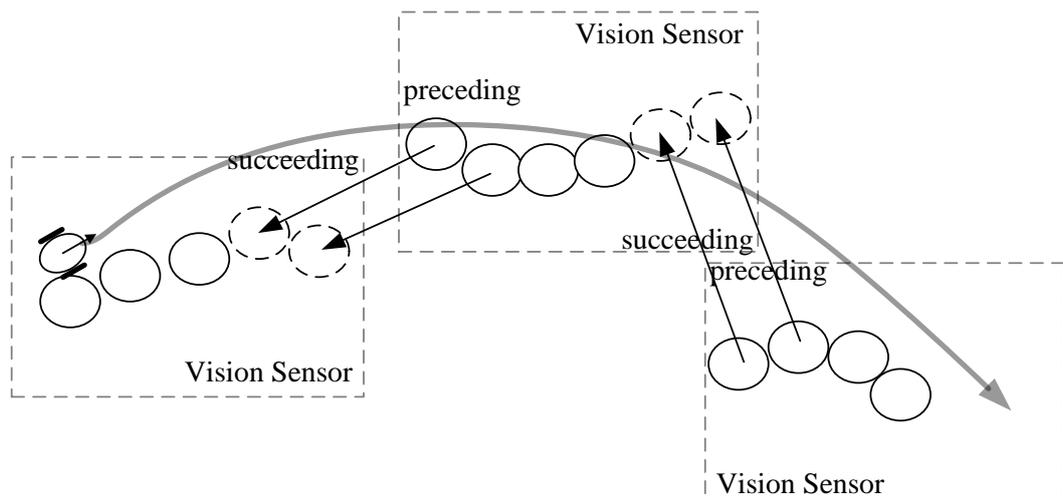


Figure 7.21 Sending border control points from preceding vision sensor to succeeding ones

The signal flow is shown in Figure 7.22. Border control point coordinates are transmitted periodically by all the vision sensors to their succeeding vision sensors if they exist. Destination address is specified in the TOS_Msg header.

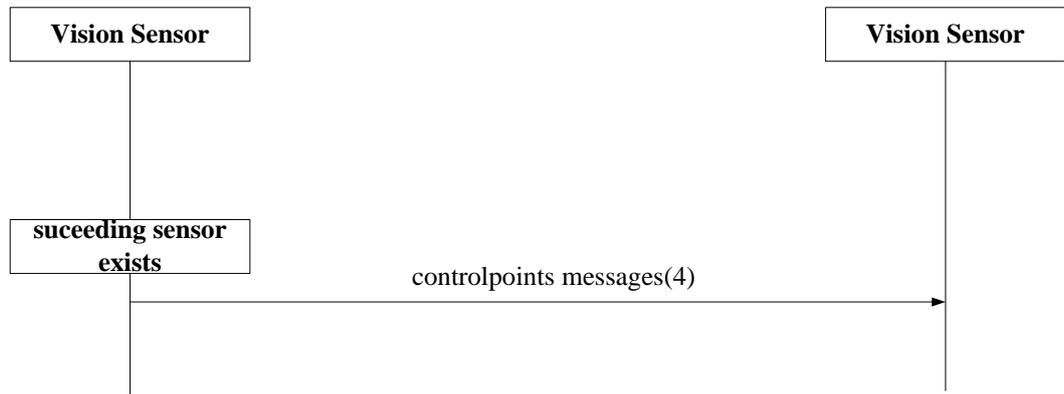


Figure 7.22 Exchange border control points signal flow

The corresponding packet format is shown in Table 7.8,

Table 7.8 Control point packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
NCP(B8)	X1H(B9)	X1L(B10)	Y1H(B11)	Y1L(B12)	X2H(B13)	X2L(B14)	Y2H(B15)
Y2L(B16)							

- CMD = 7
- B0 ~ B7 refer to 7.3.1.1
- B8: Total number of control points to be sent
- B9~B16: Control point coordinates,

$$x = X1H * 256 + X1L \quad (7.4)$$

$$y = Y1H * 256 + Y1L \quad (7.5)$$

7.3.2.3 CMD14: Handover Token Commands

At a specific time, only the coordinator can send control command to the mobile robot. There is no control centre to assign the control token

among vision sensors in the distributed environment. All vision sensors have to compete for the token based on its view of the mobile robot, shown in Figure 7.23. The areas are divided into five zones from inner area (zone0) to outer area (zone4). The purpose of this command is to initiate the negotiation procedure and determine the right vision sensor to have the control token.

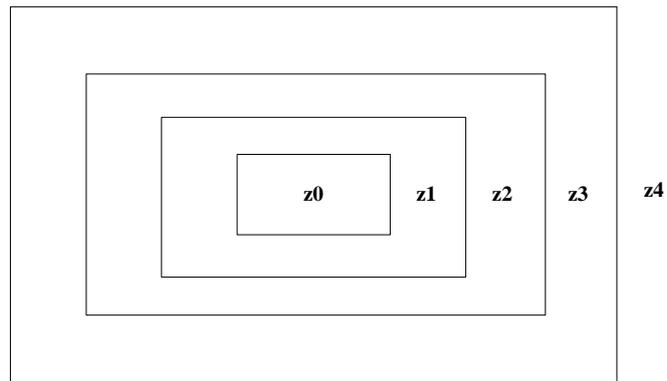


Figure 7.23 Mobile robot in observation zones in vision sensor

7.3.2.3.1 Token Handover Signal Flows

The signal flows are interpreted as follows.

Case 1: One vision sensor sends request to compete for the token and there is no other request found at the same time. A timer is set up once the command is broadcasted. If there is no other token request messages received during the timer lifetime, the vision sensor takes the token and broadcast its ownership of the token immediately. Figure 7.24 shows the signal flow.

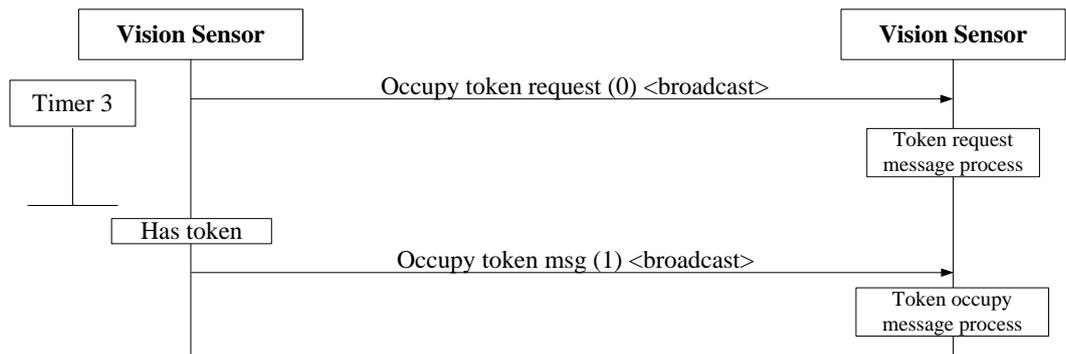


Figure 7.24 Control token init signal flow, case 1

Case 2: If during the timer lifetime, other vision sensors broadcast a token request, then the values of their addresses are used to determine the token ownership, i.e. smaller value of the address will be the winner. Figure 7.25 depicts the signal flow.

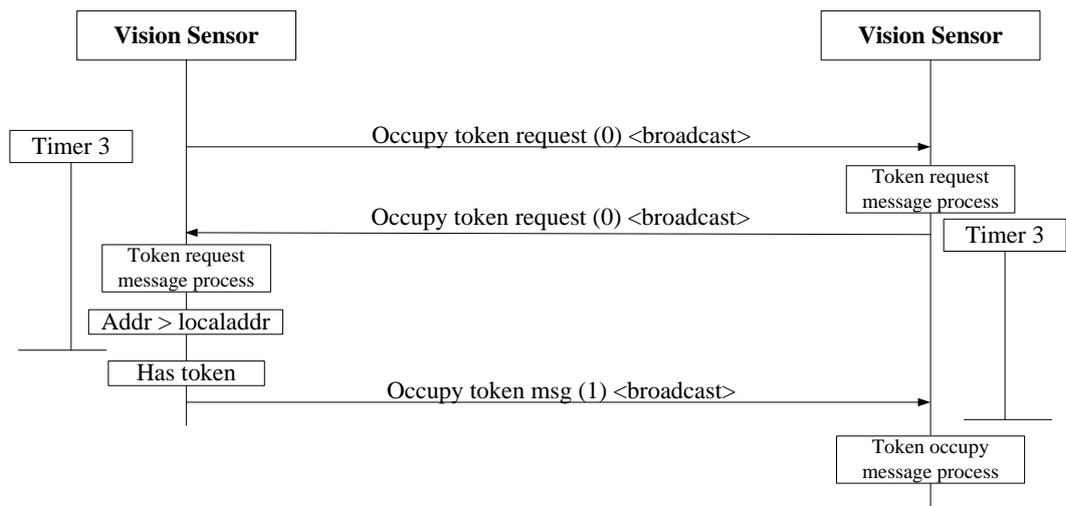


Figure 7.25 Control token init signal flow, case 2

Case 3: Once a vision sensor has the control token, it will broadcast its ownership periodically. Upon receipt of this message, other vision sensors will set up another timer. During the lifetime of this timer, it assumes that the ownership is occupied by others and will not send request message during this time. If one vision sensor sends a request for the token and during this time, it receives message indicating that the

token is already been occupied by others, it will stop competing for the token. Figure 7.26 shows the signal flow.

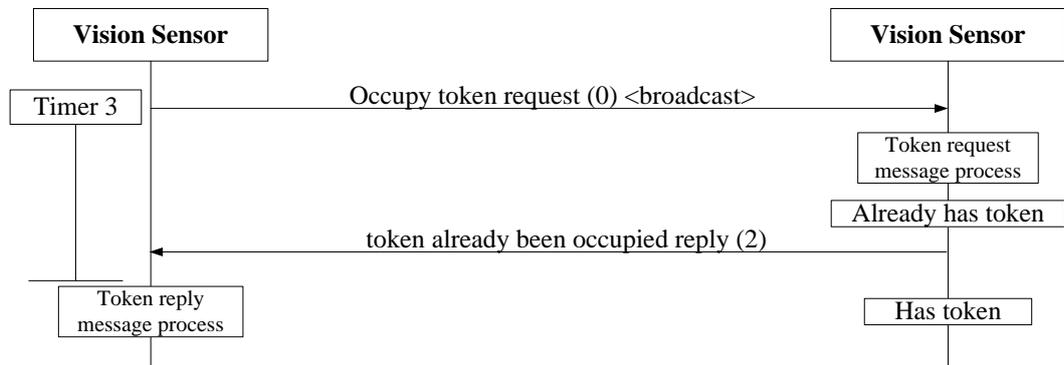


Figure 7.26 Control token init signal flow, case 3

Case 4: When the mobile robot moves from an inner area to an outer area in the vision, the coordinator will try to initiate a procedure to handover the token to other vision sensors. First it broadcasts a token handover request with its view zone value and setup a timer (Timer 1). Upon receipt of the handover message, other vision sensors will check whether they have a better view on the robot. Vision sensors with better views will send token handover reply messages back to the coordinator and setup a timer (Timer 2). If the coordinator receives the response messages before the Timer 1 expires, it will choose the vision sensor as the target and send token handover confirmation message to that target vision sensor to hand over its ownership. If token handover confirmation message is received, the target vision sensor will have the token, as shown in Figure 7.27.

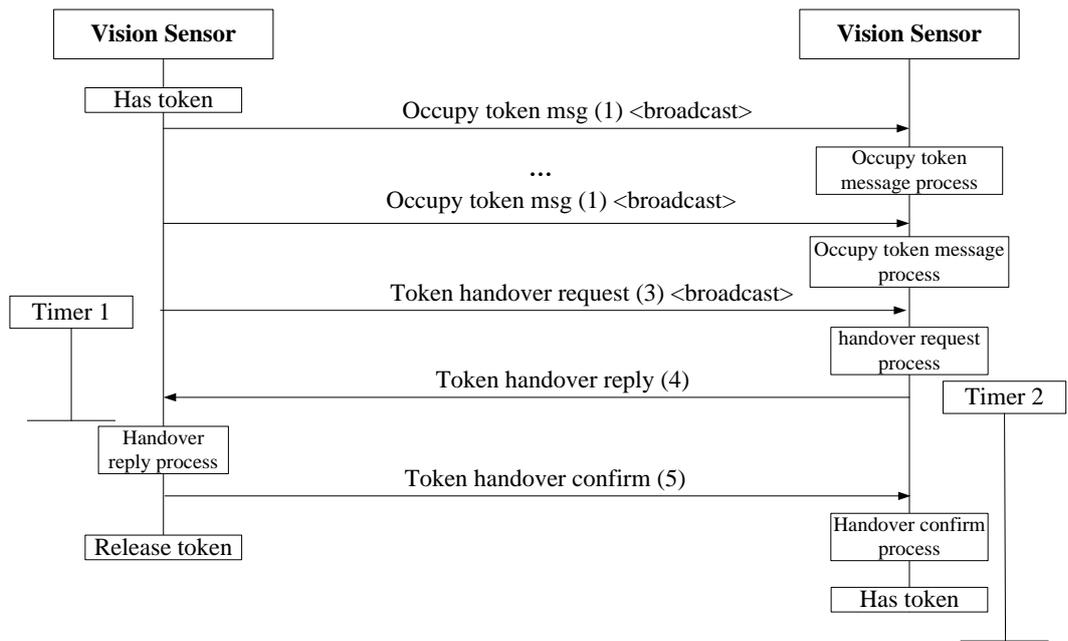


Figure 7.27 Control token handover signal flow - successful

However if no handover confirmation message received before the Timer 2 expires, that is the handover confirmation message does not reach the recipient, a token init procedure will be invoked as no other sensors apart from the coordinator has the token to broadcast the occupy token message which is shown in Figure 7.28.

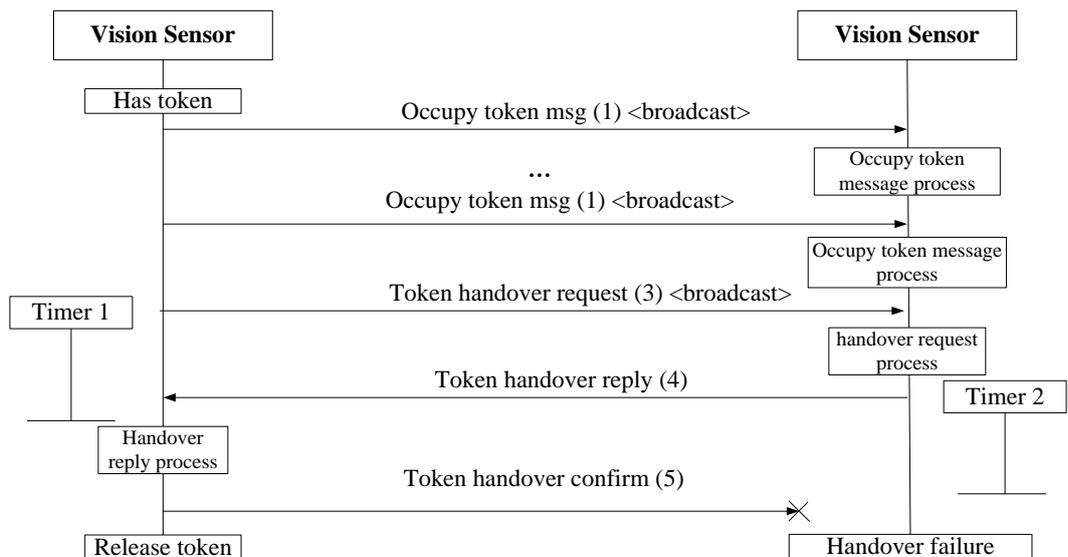


Figure 7.28 Control token handover signal flow - failure

The packet format is listed in Table 7.9.

Table 7.9 Token packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
type(B8)	zone(B9)						

- CMD = 14
- B0 ~ B7 refer to 7.3.1.1
- B8: Token message types.

The descriptions and possible values for B8 is listed in Table 7.10,

Table 7.10 Token messages

B8 value	Description
0	Init token request
1	Occupy token msg
2	token already occupied reply
3	Token handover request
4	Token handover reply
5	Token handover confirmation

- B9: view zones. It is used to indicate the quality of mobile robot being observed in one vision sensor. There are zone0, zone1, zone2, zone3 and zone4 with the values 0, 1, 2, 3 and 4 respectively, shown in Figure 7.23.

7.3.3 Communication between Vision Sensor and Robot

After planning, the dominant vision sensor will send a series of commands to the robot with time tags. The signal flow is shown in Figure 7.29.

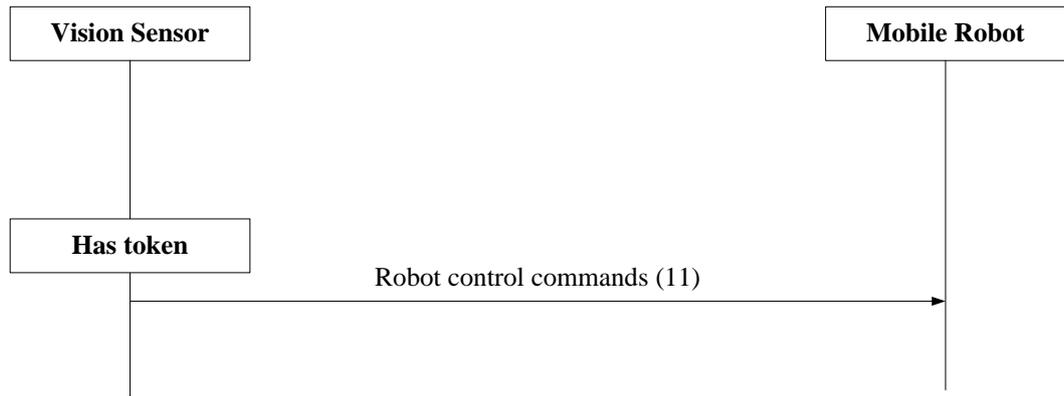


Figure 7.29 Robot control signal flow

The packet format is shown in Table 7.11,

Table 7.11 Robot control commands packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)
TotalL(B7)	Num of steps(B8)	timet1(B9)	Vvalue(B10)	Vsign(B11)	Dvalue(B12)	Dsign(B13)
(B14 ~ B111)						

- CMD = 11
- B0 ~ B7: refer to 7.3.1.1
- B8: total steps of command series
- B9~B111: commands with time tags

Timet and velocity *Vvalue* are multiplied by 100 before they are put in the packet to convert float numbers into integers. The original unit of time is in second and the value is the time offset from the previous one. The unit for velocity is meter/second before multiplication.

The value ranges are listed in Table 7.12,

Table 7.12 Robot control command fields

Field	Value
Dsign	0: left or centre, 2: right
Dvalue	0~45 degree
Vsign	0: forward or stop, 2: backward
Vvalue	0~255 cm/s

7.3.4 Communications between Vision Sensors and the Remote Console

The remote console is responsible for system parameters setting, status monitoring, vision sensor node controlling and etc.. The communication protocol between vision sensors and console is designed to provide the foundations of these functions.

As a transparent wireless bridge for the remote console, the wireless peripheral is always try to initiate and maintain a TCP connection with the remote console to establish a data exchange tunnel when it starts.

Three signal flow scenarios are involved in transferring data between the remote console and vision sensors as explained below.

- **Scenario 1:** Remote console – Vision sensor signalling: Figure 7.30 shows the signal flow of this scenario. The remote console sends control commands to vision sensors. Commands are invoked by the operator, transparently forwarded by the wireless module to the vision sensors. This scenario covers all commands forwarded sent by the remote console to vision sensors without requiring acknowledgement from vision sensors. If an error happens during transmission from the wireless module to vision sensors, re-transmission is not required because vision sensors have errors handling mechanisms for these categories of commands. The messages include restart the user program, request the vision sensor to send an image, instruct vision sensor to re-sample a background frame, switch on or off the control

points remote display and whether vision sensor continuously control robot if it has the token.

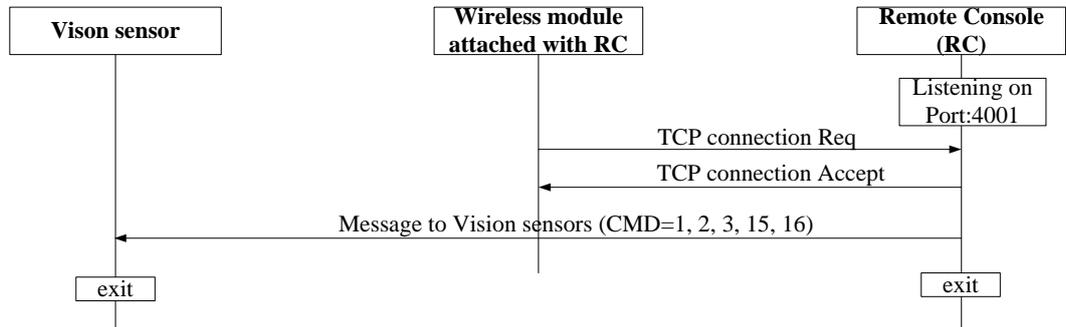


Figure 7.30 Signal flow between remote console and vision sensor - Scenario 1

- Scenario 2:** This scenario illustrates the processes of messages being sent from vision sensors to the remote console upon receipt of messages in scenario 1. It is not a reliable transmission; lost messages losing can be tolerated by the system. The messages consists of sending control points to remote console, sending image frames to remote console, reporting vision sensor system error or status info to remote console, sending A-snake control points to the remote console and sending the robot location to the remote console.

Figure 7.31 depicts the signal flow.

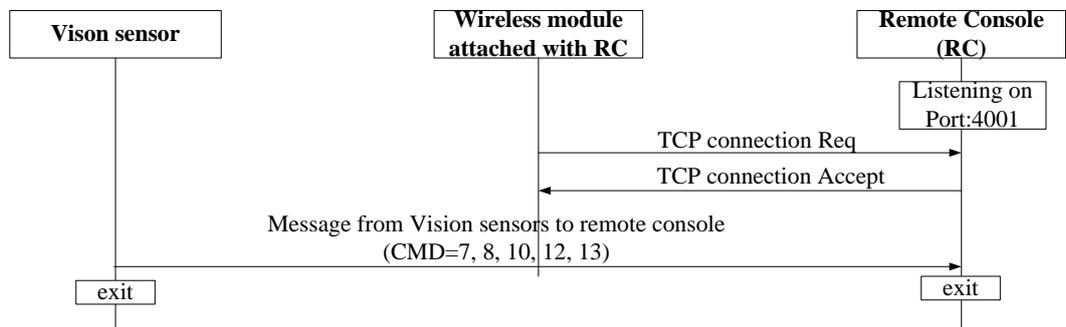


Figure 7.31 Signal flow between remote console and vision sensor - Scenario 2

- Scenario 3:** This **scenario** is proposed in the case of a need for reliable transmission, such as downloading system drivers, updating user programs, transferring setting profile files and etc.. The session is originated by the operator via the remote console. All data packets are required to be acknowledged by the recipient. Re-transmission will be invoked if no confirmation messages received within a given time. The scenario is illustrated in Figure 7.32.

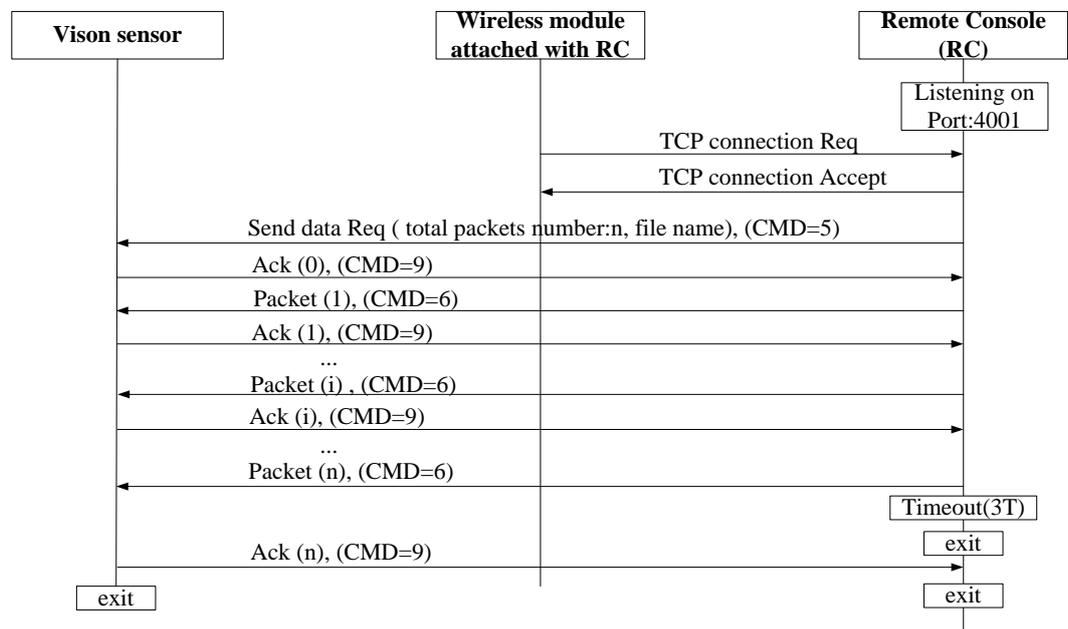


Figure 7.32 Signal flow between remote console and vision sensor - Scenario 3

Interpretations of each of the commands are as follows.

7.3.4.1 CMD1: Restart Client Program

This is a pure vision sensor control command without user payload. The packet format is shown in Table 7.13. It is designed for an operator to remotely restart the application in the vision sensors.

Table 7.13 Restart client program packet format

CHK(B0)	CMD(B1)
---------	---------

- CMD = 1
- B0 ~ B1, refer to 7.3.1.1.

7.3.4.2 CMD2: Acquire Image

The command is used to request vision sensors to send one frame of an image to the remote console for displaying and debugging purpose. The packet format is shown in Table 7.14. It is designed for an operator to remotely restart the application in the vision sensors.

Table 7.14 Acquire image packet format

CHK(B0)	CMD(B1)	imagetype(B2)
---------	---------	---------------

- CMD = 2
- B0 ~ B1, refer to 7.3.1.1.
- B2 indicates the type of image requested. The following image formats are supported,

Table 7.15 Image format

B2 Value	Descriptions
0	Gray extracted blob image
1	Original gray image
2	Gray foreground image
3	Sample colour image and store them in vision sensors

7.3.4.3 CMD3: Change Background Image

This is a pure vision sensor control command without user payload. The packet format is shown in Table 7.16. It is designed for an operator to instruct the user application to re-sample a number of images in order to calculate a new background.

Table 7.16 Change background image packet format

CHK(B0)	CMD(B1)
---------	---------

- CMD = 3
- B0 ~ B1, refer to 7.3.1.1.

7.3.4.4 CMD5: Send Data Request

This command is used to inform the vision sensor that the remote console will download a large size file with the file name indicated in the payload field. The total file size in bytes is carried within this packet for receipt to allocate storage space before data transfer. The packet format is shown in Table 7.17

Table 7.17 Send data request packet format

CHK(B0)	CMD(B1)	Total1(B2)	Total2(B3)	Total3(B4)	Total4(B5)
file name (B6~B111)					

- CMD = 5
- B0 ~ B1, refer to 7.3.1.1.
- B2 ~ B5, Total file size in bytes,

$$filesize = ((Total1 * 256 + Total2) * 256 + Total3) * 256 + Total4 \quad (7.6)$$

- B6 ~ B111, the transferred file name. The length of the file name should be less than 104 characters so that they can be fitted within one packet.

7.3.4.5 CMD6: Reliable Data Packets

Data packets with a unique integer number indicating its sequence for error re-transmission. The packet format is shown in Table 7.18.

Table 7.18 Data packet format

CHK(B0)	CMD(B1)	SNH(B2)	SNL(B3)
data (B4~B111)			

- CMD = 6
- B0 ~ B1, refer to 7.3.1.1.
- B2 ~ B3, Packet sequence.

$$SN = SNH * 256 + SNL \quad (7.7)$$

- B4 ~ B111, data payload.

7.3.4.6 CMD7: Control Points

This data packet is used to transfer control points to remote console for displaying purpose. It sends data periodically whenever new planned control points are ready. The transfer can be switched on or off by sending commands CMD15. The data format is shown in Table 7.19.

Table 7.19 Control point packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
NCP(B8)	Control points (B9 ~ B111)						

- CMD = 7
- B0 ~ B7, refer to 7.3.1.1.
- B8, number of control points to be sent. Each control points occupies four bytes in the packet,

$$Total = Round((NCP * 4 + 1) / (TOSH_DATA_LENGTH - 8)) \quad (7.8)$$

- B9 ~ B111, control points, they are organised as following,

Table 7.20 Control point payload

X1H(B9)	X1L(B10)	Y1H(B11)	Y1L(B12)
---------	----------	----------	----------	-------

7.3.4.7 CMD8: Image Data

This packet is designed for the operator to check the status of the environment in vision sensor's view, discover errors during user program debugging and etc.. The packet format is listed in Table 7.21.

Table 7.21 Image data packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
Image pixels (B8 ~ B111)							

- CMD = 8
- B0 ~ B7, refer to 7.3.1.1.
- B8 ~ B111, gray values of image pixels, 0~255. The image data are arranged as in Figure 7.33, i.e. read data along with the width first.

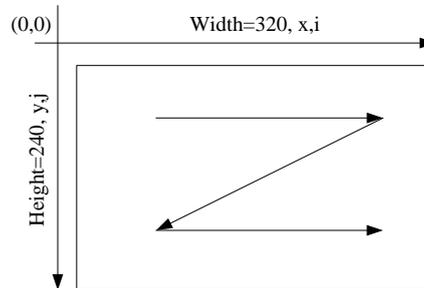


Figure 7.33 Image data structure

7.3.4.8 CMD9: ACK

This packet is used in association with CMD5 and CMD6 when vision sensor receives data from the remote console by reliable transmission. The ACK number should be increased by one if the packet received is correct and remain unchanged if the current packet is required to be re-transmitted due to errors.

Table 7.22 ACK packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
ACKH(B8)	ACKL(B9)						

- CMD = 9
- B0 ~ B7, refer to 7.3.1.1.
- B8 ~ B9, The sequence number to be acknowledged.

7.3.4.9 CMD10: iMote2 System Status Information

This packet is designed for the vision sensor to report the status of user program. It is invoked when certain kinds of situation happened without instructions from the operator.

Table 7.23 System status information packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
InfoCode(B8)							

- CMD = 10
- B0 ~ B7, refer to 7.3.1.1.
- B8, represents the information code defined as follows,

Table 7.24 iMote2 system information

B8 Value	Description
0	No error
1	Camera module init error
2	Vision sensor is not in the topological route
3	Sample background image error
4	Camera operation error (ioctl error)
5	Failure to record the debug information
6	Colour image is successfully sampled and stored
7	Robot is in the target position

7.3.4.10 CMD12: A-snake Control Points

If the robot is observed by a vision sensor, whether it has a token or not, the vision sensor needs to plan the accompanied snake so that it has robot control commands ready whenever it gets the token. The planned A-snake control points are automatically transferred to the remote console periodically for display. Table 7.25 shows the packet format.

Table 7.25 A-snake control point packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
NCP(B8)	A-snake control points (B9 ~ B111)						

- CMD = 12
- B0 ~ B7, refer to 7.3.1.1.
- B8, the number of control points to be sent. Each control point occupies four bytes in the packet,
- B9 ~ B111, control points, they are organised as the following,

Table 7.26 A-snake control point payload

X1H(B9)	X1L(B10)	Y1H(B11)	Y1L(B12)
---------	----------	----------	----------	-------

7.3.4.11 CMD13: Robot Coordinate

This packet is used to update the robot's coordinates in the remote console.

Table 7.27 Robot coordinate packet format

CHK(B0)	CMD(B1)	SrcAddrH(B2)	SrcAddrL(B3)	SNH(B4)	SHL(B5)	TotalH(B6)	TotalL(B7)
XH(B8)	XL(B9)	YH(B10)	YL(B11)	ASign(B12)	AH(B13)	AL(B14)	

- CMD = 13
- B0 ~ B7, refer to 7.3.1.1.

- B8~B11, the coordinates of the robot.
- B12 ~ B14, the direction of robot. The angular value is multiplied by 100 before transferring to avoid float numbers. B12 is 0 if the angular value is negative, 2 otherwise.

7.3.4.12 CMD15: Control Points Display Switch

This packet is used to switch on or off the control points display option.

Table 7.28 Control points display switch packet format

CHK(B0)	CMD(B1)	switch(B2)
---------	---------	------------

- CMD = 15
- B0 ~ B1, refer to 7.3.1.1.
- B2, 0 disable and 1 enable.

7.3.4.13 CMD16: Robot Control Mode

This packet is designed for debug purpose. Operator can choose whether the robot is controlled continuously by the vision sensor or is controlled only when the operator instructs it to do so.

Table 7.29 Robot control mode packet format

CHK(B0)	CMD(B1)	mode(B2)
---------	---------	----------

- CMD = 16
- B0 ~ B1, refer to 7.3.1.1.
- B2, 1 continuously being controlled, 0 being controlled once when the operator give instructions.

7.4 Software Implementation

To build the intelligent environment to aid the mobile robot navigation, user software applications are implemented to support the hardware modules and realise the algorithms. This section first gives an overview of the software organisations. Details of each of the applications and their function modules are then discussed. The user applications are written either by C/C++ or Java programming.

7.4.1 Overview of the Software Structure

Figure 7.34 illustrates the software structure in the proposed intelligent environment supported by the iMote2 vision sensors based on the hardware modules proposed in Section 7.2.1 . The function modules proposed in Figure 3.2 about the system architecture are realised in this section though they may be merged or split to allow flexible implementations. The standard system software modules and libraries are indicated by white blocks, including Linux kernel, C/C++ libraries, Java Virtual Machine (JVM), IEEE 802.15.4 and the OV7620 drivers. The user applications based on these standard modules are highlighted either in green, contributed by the thesis, or marked in brown, partially written by the author. These applications numbered from 1 to 7 are located and run in the remote console, the iMote2 main board, the iMote2 vision sensors, the Gainz IEEE 802.15.4 node and the mobile robot respectively. Different types of communication protocols used between the software applications are depicted in yellow hexagons. Specifically, a TCP/IP socket connection over a USB cable is used between the remote console and the iMote2

main board, which is used as an IEEE 802.15.4 wireless bridge; the Recommended Standard 232 (RS-232) is used between the Gainz node and the mobile robot to exchange data; and the IEEE 802.15.4 protocol links all iMote2 vision sensors as well as the Gainz node.

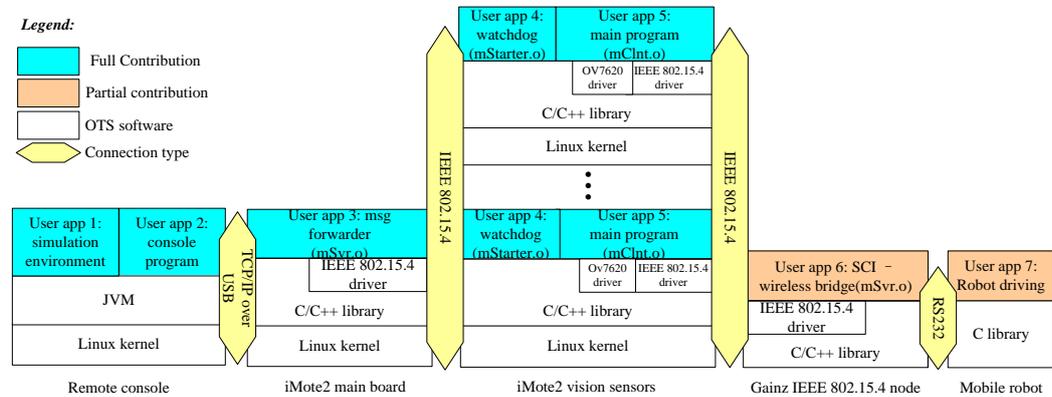


Figure 7.34 Software structures and the contributions

User application 1 and 2: Both of these two applications are programmed in java, run on JVM in the remote console. Application 1 is a pure simulation environment to simulate the snake forces, trajectory generation and robot trajectory tracking. It provides a Graphic User Interface (GUI) to display the simulation results and can be used to fine tune the algorithm parameters but it has no connections with the iMote2 sensors or the mobile robot. On the other hand, application 2 is an application to configure the intelligent environment and monitor mosaic eyes without implementing any navigation functions. It is used to download files required by the vision sensors and to receive and display the results from them. It communicates with the vision sensors by creating a TCP socket listening on port 4001 to connect to the IEEE 802.15.4 module, i.e. the iMote2 main board, which will talk to the vision sensors wirelessly.

User application 3: This program is written in C/C++ and runs on the Linux kernel in an iMote2 main board. The purpose of this application is on the one hand to encapsulate messages sent by the vision sensors into IP packets and forward them to the remote console, and on the other to analyse the IP packets from the remote console to obtain the destination addresses and sends them to the specific vision sensors.

User application 4 and 5: These are the most important programs in building the intelligent environment. They are C/C++ programs run on the embedded Linux kernel in vision sensors. All robot navigation related functions are implemented in application 5 (the iMote2 main program). If an executive file is running, it cannot re-launch itself by replacing the old file. Thus application 4 is created to fulfil this task in case the main program (App 5) is required to be replaced by a new one.

User application 6: This is a lightweight version of IEEE 802.15.4 protocol stacks realisation purely written in C/C++ libraries without any operation systems running. It converts packets from IEEE 802.15.4 to RS-232 bit streams and vice versa.

User application 7: Written in standard C libraries, this program is responsible for receiving, analysing and executing the instructions from vision sensors. It also takes care of PWM signals for controlling DC driving motors and the steering servo motor. Speed feedback control is realised by comparing the signals from the tachogenerator and the desired velocity values.

In the following sections, details of the application implementations will be discussed one after another. Rather than discussing the applications in sequence order, applications 4 and 5 are intended to be described first because they are the key modules in the intelligent environment system and they form the foundation to explain the other applications. This will be followed by a description on remote console application 2, which has the most interactions with the iMote2 vision sensors. Application 3 is the communication forwarder between application 2 and 5, so it is discussed after that. With applications 6 and 7, the complete mosaic eyes intelligent environment is achieved and they are introduced thereafter. Application 1 is presented at last for it is a separate, independent software purely for simulation purpose.

7.4.2 iMote2 Vision Sensor Software Implementation (App 4 and App 5)

The software implemented in iMote2 vision sensors consists of the main program (App 5) and an assistant one (App 4). App 4 is responsible for starting/restarting App 5. Other than that, all functions in the vision sensors are implemented in App 5.

7.4.2.1 Application 4 Software Implementation

The *msgsnd()* and *msgget()* Linux system calls are used, respectively, to send messages to and receive messages from a Linux system Inter Process Communication (IPC) message queue to realise the inter-process communication between App 4 and App 5 so that App 4 can restart App 5 on demand.

The interaction is shown in Figure 7.35. When an iMote2 vision sensor is powered on, App 4 is started along with the embedded Linux kernel startup. Then App 4 launches App 5 by calling Linux system call: *system* (“./mCInt.o”) and enters into a waiting loop for message #4666 which is a user defined IPC message identifier used to report event from App 5 to App 4. If App 5 has completed one file downloading and the file is the *mCInt.o* executive file or it has received an application restart signal from the remote console, the message #4666 will be sent to the system message queue. This message will break the message waiting loop in App 4 and initiate another waiting loop for a certain amount of time (5 seconds) for the current App 5 to clear up and exit. After that, App 4 starts a new App 5 by launching the new *mCInt.o* executive file.

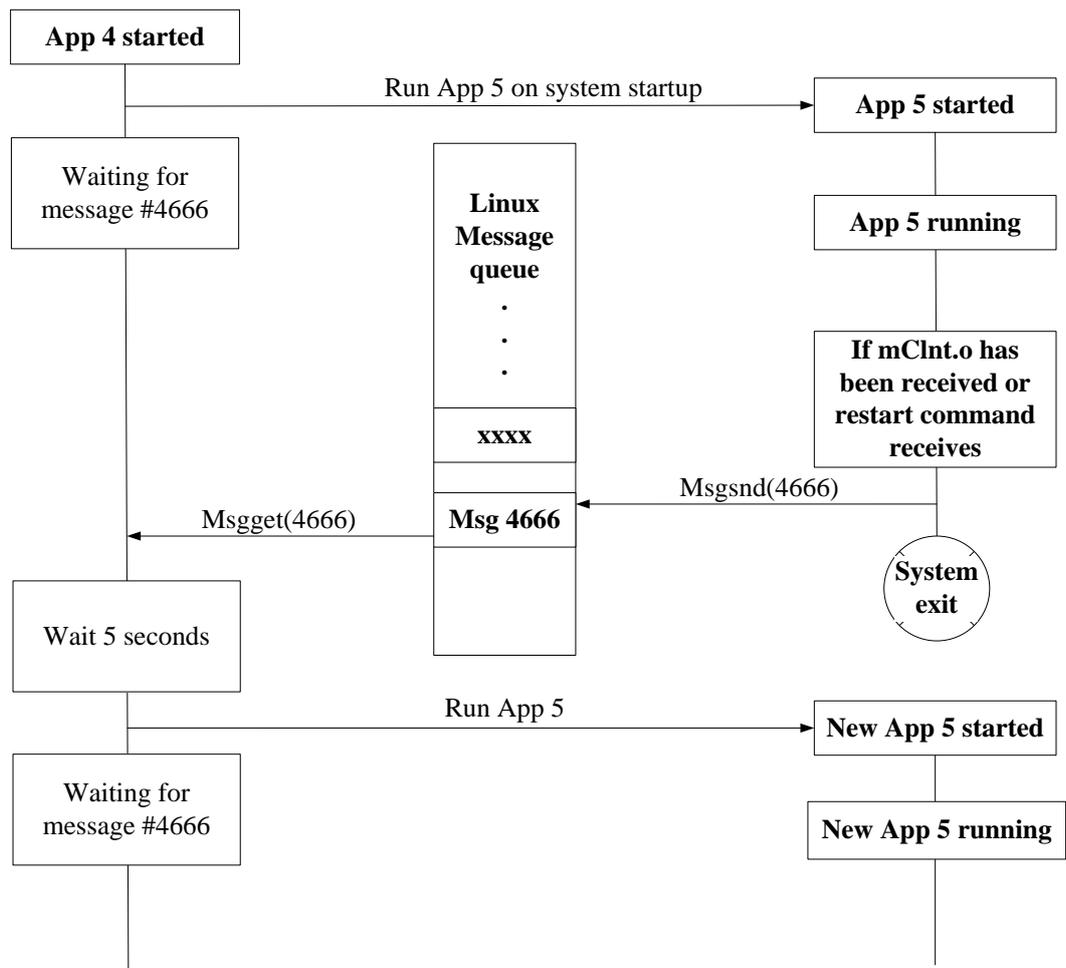


Figure 7.35 Interactions between App 4 and App 5

7.4.2.2 Application 5 Software Implementation

Figure 7.36 illustrates an overview of all the threads involved in the application implementation. Thread 1 is the main loop containing system initialisation, image acquisition and processing, R-snake planning, A-snake generating, communication and token control. Thread 2 is dedicated for receiving packets, assembling them in order, distributing them to related functions for further processing. It is blocked until a new message is arrived. Thread 3 to thread 7 represent the five timers created to assist thread 1 and thread 2 to fulfil timing related tasks, such as the monitoring lifetime of a command and etc..

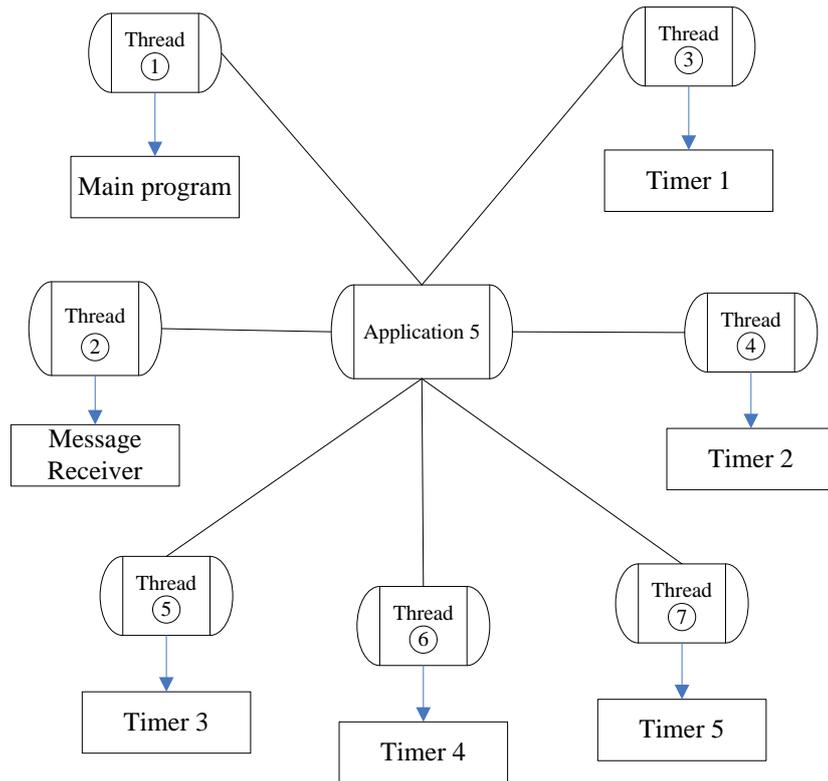


Figure 7.36 Token command concurrent threads

Three sub classes (A-snake, R-snake and Image) are constructed to support the main class *VisionSensor*. Data types defined in the sub classes can be used in the main class. The data structure TOS_Msg is only defined in Main *VisionSensor* because all communications are handled directly or indirectly within this class.

In Class *Image*, CameraParam contains all setting parameters required by the OV7620 vision driver. Images are captured and processed in this sub class to provide the robot and obstacle positions. The robot is represented by its coordinate, orientation as well as velocity and rotational speed while obstacles are defined by their coordinates. Colour structure is used to hold target colours to find interested blobs.

Route and Obstacles structures are used by the *R-snake* sub class to re-initiate and deform the snake. By exchanging information of control points and overlapping areas, the R-snake interacts with other vision sensors to achieve the global goal.

Based on the R-snake information and the robot position, the *A-snake* sub class will generate the final trajectory and control commands to control the mobile robot.

More details of data structures used in App 5 are shown in Figure 7.37.

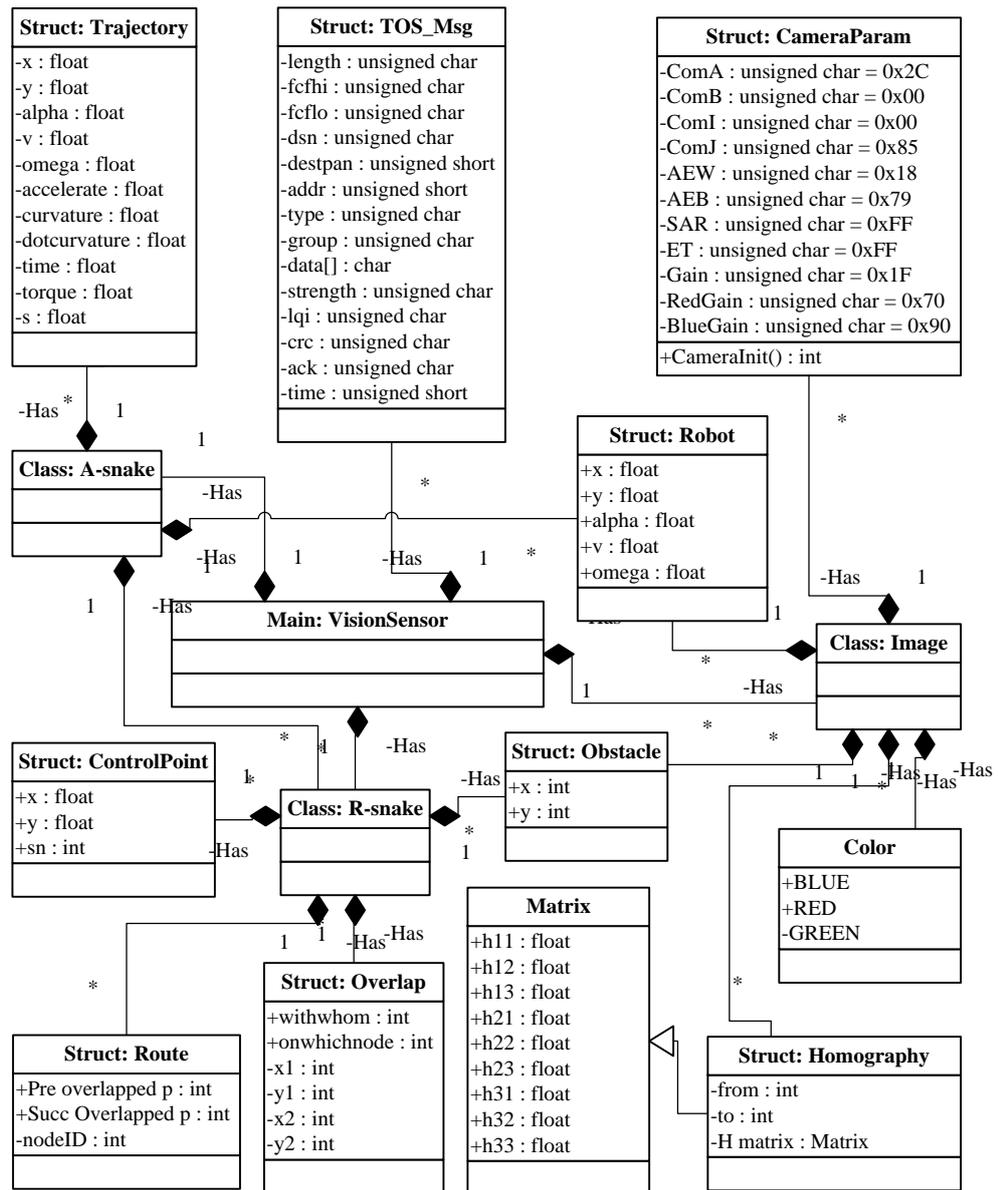


Figure 7.37 Classes and structures defined in App 5

The flow chart of main program is illustrated in Figure 7.38.

At initialisation, instances of *Image*, *R-snake* and *A-snake* subclasses are created; memories for global variables, image buffers, data structures are allocated; message receiving thread and timers are initialised; the camera is opened and a self testing is performed, the background image is calculated by sampling the 16 frames after self testing; system profile,

parameters and system setting files are read to set default values of the variables; the R-snake is initiated by the default parameters. If any errors happen, error code will be reported to the remote console for operator intervention, otherwise, the main loop starts.

The first thing in the main loop is to check whether the background re-sample signal is received. If re-sampling is set by the remote console, 16 frames will be captured continuously and the new background is calculated and stored in the buffer. This is followed by the image acquisition and processing process to be discussed in Section 7.4.2.2.1. In the case when the algorithm for the detection of obstacles by multiple cameras (described in Section 4.5.5.2), is involved, obstacles information obtained in the image processing will be transferred to a dominant vision sensor for fusion.

With all coordinates and route information available up to this step, the R-snake process, to be discussed in Section 7.4.2.2.2, will plan an obstacle free optimised reference snake from the start point to the target. As discussed in Section 7.3.2.2, if a preceding vision sensor is present, border control points will be sent to that sensor once the R-snake is planned.

To reduce the power consumption, if the mobile robot can neither be observed nor predicted, that is the robot is not in view, the main program will skip the trajectory generating process (to be discussed in Section 7.4.2.2.3) and go to the token control process directly. Otherwise the generated A-snake will be sent to the remote console for display before proceeding to the token control process. The token control process

is responsible for negotiating a dominant vision sensor to control robot.

This will be discussed in detail in Section 7.4.2.2.4.

Finally, the control points or images will be sent to the remote console for display if it is requested by the operator.

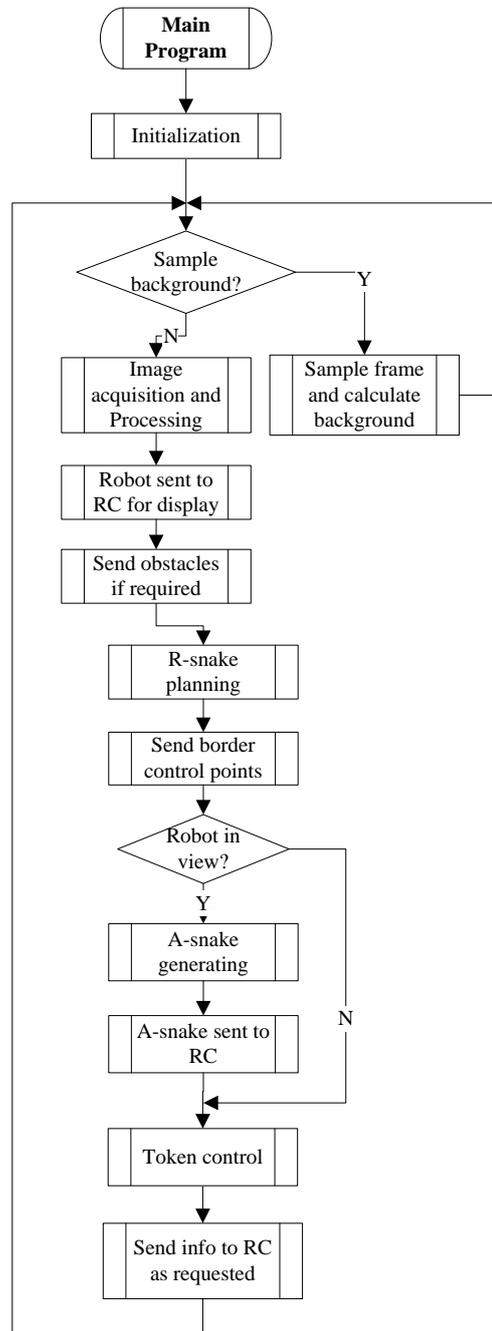


Figure 7.38 Main program flow chart

An example of the R-snake with 6 control points sent to the remote console is shown in Figure 7.39 with the packet format defined in 7.3.4.6.

```
length=33, fcfhi=1, fcflo=8, dsn=238, destpan=65535
dest addr=0, type id=0, group id=0
strength=161, lqi=186, crc=0, ack=0, time=0
data are:
29 7 0 40 0 1 0 1 6 0 104 0 -25 0 111 0 -101 0 123 0 102 0 -90 0 80 0 -23 0 80 1
```

Figure 7.39 Snake with 6 control points

7.4.2.2.1 Image Acquisition and Processing

The online image processing algorithms proposed in Section 4.3.4 are implemented in this process. The processing flowchart is shown in Figure 7.40. After capturing one frame and storing it in the allocated buffer, a subtraction is carried out between this frame and the stored background image to get the dynamic foreground image. Based on this foreground image with static information being removed, binary image is created and a run-length encoding technique is employed in order to locate the robot and the obstacles.

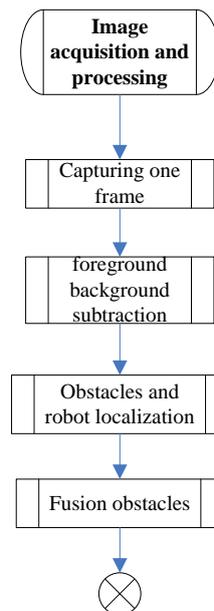


Figure 7.40 Image acquisition and processing

7.4.2.2.2 R-snake Process

Figure 7.41 illustrates the R-snake processing flows. Re-initialisation procedure can be activated by two conditions: a new route is input by an operator or a snake experiences a broken status. A Dijkstra searching on the vision sensor image will be carried out if the re-initialisation flag is set. However, there is no guarantee that the search will be successful in the first time due to possible unexpected circumstances such as obstacles blocking the whole path. If this happens, the search will continue once obstacles information is updated. The data exchange between the main program thread and the message receiving thread in one vision sensor are achieved by the shared buffers. Meanwhile, the collaborations between different vision sensors are achieved by forces reactions. Hence a snake in the main program thread must check whether the border control points sent by its neighbouring sensors have been written in the shared buffers by the receiving thread. As the control points are projected to the workspace plane by the senders, the main program can fuse them with its own control points by comparing the coordinates. After that, internal and external forces are calculated by equations (5.6) (5.7), (5.11)(5.12) and (5.17)(5.18) in Section 5.2 to reshape the snake. During forces calculation, constraints thresholds are checked by equations proposed in Section 5.3.2 so that the snake's status can be categorised for next processing loop usage.

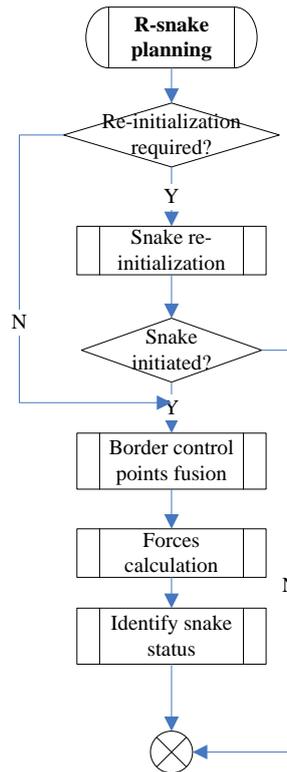


Figure 7.41 R-snake process

7.4.2.2.3 Trajectory Generating Process

Algorithms for generating mobile robot motion control commands proposed in Chapter 6 are used during this implementation. The flowchart is shown in Figure 7.42.

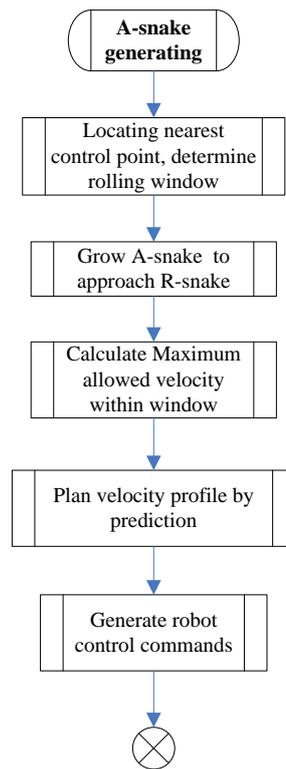


Figure 7.42 Trajectory generating process

First of all, the generating process needs to determine where to start the A-snake planning and how far it will go. The A-snake can be seen as a bridge to soft-land the robot from its current position and orientation towards the R-snake and eventually reaches the goal. For a finer A-snake design, an interpolation process is carried out between the control points. Then, the nearest control point to the robot current position in Euclidian space is selected as the A-snake start point. The length of rolling window is specified by experimental value. However the actual length of the rolling window may vary depending on the number of available control points ahead to the goal. If it is very close to the goal, the actual length of rolling window could be less than the given value. The pseudo codes depicts the process.

```

if ( index_cp + window_length < num_cp ){
    rolling window begins at index_cp with length num_cp
}
else if ( index_cp < num_cp ){
    rolling window begins at index_cp with length num_cp - index_cp
}
else ( index_cp >= num_cp ){
    robot reaches the goal
}

```

Once the start point is determined, A-snake will grow by employing Equation (6.42). As a result, a trajectory with $\{x, y, \theta, k, \partial k / \partial s, s\}$ can be achieved in this step.

The maximum allowed velocity is calculated by equation (6.15). Three factors are considered to obtain the maximum allowed velocity: maximum driving speed, saturated torque limitation and driving without slippage. They are calculated as following,

```

for ( each control point in the rolling window ){
    maximum driving velocity, v1
    calculate maximum velocity limited by saturated torque, v2
    calculate maximum velocity limited without slippage, v3
    vmax = min(v1,v2,v3)
}

```

After above preparation procedures, the expected velocity profile can be squeezed by employing the method in Section 6.3.2. The corresponding pseudo codes are as following,

```

fill in arc length values in array s
set robot current speed as the start speed and 0 as the end speed in the rolling window

while ( more segments to plan ){
    pop up the first segment and remove it from the segments list
    set i as start of the segment and j as the end of the segment
    while ( s[i] < s[j] ){
        find minimum velocity vmim within this section (from i+1 to j-1)
        mark the index number as k
        if ( v[i] <= v[j] ){

```

```

set the expected velocity at i+1 caculated by the given acceleration
mark the index number as k
if (expected velocity >= vmin ){
    if(vmin happens at i+1){
        set vmin as the velocity at i+1
    }
    else{
        set i+1 velocity to its previous value
        create two new segments (i,k) and (k, j)
    }
}
else{
set the expected velocity at j-1 calculated by the given deceleration
mark the index number as k
if (expected velocity >= vmin){
    if( vmin happens at j-1 ){
        set vmin as the velocity at j-1
    }
    else{
        set j-1 velocity to its previous value
        create two new segments (i,k) and (k,j)
    }
} } } }

```

Base on the velocity profile, the approximate arrival time at each control point can be calculated by Newton's first law,

$$time = \frac{2 \cdot (s(i) - s(i-1))}{(v(i) + v(i-1))} \quad (7.9)$$

where $s(i)$ and $v(i)$ are the arc length and velocity at control point i respectively. By (6.10), (6.11) and (6.12), other parameters such as driving forces, driving torques, rotational speeds can be obtained accordingly to form a complete time series trajectory defined in Figure 7.37. One example of the generated trajectory is shown in Table 7.30 with 15 steps predicted. From the trajectory, one can see that robot has achieved the maximum driving velocity at (0.8m/s) and the total expected travel time is 0.56 seconds.

Table 7.30 One generated trajectory

step	x	y	alpha	v	omega	acc	cur	.cur	toque	time	thedad	maxallowedv	s
1:	1.45	1.51	1.72	0.76	-0.00	0.97	-0.00	200.00	116.29	0.14	1.75	0.76	0.07
2:	1.44	1.54	1.69	0.80	-5.18	0.00	-6.48	200.00	128.00	0.18	1.76	0.80	0.10
3:	1.44	1.58	1.63	0.80	0.00	0.00	0.01	-200.00	-128.00	0.22	1.77	0.80	0.13
4:	1.44	1.61	1.65	0.80	5.19	0.00	6.49	-200.00	-128.00	0.26	1.76	0.80	0.16
5:	1.43	1.64	1.72	0.80	-0.00	0.00	-0.00	200.00	128.00	0.30	1.75	0.80	0.19
6:	1.43	1.67	1.69	0.80	-5.20	0.00	-6.50	200.00	128.00	0.34	1.76	0.80	0.23
7:	1.43	1.71	1.63	0.80	-0.00	0.00	-0.00	-200.00	-128.00	0.38	1.75	0.80	0.26
8:	1.43	1.74	1.65	0.80	5.19	0.00	6.49	-200.00	-128.00	0.42	1.74	0.80	0.29
9:	1.42	1.77	1.72	0.80	-0.00	0.00	-0.00	200.00	128.00	0.46	1.74	0.80	0.32
10:	1.42	1.80	1.69	0.80	-5.20	0.00	-6.50	200.00	128.00	0.50	1.74	0.80	0.36
11:	1.42	1.83	1.63	0.80	-0.00	0.00	-0.00	-200.00	-128.00	0.54	1.74	0.80	0.39
12:	1.41	1.87	1.65	0.80	5.19	0.00	6.49	-200.00	-128.00	0.58	1.76	0.80	0.42
13:	1.41	1.90	1.72	0.80	-0.00	0.00	-0.00	200.00	128.00	0.62	1.76	0.80	0.45
14:	1.40	1.93	1.69	0.80	-5.42	0.00	-6.77	200.00	128.00	0.66	1.79	0.80	0.49
15:	1.40	1.97	1.62	0.80	-0.00	-0.85	-0.00	-200.00	-128.00	0.70	1.79	0.80	0.52

The velocity profiles are encapsulated into robot motion instructions following a successful trajectory planning. The data format is shown in Table 7.11. Figure 7.43 shows one of the commands sent by node-40. As seen in the figure, there are 20 steps altogether.

```
length=89, fcfhi=1, fcflo=8, dsn=235, destpan=65535
dest addr=101, type id=0, group id=0
strenght=161, lqi=186, crc=0, ack=0, time=0
data are:
-6 11 0 40 0 1 0 1 20 8 37 1 0 4 75 2 35 4 80 2 35 4 80 2 35 4 80 2 35 4 80 2 35
35 4 80 2 35 4 80 2 35 4 80 2 35 4 80 2 35 4 76 2 35 9 0 1 0
```

Figure 7.43 Command sent to robot

7.4.2.2.4 Token Control Implementation

Token control is responsible for checking system status, such as whether it has a good view of the robot, whether the robot control token has been occupied and etc., in order to initiate different procedures to handle the token. It is responsible for invoking token handover or request session. It is called periodically by the main program. To fulfil the token control, four timers, Timers 1 to 4, are created to assist tracking the timeliness of the commands. Message response functions are

implemented in Thread 2, the message receiver thread, dealing with all CMD14 related packets.

7.4.2.2.4.1 Four Timers Implementation

Timers can be created, cancelled or resumed by the main processes. Timer 1, Timer 2 and Timer 3 will automatically take a certain kind of specified actions and exit once their lifetimes expire, while Timer 4 is a cyclic routine running until the end of the program. Details of the timers are shown in Figure 7.44.

If the dominant vision sensor does not have a good view or the robot gradually disappears from its view for some reasons, the token handover request message will be broadcasted. Followed by this, Timer 1 (a 100ms timer) is started and keeps an eye on the handover reply message. If token request reply message is received before Timer 1 times out, the main program will stop timer 1 and further handover procedures will be processed; otherwise a robot control instruction is sent out to stop the robot due to very poor view.

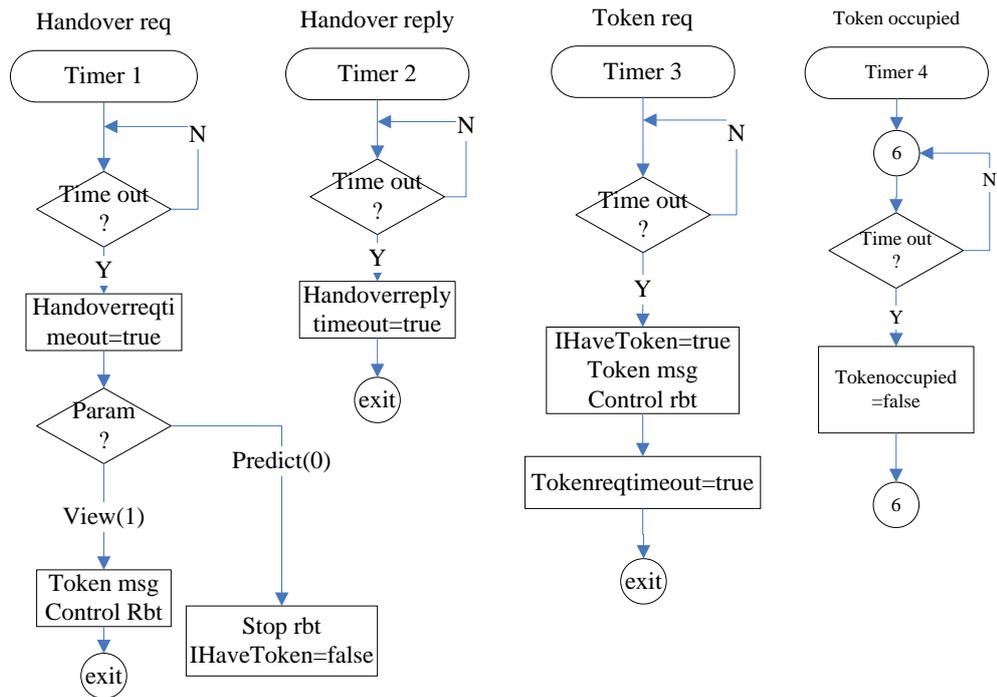


Figure 7.44 Four timers flow charts

Timer 2 will immediately start once a handover reply message is sent. It waits for the receipt of handover confirmation message until the time out limit of 50ms is reached. In case no confirmation message is received during its lifetime, the timer will inform this to the main program to terminate the token acquiring procedure.

When a vision sensor tries to compete for the control token, it will initiate a Timer 3 to allow 50ms to check conflicts. If competition is successful, it will broadcast a token occupy message to notice other vision sensors its ownership of the token and will do this periodically as long as it keeps the ownership.

Timer 4 continuously detects whether the broadcasted token occupy message has been received within 600ms from the time it received the last broadcast message. It will do nothing if token occupy message is

received; otherwise it will inform the main program to compete for the token since no one owns it anymore.

7.4.2.2.4.2 Token Control Process

It checks periodically whether it has the control token by checking the global *IHaveToken* variable which has been set once the vision sensor obtains the token or been cleared when the token is lost. In the case that a token is not on hold, it will initiate a token request procedure and reset Timer 3 if the robot has been observed and the control token is not occupied by others, otherwise the process will go to the beginning of the thread. In the case of a token has been taken, it will send control commands to the robot if the robot is still present with a good view. However, if the robot is not in a good view area, a handover request command will be sent out to check whether the other vision sensors have better observations. Timer 1 is reset accordingly. One situation may happen, that is the vision sensor has a token but cannot see the robot in its view, due for example to the robot high moving speed. In this case, if predictive values are available, a normal handover request is sent otherwise a very urgent handover request is initiated. The detailed flow chart is shown in Figure 7.45.

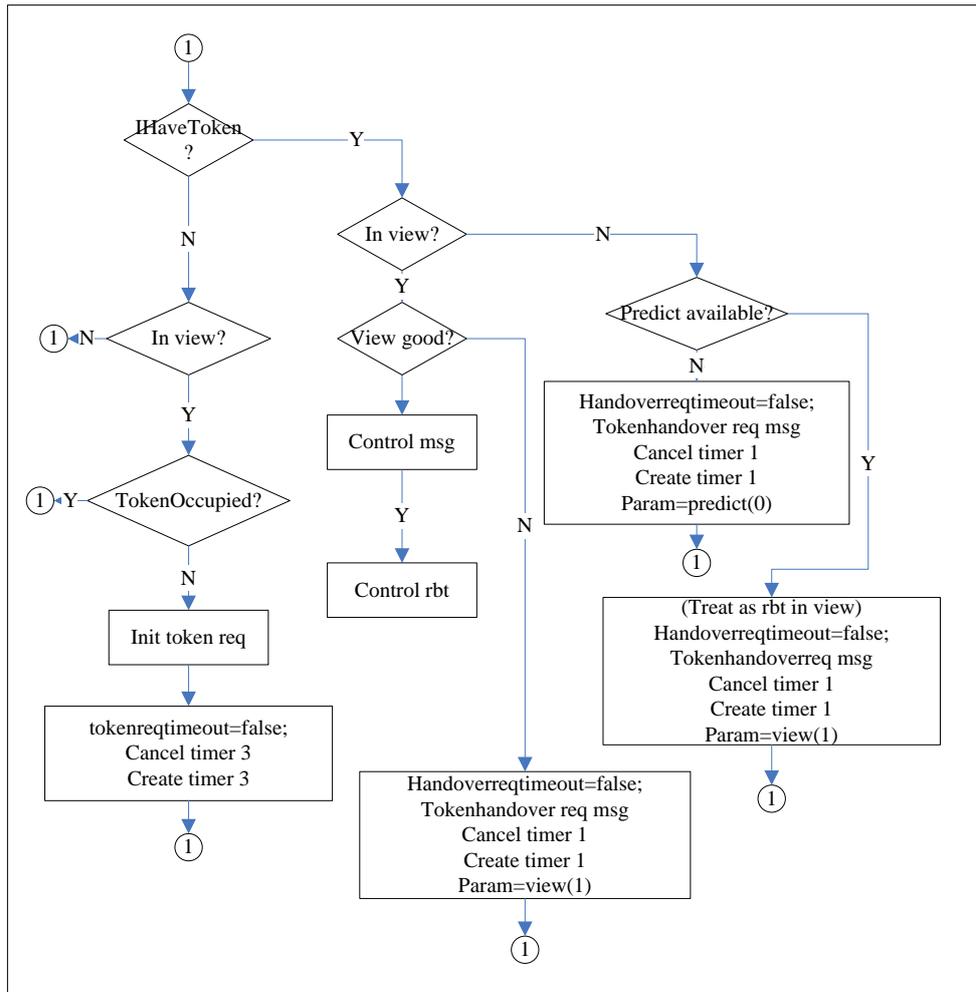


Figure 7.45 Main token process thread

7.4.2.2.4.3 CMD14 Message Response Functions Implementation

The responded messages include occupy token message, token request message, token reply message, handover request message, handover reply message and handover confirmation message.

- Occupy token message

Upon receipt of this message, Timer 4 is restarted and the token is assumed to be occupied by other vision sensors. The process is shown in Figure 7.46.

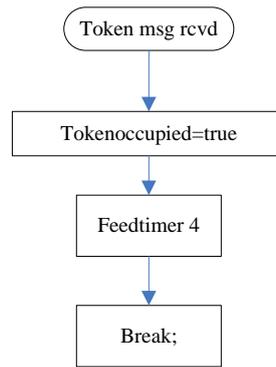


Figure 7.46 Process upon occupy token message receipt

- Token request message

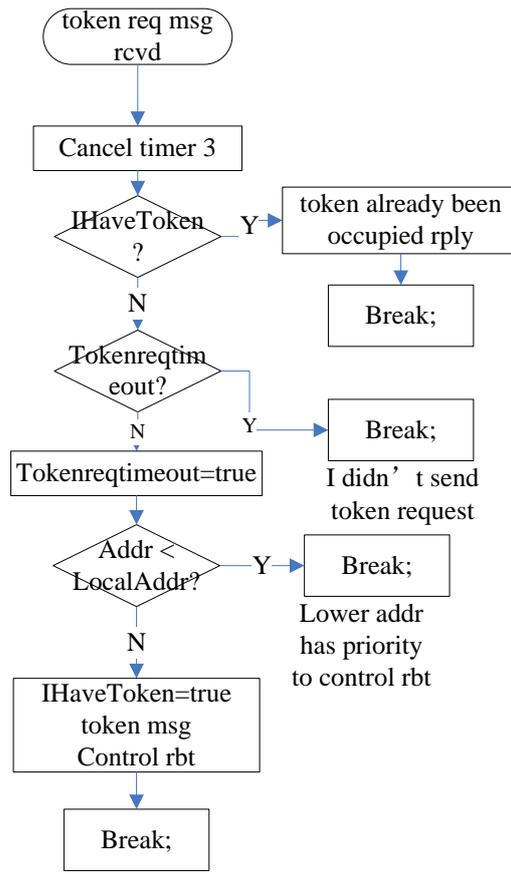


Figure 7.47 Process upon token request message receipt

Receipt of this message indicates that some vision sensors are trying to get hold of the control token. The first thing to do is to cancel Timer 3 in case Timer 3 sets the *IHaveToken* variable wrongly. A token reply message is sent to show it is already owned by the recipient if it is true, otherwise go to further process. Since Timer 3 is cancelled already, the only possibility for a timeout status of the Timer 3 is that it has not yet started. In the case that Timer 3 is not timed out yet, which means that the recipient is also competing for the token, the address values are used to decide whether the token should be taken. If the recipient has the smaller short address value, it will occupy the token, notify others and control the robot. Figure 7.47 shows the details.

- Token reply message

Upon receipt of this message, the main program should stop Timer 3 and release the token no matter what happens because it is clearly indicated that another vision sensor has the token already. It is shown in Figure 7.48.

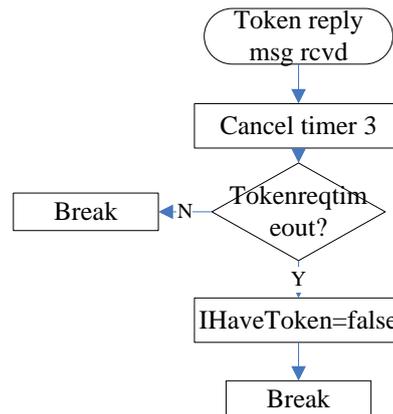


Figure 7.48 Process upon token reply message receipt

- Handover request message

If the receipt has better view, it will send a reply message to the current token owner and start Timer 2 to wait for confirmation, otherwise this message is ignored. The flow chart is shown in Figure 7.49.

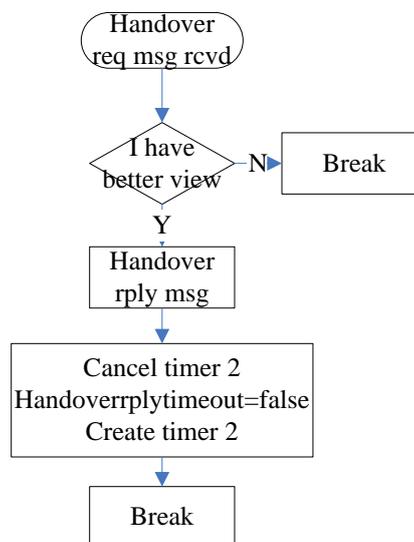


Figure 7.49 Process upon handover request message receipt

- Handover reply message

When this message is received, the token ownership should be released and handover confirmation message should be sent out.

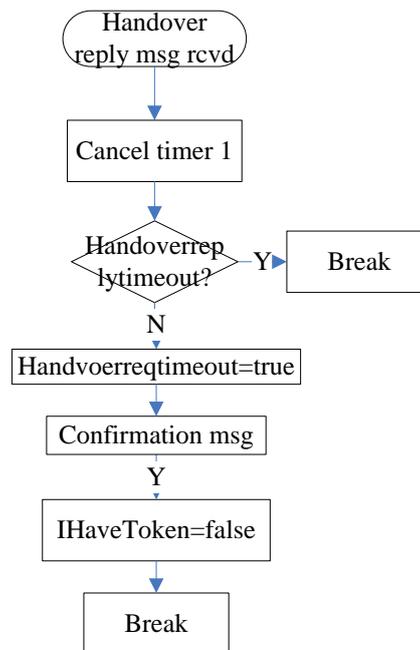


Figure 7.50 Process upon handover reply message receipt

- Handover confirmation message

This message indicates a successful handover has been completed, hence Timer 2 will be stopped, the *IHaveToken* variable will be set, the token occupy message will be broadcasted and the robot control instructions will be sent. This is shown in Figure 7.51.

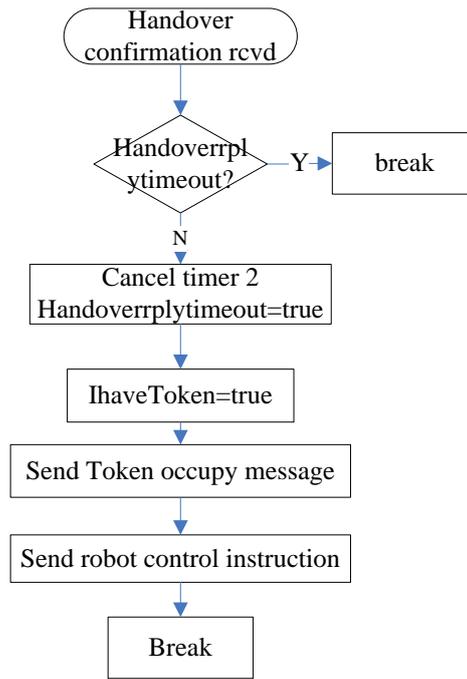


Figure 7.51 Process upon handover confirmation message receipt

7.4.3 Remote Console Implement (App 2)

With the benefit of Java hardware platform independent technology, application 2 can be run on both Windows and Linux based operation systems. According to the software modules proposed in Section 3.3.1, the scope of the App 2 is to monitor, configure and control the intelligent system rather than simulations, which are separately deployed in App 1. Therefore, three classes are implemented to meet the requirement: GUI interfaces providing interactions between the operator and the system, packets receiving and sending and generating and storing files for downloading. One screenshot of the software is shown in Figure 3.3.

7.4.3.1 Remote Console GUI Interfaces

Figure 7.52 is the remote console GUI Interface. It consists of commands area, display area and status area.

7.4.3.1.1 Commands Area

The implemented operations for an operator to interact with the system are listed in Table 7.31.

Table 7.31 Commands list of remote console GUI

Menu	operations
iMote2	Change background image Display image on screen Display blobs on screen Sample colour image and store on iMote2 node Assign correspondent points to calculate mapping matrix Assign overlap areas Associate semantic name with address Download file Reset iMote2 program Enable or disable R-snake display Control robot continuously or by operator request
Communication	Start TCP server listening

7.4.3.1.2 Display Area

The area is divided into four smaller regions. Each region is dedicated to one vision sensor. They are identified by the vision sensor addresses or their semantic names. In Figure 7.52, control points are enabled to be shown, they are represented by sparse gray circles. White obstacles are observed in the vision sensors. The R-snake is forced to bend to avoid the obstacles. The robot and the A-snake are shown in red and green circles respectively.

7.4.3.1.3 Status Area

The remote console system information or the information reported by iMote2 are displayed here, such as camera init error, robot reaches goal and etc.. This information is listed in Table 7.24.

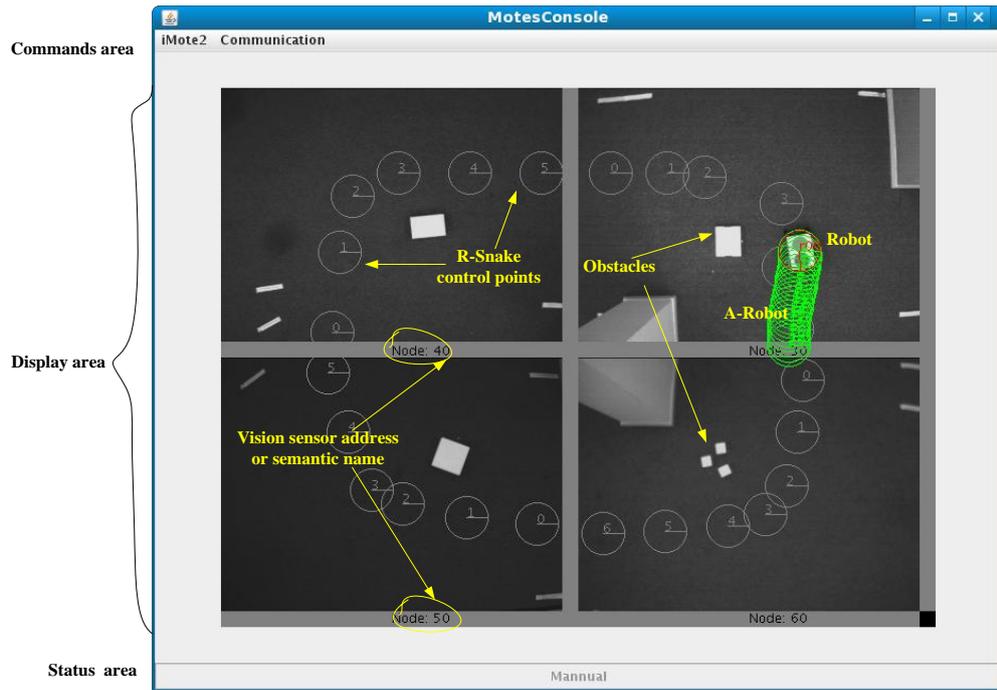


Figure 7.52 Application 2 GUI interface

7.4.3.1.4 The Implementation

All data required for display are prepared by the communication class. The GUI class accesses these data by calling the *get()* functions provided by the communication class listed in Table 7.32.

Table 7.32 Get functions provided by communication class for GUI class

Functions	Descriptions
GetControlPoint(sn)	The R-snake control points in vision sensor <i>sn</i> are returned in an object array
GetRefPoint(sn)	The A-snake control points in vision sensor <i>sn</i> are returned in an object array
GetRobotPosition(sn)	The robot position in vision sensor <i>sn</i> is returned in an object array
GetImage(<i>sn</i> , buffer)	The image in vision sensor <i>sn</i> is stored in the buffer provided by the caller

Then, the object array will be converted to their corresponding data structure as defined Figure 7.37. Finally, the *show()* function will be called by GUI to display them on the screen.

Similarly, operator commands are captured by the GUI class and sent to the communication class by the communication class *set()* functions listed in Table 7.33.

Table 7.33 Set functions provided by communication class for GUI class

Functions	Descriptions
SetCmd(sn, cmd, param)	Vision sensor, command and command parameter are set to communication class
SetSrcFileLocation(path)	The selected file path by operator is set to communication class

7.4.3.2 Remote Console Communication Module Implementation

As defined in Table 7.6, except CMD4, CMD11 and CMD14, the remote console must implement all commands to respond to or to invoke the messages to fulfil a certain kind of data exchange tasks. Figure 7.30 and Figure 7.31 show the signal flows between the remote console and the vision sensors. TCP Socket is established to connect the App 2 and App 3. When an operator choose the “TCP server starts listening” menu from the GUI commands area, remote console creates a server socket listening on the port 4001. If the attached iMote2 main board starts up, its client socket will try to connect the server to establish the connections.

Once the TCP connection is established, the message forwarder will transparently forward messages to the remote console and transfer the messages from the remote console to the vision sensors. The packets forwarded by the message forwarder from different vision sensors are mixed most of the time due to concurrent transmissions from all the vision sensors. Therefore, one message classifier is designed to reassemble messages according to the source addresses. Figure 7.53 illustrates this function. Different colours represent messages from different vision

sensors. When the remote console receives them from the message forwarder, they are mixed together. After being classified by the classifier, messages are separated and stored in different buffers.

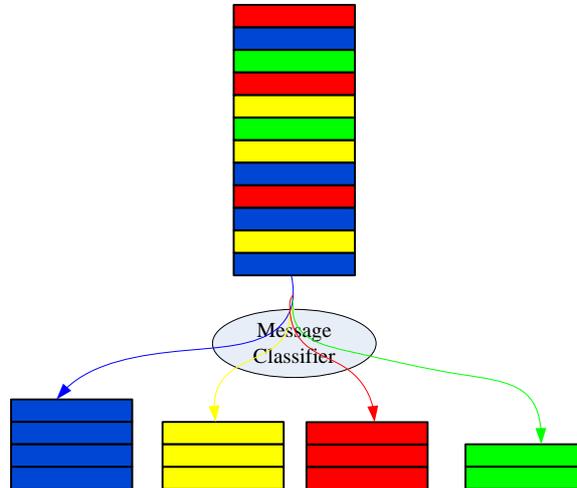


Figure 7.53 Messages classifier

A ready signal is marked on receipt of a complete message. This signal is checked by the GUI periodically to decide whether they need to update the corresponding buffers so as to refresh the display.

7.4.3.3 Profile Files Generating

From the development point of view, all vision sensors should have exactly the same program for batch production. However, due to production variations, the camera physical characteristics may differ. In addition, different vision sensor locations create differences in mapping parameters and semantics and different roles of the vision sensors require different behaviour. Profile files are the keys to solve these contradictions. With different profile files, the same program can act in different ways.

Meta-data are created according to the requirement reported in Section 3.3.1. Figure 7.54 list all data types define in App 2. Colour is used

to define the targets. CameraParams are hardware dependent characteristic parameters. Route is used to represent the topological map. The overlapping areas and mapping matrix are stored in Overlap and Mapping structures.

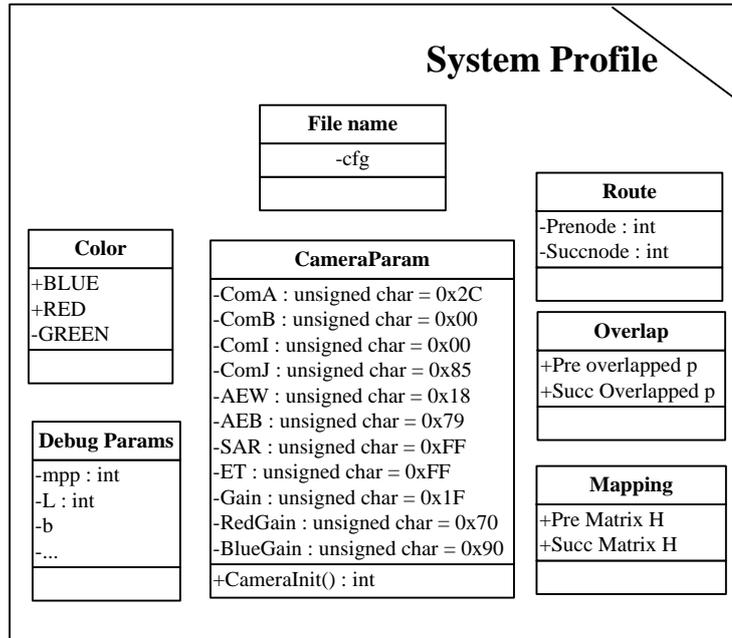


Figure 7.54 System profile meta-data

The usage of the remote console software is quite straightforward. An example of calibrating the mapping matrix between two vision sensors is illustrated as follows.

From multiple view geometry [129], if at least four corresponding points can be found in two view and the points are not in a line, a homography matrix can be obtained. In order to quickly find out the corresponding points, one grid is adopted for calibration, as shown in Figure 7.55.

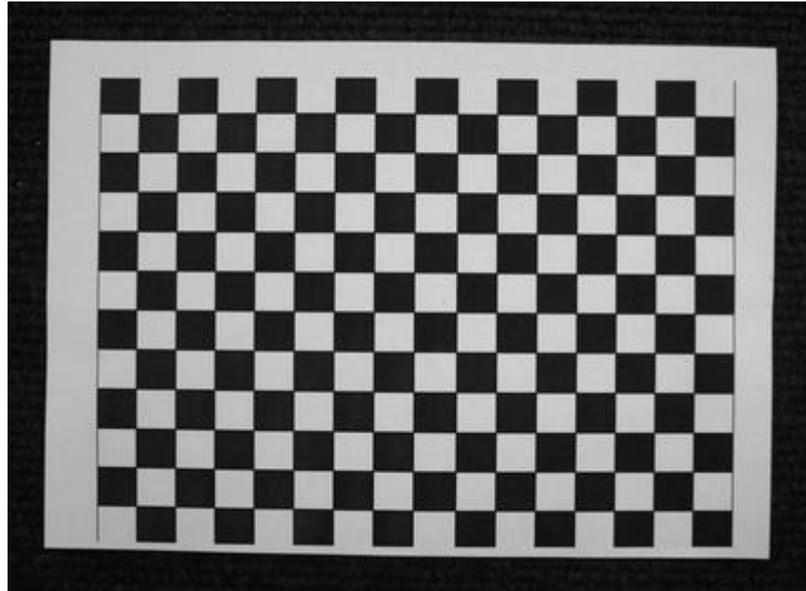


Figure 7.55 Camera calibration grid

After placing the grid in a location where both cameras can observe it, display both images on the screen as shown in Figure 7.56.

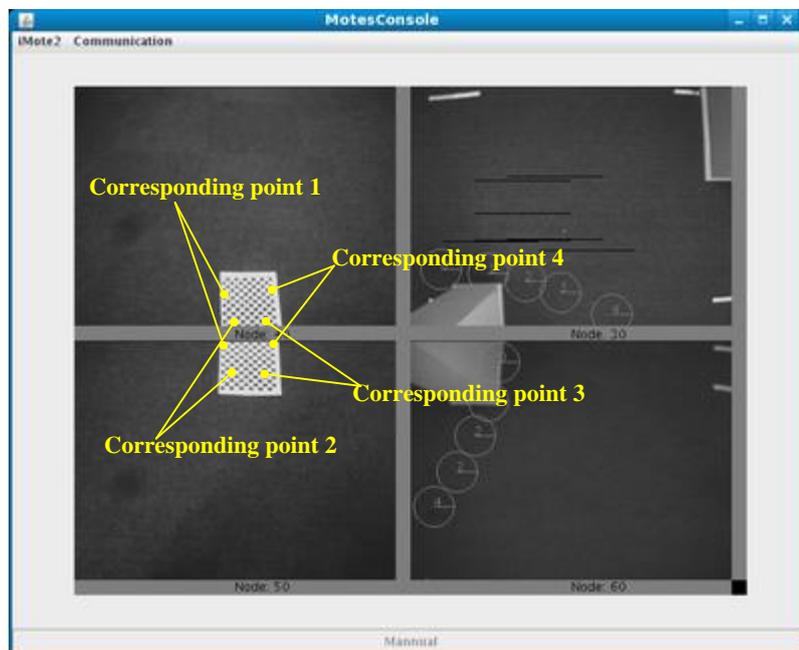


Figure 7.56 Overlap areas in two cameras

Then, from the drop down menu choose iMote2 and then choose the Get Correspondent points as shown in Figure 7.57.

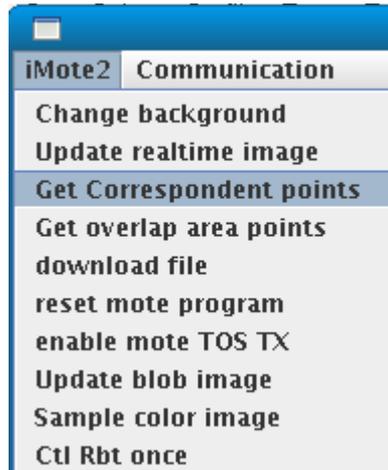


Figure 7.57 Mapping calibration step 1 of 4

Following this operation is the message in Figure 7.58.

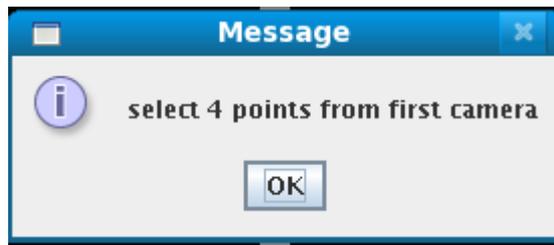


Figure 7.58 Mapping calibration step 2 of 4

After this operation, four points, shown in Figure 7.56, can be seen in both camera views but not in a line should be chosen and click on the screen one by one. After all four points been chosen in one vision sensor, a message shown in Figure 7.59 will remind the operator to choose another four points in the second camera. Four correspondent points should be selected in the same sequence as the first four.

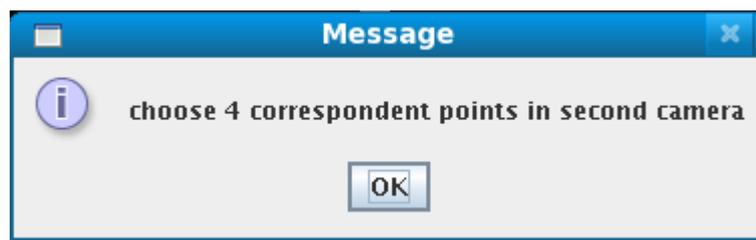


Figure 7.59 Mapping calibration step 3 of 4

By doing this, enough information is collected by the application. Then, the homography matrix is calculated and stored in the disk. Following this is a confirmation of a successful mapping calibration as shown in Figure 7.60.



Figure 7.60 Mapping calibration step 4 of 4

An example of the result mapping matrix is shown as the following,

```
50<-40:1.0030 -0.4251 87.3347 -0.0123 0.7599 -160.6899 0.0000 -0.0024 1.5421  
40<-50:1.0039 0.5696 2.4977 0.0144 1.9826 205.7725 -0.0000 0.0031 0.9729
```

The first line is the mapping matrix from vision sensor (40) to vision sensor (50), and the second line is the mapping from vision sensor (50) to (40).

7.4.4 Remote Console Wireless Peripheral Software Implementation (App 3)

As discussed above, the remote console PC has neither IEEE 802.15.4 hardware radio module nor protocol stack, thus this peripheral behaving as a message forwarder is deployed to construct an IEEE 802.15.4 frame to encapsulate the payload in the IP packet and vice versa.

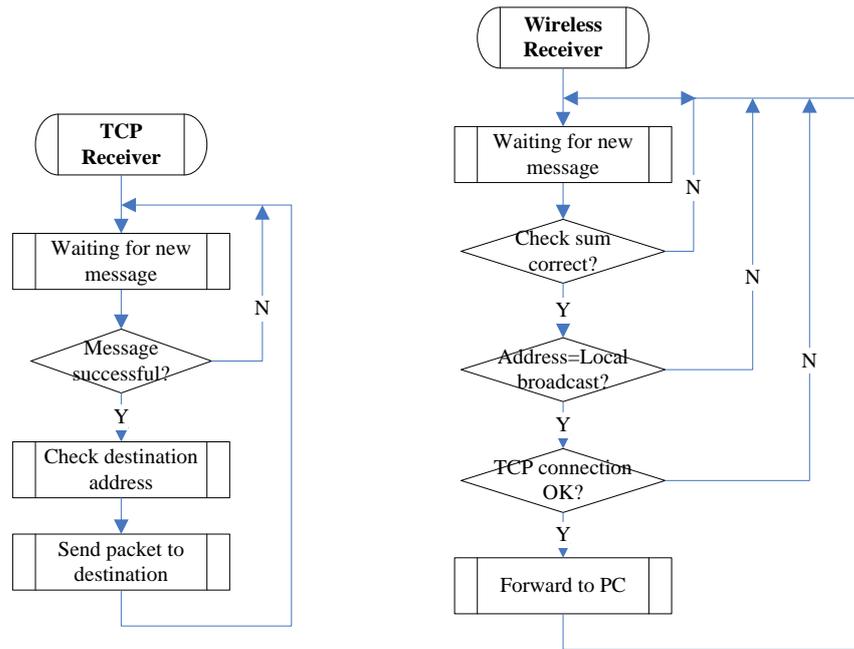


Figure 7.61 Remote console wireless peripheral software processing flow

As a message forwarder, less time delay means better performance. Hence the program is designed to be as simple as possible. The main program is responsible for detecting the existence of the TCP server and will try to establish a TCP connection. The flowcharts for two receiver threads (TCP receiver and IEEE 802.15.4 Receiver) are listed in Figure 7.61.

The TCP receiver is responsible for forwarding messages from the remote console to wireless vision sensors. The blocked thread is activated on receipt of a new message arrival. If the format and checksums are correct, the packet header will be open and the destination address is used to send the packet.

The wireless receiver is in charge of inspecting messages on air. Only the messages sent to server or broadcasted will be forwarded to the remote console, other messages will be filtered out.

7.4.5 Gainz Software Implementation (App 6)

Similar to what has been described in Section 7.4.4 , App 6 implements the message bridging functions between the vision sensors and the robot. There is no operation system running in this hardware. All codes are compiled into a single binary file, then this file is flashed to the processor via JTAG port (Joint Test Action Group, IEEE 1149.1). The programs and the IEEE 802.15.4 protocol stacks are developed by colleagues in Tongji University. However, the source codes have been modified to add additional headers and tails to mark the start and end of the packet in order to adapt it for serial port usage in this PhD work. If the packet is sent to the mobile robot by inspecting the destination address, then the packet will be encapsulated in a serial tunnel and forwarded to the mobile robot. The packet format is listed in Table 7.34. There is a possibility that the header byte happens to be the same as the payload byte, thus two-byte header and two-byte tail are used to reduce the possibility. In general, the possibility will be reduced from 1/256 to 1/65536 with two-byte isolation. With a careful selection of the binary values different from the expected packet data, the error possibility will be reduced further to avoid incorrect serial port bit stream segmentation.

Table 7.34 Gainz packet format

Head0	Head1	Payload	Tail0	Tail1
0x55(B0)	0x50(B1)	iMote2 Packet(B2~B111)	0x50(B112)	0x55(B113)

7.4.6 Mobile Robot Software Implementation (App 7)

As an end actuator, the mobile robot receives command series which are forwarded by the Gainz node from the vision sensor, and puts them in the command execution queue for actuation.

Command series received from vision sensor are velocities or forces with an execution time in milliseconds. In addition to generate a certain kinds of PWM to drive the motors according to the given speed, the mobile robot must have the ability to end one command and start another exactly in the expected time to achieve the desired performance. The timer thread created by software cannot meet the high precision requirement because the actual timing varies due to the CPU load. Therefore, one of the on chip hardware enhanced capture timers is enabled to achieve precision timing. By soft wiring the system clock output to a clock pulse counter with a preset counting value, the counter will increase its value and generate an interrupt once the counter is overflowed. The preset counting value then determines the timing of the timer. With a prescaler, the maximum timing duration can be scaled to a larger value. If the prescaler is 16, then every 16 clock will increase the counter by 1. Given the 24MHz system clock frequency and the prescaler setting with 16, 1 increment in the counter is equal to 0.000000667 seconds, thus $0.001/0.000000667 = 1499$ (0x05db) should be the preset value of the counter to achieve a 1ms timer.

The timer interrupt processing flowchart is shown in Figure 7.62. The first thing to do in the interrupt is to clear the interrupt flag and get it ready for the next interrupt. If new commands are received, they will be inserted

in the execution queue. In the case that no new commands arrive, the status of one command execution will be checked to decide whether to generate a new command for actuation or to continue executing the previous command. If there are no more commands in the queue, the robot will maintain the latest execution command.

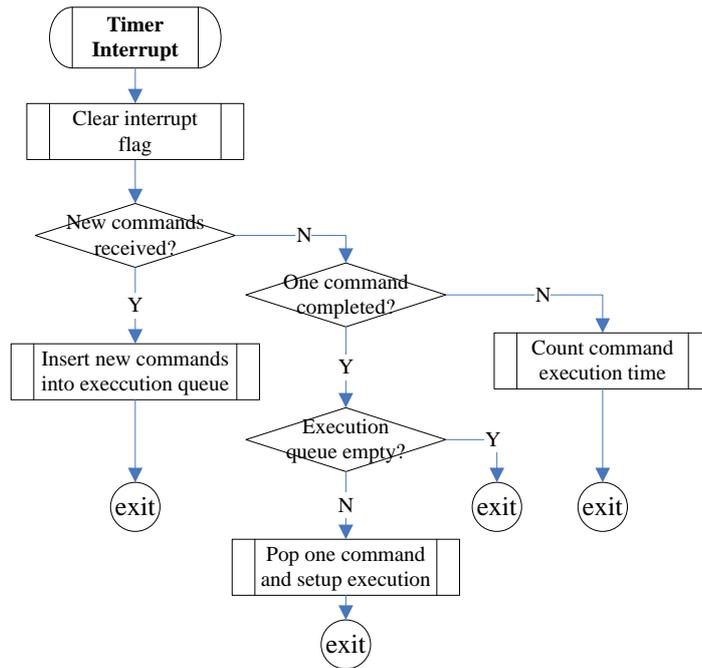


Figure 7.62 Command execution flowchart in mobile robot

7.4.7 Simulation Environment Implementation (App 1)

The simulation environment which has been used in Chapter 5 is deployed for algorithm evaluation and result visualisation. It is a separate application from application 2 to application 7. It is programmed by Java language. The screenshot of the application is shown in Figure 7.63.

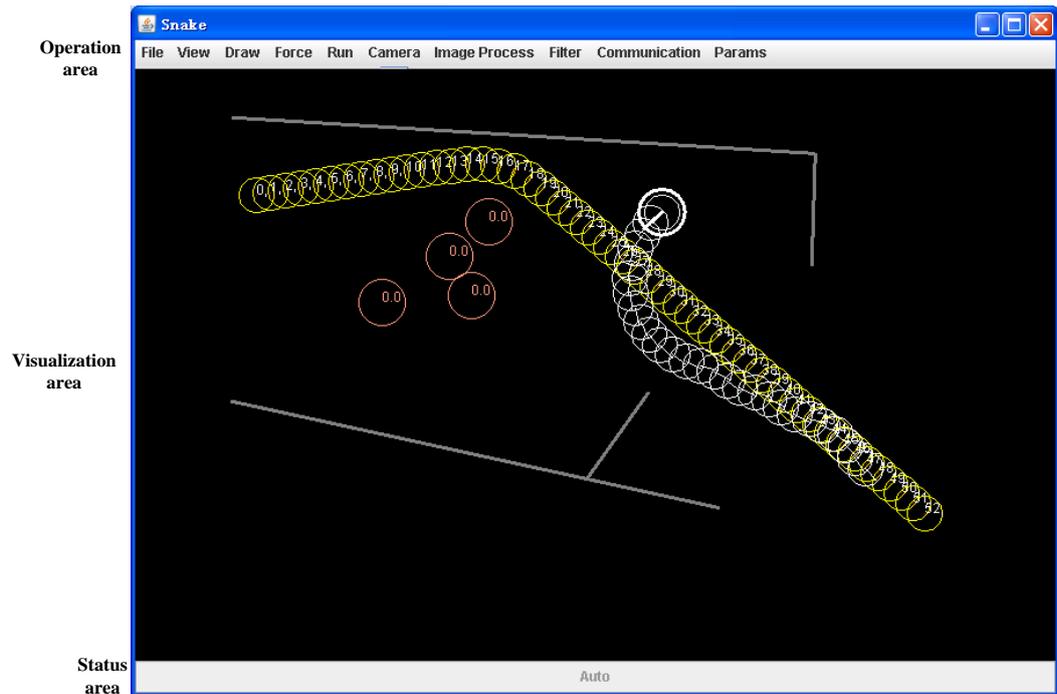


Figure 7.63 Simulation software main interfaces

The main interface includes three areas: operation area, visualisation area and status area.

Simulation options can be made in the operation area. A list of the functions and the main menus is listed in Table 7.35. This area is used to display real time image, snake control points, real time robot location, planned trajectory, obstacles and etc.. One example is shown in Figure 7.63: white lines represent walls, red circles represent obstacles, yellow circles for snake control points, white circles for the A-snake and robots.

Table 7.35 Simulation environment function list

Main menu	Function list
File	Save or load simulation scenarios.
View	Whether to show obstacles or safety area

Main menu	Function list
Draw	Input start and goal so that to initiate a snake place emulate robot set manual obstacles set walls clear all visualisation objects in areas and more
Force	Choose whether to apply following forces: Elastic force Bend force Curvature force Obstacles force Wall repel force
Run	Run automatically or run step by step manually
Camera	Choose cameras Start or stop capturing real time video
Image process	Load or save an image Change background Show background Show real time image Show blob image Setup homography parameters
Filter	Whether to apply following image filters: Blur Sharpen Negative Rotate Mapped image by homography
Comm	Start or stop a connection with other simulation instances for obstacles fusion or other information exchange Run standalone
Params	Whether to show following parameters: Control point serial number Free, rigid or broken status Hop Curvature values Forces values And others

The status areas are dedicated to show the system information such as whether it is run in automatically mode or manual mode.

Algorithms and functions are encapsulated into classes as shown in Figure 7.64.

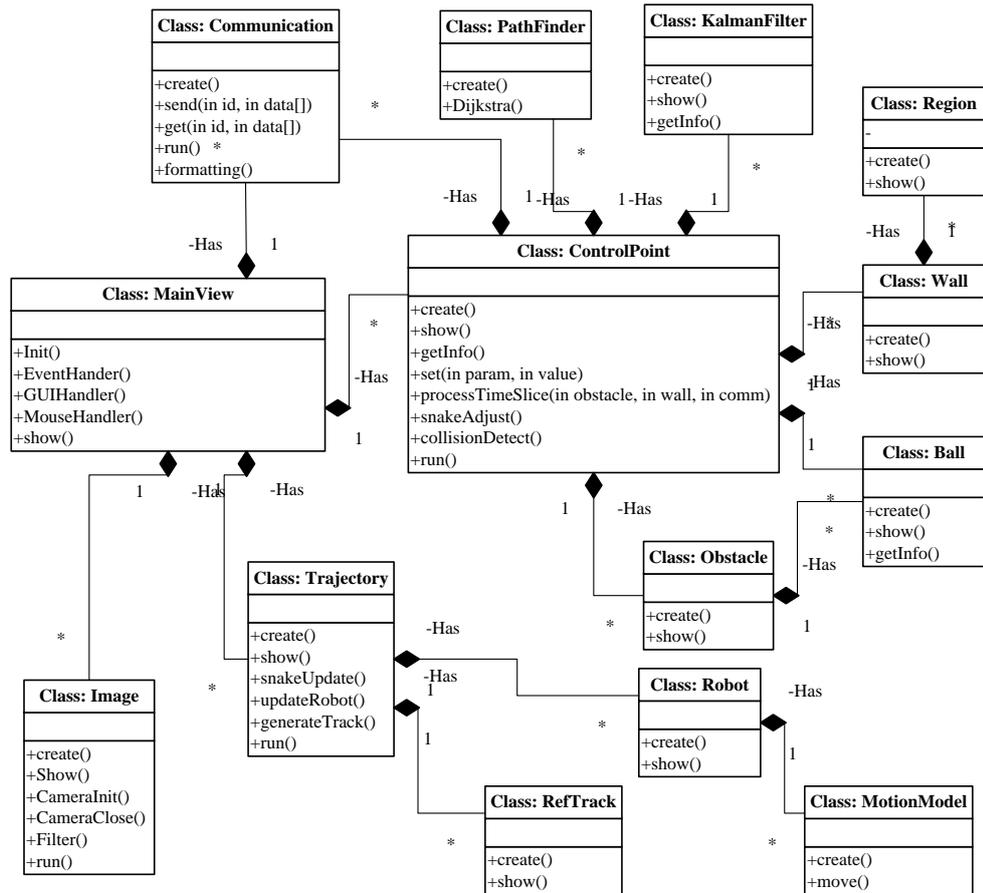


Figure 7.64 Simulation software classes structure

The main class is the Class *MainView*. It provides the main user graphic interfaces, deals with the user input, initialises instances for other object classes, displays the simulation results on screen and reports the system status. *PathFinder* is used to search a path in the image by Dijkstra global search algorithm. Based on this path, *MainView* will create multiple instances of *ControlPoint*.

The Class *ControlPoint* holds all path planning related implementations. One *ControlPoint* instance is designed to emulate a distributed mosaic eye. To simulate a distributed processing environment, the *Communication* class is used to emulate different types of

communication, such as the wireless communication, to connect all *ControlPoint* instances rather than shared memories. So a function *processTimeSlice()* is designed to be called by the *MainView* continuously rather than in a run loop in the *ControlPoint*. A run loop is a cycle thread created within a class to do the process periodically. *KalmanFilter* is used to filter and forecast the obstacles positions detected by *Image* class. Class *Walls* are created as boundaries to define moving space edges. Classes such as *Walls*, *ControlPoint* and *Obstacles*, have their own *show()* functions which will simplify the display process in *MainView*.

Class *Image* is responsible for interactions between applications and cameras. It captures real time image, applies filters on the image, extracts obstacles information and shares all these data with other classes for further processing or displaying.

Class *Communication* can process both data exchanges between control points internally and between this simulation instance with others externally. The primitive operations exposed to other classes are *send()* and *get()*.

The trajectory generating algorithm in Chapter 6 is implemented in Class *Trajectory* which will generate a time series commands for a robot model, Class *Robot* to navigation from robot current location to the goal position.

7.5 Discussion and Evaluation

7.5.1 Processing Time for Software Modules

Figure 7.66 shows two running loops with the recorded time for all the processing modules in one of the vision sensors. Sampling and buffering an image takes around 89ms; distinguishing foreground from background and identifying objects consumes about 146ms, which is the major time used in the whole processing loop; R-snake adjusting utilises 10ms. The time for generating the trajectory is less than 2ms because this processing is ignored due to the absence of the robot. As one can see from the data, the image processing time takes more than half the total processing time, around 260ms.

```
read one frame has run for: 89.203003 ms
get foreground frame has run for: 129.796997 ms
extract obstacles has run for: 16.617001 ms
adjust snake has run for: 10.219999 ms
send borerpoints has run for: 3.796000 ms
get robot locations has run for: 2.454000 ms
generate track has run for: 1.797000 ms
have token process has run for: 1.807000 ms
send ref to pc has run for: 1.821000 ms
one loop ends has run for: 4.637000 ms
read one frame has run for: 88.086998 ms
get foreground frame has run for: 129.607010 ms
extract obstacles has run for: 16.605999 ms
adjust snake has run for: 7.778000 ms
send borerpoints has run for: 1.706000 ms
get robot locations has run for: 4.024000 ms
generate track has run for: 2.326000 ms
have token process has run for: 1.825000 ms
send ref to pc has run for: 1.813000 ms
one loop ends has run for: 3.954000 ms
```

Figure 7.65 Software modules processing time: not sending display control points, no robot in view

Figure 7.66 shows the time consumed by different modules when the vision sensor observes the robot and the control points are requested to

be sent to the remote console. The time for sampling and buffering are now 89ms and 175ms respectively. Trajectory generating utilises around 56ms to plan the motion commands. The processing of sending A-snake to the remote console consumes about 19ms.

```
read one frame has run for: 89.469002 ms
get foreground frame has run for: 175.250000 ms
extract obstacles has run for: 17.677999 ms
adjust snake has run for: 12.990000 ms
send borerpoints has run for: 1.827000 ms
get robot locations has run for: 2.235000 ms
generate track has run for: 56.079998 ms
have token process has run for: 26.111000 ms
send ref to pc has run for: 28.410999 ms
one loop ends has run for: 39.949001 ms
read one frame has run for: 96.678001 ms
get foreground frame has run for: 167.983994 ms
extract obstacles has run for: 17.431000 ms
adjust snake has run for: 13.020000 ms
send borerpoints has run for: 1.833000 ms
get robot locations has run for: 2.210000 ms
generate track has run for: 56.125999 ms
have token process has run for: 26.291000 ms
send ref to pc has run for: 18.722000 ms
one loop ends has run for: 39.705997 ms
```

Figure 7.66 Software modules processing time with robot in view

A comparison diagram is shown in Figure 7.67. The appearance of the robot does not increase the processing time for reading a frame, extracting obstacles and adjusting snake. But dramatic increments of the processing time occur in generating trajectory, token process and sending A-snake to the remote because these three processes were not involved if the robot is not in view and they take time with the presence of the robot. From Figure 7.67, about 20~30ms can be seen for transferring one packet. The time for extracting foreground frames also increases due to more time is used for distinguishing foreground from background because the presence of robot increases the complexity of the image processing.

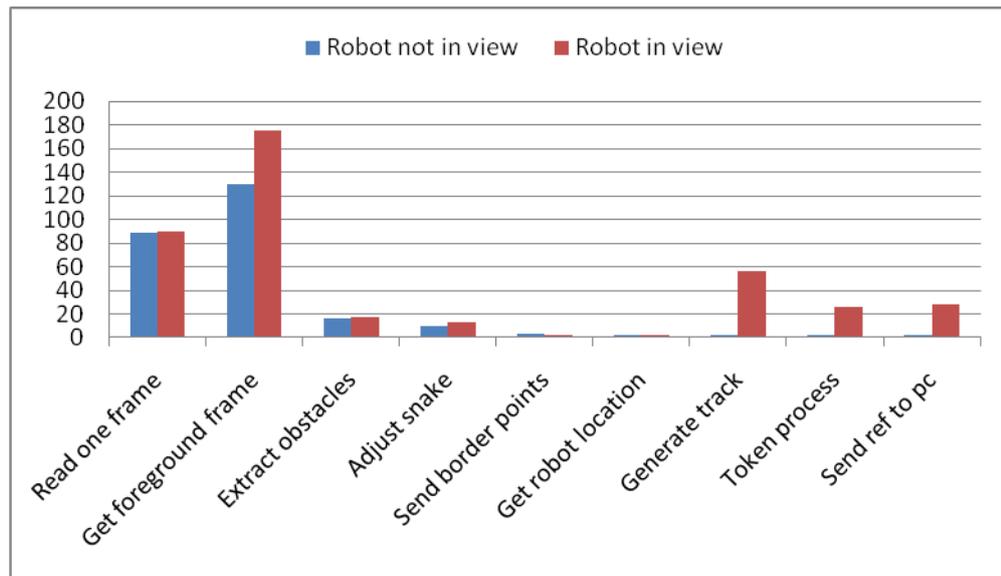


Figure 7.67 Running time comparison

7.5.2 The Impact of the Differences in Camera Physical Characteristics

The OV7620 can be configured in automatic operation mode. In this mode, exposure time and gains will be adjusted based on the average lightness on the image by different algorithms. However, this advantage has a negative impact on the system especially when the camera is too sensitive to the environment. If the background image is not sampled in accordance with the parameters change, the threshold for distinguishing foreground from background should be set to a very large value to allow for environment change. This will increase the possibility of taking background as foreground. On the other hand, if the background image is sampled on the change of the parameters, this will increase the CPU burden since multiple background images are sampled for an average background frame. Thus manual settings of camera parameters are adopted.

However, even when different cameras have exactly the same parameters, the pixel colours of the images captured on the same object by different cameras vary. Figure 7.68, Figure 7.69 and Figure 7.70 show images taken by three cameras with the same settings. As one can see, the colours differ dramatically from case 1 and case 2. Even the parameters work perfectly for case 1, the robot cannot be recognized by the camera in case 2.

The system profile files for different vision sensors store the specific settings for each of the cameras; this increases the flexibilities for choosing parameter as well as for a more reliable performance.

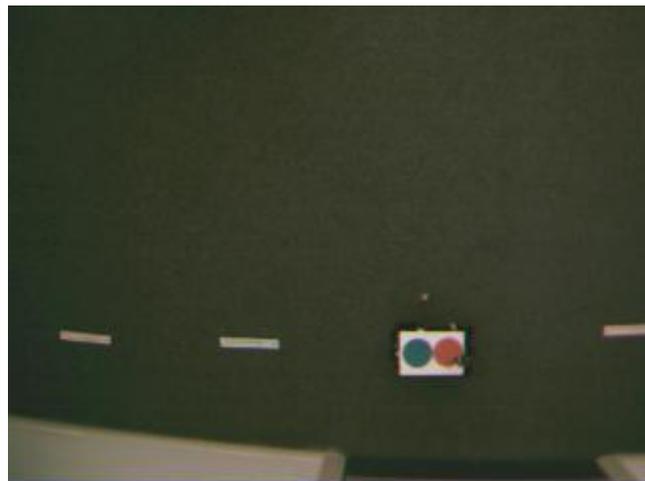


Figure 7.68 Image taken by different cameras - 1



Figure 7.69 Image taken by different cameras - 2

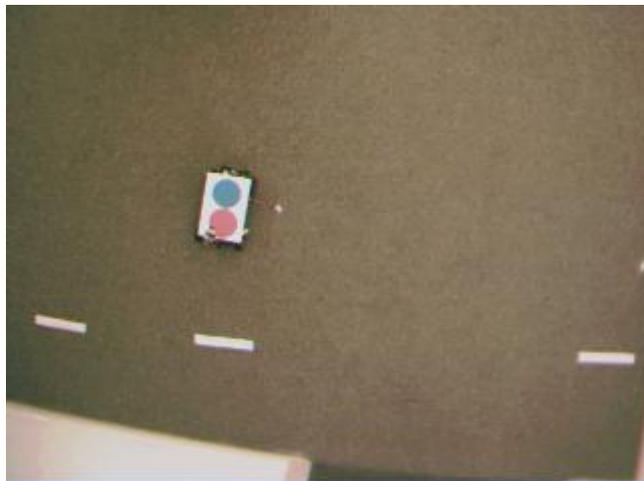


Figure 7.70 Image taken by different cameras - 3

7.6 Conclusion

This chapter discussed the complete implementation of the intelligent environment, including the hardware components, software applications as well as the communication protocols.

Vision sensors are mounted in the building to provide the fundamental infrastructure of the system. To support real time image processing and wireless communication, the iMote2 wireless sensor hardware platform is selected for its powerful processing capability, flexibility to interface with

various sensor boards and its onboard CC2420 for wireless communication. Base on this platform, a wireless vision sensor node is developed by integration with an OV7620 camera module. The details of the hardware design and implementations are presented in this chapter.

Communication protocols on top of IEEE802.15.4 are developed to serve the specific needs for path planning, exchange of control commands and data for robot navigation. Depending on different types of commands, communication can be invoked either by a remote console or by vision sensors. Details of signal flows between different entities, the packet formats and field definitions are described.

As the functional components of the system, seven software applications are developed to support distributed system configuration and robot navigation. A clear image of the software structure and the details of the functional modules are discussed. The software programs running in the remote console are developed in Java programming language while others are coded by C/C++. In addition to the applications for robot navigation in an intelligent environment, a snake simulation program with a GUI is developed to evaluate the algorithms and tune the control parameters.

Discussions have also been carried out on the implementation to highlight the merits and limitations of the system.

Chapter 8 : CONCLUSION AND FUTURE WORK

8.1 Conclusion

Autonomous navigation is a traditional research topic in intelligent robotics and vehicles, which requires a robot to perceive an environment through its onboard sensors, such as cameras and laser scanners, and drive automatically to a goal.

In order to achieve autonomous navigation, considerable efforts have been paid to cognitive aspects and knowledge representation that answer some fundamental questions of “Where am I?”, “How do I get there?” and “Where have I been?”, corresponding to localisation, path planning and map-making, respectively. Though human can answer these questions effortlessly based on proper information resources and prior knowledge, massive computation and memory space are necessary for a robot.

Most research to date has focused on the development of a large and smart “brain” to gain autonomous capability for robots. Carrying a “supercomputer” with a wide range of sensors, the robot has to perform centralised information processing and conduct complex decision making from perception, localisation and path planning to behaviour coordination and mission level task planning. Although the techniques have been developed fruitfully, they are facing a bottleneck of complexity due to dynamic changes and unstructured environments.

However, insect compound eyes with very small nervous systems can work in a highly efficient way that no computer today can contest with. The

bio-mechanism behind this seems to be the hardwired nervous systems from birth with each neuron having its own special predetermined links and functions.

Inspired by the biological insect eyes system, an intelligent environment architecture supported by mosaic eyes mounted in a building is proposed by the thesis to navigate unintelligent robots in the environment. With the distributed sensors and associated distributed information, the massive robot intelligence is released to its living environment. Further to this architecture, bio-mimetic snake path planning is proposed to coordinate all the decentralised vision sensors so as to navigate the mobile robots by generating and sending motion commands. A complete solution is proposed and tested for integrating communication with global path planning, trajectory generation and motion control of mobile robots. The scheme taking a snake as a coordination mechanism can be applied to general motion control problems using wireless sensor networks. To the best knowledge of the author, this scheme is the first work to have a unified strategy that integrates communication with control. Previous works in navigation by a wireless sensor network either separate network route planning from robot motion control or conduct merely snake based path planning while ignoring the low level motion control.

The following subsections discuss the major contributions of this thesis and the achieved objectives.

8.1.1 A Distributed Intelligent Environment Architecture

Steering away from the traditional centralised approaches, this thesis proposed one mosaic eyes based intelligent environment architecture for mobile robot navigation. Three components were defined with clear inter-related functional modules:

- The **remote console** is responsible for system configuration, global topological map configuration, data download, vision sensors control, mosaic eyes calibration and human-machine interaction. The topological map, system profile files and some general purpose files that must be contained in this component are also defined;
- **Vision sensors** are the brains of the intelligent environment. Each sensor covers a certain area of the workspace and provides services within that area. As the main component in the architecture, vision sensors are defined to have the largest range of functional modules, from single view image acquisition and processing to multiple views projections and mapping; from path initialisation, adjustment, trajectory generating to eventually robot motion control; from wireless packets receiving, sending and analysing to all kinds of interested data reporting.;
- The wireless **controlled robot** used was an off-the-shelf model car with limited on-board computational power and IEEE 802.15.4 wireless communication capability to receive and execute the instructions from vision sensors.

The proposed distributed intelligent environment releases the massive computation into different sensors, simplifies the localisation problem and promises a scalable environment variation.

8.1.2 The Distributed Snake Algorithm

A novel distributed snake algorithm by collaboration of distributed wireless vision sensors was proposed for collision-free path planning in a decentralised environment. To solve the curvature constraint problem, an original idea of using a three state snake was conceived.

The snake is split into small portions and maintained by different vision sensors. By adjusting the snake control points collaboratively with neighbouring vision sensors to reach a global balance, each portion of the snake performs globally. The three states machine was designed to control the snake control point movement such that the curvature constraints can always be satisfied. When control points are in different states, portions of the snake may become flexible, rigid or broken depending on the proposed uniform internal forces. Simulations and experiments were carried out and verified that the proposed snake based approach would be an effective distributed solution for simultaneously conducting global and local path planning.

8.1.3 A-snake Based Trajectory Tracking

A novel A-snake approach was proposed for robot motion control under the dynamic constraints. The A-snake generation and the proof of the A-snake convergence as well as the time-optimal control were

reported in detail. The proposed A-snake is in fact a distributed feedback control strategy to track the reference snake path in spite of external disturbances.

Both simulations and experiments were carried out to verify that the proposed method would offer an effective distributed navigation solution for integrating path planning, trajectory generation and motion control into a unified snake based control mechanism.

8.1.4 A Control-Oriented Communication Protocol

As a special mobile robot control system, the control task has to be solved via collaborations between wireless sensors. A communication protocol on top of IEEE802.15.4 was proposed to provide both reliable and unreliable data exchanges between different components, taking into account the factors of path planning, control instructions, file transferring, debugging and monitoring.

The protocols were deployed in the test bed system to link the different components successfully.

8.1.5 Applying Multiple Bloom filters in Topological Routing

Multiple Bloom filters were used for off-line global map building, compressing and storing, taking into account the fact that wireless sensors have limited memories and low rate communication bandwidth. After splitting and downloading all planned global map into individual vision sensors, the global topological path planning was achieved by on-line

Bloom filters querying. It was proved to be an efficient approach for distributed path planning by providing unambiguous spatial semantics to individual wireless sensors with a low memory footprint.

8.1.6 Occupancy Map Building and Geometry Obstacles Extraction

A Bayesian probabilistic model based fusion algorithm was proposed for determination of area occupancy from cooperation of distributed wireless vision sensors. Simulation results demonstrated that the algorithm had provided a more precise obstacle localisation.

8.1.7 The Intelligent Environment Test Bed Development

A complete intelligent environment test bed including the hardware components, software applications as well as the communication protocols was created as a result of the PhD research.

A sensor board was designed to integrate the OV7620 camera module with the iMote2 main board to form the vision sensor. The vision network was constructed by mounting the vision sensors in the ceiling inside the building. A wireless controlled mobile robot was developed by integrating one model car chassis, one Motorola MCU with one RS-380SH-4045 DC motor and one Futaba s3010 servo motor. The remote console is an AMD 2.0GHz Linux based PC.

Base on the hardware platform, software applications were developed to enable the hardware function. The communication protocols were

implemented along with the software applications to provide wireless as well as wired data exchange between different physical entities.

8.1.8 Creation of a Simulation Environment for Snake Algorithm Evaluation

A Java based software simulation platform was developed to enable various functions to be built in including concept proofing, algorithms performance evaluation, parameters tuning of the distributed snake algorithm. The coefficient constants of external and internal forces of snake, the sliding mode switch plane design, the image processing and the multiple obstacles extraction algorithms and etc. were all tested in the simulation environment before deploying to the vision sensors.

8.2 Future Work

The PhD work is one attempt to tackle the distributed navigation and motion control problem inspired by biological systems. The simulation environment and the experiment test bed developed within this PhD programme provide some promising results for intelligent environment aided robot control and form an infrastructural platform for further research.

While much have been achieved during the three-year study, several problems remain to be resolved and investigated in future research. The followings are some areas and topics that require further effort for research in effective robot control through environment intelligence.

8.2.1 Image Based Distributed Visual Servoing

Multiple view geometry of distributed cameras introduces difficulties to provide an accurate interpretation of a scene and mapping from image planes onto driving forces and steering torques of a robot. Traditionally, there are two approaches to achieve visual servoing: position based and image based. In the case of position based visual servoing, as adopted in this thesis, spatial coordinates are measured in the image plane and then projected to the work plane for trajectory planning and control. Curvature constraints, forces, torques and etc. are checked or planned in the same plane as the mobile robot. This method is a straightforward idea to coordinate multiple cameras by mapping images to a common workspace but it reduces the system efficiency due to the computation-intensive and ill-posed nature of the perspective projection, resulting in increased data exchanges in a wireless sensor network. More specifically, a very short line in a 2D image plane may imply a very long line in the workspace due to different perspective views, thus the size or position of an obstacle being extracted based on this short line might be error-prone when being projected to the work plane. Therefore the corresponding pixels in other cameras have to be projected to the work plane and fused. This will increase the communication burden for the purpose of information fusion accordingly.

Doing all planning and control in the image plane directly and project the forces and torques back to work plane can be efficient but it brings a challenge to distributed image based control. Some successful reactive control applications based on image features have been reported. A main

stream way is to utilise features matching between the captured image and the desired image in a real environment. As far as control is conducted in the image plane, the image Jacobian matrix has to be considered as a nonlinear coupling matrix that can affect the directions as well as the magnitudes of forces projection significantly. For example, the constraints, such as curvature, satisfaction in the image plane will not guarantee the satisfaction in the work plane. How to support robot servo control in image planes directly in an environment with distributed cameras may be a very interesting and promising topic for future research.

8.2.2 Self-Calibration of a Wireless Vision Sensor Network

Whenever there is a need for establishing a relationship between the captured 2D images and the 3D world scene, a camera network calibration is a prerequisite. Such relationship will be used to infer 3D structure of the world scene from the 2D images, and vice versa.

Wireless vision sensor network is bringing new research opportunities and challenges in image processing as well as wireless communication. The general idea of wireless sensor network is to integrate wireless communication, sensors and recognition technologies to provide pervasive intelligence. The quantity of sensors involved could be huge. It implies that the number of cameras required for calibration can be tedious and time consuming. It is not just mapping projection between one or two vision views, hundreds or thousands of cameras have to be calibrated before their cooperative operations. In addition to the calibration of vision projective relationships, the communication infrastructure needs to be

calibrated. Due to the dispersion of sensors in space, the communication ranges and routes have to be considered. The vision topology of the sensor network is usually different from that of the communication. Two vision sensors having overlap areas does not imply they can be linked directly if they are placed in a distance out of communication range but facing to a common area. Two vision sensors being placed in the same location with direct communication route may have entirely different views if they are mounted back to back.

The workload will be huge to calibrate or re-calibrate these two networks: the visual network and the communication network. How to create the topological map along with the projection relationships and the communication routes automatically? How to optimise the structure of the two networks and make the most out of their fusion? These could be important research areas in the future.

8.2.3 Automatic Background Image Adaptation

In the intelligent environment proposed in this thesis, vision sensors are fixedly mounted inside a building. Background image discrimination is the most straightforward yet useful technique to detect dynamic objects. This approach usually pre-processes the images for extracting intruding objects from the static background in order to achieve fast object detection and classification. However in reality the background is dynamic caused by variable illumination and structured changes.

How to adapt to the background changes is a hotspot of research in computer vision. The luminance change is the major factor to be

considered. A schedule could be set up to load the sampled background image at specific times or alternatively an adaptive mechanism for background update along with the passage of time could be developed. Obviously the former approach requires a vast amount of manual work due to the large scale of the sensor network. Further to this, due to structure changes, a pre-sampled environment image may not reflect the real situation. For the latter method, algorithms to determine a right time to sample the background, to eliminate dynamic objects from the background or to add the new intrusive objects but with no further movement for a while as background have attracted lots of research, which could be added as one of the optional future works.

8.2.4 Bio-Mimetic Technique Researches

A deeper involvement of bio-mimetic techniques could be another direction of the future work. This includes but not limited to sensor network region partitioning and organising, enhancement of communication efficiency, optimum protocol design and decentralised control, etc..

One robot control approach based on arthropod nervous systems is reported in [16] intending to answer how the brain, which transmits chemical signals between neurons in a relatively sluggish thousandth of a second, ends up performing some tasks faster and more efficiently than the most powerful digital processors. Another example is the biological discovery of some insects possessing a small nervous system but tens of thousands of eyes. How can this high throughput information exchange be achieved by slow signal transmission? Can the bio-mechanism be used to

improve the communication efficiency of wireless sensor networks? If the small nervous systems of insects and other invertebrates are hardwired from birth with each neuron having its special predetermined links and functions, how can the high efficient neuron organisation be applied to a large scale and complex environment?

Wolf spider has multiple eyes that have various roles. Some provide forward vision while others may scan and provide peripheral vision. The eye signals going to the brain are combined to complete motion detection, distance estimation and image formation. Can a similar mechanism being used to partition the sensor network in regions or to assign specific roles to different vision sensors to achieve better performance?

The aggregate motion of a herd of land animals, a flock of birds, a school of fish, a colony of ants or growth of bacteria in the natural world exhibits different kinds of collective behaviours being achieved by decentralised, self-organised biological systems. Research on their patterns has inspired many research areas such as Swarm Intelligence (SI) [170] and Particle Swarm Optimisation (PSO) [171]. Can these types of research be applied for protocol design to achieve global performance by short range decentralised communications? Or can the snake adjustment be optimised by adopting the above mechanisms? Biologically inspired methods will remain to be a much sought after research in the future.

REFERENCE

- [1] R. R. Murphy, *Introduction to AI robotics*. Cambridge: MIT Press, 2000.
- [2] R. C. Arkin, *Behavior-based Robotics*. Cambridge: MIT Press, 2000.
- [3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, pp. 14-23, 1986.
- [4] Neumann, C. Fermuller, and Y. Aloimonos, "Eyes from eyes: new cameras for structure from motion," presented at IEEE Workshop on Omnidirectional Vision, pp. 19-26, 2002.
- [5] A. M. Tekalp, *Digital video processing*. NJ: Prentice-Hall, 1995.
- [6] J. Vogel and B. Schiele, "A semantic typicality measure for natural scene categorization," presented at DAGM'04 Annual Pattern Recognition Symposium, Berlin, 2004.
- [7] M. L and D. E. Nilsson, *Animal Eyes*: Oxford University Press, 2002.
- [8] L. P. Lee and R. Szema, "Inspirations from biological optics for advanced photonic systems," *Science*, vol. 310, pp. 1148-1150, 2005.
- [9] J.Feng, *Computational Neuroscience: A Comprehensive Approach*: Taylor & Francis CRC Press, 2003.
- [10] T. W. Cronin and J. Marshall, "Parallel processing and image analysis in the eyes of mantis shrimps," *The Biological Bulletin*, pp. 177-183, 2001.
- [11] F. Boussaid, C. Shoushun, and A. Bermak, "A scalable low power imager architecture for compound-eye vision sensors," presented at Fifth International Workshop on System-on-Chip for Real-Time Applications (IWSOC'05), pp. 203-206, 2005.
- [12] J. Tanida, T. Kumagai, K. Yamada, S. Miyatake, K. Ishida, T. Morimoto, N. Kondou, D. Miyazaki, and Y. Ichioka, "Thin observation module by bound optics (tombo): concept and experimental verification," *Applied Optics-IP*, vol. 40, 2001.
- [13] R. Serrano-Gotarredona and et al, "AER building blocks for multi-layer multi-chip neuromorphic vision systems," *NIPS*, 2005.
- [14] K. Boahen, "Neuromorphic Microchips," *Scientific American*, May 2005.
- [15] "Development of biomorphic flyers (NPO-30554)," *NASA Tech Briefs*, vol. 28, 2004.
- [16] B. Webb, R. R. Harrisonb, and M. A. Willis, "Sensorimotor control of navigation in arthropod and artificial systems," *Arthropod Structure & Development*, vol. 33, 2004.
- [17] D. Gu and H. Hu, "Neural predictive control for a car-like mobile robot," *International Journal of Robotics and Autonomous Systems*, vol. 39, 2002.
- [18] Y. Cheng, P. Jiang, J. Zhu, and F. Guo, "Application of limit-cycle navigation improved by potential field approach to mobile robots," *Robot*, vol. 26, pp. 133-138, 2004.
- [19] H.Lin, J.Xiao and Z.Michalewicz, "Evolutionary algorithm for path planning in mobile robot environment," presented at Proc.1st IEEE Conf. on Evolutionary Computation Florida, USA, pp. 211-215, 1994.
- [20] J.Lee and Z.Bien, "Collision-free trajectory control for multiple robots based on neural optimization," *Network Robotica*, vol. 8, pp. 185-194, 1990.
- [21] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Transactions on Networking*, vol. 10, pp. 604-612, 2002.
- [22] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal, "The bloomier filter: an efficient data structure for static support lookup tables," presented at In

- Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia: SIAM, pp. 30-39, 2004.
- [23] S. Cohen and Y. Matias, "Spectral bloom filters," presented at In Proceedings of the 2003 ACM SIGMOD International Conference on the Management of Data, New York, pp. 241-252, 2003.
- [24] P. Maes, *Designing Autonomous Agents*. Cambridge: MIT/Elsevier, 1990.
- [25] J. Efken and R. Shaw, "Ecological Perspectives on the new artificial intelligence," *Ecological Psychology*, vol. 4, pp. 247-270.
- [26] R. Brooks, "New approaches to robotics," *Science*, pp. 1227-1232, 1991.
- [27] D. Lyons and A. Hendriks, "Reactive planning," *Encyclopedia of artificial intelligence*, 1992.
- [28] S. G. Goodridge and R. C. Luo, "Fuzzy behavior fusion for reactive control of an autonomous mobile robot: MARGE," presented at IEEE International Conference on Robotics and Automation, pp. IEEE International Conference, May 1994.
- [29] M. G. Kay and R. C. Luo S. G. Goodridge, "Multi-layered fuzzy behavior fusion for reactive control of an autonomous mobile robot," presented at Proceedings of the Sixth IEEE International Conference on Fuzzy Systems, pp. 579-584, 1997.
- [30] D. Kortenkamp, I. Nourbakhsh, and D. Hinkle, "The 1996 AAAI mobile robot competition and exhibition," *AI Magazine*, vol. 18, pp. 55-64, 1997.
- [31] W. G. Walter, *The Living Brain*. New York: W.W.Norton, 1953.
- [32] V. Braitenberg, "Vehicles: Experiments in Synthetic Psychology," 1984.
- [33] R. C. Arkin, "The impact of cybernetics on the design of a mobile robot system: a case study," *IEEE Transactions on System, Man, and Cybernetics*, vol. 20, pp. 1245-1257.
- [34] R. Ritzmann and T. McKenna R. Beer, *Neuroethology and Robotics*. Neuroethology and Robotics: Academic Press.
- [35] M. A. Arbib, *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory*. New York: Wiley, 1972.
- [36] M. Arbib and D. House, "Depth and detours: an essay on visually guided behavior," *Vision, Brain, and Cooperative Computation*, pp. 139-163, 1987.
- [37] D. Norman and T. Shallice, "Attention to action: willed and automatic control of behavior," *Consciousness and Self-regulation: Advances in Research and Theory*, vol. 4, pp. 1-17.
- [38] R. C. Arkin, "Integrating behavioral, perceptual and world knowledge in reactive navigation," *robotics and autonomous systems*, vol. 6, pp. 105-122, 1990.
- [39] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots," presented at AAAI-92, pp. 809-815, 1992.
- [40] T. Lozano-Perez, "Spatial planning: a configuration space approach," *IEEE Trans. on Comput.*, vol. 32, pp. 108-120, 1983.
- [41] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man, Cybern*, vol. 11, pp. 681-698, 1981.
- [42] R. A. Brooks, "Solving the find-path problem by good representation of free space," presented at in Proc. Nat. Conf. Artificial Intelligence, AAAI 82, pp. 381-386, 1982.
- [43] H. Choset and J. Burdick, "Sensor based motion planning: the hierarchical generalized voronoi graph," *International Journal of Robotics Research*, 2000.
- [44] B. Schiele and J. Crowley, "A comparison of position estimation techniques using occupancy grids," presented at Proc. 1994 IEEE

- International Conference on Robotics and Automation (ICRA-94), San Diego, CA, pp. 1628-1634, 1994.
- [45] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware," *Computer Graphics*, vol. 24, pp. 327-335, 1990.
- [46] Gardner and J. Richard, "The brunn-minkowski inequality," *Bull. Amer. Math. Soc.*, vol. 39, pp. 355-405, 2002.
- [47] J. Barraquand and J. C. Latombe, "A Monte-Carlo algorithm for path planning with many degrees of freedom," presented at In IEEE Int. Conf. Robot. & Autom., pp. 1712-1717, 1990.
- [48] K. Kondo, "Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration.," *IEEE Trans. Robot. & Autom.*, vol. 7, pp. 267-277, 1991.
- [49] E. Mazer, J. M. Ahuactzin, and P. Bessiere, "The Ariadne's clew algorithm," *J. Artificial Intell. Res.*, vol. 9, pp. 295-316, 1998.
- [50] E. Mazer, G. Talbi, J. M. Ahuactzin, and P. Bessiere, "The Ariadne's clew algorithm," presented at In Proc. Int. Conf. of Society of Adaptive Behavior, Honolulu, 1992.
- [51] S. M. LaValle and J. J. Kuffner, *Rapidly-exploring random trees: Progress and prospects, Algorithmic and Computational Robotics: New Directions*, 2001.
- [52] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. & Autom.*, vol. 12, pp. 566-580, 1996.
- [53] B. Glavina, "Solving findpath by combination of goal-directed and randomized search," presented at In IEEE Int. Conf. Robot. & Autom., pp. 1718-1723, 1990.
- [54] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," presented at In IEEE Int. Conf. Robot. & Autom., 2000.
- [55] O. Khatib, "Real time obstacle avoidance for manipulators and mobile robots," presented at ASME Winter Annual Meeting on Control of Manufacturing Processes and Robot Systems, Boston, pp. 243-251, 1983.
- [56] J. R. Andrews and N. Hogan, "Impedance control as a framework for implementing obstacle avoidance in a manipulator," presented at In Control of Manufacturing Processes and Robot Systems presented at the ASME Winter Annual Meeting, Boston, pp. 243-251, 1983.
- [57] N. Hogan, "Impedance control: an approach to manipulation: Parts I,II,III," *ASME Journal of Dynamic Systems, Measurement, and Control*, vol. 107, pp. 1-24, 1985.
- [58] W. S. Newman and N. Hogan, "High speed robot control and obstacle avoidance using dynamic potential functions," presented at Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh, pp. 14-24, 1987.
- [59] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," presented at Proc. IEEE Conf. Robotics and Automation, Sacramento, CA, pp. 1398-1404, Apr 1991.
- [60] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 1179-1187, 1989.
- [61] J. H. Chuang and N. Ahuja, "An analytically tractable potential field model of free space and its application in obstacle avoidance," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 729-736, 1998.
- [62] M. Okutomi and M. Mori, "Decision of robot movement by means of a potential field," *Advanced Robotics*, vol. 1, pp. 131-141, 1986.

- [63] P. Khosla and R. Volpe, "Superquadric artificial potentials for obstacle avoidance and approach," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia, PA, pp. 1778-1784, 1988.
- [64] D. E. Koditschek, "Exact robot navigation by means of potential functions: Some topological considerations," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, Rayleigh, NC, pp. 1-6, 1987.
- [65] E. Rimon and D. E. Koditschek, "Exact robot navigation using cost functions: The case of distinct spherical boundaries in e^n ," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia, PA, pp. 1791-1796, 1988.
- [66] B. H. Krogh and C. E. Thorpe, "Integrated path planning and dynamic steering control for autonomous vehicles," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, pp. 1664-1669, 1986.
- [67] P. Tournassoud, "A strategy for obstacle avoidance and its application to multi-robot systems," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, pp. San Francisco, 1986.
- [68] Y. Kanayama, "Least cost paths with algebraic cost functions," presented at In Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia, PA, pp. 75-80, 1988.
- [69] B. R. Donald, "Motion planning with six degrees of freedom," MIT Artificial Intelligence Laboratory 1984.
- [70] M. Kass, A. Witkin, and D. Terzopoulos, "Snake: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321-331, 1988.
- [71] D. Geiger, A. Gupta, L. A. Costa, and J. Vlontzos, "Dynamic programming for detecting, tracking and matching deformable contours," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 294-302, 1995.
- [72] S. R. Gunn and M. S. Nixon, "A robust snake implementation: A dual active contour," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 19, pp. 63-68, 1997.
- [73] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," *Int. Journal Comput. Vis.*, vol. 22, pp. 61-79, 1997.
- [74] J. A. Sethian, "Fast marching methods," *SIAM Rev.*, vol. 41, pp. 199-235, 1999.
- [75] S. J. Osher and J. A. Sethian, "Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation," *J. Comput. Phys.*, vol. 79, pp. 12-49, 1988.
- [76] L. D. Cohen and R. Kimmel, "Global minimum for active contour models: A minimal path approach," *Int. J. Comput. Vis.*, vol. 24, pp. 57-78, 1997.
- [77] A. E. Hassanien and M. Nakajima, "Feature-specification algorithm based on snake model for facial image morphing," *IEICE Trans. Inform. Syst.*, vol. E82-D, pp. 439-445, 1999.
- [78] S. Quinlan, "Real-time modification of collision-free paths," in *Department of Computer Science*, vol. PhD: Stanford, 1994.
- [79] S. Quinlan and O. Khatib, "Elastic bands: connecting path planning and control," presented at IEEE Int. Conf. Robotics and Automation, Atlanta, USA, 1993.
- [80] A. Mclean, "Dealing with geometric complexity in motion planning," presented at Int. Conf., IEEE in Robotics and Automation, 1996.
- [81] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed: MIT Press and McGraw-Hill, 2001.

- [82] Y. Cheng, Y. F. Hu, and P. Jiang, "A distributed snake algorithm for mobile robots path planning with curvature constraints," presented at IEEE Int. Conf. on SMC, Singapore, pp. 2056-2062, 2008.
- [83] S. S. Ge, X. C. Lai, and A. A. Mamun, "Sensor-based path planning for nonholonomic mobile robots subject to dynamic constraints," *Robotics and autonomous systems*, vol. 55, pp. 513-526, 2007.
- [84] T. Liang, J. Liu, G. Hung, and Y. Chang, "Practical and flexible path planning for car-like mobile robot using maximal-curvature cubic spiral," *Robotics and Autonomous Systems*, vol. 52, pp. 312-325, 2005.
- [85] W. L. Nelson, "Continuous steering-function control of robot carts," *IEEE Trans. Industrial Electronics*, vol. 36, pp. 330-337, 1989.
- [86] M. Khatib, "Sensor-Based Motion Control for Mobile Robots," in *LAAS-CNRS*, vol. PhD, 1996.
- [87] D. Soetanto, L. Lapierre, and A. Pascoal, "Adaptive, non-singular path-following control of dynamic wheeled robots," presented at Proceedings of the 42nd IEEE Conference on Decision and Control, Maui, Hawaii USA, pp. 1765-1770, 2003.
- [88] R. Brockett, "Asymptotic stability and feedback stabilization," *Differential Geonletric Control Theory*, pp. 181-191, 1983.
- [89] C. C. D. Wit, H. Khenouf, C. Samson, and O. J. Sordalen, "Nonlinear control design for mobile robots," *World Scientific Series in Robotics and Automated Systems*, vol. 11, pp. 121-156, 1993.
- [90] J. Godhavn and O. Egeland, "A Lyapunov approach to exponential stabilization of nonholonomic systems in power form," *IEEE Transactions on Automatic Control*, vol. 42, pp. 1028-1032, 1997.
- [91] A. Aguiar, A. Atassi, and A. Pascoal, "Regulation of a nonholonomic dynamic wheeled mobile robot with parametric modeling uncertainty using Lyapunov Function," presented at Proc. 39th IEEE Conference on Decision and Control, Sidney, Australia, Dec, 2000.
- [92] C. C. D. Wit and O.Sordalen, "Exponential stabilization of mobile robots with nonholonomic constraints," *IEEE Transactions on Automaric Control*, vol. 37, pp. 1791-1797, 1992.
- [93] J. Hespanha, "Stabilization of Nonholonomic integrators via logic-based switching," presented at Proc. 13th World Congress of IFAC, Francisco, CA, USA, pp. 467-472, 1996.
- [94] E. Freund and R. Mayr, "Nonlinear path control in automated vehicle guidance," *IEEE Transactions on Robotics and Automation*, vol. 13, pp. 49-60, 1997.
- [95] G. Walsh, D. Elbury, S. Sasuy, R. Murray, and J. Laumond, "Stabilization of trajectories for systems with nonholonomic constraints," vol. 39, pp. 216-222, 1994.
- [96] R. Fierro and F. Lewis, "Control of a nonholonomic mobile robot: backstepping kinematics into dynamics," presented at Proc. 34th IEEE Conference on decision and control, New Orleans, LA, USA, pp. 3805-3810, 1995.
- [97] C. Samson and K. Ait-Abderrahim, "Mobile robot control part 1: feedback control of a non-holonomic mobile robots," *Technical Repon No. 1281, INRIA*, 1991.
- [98] A. Micaelli and C. Samson, "Trajectory tracking for unicycle-type and two-steering wheels mobile robots," *Technical Repon No. 2097, INRIA*, 1993.
- [99] Z. Jiang and H. Nijmeijer, "A recursive technique for tracking control of nonhohonomic systems in chained form," *IEEE Transactions on Robotics and Automation*, vol. 44, pp. 265-279, 1999.
- [100] P. Encarnacao and A. Pascoal, presented at 3D path following for autonomous underwater Vehicles, Sydney, Australia, 2000.

- [101] P. Encarnacao, A. Pascoal, and M. Arcak, "Path following for marine vehicles in the presence of unknown currents," presented at Proc. SYROCO 2000, 6th IFAC Symposium on Robot Control, Vienna, Austria, 2000.
- [102] H. Baldus, K. Klabunde, and G. Muesch, "Reliable set-up of medical body-sensor networks," presented at In Proc. EWSN 2004, Berlin, 2004.
- [103] R. Beckwith, D. Teibel, and P. Bowen, "Pervasive computing and proactive agriculture," presented at In Adjunct Proc. PERVASIVE 2004, Vienna, April, 2004.
- [104] Z. Butler, P. Corke, R. Peterson, and D. Rus, "Networked Cows: virtual fences for controlling cows," presented at In WAMES 2004, Boston, Jun, 2004.
- [105] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with zebraNet," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 96-107, 2002.
- [106] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson, "Wireless sensor networks for habitat monitoring," presented at In WSNA, Atlanta, USA, Sep, 2002.
- [107] I. W. Marshall, C. Roadknight, I. Wokoma, and L. Sacks, "Self-organizing sensor networks," presented at In UbiNet 2003, London, UK, Sep, 2003.
- [108] K. Martinez, R. Ong, J. K. Hart, and J. Stefanov, "GLACSWEB: a sensor web for glaciers," presented at In Adjunct Proc. EWSN 2004, Berlin, Germany, Jan, 2004.
- [109] F. Michahelles, P. Matter, A. Schmidt, and B. Schiele, "Applying wearable sensors to avalanche rescue," *Computers and Graphics*, vol. 27, pp. 839-847, 2003.
- [110] R. RiemVis, "Cold chain management using an ultra low power wireless sensor network," presented at In WAMES 2004, In WAMES 2004, Jun, 2004.
- [111] G. Simon, A. Ledezczi, and M. Maroti, "Sensor network based countersniper system," presented at In Proc. SenSys, Baltimore, USA, Nov, 2004.
- [112] ARGO - Global Ocean Sensor Network, " www.argo.ucsd.edu."
- [113] The 29 Palms Experiment: Tracking vehicles with a UAV-delivered sensor, " network.tinyos.millennium.berkeley.edu/29Palms.htm."
- [114] C. Kappler and G. Riegel, "A real-world, simple wireless sensor network for monitoring electrical energy consumption," presented at In Proc. EWSN 2004, Berlin, Germany, Jan, 2004.
- [115] "http://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes."
- [116] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422-426, 1970.
- [117] J. K. Mullin and D. J. Margoliash, "A tale of three spelling checkers," *Software - Practice and Experience*, vol. 20, pp. 625-630, 1990.
- [118] M. D. McIlroy, "Development of a Spelling List," *IEEE Transactions on Communications*, vol. 30, pp. 91-99, 1982.
- [119] J. K. Mullin, "Optimal semijoins for distributed database systems," *IEEE Transactions on Software Engineering*, vol. 16, pp. 558, 1990.
- [120] K. Bratbergsengen, "Hashing methods and relational algebra operations," presented at In Proceedings of the Tenth International Conference on Very Large Databases, San Francisco: Morgan Kaufmann, pp. 323-333, 1984.
- [121] P. Valdurez and G. Gardarin, "Join and semijoin algorithms for a multiprocessor database machine," *ACM Transactions on Database Systems*, vol. 9, pp. 133-161, 1984.

- [122] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 281-293, 2000.
- [123] S. Czerwinski, B. Y. Zhao, T. Hodes, A. D. Joseph, and R. Katz, "An architecture for a secure service discovery service," presented at In Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), New York, pp. 24-35, 1999.
- [124] J. Borenstein L. Feng, B. Everett, "Where am I? Sensors and methods for autonomous mobile robot localization," *Technical Report, The University of Michigan UM-MEAM-94021*, 1994.
- [125] W. Bernhard, "Interference cancelation in ultrasonic sensor arrays by stochastic coding and adaptive filtering," presented at IEEE International Conference on Intelligent Vehicles, pp. 376-380, 1998.
- [126] J. Lee, K. Morioka, and H. Hashimoto, "Mobile Robot Control in Intelligent Space for People Support," *Journal of Robotics and Mechatronics*, vol. 14, pp. 390-399, 2002.
- [127] "<http://www.xbow.com/Products/productdetails.aspx?sid=253>."
- [128] S. Crocel, F. Marcellon, and M. Vecchio, "Reducing power consumption in wireless sensor networks using a novel approach to data aggregation," *The Computer Journal*, vol. 51, pp. 227-239, 2008.
- [129] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, 2003.
- [130] J. K. Mullin, "A second look at bloom filters," *Communications of the ACM Transactions on Database Systems*, vol. 26, pp. 570-571, 1983.
- [131] G. Welch and G. Bishop, "An introduction to the Kalman Filter," *UNC-Chapel Hill, TR 95-041*, 2004.
- [132] H.F. Durrant-Whyte J.J. Leonard, I.J. Cox, "Dynamic map building for an autonomous mobile robot," *Internat. J. Robotics Res*, vol. 11, pp. 89-96, 1992.
- [133] W. D. Rencken, "Concurrent localisation and map building for mobile robots using ultrasonic sensors," presented at Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-93), Yokohama, Japan, pp. 2129-2197, 1993.
- [134] R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O. Sullivan, "A layered architecture for office delivery robots," presented at Proc. 1st International Conference on Autonomous Agents, Marina del Rey, CA, 1997.
- [135] C. Wetzler, G. Wei, and E. Puttkamer, "Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans," presented at Proc. International Conference on Intelligent Robots and Systems (IROS-94), Munich, Germany, pp. 595-601, 1994.
- [136] I. J. Cox, "Blanche-An experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 193-204, 1991.
- [137] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, pp. 99-141, 2001.
- [138] J. Borenstein, B. Everett, and L. Feng, *Navigating Mobile Robots: Systems and Techniques*: A.K. Peters, Wellesley, MA, 1996.
- [139] W. Burgard, A. Derr, D. Fox, and A. B. Cremers, "Integrating global position estimation and position tracking for mobile robots: The dynamic markov localization approach," presented at Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), Victoria, BC, 1998.

- [140] P. Jensfelt and S. Kristensen, "Active global localisation for a mobile robot using multiple hypothesis tracking," presented at Proc. IJCAI Workshop on Reasoning with Uncertainty in Robot Navigation, Stockholm, Sweden, pp. 13-22, 1999.
- [141] G. A. Bekey S. I. Roumeliotis, "Bayesian estimation and Kalman filtering: A unified framework for mobile robot localization," presented at Proc. IEEE International Conference on Robotics and Automation (ICRA-2000), San Francisco, CA, pp. 2985-2992, 2000.
- [142] S. Engelson and D. McDermott, "Error correction in mobile robot map learning," presented at Proc. 1992 IEEE International Conference on Robotics and Automation, Nice, France, pp. 2555-2560, 1992.
- [143] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *J. Artificial Intelligence Res*, vol. 11, pp. 391-427, 1999.
- [144] D. Estrin, D. Culler, and K. Pister, "Connecting the physical world with pervasive networks," *IEEE Pervasive Computing*, vol. 1, pp. 59-69, 2002.
- [145] W. Su I.F. Akyildiz, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communication Magazine*, pp. 102-114, 2002.
- [146] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. Cambridge: USA: MIT Press, 2004.
- [147] Q. Li and D. Rus, "Navigation protocols in sensor networks," *ACM Trans. on Sensor Networks*, vol. 1, pp. 3-35, 2005.
- [148] D. Xue, P. Jiang, and J. Zhu, "A self-organising diffusion protocol of sensor networks for robot navigation in a dynamic environment," presented at The International Conference 2007 on Information Computing and Automation, Chengdu, China, 2007.
- [149] "<http://www.equasys.de/colorformat.html>."
- [150] G. Pingali, Y. Jean, and I. Carlborn, "Real time tracking for enhanced tennis broadcasts," presented at Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 260-265, 1998.
- [151] K. Sobottka and I. Pistas, "Segmentation and tracking of faces in color images," presented at Proc. of International Conference on Automatic Face and Gesture Recognition, pp. 236-241, 1996.
- [152] S. Mckenna, Y. Raja, and S. Gong, "Tracking colour objects using adaptive mixture models," *Image and Vision Computing*, pp. 225-231, 1999.
- [153] Y. Wu, Q. Liu, and T. S. Huang, "Robust real-time human hand localization by self-organizing color segmentation," presented at In Proc. of ICCV99 Workshop RATFG-RTS, 1999.
- [154] J. Bruce, T. Balch, and M. Veloso, "Fast and inexpensive color image segmentation for interactive robots," presented at IROS '00, pp. 2061-2066, 2000.
- [155] H. N. E. Bystrom, "Extreme value theory and extremely large electricity price changes," *International Review of Economics and Finance*, vol. 14, pp. 41-55, 2005.
- [156] A. Hoover and B. D. Olsen, "A real-time occupancy map from multiple video streams," presented at IEEE Int'l Conf. on Robotics and Automation, pp. 2261-2266, 1999.
- [157] M. R. Seiegel and Y. Proykova, *Schaum's outline of theory and problems of theoretical mechanics: with an introduction to Lagrange's equations and Hamiltonian theory*. NY: MacGraw-Hill, 1980.
- [158] J. M. Maciejowski, *predictive control with constraints*, 1 ed.
- [159] D. M. Prett C. E. Gacia, and M. Morari, "Model predictive control: theory and practice-a survey," *Automatica*, vol. 25, pp. 335-348, 1989.

- [160] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: stability and optimality," *Automatica*, vol. 36, pp. 789-814, 2000.
- [161] G. Klancar and I. Skrjanc, "Tracking-error model-based predictive control for mobile robots in real time," *Robotics and Autonomous Systems*, vol. 55, pp. 460-469, 2007.
- [162] Y. G. Xi and C. G. Zhang, "Rolling path planning of mobile robot in a kind of dynamic uncertain environment," *Acta Automatica Sinica*, vol. 28, pp. 161-175, 2002.
- [163] W. Hahn, "Theory and Application of Liapunov's Direct Method," 1963.
- [164] J. J. Slotine, "Applied nonlinear control," 1991.
- [165] C. Samson, "Control of Chained Systems Application to Path Following and Time-Varying Point Stabilization of Mobile Robots," *IEEE Transactions on Automatic Control*, vol. 40, pp. 64-77, 1995.
- [166] G. Oriolo, A. D. Luca, and M. Vendittelli, "WMR Control Via Dynamic Feedback Linearization: Design, Implementation, and Experimental Validation," *IEEE Transactions on Control Systems Technology*, vol. 10, pp. 835-851, 2002.
- [167] "Intel Mote 2 Engineering Platform Data Sheet."
- [168] OmmVision, "OV7620 SINGLE-CHIP CMOS VGA COLOR DIGITAL CAMERA."
- [169] Texas Instruments, "2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver," 2007.
- [170] J. B. Waldner, *Nanocomputers and Swarm Intelligence*, 2008.
- [171] K. Parsopoulos and M. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*, 2010.