Cooperative and Intelligent Control of Multi-robot Systems Using Machine Learning

by

Ying Wang

B.A.Sc., Shanghai Jiao Tong University, 1991
M.A.Sc., Shanghai Jiao Tong University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

March 2008

ABSTRACT

This thesis investigates cooperative and intelligent control of autonomous multi-robot systems in a dynamic, unstructured and unknown environment and makes significant original contributions with regard to self-deterministic learning for robot cooperation, evolutionary optimization of robotic actions, improvement of system robustness, vision-based object tracking, and real-time performance.

A distributed multi-robot architecture is developed which will facilitate operation of a cooperative multi-robot system in a dynamic and unknown environment in a self-improving, robust, and real-time manner. It is a fully distributed and hierarchical architecture with three levels. By combining several popular AI, soft computing, and control techniques such as learning, planning, reactive paradigm, optimization, and hybrid control, the developed architecture is expected to facilitate effective autonomous operation of cooperative multi-robot systems in a dynamically changing, unknown, and unstructured environment.

A machine learning technique is incorporated into the developed multi-robot system for self-deterministic and self-improving cooperation and coping with uncertainties in the environment. A modified Q-learning algorithm termed Sequential Q-learning with Kalman Filtering (SQKF) is developed in the thesis, which can provide fast multi-robot learning. By arranging the robots to learn according to a predefined sequence, modeling the effect of the actions of other robots in the work environment as Gaussian white noise and estimating this noise online with a Kalman filter, the SQKF algorithm seeks to solve several key problems in multi-robot learning.

As a part of low-level sensing and control in the proposed multi-robot architecture, a fast computer vision algorithm for color-blob tracking is developed to track multiple moving objects in the environment. By removing the brightness and saturation information in an image and filtering unrelated information based on statistical features and domain knowledge, the algorithm solves the problems of uneven illumination in the environment and improves real-time performance.

In order to validate the developed approaches, a Java-based simulation system and a physical multi-robot experimental system are developed to successfully transport an object of interest to a goal location in a dynamic and unknown environment with complex obstacle distribution. The developed approaches in this thesis are implemented in the prototype system and rigorously tested and validated through computer simulation and experimentation.

TABLE OF CONTENTS

Abstract ii
Table of Contents iv
List of Figures vii
Nomenclature xi
Acknowledgements xiv
Chapter 1 Introduction 1
1.1 Goals of the Research 2
1.2 Problem Definition 3
1.3 Related Work 5
1.3.1 Multi-robot Architectures 9
1.3.2 Machine Learning in Multi-robot Systems 13
1.3.3 Computer Vision and Sensing Technologies for Multi-robot Systems 17
1.4 Contributions and Organization of the Thesis 20
Chapter 2 Learning/Planning Based Hierarchical Multi-robot Architecture 22
2.1 Overview 22
2.2 General Architecture 25
2.3 Machine Learning Unit 28
2.4 Planning Unit 29
2.4.1 Planning and Learning in Multi-robot Applications 29
2.4.2 Local Path Planning for Multi-robot Cooperative Transportation 32
2.5 Integrating Learning with Planning 35
2.6 Behavior-based Distributed Control 36
2.6.1 Behavior Representation 37
2.6.2 Behavior Composition 38
2.6.3 Group Coordination 40
2.6.4 Communication Behaviors 40
2.6.5 Coordinated Object Transportation 41
2.7 Low-level Control 43
2.7.1 Robotic Manipulators 43
2.7.2 Mobile Robots 44
2.8 Distributed Control Issues 47
2.9 Summary 49
Chapter 3 Machine Learning for Multi-robot Decision-making 51
3.1 Overview 51

3.2 Markov Decision Process (MDP) and Reinforcement Learning 52
3.2.1 Reinforcement Learning 55
3.3 Multi-robot Transportation Using Machine Learning: First Version 57
3.3.1 Multi-agent Infrastructure 59
3.3.2 Cooperation Based on Machine Learning 61
3.3.3 Simulation Result 68
3.3.4 Experimentation 72
3.4 Distributed Multi-robot Q-learning in Complex Environments 74
3.4.1 Distributed Q-learning 74
3.4.2 Task Description 76
3.4.3 Multi-robot Box-pushing With Distributed Q-learning 76
3.4.4 Simulation Results 82
3.5 Team Q-learning Based Multi-robot Cooperative Transportation 90
3.5.1 Stochastic Games and the Team Q-learning Algorithm 91
3.5.2 Simulation Results 92
3.5.3 Impact of Learning Parameters 95
3.5.4 Comparing Team Q-learning with Multi-robot Distributed Q-learning 102
3.6 Summary 110
Chapter 4 Sequential Q-learning with Kalman Filtering (SQKF) 112
4.1 Overview 112
4.2 Sequential Q-learning with Kalman Filtering 113
4.2.1 Sequential Q-learning 113
4.2.2 Kalman Filtering based Reward Estimation 115
4.3 Simulation Results 120
4.4 Summary 131
Chapter 5 Computer Vision in Multi-robot Cooperative Control 133
5.1 Overview 133
5.2 Multi-robot Transportation System 135
5.3 The Fast Color-blob Tracking Algorithm 137
5.3.1 Removal of the Lighting Disturbance 137
5.3.2 Statistical Feature Filtering 139
5.3.3 Color-blob Template Matching 140
5.3.4 Coordinate Transformation and Pose Reconstruction 140
5.4. Experimental Result 141
5.4.1 The Test-bed 141
5.4.2 Experimental Result 142

5.5 Multi-robot Route Planning 147
5.6 Summary 149
Chapter 6 Experimentation in Distributed Multi-robot Cooperative Transportation 150
6.1 Overview 150
6.2 Test Bed 151
6.3 JAVA RMI 155
6.3.1 Remote Interface 158
6.3.2 Server Side 158
6.3.3 Client Side 160
6.4 A Physical Multi-robot Transportation System 161
6.5 Force/Position Hybrid Control 165
6.6 Summary 172
Chapter 7 Conclusions 173
7.1 Primary Contributions 173
7.2 Limitations and Suggested Future Research 175
7.2.1 Improvement of the Model and Algorithm of SQKF 176
7.2.2 GA-based Learning Space Compression 176
7.2.3 Local Modeling and Low-level Control 176
7.2.4 Integrating Planning and Learning 177
7.3 Summary 177
Bibliography 178
Appendix: JAVA Documents of the Developed System 190

LIST OF FIGURES

Figure 1.1 Schematic representation of the developed system 3
Figure 2.1 A general and hierarchical multi-robot architecture 26
Figure 2.2 The principle of the machine learning unit 28
Figure 2.3 An example of the local minimum phenomenon in a purely learning-based multi-robot cooperative box-pushing task 30
Figure 2.4 The bounded local environment in navigation potential function planning 33
Figure 2.5 The hierarchical behavior-based control layer 36
Figure 2.6 A SR diagram of a simple behavior 37
Figure 2.7 The diagram of a composite behavior with a competitive operator 39
Figure 2.8 The diagram of a composite behavior with a cooperative operator 40
Figure 2.9 Behavioral hierarchy describing the coordinated transport sub-task 41
Figure 2.10 A FSA describing the transitions of the group behaviors in the project of coordinated object transportation 43
Figure 2.11 Control of a mobile robot 45
Figure 2.12 The block diagram of motion control of a mobile robot 46
Figure 3.1 The value iteration algorithm to calculate the utilities 55
Figure 3.2 The single-agent Q-learning algorithm 56
Figure 3.3 The developed multi-robot system 59
Figure 3.4 The multi-agent architecture used in the developed system 59
Figure 3.5 The world state representation 62
Figure 3.6 Representation of the cooperation strategy (action) 63
Figure 3.7 The Genetic Algorithm 64
Figure 3.8 The integration scheme of RL and GA 68
Figure 3.9 The robots move randomly and locate the box 70
Figure 3.10 The whole transportation process 71
Figure 3.11 The goal location was suddenly changed from (380,370) to (385,60) at t = 34s while the robots were transporting the box 72

Figure 3.12 The robots have selected the "sweeping" action to remove a movable obstacle in the path 73
Figure 3.13 Description of a multi-robot box-pushing task 76
Figure 3.14 The binary coding of the environmental states 77
Figure 3.15 Coding rule of the environmental states 78
Figure 3.16 Pushing actions available to each robot 79
Figure 3.17 The dynamics of the box pushed by three robots 80
Figure 3.18 Three box-pushing snapshots under different obstacle distributions and goal locations 84
Figure 3.19 The environmental state with binary value "1000000" and six pushing actions available to the robots 87
Figure 3.20 The Q values under the state of "1000000" in 100,000 rounds of box-pushing 88
Figure 3.21 Probability density estimate of Q values under state of "1000000" in 100,000 rounds of box-pushing 88
Figure 3.22 Probability density estimate of action selection probability under the state of "1000000" in 100,000 rounds of box-pushing 89
Figure 3.23 The box trajectory in a round of box-pushing 93
Figure 3.24 The rewards received by the robots in a round of box-pushing 94
Figure 3.25 The surface of the Q-table (the states from 1 to 32) after 10,000 training cycles 94
Figure 3.26 The impact of training 96
Figure 3.27 Impact of the lower bound of the random action probability 97
Figure 3.28 The impact of the detection radius 98
Figure 3.29 The impact of the pushing force 99
Figure 3.30 The rewards received by the robots in a round of box-pushing 101
Figure 3.31 Simulation results with the single-agent Q-learning algorithm and the team Q-learning algorithm 103
Figure 3.32 Performance index comparison of the single-agent Q-learning algorithm before and after the training stage 106

Figure 3.33 Performance index comparison of the Team Q-learning algorithm before and after the training stage 109
Figure 4.1 The simulated multi-robot box-pushing system 120
Figure 4.2 A successful example of multi-robot box pushing with the new SQKF algorithm 121
Figure 4.3 Probability density estimate of the number of steps per round before and after training 122
Figure 4.4 Probability density estimate of the average global reward per round before and after training 123
Figure 4.5 The number of steps per round in 1,000 rounds of continuous box-pushing 124
Figure 4.6 The average global reward per round in 1,000 rounds of continuous box-pushing 125
Figure 4.7 The global rewards received by the robots in a round of box-pushing before training 126
Figure 4.8 The probability density estimate of the global rewards in Figure 4.7 127
Figure 4.9 The global rewards received by the robots in a round of box-pushing after training 128
Figure 4.10 The probability density estimate of the global rewards in Figure 4.9 128
Figure 4.11 Probability density estimate of number of steps per round in 500 rounds of box-pushing 129
Figure 4.12 Probability density estimate of average global reward per round in 500 rounds of box-pushing 130
Figure 5.1 The developed physical multi-robot system 135
Figure 5.2 The color-blob tracking algorithm 137
Figure 5.3 A sample image of a color blob 139
Figure 5.4 The experimental system in IAL at UBC 142
Figure 5.5 An original image captured by the CCD camera 143
Figure 5.6 The hue component of the original color image in the HSI color space 144
Figure 5.7 The hue component image after the background is filtered 145
Figure 5.8 The result after filtering Figure 5.7 using template matching 146

Figure 5.9 The reconstructed result of the poses of the robots and the box 147
Figure 6.1 The multi-robot object transportation system developed in IAL 151
Figure 6.2 The color blobs used to identify the orientation of the box 153
Figure 6.3 The PioneerTM 3-DX mobile robot 154
Figure 6.4 The PioneerTM 3-AT mobile robot 154
Figure 6.5 JAVA RMI distributed computing 157
Figure 6.6 Multi-robot cooperative transportation 164
Figure 6.7 The robot hybrid force/position control scheme 166
Figure 6.8 The robots move to the desired position and push the box under hybrid force/position control 168
Figure 6.9 Position control of the robotic arm 170
Figure 6.10 The force control of the robotic arm 171

NOMENCLATURE

Notations

$\gamma_k(x,y)$	Attraction function in the potential field approach
$\beta(x,y)$	Repulsion function in the potential field approach
$\varphi(x,y)$	Potential function
$\nabla\varphi(x_i,y_i)$	Gradient of a potential field
$D(q)$	Inertia matrix of the manipulator
$C(q,\dot{q})$	Coriolis matrix
τ	Joint torque (see Chapter 2)
$S = \{s_1, s_2, \ldots, s_n\}$	Set of states in the environment
$A = \{a_1, a_2, \ldots, a_m\}$	Set of actions available to a robot
$T(s,a,s')$	Transition function, which decides the next environmental state s' when robot selects action a_i under current state s.
$\prod(s)$	Probability distribution over the states
$R: S \times A \rightarrow \Re$	Reward function. Gives the immediate reward when robot takes action a_i under current state s
π	Policy with which a robot selects its actions
π^*	Optimal policy
β (Chapter 2)	Mapping function
β (Chapter 3)	Discount factor
$U(s)$	Utility of a state
ε	Maximum error allowed in the utility of any state
δ	Maximum change in the utility of any state in an iteration
$Q(s_i,a_j)$	An entry in the Q-table
η	Learning rate
r	Direct reward

τ (Section 3.2)	" Temperature " parameter (see Chapter 3)
τ (Section 3.3)	The angle between the net force vector and the position vector of the closest obstacle
Γ	Net torque applied by the robots
$\sum F$	Net force applied by the robots
Λ_i	Action set of the i^{th} robot
Ψ	A set including all actions selected by the robots thus far
ϕ	Empty set
Δ	Set including the currently available actions
g_t	Global reward
b_t	An additive noise process which models the effect of the unobservable states on the global reward
z_t	A Gaussian random variable
μ	Mean
σ_w^2	Variance
$\hat{P}_t$	Covariance matrix
$\|S\|$	Total number of world states
L	Direction constraint matrix
σ	Standard deviation
ρ	Composite behavior

Abbreviations

MDP	Markov Decision Process
GA	Genetic Algorithm
RL	Reinforcement Learning
SQKF	Sequential Q-learning with Kalman Filtering
AI	Artificial Intelligence
GLIE	Greedy in the Limit with Infinite Exploration
SG	Stochastic Game
RGB	Red, Blue, and Green
HSI	Hue, Saturation, and Intensity
LAN	Local Area Network
TCP/IP	Transmission Control Protocol/Internet Protocol
RMI	Remote Method Invocation

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my supervisor, Prof. Clarence W. de Silva. Four years ago, when I first contacted him to seek a Ph.D. student position in his group, he was so kind and warm to accept my request, which enabled my memorable Ph.D. career at The University of British Columbia. In the past 4 years, Prof. de Silva has given me constant supervision, support, advice, and help. These include, to name a few, supervising my Ph.D. research, suggesting and sharpening the research ideas, discussing the research problems and helping to solve them, carefully revising and improving my writing, generously funding me to attend top conferences in the robotics and controls field and providing financial support for my Ph.D. studies, purchasing state-of-the-art robots and other equipment for my experiments, kindly appointing me as Lab Manager of the Industrial Automation Laboratory, and writing excellent recommendation letters for scholarships and job applications. In particular, it was Prof. de Silva who suggested the idea of integrating machine learning with multi-robot architectures for my Ph.D. research, which has been proved to be an excellent and innovative idea as I carried out my research in the subsequent years. In a word, I am very grateful to Prof. de Silva who has helped me so much, and in my heart, he is not only a fantastic supervisor but also another respected father.

I want to thank Dr. Otman Basir from University of Waterloo, Dr. Farrokh Sassani, Dr. Jose Marti, Dr. Ian Yellowley, and Dr. Robin Turner to work as the examining committee for my Ph.D. defense exam, who carefully assessed my thesis and gave me valuable opinions. I also wish to express my appreciation to Prof. Farrokh Sassani, Prof. Mohamed Gadala, and Dr. Roya Rahbari, who have kindly served as members of my research committee. It is they who have helped and encouraged me to improve my research in these years. In addition, I also want to thank Prof. Elizabeth Croft, Prof. Ian Yellowley, Prof. Yusuf Altintas, Prof. Mu Chiao, Prof. Ryozo Nagamune and Prof. Xiaodong Lu for their valuable suggestions and assessments during our annual graduate research seminars.

I would also want to thank Ningbo University in China and K.C.Wong Education Foundation (Hong Kong) for offering me an amazing Ph.D. fellowship. In addition, I am greatly indebted for the Ph.D. tuition award from The University of British Columbia. The appreciation should also go to other generous financial support, including research assistantships and equipment funding from the Natural Sciences and Engineering Research Council (NSERC) of Canada and the Canada Foundation for Innovation (CFI) through Prof. de Silva as the Principal Investigator.

I take this opportunity to thank my close friends and colleagues in our research group; especially, Dr. Poi Loon Tang, Mr. Roland (Haoxiang) Lang, Dr. Tao Fan, Mr. Duminda Wijewardene, Mr. Jason J. Zhang, Mr. M. Tahir Khan, Mr. Arun Gupta, Mr. Mohammed Alrasheed, Prof. Lalith Gamage, Mr. Richard McCourt, Dr. Yang Cao, and Ms. F. Nazly Pirmoradi. You guys have made my life at UBC memorable and pleasant. In addition, I want to thank my previous colleagues at Ningbo University in China, especially Prof. Zhimin Feng, Dr. Chao Hu and Prof. Shirong Liu, for their warm and constant support.

Finally, I want to thank my family for their powerful support. Above all, I wish to express my exceptional appreciation to my wife Jiangfei Sun, my daughter Shihan (Lucy) Wang, my father and mother, my sister and her family, for their continuous love, support, and encouragement throughout. *I dedicate this thesis to them.*

Chapter 1
Introduction

A multi-agent robotic system is a group of autonomous robots, which are organized into a multi-agent architecture so as to cooperatively carry out a common task in a possibly dynamic environment with or without obstacles. In such a system, each robot is regarded as an agent, which possesses capabilities of independent sensing, actuation, and decision making. Besides the physical agents (e.g., robots), some pure software agents may be included into the system to help coordinate the physical agents, or implement complex machine intelligence. The group of agents constitutes an "agent society," where they cooperate, coordinate, and even compete with each other to achieve a common goal.

In the past decade, multi-agent robotic systems have received increased attention in the robotics community as they possess important advantages over a single robot system. First, a multi-agent robotic system is a parallel system, which can provide improved efficiency, especially in such tasks as mapping, searching, and space exploration. Second, through cooperation and communication, a multi-agent robotic system may become more powerful than a single complex robot, at the same cost, and may be able to complete some special tasks which a single robot is unable to carry out. Third, multi-agent robotic systems have better fault tolerant capabilities than a single robot. For example, if one robot is damaged or malfunctioned, the remaining robots in the system can takeover its role and continue to complete the task, perhaps at a reduced level of performance.

Multi-agent robotic systems face some special challenges as well. These include synchronization with regard to the locations, applied forces and actions of the robots; determination of a cooperation strategy; and the way to learn and improve such a strategy through interaction with the environment.

An important research topic in the multi-agent robotics is the multi-robot object transportation problem. In this problem, several autonomous robots move cooperatively to

transport an object to a goal location and orientation in either a static or a dynamic environment, which may contain fixed or removable obstacles. The object may be so heavy or large that no single robot will be able to handle it alone. This is a rather challenging problem. For example, in the transportation process, each robot needs to sense possible changes in the environment, the locations of the obstacles, and the other robots in the neighborhood. Then it needs to communicate with the peers, exchange sensing information, discuss cooperation strategies, suggest obstacle avoidance schemes, plan robot trajectories, assign or receive subtasks and roles, and coordinate robot actions so as to transport the object quickly and successfully to the goal location. Because of many challenges that belong to different research fields, the multi-robot object transportation problem is a good benchmark to assess the effectiveness of various multi-agent architectures and cooperation strategies. Furthermore, multi-robot cooperation has many practical applications in fields such as space exploration, intelligent manufacturing, deep sea salvage, dealing with accidents in nuclear power plants and other hazardous environments, and robotic warfare. Therefore, multi-robot object transportation problem has become an important topic in the area of multi-agent robotics.

1.1 Goals of the Research

In this thesis, a physical multi-robot transportation system operating in a dynamic and unknown environment with complex obstacle distribution will be developed. In order to complete this challenging task, several key approaches will be established. The four primary research goals of the thesis are to:

- develop a fully distributed multi-robot architecture that accommodates behavior coordination, resource assignment, and dynamic unknown environment.
- develop a new machine learning algorithm that will enable multi-robot systems to deal with uncertainties in the environment.
- develop technologies of computer vision and sensing for carrying out multi-robot tasks.
- study performance issues such as robustness, cooperative behavior, self-learning and adapting capability, speed, and accuracy of the developed methodology.

1.2 Problem Definition

A primary objective of the present work is to develop a physical multi-robot system, where a group of intelligent autonomous robots work cooperatively to transport an object to a goal location and orientation in an unknown and dynamic environment where obstacles may be present or even appear randomly during the transportation process. Robot control, multi-agent planning, and machine learning are integrated into the developed physical platform to cope with the challenges of the problem. A schematic representation of the developed system is shown in Figure 1.1.

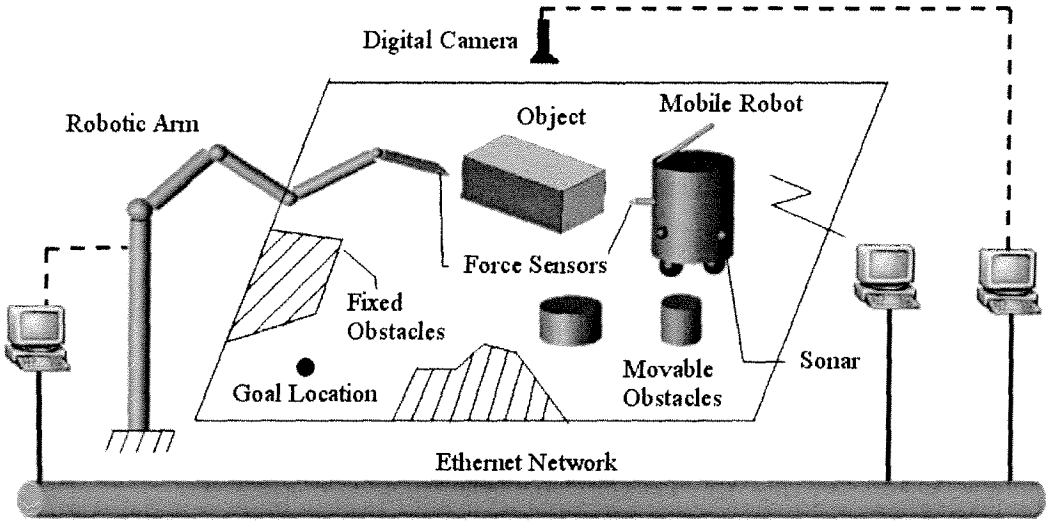


Figure 1.1: Schematic representation of the developed system.

In this system there are three or more autonomous mobile robots or fixed robotic arms, which cooperatively transport an object to a goal location. During the course of the transportation, they have to negotiate the cooperation strategy and decide on the optimal locations and amplitudes of the individual forces applied by them so that the object is transported quickly and effectively while avoiding any obstacles that may be present in the path. In some special cases, they also need to consider whether to remove the obstacles rather than negotiating around them. Other considerations such as the level of energy utilization and avoiding damage to the transported object may have to be taken into account as well. Charge Coupled Device (CCD) camera systems are used to monitor and measure the current location and orientation of the object. The environment is dynamic and unpredictable, and contains movable obstacles, which may appear randomly, and

some fixed obstacles. The robots, the camera and the sensors are separately linked to their host computers, which are connected through a local network, to implement complex controls and machine intelligence.

There are three major challenges in the developed systems, which are all key issues in the multi-robot domain. The first one is the dynamic environment. In particular, the robots are working in an environment with dynamic random obstacle distribution. The obstacles maybe appear and disappear randomly. In addition, because there are multiple robots working in parallel within the same environment, while one robot makes decisions, some other robots may have changed the environment. This also can make the environment dynamic from the standpoint of a single robot. The dynamic environment makes it very difficult for the robots to make decisions and learn useful policies because there will be hidden environment states, which are unobservable for the robots.

The second challenge results from the local sensing capabilities of the the robots. In this project, it is assumed that each robot is able to only detect an object or obstacles within a limited detection radius. It means that the robots possess local sensing capabilities only, and the global environment is unknown to them. Before a robot moves close to an obstacle, it does not know the location or the shape of the obstacle. The unknown environment is another major challenge in the multi-robot domain. Because the robots cannot know the complete environment in advance, in part due to the unfamiliarity and in part due to the dynamic nature, they cannot employ the traditional planning technologies for decision making associated with the execution of the task.

The third challenge arises due to the unreliability of communication among robots, as assumed in the present project, where the disturbances are represented by white noise. Due to unreliable communication, sometimes the robots may end up selecting a wrong cooperation strategy. Because multi-robot systems are usually expected to work in some extreme environments (for example, planetary surfaces) where communication may be unreliable due to unknown disturbances, how to improve the robustness of multi-robot decision-making in an environment with unreliable communication is a key issue in the multi-robot domain.

A typical natural environment of a multi-robot system is usually dynamic and unknown. In addition, robot communication is usually unreliable in such an environment.

In this thesis, several approaches will be developed to meet these key challenges in the multi-robot domain and enable multi-robot systems to work effectively in an unstructured natural environment.

1.3 Related Work

A considerable amount of work has been done in multi-robot cooperative control and decision-making by the robotics community in the past 15 years, particularly to support effective and robust operation of multi-robot systems in both dynamic and stationary environments. Some pertinent work is surveyed below.

Kube and Bonabeau (2000) studied cooperative transport by ants and robots, and found a "Phase Transition" phenomenon. Specifically, the ants were found to suddenly improve their performance after a learning stage. They also noticed that the ants coped with the deadlock problem in collective transportation using the method of "Realignment and Repositioning." Based on these observations, a formalized model of cooperative transportation was proposed, which included an FSM (Finite State Machine) based controller, a reactive architecture, perceptual cues, and taxis-based or kinesthetic-based actions. This biologically inspired approach was applied to a multi-robot box-pushing task and they observed that local interactions among the robots resulted in a global action. The main shortcoming of their approach stems from inefficiency for real multi-robot applications.

Cao et al. (2006) studied a multi-robot hunting task in an unknown environment. They proposed a distributed control approach involving local interactions with local coordinate systems. This approach can cope with the cumulative errors of wheels and imperfect communication networks. Computer simulations showed the validity of the proposed approach.

Rus et al. (1995) employed a team of autonomous robots to move furniture. In this project, they proposed four cooperative protocols and analyzed them in an information invariant framework, and studied whether a global controller, a global planner, explicit communication and a geometric model of the pushed object were necessary for completing the task. They also presented a pushing-tracking reorientation method for a multi-robot box-pushing task.

Kumar and Garg (2004) developed a multi-robot cooperative manipulation project with two industrial robotic arms. They employed a fuzzy logic-based approach to coordinate the motions of the two robotic manipulators so that the internal forces among them became a minimum. In addition, they used Kalman Filtering to estimate the external force on the end effector based on information from the force/torque sensor mounted on the robot wrist.

Gustafson et al. (2006) studied the scaling issues of multi-robot systems. They used several abstract models to elucidate that it was harmful to increase the number of robots or the sensor strength in multi-robot systems. Stone et al. (2006) investigated the issue of uncertainty in multi-robot systems. They identified several uncertainty sources and introduced methods for reducing uncertainty and making decisions in the face of uncertainty. These contributions enable effective planning under uncertainty for robots engaged in goal-oriented behavior within a dynamic, collaborative and adversarial environment.

Mataric et al. (1995) developed a behavior-based multi-robot box-pushing system. In their project, they proposed a behavior-based reactive architecture for multi-robot systems with explicit communication. The cooperative strategies among the robots were studied and two legged robots with minimum computational capabilities were used to demonstrate the proposed approach. In their latest work Lerman et al. (2006) presented a mathematical model for dynamic task allocation in multi-robot systems, based on stochastic processes. They assumed a large scale multi-robot system with local sensing, behavior-based controller and distributed task allocation capabilities. Through storing the history of the environment observed in the internal state of a robot, they tried to cope with the local sensing and indirect communication issues in multi-robot systems. The model proposed by them was demonstrated in a multi-robot foraging task and some statistical results and comparisons were given. In another paper written by them (Jones and Mataric, 2004a), a principled framework suitable for describing and reasoning about the intertwined entities involved in any task-achieving multi-robot systems was addressed. Using this framework, they presented a systematic procedure to synthesize controllers for the robots in a multi-robot system such that a given sequential task would be correctly executed. Gerkey and Mataric (2004) also proposed a taxonomy method for MRTA (multi-robot task allocation).

They proposed three axes for use in describing MRTA problems: (1) Single-task robots (ST) versus multi-task robots (MT). (2) Single-robot tasks (SR) versus multi-robot tasks (MR). (3) Instantaneous assignment (IA) versus time-extended assignment (TA). Based on this method they compared six typical multi-robot architectures and their MRTA approaches.

There are several useful surveys in the multi-robot domain. An older one on "Cooperative Mobile Robotics: Antecedents and Directions" was done by Cao et al. (1997). A newer overview is presented by Arai et al. (2002), where they identify seven principal areas of multi-robot systems: biological inspirations, communication, architectures/task allocation/control, localization/mapping/exploration, object transportation and manipulation, motion coordination, and reconfigurable robots. They also point out that the recent progress is beginning to advance the areas of reconfigurable robotics and multi-robot learning. Dudek et al. (2002) also proposed a taxonomy of multi-robot systems, which included seven dimensions: Collective size, Communication range, Communication topology, Communication bandwidth, Collective Reconfigurability, Processing ability, and Collective composition. In addition, a survey done by Parker (2000b) studied the existing multi-robot architectures, and pointed out several challenges in typical multi-robot tasks. In particular, she summarized that there were three multi-robot architectures in the literature, identified as: general architecture, specialized architecture and hybrid architecture. Further, she identified three typical multi-robot tasks and their main challenges. For example, in multi-robot object transportation tasks, a major challenge is uneven terrain; in multi-robot motion coordination, the key challenges are physical demonstration, extending from a 2D environment to a 3D environment, and computational complexity; For multi-robot learning, an important issue is how to assign credits among robots. Moreover, Parker argued that an important direction in the multi-robot domain was to extend the multi-agent learning approaches to multi-robot systems, where real-time constraints challenged most existing multi-robot systems.

A recent survey on multi-robot systems is the one completed by Farinelli et al. (2004). In this survey, they proposed a taxonomy-like methodology for multi-robot systems, which focused on coordination among robots. They addressed the differences between multi-robot systems (MRS) and multi-agent systems (MAS). They also classified multi-

robot tasks as unaware systems, aware but not coordinated systems, weakly coordinated systems, strongly coordinated and strongly centralized systems, strongly coordinated and weakly centralized systems, and strongly coordinated and distributed systems. Furthermore, they pointed out several research trends in the area, which involved complex coordination, distributed systems, uncertain environments and sensing, coordination model, conflict resolution, large scale systems, more complex social deliberation architecture, and weakly centralized approaches.

As Parker pointed out (Parker, 2000b), multi-robot learning has received increased attention in the past decade in order for multi-robot systems to work in dynamic and unknown environments. There are several surveys that summarize multi-robot learning or multi-agent learning; for example, the survey done by Yang and Gu (2004). In this survey, they identified that scaling an individual reinforcement learning algorithm to multi-robot systems would violate its Markov assumption, which is the drawback of many existing multi-robot reinforcement learning approaches. They also identified four frameworks of multi-agent reinforcement learning; namely, the Stochastic Games (SGs) based framework, the Fictitious Play framework, the Bayesian framework, and the Policy Iteration framework. Next, they introduced seven typical multi-agent reinforcement algorithms for possible application to multi-robot systems. They argued that there were four challenges to applying reinforcement learning to multi-robot systems; namely, the Markov assumption, continuous spaces, uncertainties, and incomplete information. Three future directions were pointed out as well; as, the behavior-based approach with reinforcement learning, fuzzy approximation functions, and continuous reinforcement learning.

Shoham et al. (2003) presented a critical survey of multi-agent reinforcement learning (Shoham et al., 2003), where they criticized that the researchers had been focusing on Nash equilibrium, which is very awkward. In addition, they identified four well-defined problems in multi-agent reinforcement learning. Another famous survey in this area was done by Stone and Veloso (2000). They proposed a taxonomy methodology for multi-agent systems from a machine learning perspective, which classified multi-agent systems as homogenous non-communicating agents, homogenous communicating agents, heterogeneous non-communicating agents, and heterogeneous communicating agents. Opportunities for applying machine learning to these multi-agent systems were

highlighted, and robotic soccer was presented as an appropriate test bed for multi-agent systems.

Panait and Luke (2005) completed a survey on cooperative multi-agent learning, where they pointed out that the huge learning space and dynamic environment were two main challenges in cooperative multi-agent learning. Simultaneously, they identified two types of learning: Team learning (Centralized learning) and Concurrent learning. Then four issues in the area of concurrent learning were discussed, as: credit assignment, learning dynamics, teammate modeling, and relationship between learning and communication. In addition, some future directions in cooperative multi-agent learning were presented, as: large-scale multi-robot systems, heterogeneity, agents with internal states (e.g., more complex computational intelligence), and dynamic team size.

1.3.1 Multi-robot Architectures

To develop a general and flexible architecture for multi-robot cooperation and coordination has been a standing objective in the robotics community. Five primary types of multi-robot architectures are discussed in the literature; namely, the behavior-based architecture, the planning-based architecture, the market-based architecture, the distributed control based architecture, and the learning-based architecture. A brief description of these architectures is given next.

1.3.1.1 Behavior-based multi-robot architecture

Parker (1998, 2000a) proposed a distributed and behavior-based multi-robot architecture called L-ALLIANCE. It uses the concepts "Impatient" and "Acquiesce" to dynamically motivate behaviors. Moreover, by assessing the performance of the teammates and dynamic changes in the environment, L-ALLIANCE autonomously updates its parameters to adapt to those changes. L-ALLIANCE was validated in a cooperative box-pushing task with two heterogeneous mobile robots. Similar research in multi-robot transportation can be found in (Rus et al., 1995), where a team of autonomous robots were employed to move furniture, and in (Yamada and Saito, 2001) where a multi-robot box-pushing system was adopted based on a reactive behavior architecture.

The behavior-based multi-robot architecture, CAMPOUT, has been employed in the Robot Work Crew (RWC) project developed in Jet Propulsion Laboratory (JPL) of NASA (Huntsberger et al., 2003; Schenker et al., 2000, 2003; Trebi-Ollennu et al., 2002; Bouloubasis et al., 2003). CAMPOUT is a fully distributed, behavior-based multi-robot architecture. It employs the leader-follower decentralized control strategy. CAMOUT was validated in a multi-robot cooperative transportation task in a PV tent autonomous deployment project on a planetary surface and a cliff robot project. In their latest work Stroupe et al. (2005, 2006) presented a multi-robot autonomous construction task based on the CAMPOUT architecture.

Pimentel et al. (2002) developed a behavior-based architecture with application in a two-robot cooperative carrying project. In this project, they considered simple obstacles in the environment and compared the effects of implicit communication with explicit communication.

Goldberg and Mataric (2002) demonstrated a successful application of behavior-based control in a task of distributed multi-robot collection. They attempted to develop controllers which were robust to noise and robot failures. Three versions of the collection task were presented and were evaluated in a spatio-temporal context using interference, time to completion, and distance traveled as the main diagnostic parameters.

Balch and Arkin (1998) proposed a formation control approach for multi-robot teams. This is a behavior-based architecture using the motor scheme approach. In this architecture, multiple behaviors can be activated simultaneously and combined cooperatively. Four types of robot team formation were controlled and switched with the proposed architecture. The developed system employed explicit communication and worked in an environment with static obstacles. In addition, they developed a behavior-based control project of a non-holonomic robot for pushing tasks where they proposed to integrate the planning approach with the behavior-based approach to cope with the local minimum problem (Emery and Balch, 2001).

In their pioneering work, Konidaris and Hayes (2005) proposed to integrate the behavior-based approach with Reinforcement Learning for multi-robot coordination. Furthermore, they suggested using topological maps to reduce the learning space in Reinforcement Learning.

Camacho et al. (2006) developed a behavior-based architecture for coordinating robot soccer agents. In their work, they assigned each robot a role for the task execution. A rule-based RECCA control algorithm was presented as well.

1.3.1.2 Planning-based multi-robot architecture

Miyata, et al. (2002) studied cooperative transportation by multiple mobile robots in unknown static environments. A centralized task planning and assignment architecture was proposed in their work. Priority-based assignment algorithms and motion planning approaches were employed. In their later paper (Yamashita et al., 2003), a motion planning approach in a 3D environment was presented.

1.3.1.3 Distributed-control-based multi-robot architecture

Wang et al. (2003a, 2003b, 2003c, 2004a, 2004b, 2005), Kume et al. (2002), Zaerpoora et al. (2005), Pereira et al. (2004), Sugar and Kumar (2002), and Chaimowicz et al. (2004) studied the caging formation control problem in multi-robot cooperative transportation. They proposed the "Object Closure" strategy to move the object with multiple mobile robots while maintaining the formation. The key issue was to introduce a bounded movable area for the object during its transportation and manipulation. In this case, the contact between the object and the robots did not need to be maintained by the controller of each robot. They termed it "Object Closure", which was employed as a type of distributed formation control strategy for multi-robot cooperative carrying tasks.

Basically, the "Object Closure" approach is a sub-category of behavior-based multi-robot coordination. However, it places its emphasis on the distributed control features.

Moreover, some other distributed-control strategies were employed by researchers to implement task allocation among multiple robots or maintain formations of a robot team. For example, Dahl et al. proposed to allocate tasks among multiple robots through vacancy chains (Dahl et al., 2003). They also demonstrated how Reinforcement Learning can be used to make vacancy chains emerge in a group of behavior-based robots. Experiments showed that the proposed algorithm outperformed random or static strategies. On the other hand, Marshall et al. proposed another distributed-control strategy for multi-robot coordination, which was called "cyclic pursuit". They demonstrated that the "cyclic

pursuit" approach was surprisingly robust in the presence of un-modeled dynamics and delays due to sensing and information processing.

1.3.1.4 Market-based multi-robot architecture

Multi-robot task allocation is a quite complex subject. It has been shown that conventional task allocation approaches are not successful in this area, especially when the number of the robots is large. However, market-based approaches seem promising in solving the complex multi-robot task allocation problem, and they are becoming popular in the multi-robot domain. For example, Gerkey and Mataric (2002a, 2002b) developed an auction-based approach for multi-robot coordination. They used the publish/subscribe first-price auction method and validated it in a multi-robot box-pushing task. A "watcher" robot observed and monitored the course of the box transportation, and "published" the required actions for use by the "pusher" robots in the same environment. The "pusher" robots understood which kinds of tasks were available at this point through "subscribing" to the information from the "watcher" robot. By matching its own skills to the required tasks, each "pusher" robot bids for a task. When the "watcher" robot received all bids from the "pusher" robots, it would select the most suitable one for each task and broadcast its decisions to the "pusher" robots. This auction-based approach was demonstrated to be quite successful in a multi-robot box-pushing task in a static environment without obstacles.

A well-known work in this area was completed by Zlot and Stenz (2006). In this paper, they integrated the market-based approach with a hierarchical task tree for complex task allocation in a multi-robot system. They demonstrated their approach in a cooperative reconnaissance task which required sensor coverage by a team of robots over a set of defined areas of interest.

1.3.1.5 Learning-based multi-robot architecture

Multi-robot systems usually work in a dynamic and unknown environment where the traditional behavior-based robots can easily fail because it is very difficult to design a sufficiently extensive behavioral rule base to cope with all possible world states encountered by the robots. Therefore, the learning capabilities are important for multi-

robot systems that work in such an environment. Most existing work in multi-robot learning employs Reinforcement Learning due to its simplicity and good real-time performance. For example, Mataric (1997) proposed a Reinforcement Learning approach for a multi-robot foraging task. In this paper, she integrated the behavior-based approach with Reinforcement Learning so as to condense the state/action space of Reinforcement Learning. She also designed a new credit assignment method called the social credit assignment, to speed up the convergence of the learning algorithm. Similar work can be found in the paper by Elahibakhsh et al. (2004) which proposed a distributed form closure strategy with Reinforcement Learning for a multi-robot cooperative carrying task.

Another interesting work in this area was completed by Ito and Gofuku (2004). They proposed a two-layer multi-robot architecture for a multi-robot cooperative transportation task. In the top level of this architecture, they employed a centralized machine learning method for decision-making. For the lower level, a distributed rule-based control strategy was developed, to control the robot movement to reach a specified position and take a specific action according to the commands from the upper level. In addition, they integrated Reinforcement Learning with Genetic Algorithms to reduce the learning space.

1.3.2 Machine Learning in Multi-robot Systems

As stated before, learning capabilities are desirable for multi-robot systems, especially when they are required to work in a dynamic and unknown environment, and machine learning technologies have been employed commonly for this purpose. Among these machine learning approaches, Reinforcement Learning and especially Q-learning, has been quite popular due to its simplicity and good real-time performance.

Parker et al. (2000a, 2002) studied multi-robot learning with regard to two aspects: Learning new behaviors and learning parameter adjustments. In order to make robots learn new behaviors autonomously, they proposed two approaches. The first one, distributed pessimistic lazy Q-learning, combines lazy (or instance-based) learning with Q-learning and uses a Pessimistic Algorithm for dealing with the credit assignment problem. The second approach, called Q-learning with VQQL(Vector Quantization with Q-learning) and Generalized Lloyd Algorithm, addresses the generalization issue in reinforcement learning. The two approaches have been validated in a CMOMMT (Cooperative Multi-robot

Observation of Multiple Moving Targets) project. In the area of learning for parameter adjustment, the L-ALLIANCE multi-robot architecture enables robots to autonomously update their control parameters so that they can adapt their behavior over time in response to changes in the capabilities of the robots, team composition, and the environment (Parker et al., 2000a, 2002; Fernandez and Borrajo, 2005).

Kapetanakis and Kudenko (2002) have summarized machine learning for autonomous agents and multi-agent systems. They established that there were two kinds of multi-agent learning: multiple single-agent learning and social multi-agent learning. In multiple single-agent learning, a learning agent regards other agents as a part of the environment and has no explicit awareness of their existence, let alone their goals and beliefs. However, in social multi-agent learning, agents can have a high awareness of other agents and incorporate this knowledge in the learning process, potentially using communication, coordination, and agent modeling techniques to support the learning task. In addition, they studied the relationships among learning, communication and roles.

Martinson and Arkin (2003) developed a multi-robot foraging task with Q-learning. They integrated Q-learning with the behavior-based approach. The learning space was reduced by the behavior representation, and Q-learning was employed to dynamically select the roles of robots. They compared the Q-learning strategy with a hand-crafted strategy and the simulation results showed that the former outperformed the latter. In another work (Martinson et al., 2002), they employed Q-learning to select behaviors for a single robot.

Park et al. (2001) proposed a modular Q-learning based multi-agent cooperation for robotic soccer. The basic idea was to use modular Q-learning to cope with the large learning space problem. Ishiwaka et al. (2003) also developed an approach to the pursuit problem in a heterogeneous multi-agent system using reinforcement learning. In their paper, they proposed the angle-based state representation in Q-learning and employed POMDP (Partially Observable Markov Decision Processes) to construct predictions of the target position and its next moving direction. In addition, Q-values for each state and action were updated via a radial basis function (RBF) due to the continuity of the sate-action space.

Dahl et al. (2004) developed a Q-learning based multi-robot cooperative transportation task in a simple environment. They proposed a two-layer multi-robot architecture where a behavior-based reactive subsystem was placed in its lower level while a Q-learning based behavior-selection subsystem was placed in the upper level. They also introduced two communication strategies to speed up the convergence of Q-learning.

Svinin et al. (2000) employed another type of reinforcement learning, Classifier System, in a multi-robot cooperative lifting task. In their project, they considered the issue of continuous space in reinforcement learning. Kawakami et al. (2001) developed a multi-robot cooperative carrying project by combining two reinforcement learning algorithms. The first reinforcement learning algorithm was used to predict the states of other robots while the second reinforcement learning algorithm was used to select the action of the current robot.

Kovac et al. (2004) addressed a "pusher-watcher" multi-robot box-pushing problem with reinforcement learning. In this project, one robot acted as a "watcher" which observed the current world state and broadcasted it to other robots. Other robots in the project acted as "pushers" which selected respective pushing actions with Q-learning. This architecture had to rely on a fully robust communication network, which represents a weakness of the approach.

Inoue et al. (2004) developed a reinforcement learning based multi-robot transportation project in which they employed machine learning to correct the robot positions in the course of transportation. Moreover, in this project they compared Q-learning with Classifier System.

Talor and Stone (2005) addressed an interesting issue in reinforcement learning: how to transfer the learned knowledge from one task to another one that has a different state-action space. The main benefit of this kind of transfer is the reduction of the training time in reinforcement learning. They used the specific domain knowledge to construct a knowledge mapping function between two tasks so that the knowledge transfer happened effectively.

Tangamchit et al. (2003) addressed several crucial factors affecting decentralized multi-robot learning in an object manipulation task. They first gave a taxonomy of multi-robot systems with the following dimensions: individual architecture (reactive,

deliberative, hybrid), group architecture (centralized vs. decentralized), number of robots (small vs. large), diversity (homogenous vs. heterogeneous), learning type (decentralized vs. centralized), learning reward (local reward vs. global reward) and learning algorithm (Q-learning vs. Monte Carlo). Then, they identified four factors affecting distributed multi-robot learning, which were the reward type, the learning algorithm, diversity, and the number of robots. They concluded that the reward type and the learning algorithm affected the final results significantly, and diversity and the number of robots could affect the learning speed but had little effect on the final results.

In order to reduce the state-action space (learning space) of reinforcement learning in multi-robot systems, Ferch and Zhang (2002) proposed a kind of graph representation of the state-action space. They demonstrated their approach in a cooperative assembly task.

It is clear that Reinforcement Learning is rather popular in multi-robot coordination. However, the applications used in this context do not have strong theoretical basis. The environment in a multi-robot task is usually dynamic because it is changed by the robots themselves. Simply extending the single-agent Reinforcement Learning algorithm to the multi-robot field violates its assumption of stationary environment (Yang and Gu, 2004). However, it is believed that this type of extension is feasible in a purely cooperative task such as multi-robot transportation.

Several multi-agent reinforcement algorithms have been developed in the multi-agent community. They include MiniMax Q-learning algorithm (Littman, 1994), Nash Q-learning algorithm (Hu and Wellman, 1998), and Friend-or-Foe Q-learning algorithms (Littman, 2001b). In particular, for purely cooperative games, Littman (2001a) suggested the Team Q-learning algorithm, which is a simplified version of the Nash Q-learning algorithm. All these algorithms assume a dynamic environment and allow each robot to observe the actions of its peers before it makes decisions. Under some conditions, these algorithms are proved to converge to the optimal policy (Yang and Gu, 2004).

The main disadvantage of the multi-agent reinforcement learning algorithms is that they result in a tricky and extensive learning space (state/action space) because each robot needs to monitor not only its own actions but also the actions of its peers. When the number of robots increases, the resulting extensive learning space will make the algorithm less effective. However, when the number of robots increases, the single agent

Q-learning algorithm still can work because its learning space is fixed.

Some researchers have tried to employ the neuro-fuzzy approach to coordinate multiple robots. For example, Pham and Awadalla (2002) presented a neuro-fuzzy based action selection architecture for cooperative robots to push a box in an obstacle-free environment. However, the neuro-fuzzy approach is not popular in the multi-robot community due to the need of complex training.

Another trend in multi-robot learning is to integrate Genetic Algorithms (GAs) with Reinforcement Learning to reduce its large learning space. A representative work in this area is done by Ito and Gofuku (2004). They tried to use GAs to search an optimal learning space for Reinforcement Learning so that it can operate within a much smaller space. Several papers discussed the benefits of integrating learning with evolution (Sen, 1998; Nolfi and Floreano, 1999). However, research in this area is still in its infancy. There are many open questions yet to be explored.

1.3.3 Computer Vision and Sensing Technologies for Multi-robot Systems

In a multi-robot system, it is important for a robot to know the latest poses (positions/orientations) of other robots and potential obstacles in the environment so as to make rational decisions. There are many methods to detect the poses of objects or peer robots in the environment; for example, using global/geographical positioning systems (GPS), sonar, laser distance finders, wireless beacons, and digital CCD (charge-coupled device) cameras. Today, the CCD camera, sonar and laser distance finder have become the typical sensors included in a standard configuration of mobile robots. As a result, there has been a need in the robotics community to develop sensing algorithms to recognize, identify and estimate features or poses of objects in a robotic work environment. For example, Stone and Veloso (1998) studied a multi-robot soccer system. In their work, they used a global camera to monitor the positions of the robots and objects in the game. Rocha et al. (2005) developed a cooperative multi-robot system to build 3-D maps of an unknown environment using the vision-based approach. Kato et al. (2003) proposed a method for identifying a robot and obtaining its relative position in a multi-robot system using an omnidirectional vision sensor. They validated their approach using the simulation and experiment results. Similar work has been done by Hajjdiab et al. (2004) where a

vision-based approach for multi-robot Simultaneous Localization and Mapping (SLAM) was proposed. They proposed to calculate the locations of the robots using a collection of sparse views of the planar surfaces on which the robots were moving on. The camera motions were estimated using inter-image homographes computed by matching the overhead transformed views.

Simultaneous tracking of multiple moving objects with vision sensors is an important topic in road traffic control systems. For example, Veeraraghavan et al. (2003) developed a computer vision algorithm to track the vehicle motion at a traffic intersection. A multilevel approach using a Kalman filter was presented by them for tracking the vehicles and pedestrians at an intersection. The approach combined low-level image-based blob tracking with high-level Kalman filtering for position and shape estimation. Maurin et al. (2005) presented a vision-based system for monitoring crowded urban scenes. Their approach combined an effective detection scheme based on optical flow and background removal that could monitor many moving objects simultaneously. Kalman filtering integrated with statistical methods were used in their approach. Chen et al. (2005) presented a framework for spatiotemporal vehicle tracking using unsupervised learning-based segmentation and object tracking. In their work, an adaptive background learning and subtraction method was applied to two real-life traffic video sequences in order to obtain accurate spatiotemporal information on vehicle objects. Rabie et al. (2005) developed a mobile bus-mounted machine vision system for transit and traffic monitoring in urban corridors. They used a new computer vision approach: the active vision paradigm, which has mechanisms that can actively control camera parameters such as orientation, focus, and zoom in response to requirements of the task and external stimuli. Siyal (2004) proposed a novel neural network and window-based image processing technique for road traffic applications. In particular, they used morphological edge detection techniques to detect vehicles. Once the vehicles were detected, a back-propagation (BP) neural network was used for calculating various traffic parameters. Deng et al. (2005) developed an adaptive urban traffic signal control system using vision sensors. They integrated and performed vision-based methodologies that included object segmentation, classification and tracking methods to determine the real-time measurements in urban roads.

The laser distance finder and the CCD camera are two standard sensing devices which are commonly used in mobile robots today. The laser distance finder can accurately detect objects within a radius of 50 meters. It can scan the surrounding environment and return a distance measurement for each degree in a 180 degree view. Alternatively, the CCD camera can return rich vision information instead of the distance information. It can detect

the shape, color and surface features of the objects of interest. Therefore, it is a natural way for researchers to combine the two sensors for object recognition and localization. For example, Alenya et al. (2005) used laser and vision to locate a robot in an industrial environment. They combined the precision of laser-based local positioning with the flexibility of vision-based robot motion estimation. Their proposed approach was validated in the warehouse of a beer production factory. Tomono (2005) employed a mobile robot equipped with a laser distance finder and a monocular camera to build a 3-D environmental model. In this framework, they first built a 2-D map by scanning objects in the environment with a laser sensor, and then verified the detected candidates by using vision-based object recognition.

Takahashi and Ghosh (2005) integrated a laser range finder sensor with a vision sensor to identify the parameters of a moving object. They showed that this approach could reduce the dimension of the parameter ambiguity. Murarka et al. (2006) presented a hybrid method using laser range-finders and vision for building local 2D maps to help an intelligent robotic wheelchair for distinguishing between safe and hazardous regions in its immediate environment. In this project, laser range-finders were used for localization and mapping of the obstacles in the 2D laser plane while vision was used for detection of the hazards and other obstacles in the 3D space. The information on the hazard and the obstacle was then combined to construct a local 2D safety map.

Frontoni and Zingaretti (2005) compared the sonar-based Monte Carlo Localization approach with the vision-based localization approach in terms of localization accuracy, and found that both approaches bore some weaknesses in an environment with high perceptual aliasing. However, it was shown that the overall accuracy was improved when the approaches were combined. Asensio et al. (1999) developed a goal directed reactive robot navigation system with relocation using laser and vision. In their project, the robot first selected the initial goal location for the navigation task, and laser was used to accomplish reactive navigation while avoiding obstacles. An extended Kalman filter was used to solve the data association problem. Ortin et al. (2004) presented a method for solving the localization problem using 2D laser and vision. A 2D laser scan together with its corresponding image was obtained and it was segmented into textured vertical planes which allowed safe plane recognition. Gopalakrishnan et al. (2005) also developed a mobile robot navigation project which integrated laser and vision. The laser-based localization, vision-based object detection, and route-based navigation techniques for a mobile robot were combined.

1.4 Contributions and Organization of the Thesis

This thesis researches and develops new techniques and expertise which will help multi-robot systems operate in a robust and effective manner in dynamic and unknown environments. There are four main contributions in the thesis, as follows:

- A fully distributed hierarchical multi-robot architecture is developed, which integrates the techniques of machine learning, path planning, behavior-based paradigm and low-level control. This architecture enables multi-robot systems to plan their sub-tasks dynamically, learn and adapt to new unknown environments, promote cooperation among robots, detect any local minimum problems, switch decision-making methods online, and complete the desired tasks in robust and effective manner.
- A modified reinforcement learning algorithm termed the Sequential Q-learning with Kalman Filtering (SQKF), is developed, which is designed to deal with some crucial issues in multi-robot learning, such as credit assignment, reducing the learning space, and Markov assumption. The proposed SQKF algorithm is compared with the conventional single-agent Q-learning algorithm and Team Q-learning algorithm through computer simulation, and it is shown that the SQKF algorithm outperforms the conventional Q-learning algorithms.
- A fast computer vision algorithm is developed to track multiple objects simultaneously, and applied to a multi-robot object transportation task. The algorithm is validated using physical experimentation.
- A physical multi-robot transportation project, which integrates the developed techniques, is developed in-house. This multi-robot system works in a dynamic and unknown real-world environment and shows adaptivity, effectiveness, and overall good performance.

The organization of this thesis is as follows. In Chapter 1, the background of multi-robot systems and the research objectives of the thesis are introduced. Then the research problem of the thesis is defined and a literature survey is carried out to establish the related background work. In Chapter 2, a learning/planning based hierarchical multi-robot architecture is presented. Various techniques included in this architecture, such as the behavior-based approach, the machine learning and planning

techniques, are addressed. Their relationships are also studied. Finally, several pertinent issues of distributed control are discussed. In Chapter 3, two conventional reinforcement learning algorithms--the single-agent Q-learning and the Team Q-learning--are extended to the multi-robot domain to facilitate decision making for robots in a dynamic and unknown environment. The two algorithms are assessed through a multi-robot transportation project. The simulation results are analyzed, and the advantages and disadvantages of the two algorithms are assessed. In Chapter 4, a modified reinforcement learning algorithm termed the Sequential Q-Learning with Kalman Filtering (SQKF), which is suitable for multi-robot decision-making, is proposed and developed. Simulation results are presented to show that the modified algorithm outperforms the conventional Q-learning algorithms in multi-robot cooperative tasks. In Chapter 5, a fast color-blob tracking algorithm is developed to support robots to track multiple objects of interest in an environment with uneven illumination. The algorithm is validated in a multi-robot object transportation project. In Chapter 6, a physical multi-robot cooperative transportation system operating in a dynamic and unknown environment is developed and presented. A JAVA RMI distributed computing model is introduced to allow robots to communicate and cooperate effectively. Multiple experimental results are presented to demonstrate the performance of the developed system, particularly concerning adaptivity and robustness. Chapter 7 summarizes the primary contributions of the thesis and indicates some relevant issues for future research.

Chapter 2
Learning/Planning Based Hierarchical Multi-robot Architecture

2.1 Overview

A multi-robot architecture represents the organization structure of the key components and their function descriptions in a multi-robot system. It has two primary functions: to promote cooperation or coordination among robots; and to enable a member robot to make rational decisions and effectively complete the corresponding tasks.

As presented in section 1.3.1, there are five main types of multi-robot architectures as given in the literature. They are the behavior-based architecture, the planning-based architecture, the distributed-control based architecture, the market-based architecture and the learning-based architecture. Especially, the behavior-based architecture, profiting from its simplicity and robustness, is the most popular one and has enjoyed many successful applications in industrial and academic fields (Parker, 1998, 2000b; Huntsberger et al., 2003; Schenker et al., 2003; Balch and Arkin, 1998). If the designer can foresee all possible world states the robots will encounter, designs the corresponding behavior rules and carefully adjust their preference levels, a behavior-based multi-robot system will work very well in a known and structured environment.

However, most multi-robot systems need to work in dynamic and unknown environments; for example, planet surfaces or deep ocean floors, where they face several challenges. First, the behavior-based approach requires a human designer to design all behavior rules and determine their preferences for the individual robots in advance. Because it is almost impossible for a designer to foresee all possible world states or situations that the robots will encounter and design the corresponding behavior rules for them, a behavior-based robot can easily fail in a dynamic and unknown environment with a large number of states. Second, since for a dynamic and unstructured environment usually there will be a large number of possible "world" states, the designer will be forced to design an extensive rule base to deal with all these states. However, it is very difficult

and sometimes impossible to arrange the preference for each behavior rule in an extensive rule base due to the "combination explosion" problem. It is clear then that a purely behavior-based multi-robot system is doomed to be weak and not robust in a dynamic and unknown environment.

The second multi-robot architecture is the planning-based architecture which employs planning techniques of traditional Artificial Intelligence (AI) to find optimal decisions for the robots (Miyata et al., 2002). There are two types of planning-based multi-robot architectures: the centralized planning architecture and the distributed planning architecture. In a centralized planning architecture, one robot will plan actions and assign sub-tasks for all other robots in the team. Because task allocation among multiple robots is a NP (Non-deterministic Polynomial time) problem in the computational complexity theory, the main disadvantage of this approach is its computational complexity. When the number of robots is large, it is very difficult and even impossible to find an analytical solution. The only appropriate method might be to employ an AI-based optimization technique to approximate it. In addition, the fragility and communication bottleneck of a centralized system will make it weak. On the other hand, the distributed planning multi-robot architecture attempts to enable each robot to plan its actions independently, and reach a global optimal decision by merging the individual decisions of all robots. Due to its difficulty, as reported in the literature, there have few successful examples in this field. Furthermore, both centralized-planning architecture and distributed-planning architecture will easily fail in a dynamic and unknown environment because they cannot adjust the policies (action sequences) of robots quickly to adapt to a fast changing environment.

The third type of multi-robot architecture is based on distributed control strategies (Pereira et. al, 2004; Sugar and Kumar, 2002; Wang et al., 2003a, 2003b, 2003c, 2004a, 2004b, 2005). They employed a distributed control strategy such as "Form Closure," "Object Closure," or "Leader-Follower Control" to allocate sub-tasks for robots so as to control the team formation. The main disadvantage of this type of multi-robot architecture is that it is only suitable for a special multi-robot task in the class of Multi-robot Cooperative Transportation, and it is not a general architecture. In particular, the "Object Closure" approach cannot be extended to other multi-robot systems such as multi-robot

cooperative foraging and multi-robot cooperative observation. In addition, this type of multi-robot architecture also suffers from the computational complexity problem.

The fourth type of multi-robot architecture is the market-based architecture (Zlot and Stenz, 2006; Gerkey and Mataric, 2002b). Rooted in the multi-agent coordination theory, it is a new and promising approach for coordinating multiple robots in a complex environment. However, there are still many open questions in this area, and its main challenges are related to computational complexity and real-time performance.

The fifth type of multi-robot architecture is the machine learning-based architecture. Rooted in iteration and dynamic programming techniques, machine learning technologies approximate the optimal polices of the robots and attempt to avoid the computational complexity problem which affect other multi-robot architectures. In particular, due to its simplicity and good real-time performance, reinforcement learning, especially the Q-learning algorithm, has been used commonly (Ferch and Zhang, 2002; Ito and Gofuku, 2004; Martinson and Arkin, 2003; Mataric, 1997; Stone and Veloso, 1998; Yang and Gu, 2004; Liu and Wu, 2001). In a survey paper, Arai et al. (2002) have suggested that area of learning-based multi-robot system was one of two emerging trends in the multi-robot domain.

As discussed in Chapter 1, the learning capability is crucial for multi-robot systems to work in a dynamic and unknown environment. It will help robots to identify new world states and learn how to adapt to them. Based on the experiences of past success and failure in a dynamic environment, the robots with a learning capability can continuously adjust their action policies and improve the performance, thereby completing the required tasks in a quicker and more robust manner.

It has been shown, however, that simply extending learning algorithms that are designed for a single robot to a multi-robot system is not appropriate (Yang and Gu, 2004). In particular, our own work has shown that extending the single-agent Q-learning algorithm to the multi-robot domain violated its assumption of a stationery environment, and made the learning algorithm not converge to its optimal policy. In addition, a purely learning-based multi-robot system cab be easily trapped into a local minimum (Wang and de Silva, 2006a, 2006b).

From these observations it is clear that the problem of multi-robot decision and control is a challenging one. No single approach can solve all the problems associated with it. Accordingly, the combination of several existing mainstream techniques may be necessary to build a new multi-robot architecture that can support multi-robot cooperative operations in a dynamic and unknown environment, effectively and in a robust manner. In this chapter, a new hierarchical multi-robot architecture is proposed, which integrates machine learning, planning, and behavior-based approaches. In section 2.2, a preliminary introduction of this general architecture is given. Its detailed implementation is presented in the subsequent sections. Finally, some distributed control issues in multi-robot systems are discussed. The multi-robot architecture developed in this chapter will be validated with computer simulations and physical experiments in the subsequent chapters.

2.2 General Architecture

In order to support multi-robot systems that work in a dynamic and unknown environment, a hierarchical multi-robot architecture that offers improved robustness and real-time performance is developed in this chapter. This architecture, presented in Figure 2.1, will integrate machine learning, planning and behavior-based approaches for robotic decision-making.

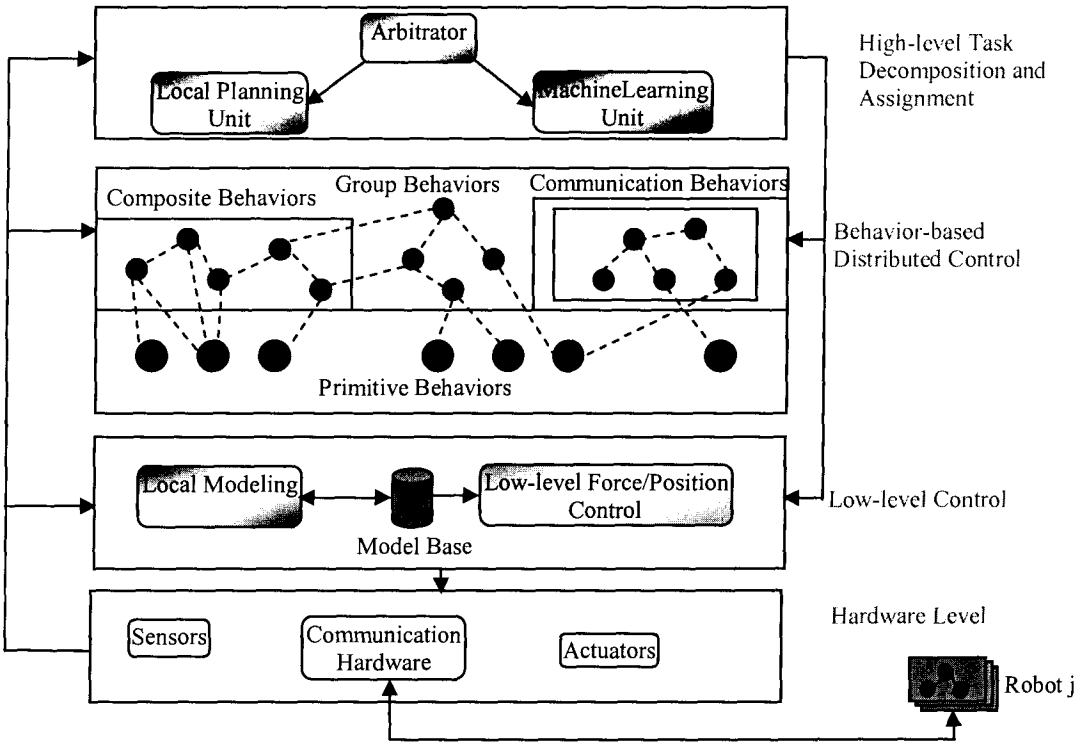


Figure 2.1: A general and hierarchical multi-robot architecture.

This is a fully distributed multi-robot architecture which attempts to integrate several popular artificial intelligence (AI) techniques for decision-making in a dynamic, unstructured, and complex environment. By organizing different techniques in a hierarchical manner, this architecture provides the robots with flexible and robust capabilities of decision-makings to adapt to the external environment dynamically. It has four layers which are: High-level Task Decomposition and Assignment, Middle-level Behavior-based Control, Low-level Robot Force/Position Control, and Hardware Level.

In its top level, there is a learning unit and a planning unit, which are used to implement inter-robot coordination. In particular, a specific robot will collect information from its own sensors and from the sensors of the peer robots through the communication network, determine the current world state, and decide its actions with its learning-based or planning-based units to reach the goal of the robot team. Therefore, this level will carry out inter-robot task decompositions and assignment.

As noted before, when making decisions, each robot may use a learning-based unit or a planning-based unit. An arbitrator will monitor the current world state and decide which

unit will be used to make decisions for the robot. For example, when the robots work in a dynamic environment, the learning-based unit is usually selected by the arbitrator to decide actions for the robot. However, once the arbitrator realizes that the whole multi-robot system is trapped in a local minimum, it will turn off the learning unit temporarily and switch on the planning unit to plan actions for the robots in a local environment. When the robots escape the local minimum, the arbitrator will return the decision-making capability to the learning unit. Through this type of switch mechanism, the multi-robot architecture benefits from both learning and planning, and can flexibly adapt to various environments.

In the middle level of the architecture, a popular behavior-based approach is employed to implement robust control of robot actions. As stated in Section 2.1, the behavior-based approach is known to be effective in multi-robot cooperative control. Here, it is employed to implement sub-tasks as decided by the top level. When the top level of the architecture establishes the current sub-tasks for the robots, they will be sent down to the behavior-based control level so as to be decomposed in further detail of robot behavior. There are four types of behavior: group behaviors, composite behaviors, communication behaviors and primitive behaviors. The group behaviors implement coordination among robots; for example, "Follow the leader robot," and they are usually fired by the learning unit or the planning unit in the top level of the robot architecture. The communication behaviors are responsible for communication among robots. Composite behaviors control the actions of individual robots; for example, "Avoid obstacle," and they can be decomposed further into primitive behaviors such as "Turn left" or "Go straight." Through the design of hierarchical behaviors and by carefully arranging their preferences, the robots can complete the required tasks as defined by the top level, in a effective and robust manner.

The third level of the architecture is the low-level control layer. In this layer, various low-level "hard" controllers, such as proportional-integral-derivative (PID) controllers or force controllers, can be designed to accurately control the actions of the robots. The required values of the controllers are established by receiving commands from the behavior-based control layer. In addition, a local modeling unit may be designed to model local interactions among the robots or between one robot and the manipulated object (or, work environment) based on the online position/force information of the robots.

The lowest level of the architecture is the hardware level. It includes sensors, actuators, communication hardware, video cameras of robots. They constitute the hardware platform of the multi-robot architecture.

The first two levels of the architecture implement "soft" control of a robot while the third level implements "hard" control. In particular, the top level is employed to decompose and assign tasks among robots; the behavior-based control layer decomposes the actions of a single robot further into more detailed primitive behaviors; and the third level controls motions and forces of the robot with "hard" control laws. By combining established mainstream approaches in multi-robot coordination, such as learning, planning and behavior-based methods, this multi-robot architecture provides more flexiblity and cooperating power in a dynamic, unknown and unstructured environment than those presented in literature.

The proposed multi-robot architecture will be validated through computer simulation and experimentation, as presented in the following chapters.

2.3 Machine Learning Unit

A machine learning unit is designed in the proposed multi-robot architecture and is placed in its top layer. The objective of this unit is to learn optimal decision-making policies in a given environment through continuously observing actions of the robots and the corresponding environmental rewards. The principle of the machine learning unit is presented in Figure 2.2.

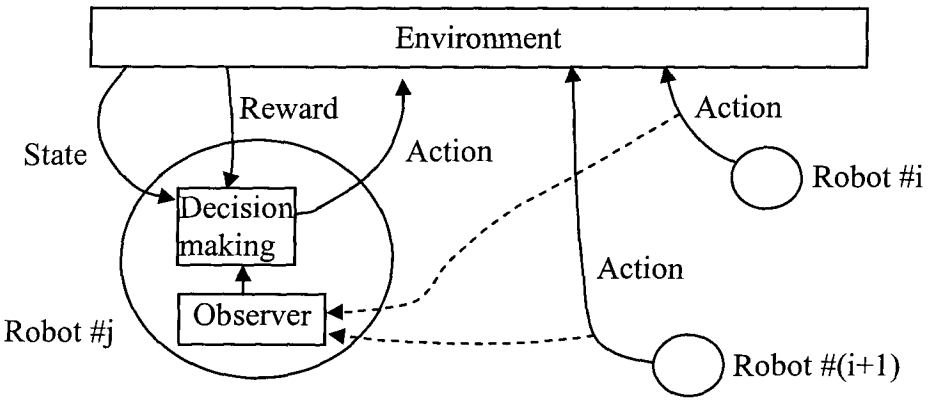


Figure 2.2: The principle of the machine learning unit.

In Figure 2.2, each robot gains experience by continuously observing the current environmental state, selecting and taking an action, and getting a reward from the environment. Simultaneously, it also needs to observe actions of its peer robots and assess their effects on the environment. Based on the past experience with the work environment, each robot employs a machine learning technique, as developed in the thesis, to continuously improve its performance and approximate its optimal decision-making policy.

In particular, with the help of the machine learning unit, the robot dynamically adjusts the mapping relationship between the world (work environmental) states and its actions. Therefore, it will outperform a robot that may use a conventional behavior-based approach whose behavior rules and rule preferences are fixed.

In this thesis, Q-learning will be employed and enhanced to learn and improve the mapping relationship between the world states and the robot actions. Here, the main challenge is how to extend the single-agent Q-learning algorithm to the multi-robot domain. Chapter 3 and Chapter 4 will discuss this issue in detail.

2.4 Planning Unit

2.4.1 Planning and Learning in Multi-robot Applications

A multi-robot architecture integrating the behavior-based approach with machine learning works well in a complex environment even when the environment is dynamic and unknown. However, in our previous research (Wang and de Silva, 2006a), it was shown that with this type of architecture, sometimes the multi-robot system may be trapped in a local minimum, as exemplified in Figure 2.3.

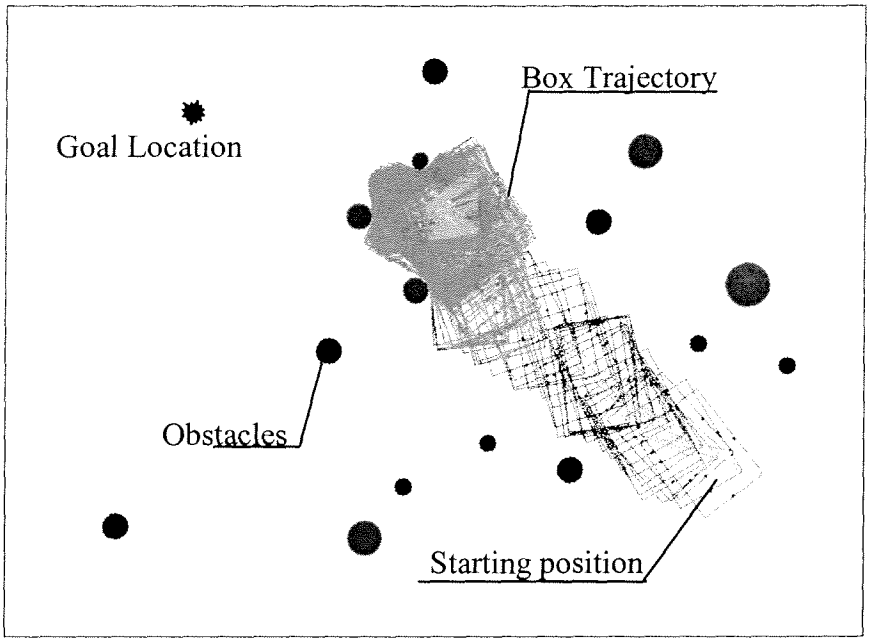


Figure 2.3: An example of the local minimum phenomenon in a purely Q-learning-based multi-robot cooperative box-pushing task.

Figure 2.3 shows a multi-robot cooperative box-pushing task which employs the distributed Q-learning algorithm. This project will be described in detail in the next chapter. In the present section, it is used just to demonstrate why planning capabilities are still necessary for multi-robot tasks. In Figure 2.3, there are three autonomous robots which attempt to push a rectangle box cooperatively from its starting position to a goal location. Many challenges exist. For example, there are obstacles distributed randomly in the environment and each robot possesses local sensing capabilities only, which means a robot does not know about an obstacle before it moves sufficiently close to the obstacle. Another challenge is that the box is very long and heavy so that no single robot will be able to handle it alone. The robots have to find optimal cooperative pushing strategies to move the box to the goal location while avoiding collision with any obstacles in the environment. In our previous work (Wang and de Silva, 2006a, 2006b), a Q-learning based distributed multi-robot architecture was proposed to complete the task. While it was successful in most of trials, some failure examples like the one shown in Figure 2.3 were observed.

In Figure 2.3, it is noted that the multi-robot system enters a local minimum area and the box is stuck there. Even though there was a possible trajectory for the robots to move the box out of the local minimum area, the learning-based robots could not find it. This example shows that a purely learning-based multi-robot system sometimes fails in a complex environment, due to the local minimum problem. In other words, a multi-robot system with learning capability alone is not strong enough to deal with all possible environmental states.

However, if the multi-robot system in Figure 2.3 possesses some local path planning capabilities, it will be able to get out of the local minimum easily because of the presence of at least one path for the robots to avoid the obstacles and move the box to the goal location.

It follows that, both learning and planning are important for a multi-robot system to successfully complete cooperative tasks in a dynamic and unknown environment in the presence of complex obstacle distribution. The learning capability will help robots to adapt to a fast changing environment and improve their performance continuously while the planning capability will overcome shortcomings of a purely learning-based decision-making system and make it escape any local minimum.

In the top level of the proposed multi-robot architecture shown in Figure 2.1, a local planning unit is integrated with a machine learning unit to decompose tasks among the robots. The so-called "local planning" means that planning is executed in a local world or local environment instead of a global one. In Section 2.1, it is stated that a global planning architecture suffers from shortcomings such as computational complexity and communication bottleneck. However, a local planning unit does not possess these problems or these problems are not very serious because it only needs to deal with local decision-making issues in a part of the environment. The cost of a local planning unit is it is not strong enough to make all decisions necessary for the robots to complete a task. It has to combine with other AI techniques, such as machine learning and the behavior-based paradigm, to constitute a more powerful and flexible decision-making unit for multi-robot systems.

2.4.2 Local Path Planning for Multi-robot Cooperative Transportation

As presented in Section 1.1, one of the objectives of this thesis is to develop a multi-robot cooperative transportation system which works in a dynamic and unknown environment with complex obstacle distribution. In order to reach this goal, a local path planning unit is designed. This planning unit will be activated when the learning-based decision-making unit is trapped at a local minimum, and it will temporarily plan local moving paths of the robots so as to escape the local minimum. Once the robots are away from the local minimum, the decision-making right will be returned to the learning-based unit from the planning unit.

The local planning unit is a centralized planning unit. In other words, when a local minimum is detected, all learning units of robots will stop working and transfer their decision rights to the planning unit of a leader robot whose role is designated in the design stage. Then, the leader robot will merge the sensor information of all robots and establish the current local world state. Based on the local world state, it will employ a potential-function-based path planning algorithm to plan the moving paths for all robots in the system.

Although the centralized planning approach is criticized in Section 2.1, it is acceptable in the present multi-robot local planning unit. The first reason for this is that computational complexity is not high here because only a small part of the environment is considered when planning the moving paths for the robots. The second reason is that the local planning unit is seldom activated when a multi-robot system is working. It is activated only when a local minimum is detected. Once the local minimum is escaped, it will transfer the decision-making right to the machine learning unit in the top level of the architecture. Therefore, the local planning unit only works for a very short period of time and during most of time the machine learning unit determines the action policy of a robot.

In this thesis, instead of the conventional attractive/repulsive potential function approach, the navigation potential function approach developed by Rimon and Koditschek (1992) is employed to plan the moving paths of multiple robots. The advantage of this approach is that it has only one minimum so that the planner will not be trapped at a local minimum. The navigation potential function approach is presented below.

This approach initially assumes that the local environment is bounded by a circle centered at (x_0, y_0) and has n spherical obstacles centered at $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, as presented in Figure 2.4. The objective of the planning algorithm is to find a path from (x_{start}, y_{start}) to (x_{goal}, y_{goal}) while avoiding all obstacles in the local environment.

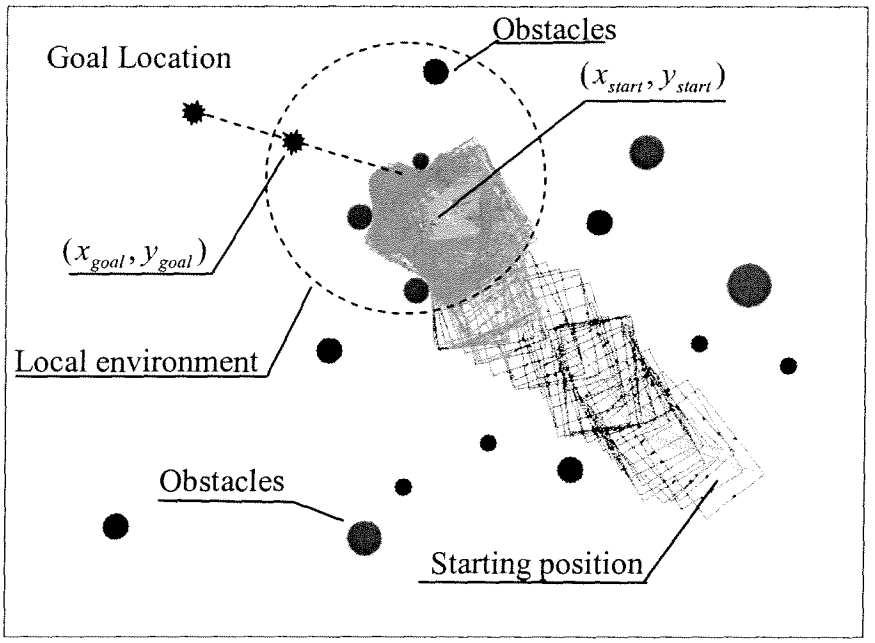


Figure 2.4: The bounded local environment in navigation potential function planning.

In order to set up the potential function of the local environment, the distance functions at a point (x, y) inside the local environment are defined as

$$\beta_0(x, y) = -(x - x_0)^2 - (y - y_0)^2 + r_0^2 \tag{2.1}$$

$$\beta_i(x, y) = (x - x_i)^2 + (y - y_i)^2 - r_i^2, \quad i = 1, 2, \cdots, n \tag{2.2}$$

where r_0 is the radius of the local environment with the center at (x_0, y_0), and r_i is the radius of the i^{th} obstacle in the local environment. Instead of considering the distance to the closest obstacle or the distance to each individual obstacle, the sum of the distance functions is used as the repulsive function of this potential field approach; specifically,

$$\beta(x, y) = \sum_{i=0}^{n} \beta_i(x, y) \tag{2.3}$$

In addition, the attractive function is defined as

$$\gamma_k(x,y) = ((x - x_{goal})^2 + (y - y_{goal})^2))^k \tag{2.4}$$

where k is a constant. Finally, the total potential function in the local environment is defined as

$$\varphi(x,y) = (\frac{\gamma_k(x,y)}{\beta(x,y) + \gamma_k(x,y)})^{1/k} = \frac{(x - x_{goal})^2 + (y - y_{goal})^2}{(((x - x_{goal})^2 + (y - y_{goal})^2)^k + \beta(x,y))^{1/k}} \tag{2.5}$$

It is shown that the potential function $\varphi(x,y)$ has a unique minimum in the local environment (Rimon and Koditschek, 1992) if k is sufficiently large. Therefore it is desirable to employ a gradient descent algorithm on the potential field to find the local goal location (the unique minimum) . The gradient descent algorithm is presented below.

Algorithm 2.1 Gradient Descent

Input: A means to compute the gradient $\nabla\varphi(x,y)$ at a point $q(x,y)$

Output: A sequence of points $\{q(x_0,y_0), q(x_1,y_1), \cdots, q(x_i,y_i)\}$

1: $q(x_0,y_0) = q(x_{start}, y_{start})$
2: i=0
3: **While** $|\nabla\varphi(x_i,y_i)| > \varepsilon$ **do**
4: $q(x_{i+1},y_{i+1}) = q(x_i,y_i) + \alpha(i)\nabla\varphi(x_i,y_i)$
5: $i = i + 1$
6: **end While**

In algorithm 2.1, ε is chosen to be a sufficiently small positive number based on the task requirement. In addition, the value of the scalar $\alpha(i)$ determines the step size at the i_{th} step. It is important to select a good $\alpha(i)$ because a small value will result in long and unnecessary computation and a big value will cause the robots to "jump" into the obstacles. Usually $\alpha(i)$ is selected based on empirical results. For example, it is determined according to the distance to the nearest obstacle or to the goal.

2.5 Integration of Learning with Planning

As discussed in Section 2.1, multi-robot planning faces challenges such as computational complexity and communication bottleneck. In particular, a planning-based multi-robot system can easily fail in a dynamic and unknown environment because it does not have the knowledge of the global environment and cannot adjust its planned policy sufficiently fast to adapt to a fast changing environment. Therefore, learning capabilities are crucial for a multi-robot system to work robustly in a dynamic environment. However, as stated in Section 2.4.1, our previous research shows that a purely learning-based multi-robot system can be easily trapped at a local minimum in a complex environment. It indicates that both learning and planning capabilities are very important for multi-robot systems when carrying out tasks in a dynamic and unknown environment.

Accordingly, in Figure 2.1, an arbitrator is designed in the top layer of the proposed architecture to select either the learning unit or the local planning unit for multi-robot decision-making. When a multi-robot system begins its work, the arbitrator selects the machine learning unit as its decision-making unit to determine its action policy, which is then communicated to its lower level: the behavior-based distributed control layer. By observing the world states, the robotic actions and the rewards received when the multi-robot system is in operation, the machine learning unit gains knowledge of cooperation among the robots and improves its performance continuously. At the same time, the arbitrator continuously monitors the progress of the multi-robot task and determines whether it is trapped at a local minimum. If the arbitrator finds that the multi-robot system is trapped at a local minimum, it will temporarily close the machine learning unit of the current robot and transfer its decision making right to a leader robot which is designated in advance. After all robots have transferred their decision-making rights to the leader robot, a centralized planning architecture is formed temporarily. Through fusing sensor information of all robots, a local environment is estimated by the leader robot. Based on this local environment, the leader robot plans the actions of the robots by employing the navigation-potential-function approach which is presented in Section 2.4.2. All the robots take actions planned by the leader robot until they escape the local minimum. When the robots are away from the local minimum, the arbitrator of each robot will return the decision-making right to its machine learning unit. The temporarily centralized planning

architecture is then dismissed and the multi-robot system returns to a learning-based distributed decision-making system.

In addition, a heuristic approach is used to recognize a local minimum for implementing the planning/learning switch mechanism, as outlined above. In particular, if $n(n \geq 3)$ successive $q(x_i, y_i)$ are found to lie within a small region of the local environment, it is possible that a local minimum exists. For example, if for a small positive ε_m , $|q(x_i, y_i) - q(x_{i+1}, y_{i+1})| < \varepsilon_m$, $|q(x_i, y_i) - q(x_{i+2}, y_{i+2})| < \varepsilon_m$ and $|q(x_i, y_i) - q(x_{i+3}, y_{i+3})| < \varepsilon_m$ are detected, then it is concluded that $q(x_i, y_i)$ is close to a local minimum.

2.6 Behavior-Based Distributed Control

In the proposed multi-robot architecture as presented in Figure 2.1, a behavior-based distributed control layer is designed to control the local actions of a robot. The behavior control layer is situated between the high-level task decomposition and assignment layer and the low-level "hard" control layer. Actions selected by the high-level task decomposition and assignment layer are usually quite abstract and cannot be directly sent to the low-level "hard" control layer for execution. Therefore, these high-level abstract actions need to be decomposed into more detailed actions in the layer of behavior-based distributed control.

Here, a hierarchical behavior control architecture, which is based on the CAMPOUT architecture developed at the NASA Jet Propulsion Laboratory or JPL (Huntsberger et al., 2003), is developed to control the robot behavior in a robust manner. The general idea of the behavior control layer is presented in Figure 2.5.

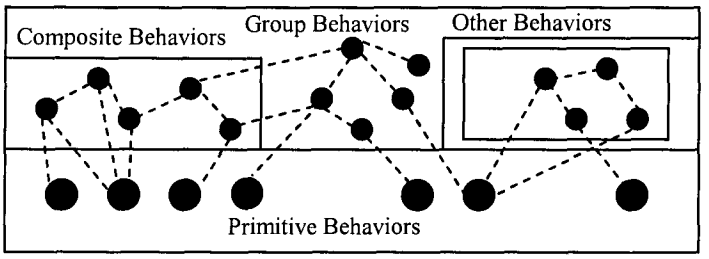


Figure 2.5: The hierarchical behavior-based control layer.

The so-called "Behavior-based systems" are systems that use behaviors as basic building blocks of robotic actions. Here, a behavior is a reaction to a stimulus. Therefore, in some literature, behavior systems are also called reactive systems. In such a system, the use of an explicit abstract knowledge representation is usually avoided. Actions of robots are coupled tightly with their perceptions without a knowledge reasoning process or time history. The ideas of behavior-based systems are derived from animal models, which have been proved to be quite successful in AI research.

There are several advantages of behavior-based systems over planning-based systems. First, a behavior-based system is simple and easy to design. By tightly coupling perceptions with actions and avoiding complex time-consuming reasoning and planning, it is much easier to design a behavior-based system than a traditional AI system with planning and reasoning. In addition, it is shown that planning is often "useless" in a fast changing environment because it cannot adapt to it. Second, the behavior-based approach has shown good performance of robustness in simulated and physical robotic systems. If the designer can foresee all possible world states and arrange the corresponding behavior rules to deal with them, a behavior-based system can be quite successful even in a dynamic and unknown environment. Third, through the interaction of existing behaviors, some "emergent behaviors" or new behaviors generated by uncertainties in the environment will occur automatically, which can surprise the designer.

2.6.1 Behavior Representation

Here, behaviors are expressed using Stimulus-Response (SR) diagrams. Figure 2.6 shows a SR diagram.

Figure 2.6: A SR diagram of a simple behavior.

In Figure 2.6, a behavior can be expressed as a 3-tuple (S, R, β) where S denotes the set of all possible stimuli, R represents the range of the response, and β denotes the mapping function between S and R.

An individual stimulus s ($s \in S$) is coded as a binary tuple (p, λ) where p indicates the type of stimulus and λ denotes its strength. In addition, there is a threshold value τ for each stimulus. If the stimulus strength is bigger than τ, a response will be generated.

Further, for mobile robots, a response r can be expressed with a six-dimensional vector consisting of six subcomponents. Each subcomponent encodes the magnitude of the translational and orientational responses for each of the six degrees of freedom of motion.

$$r = [x, y, z, \theta, \phi, \varphi], \quad where \ r \in R \tag{2.6}$$

For ground-based mobile robots, the response vector can be further simplified as:

$$r = [x, y, \theta] \tag{2.7}$$

where x and y denote the position of the mass center of the robot, and θ represents its heading.

For a particular behavior β, a gain value g is to be defined to modify the magnitude of its response further. Finally, a behavior is expressed as

$$r' = g * r = g * \beta(s) \tag{2.8}$$

2.6.2 Behavior Composition

In Figure 2.5, simple primitive behaviors can be combined to build more abstract composite behaviors. Composite behaviors are behavior packages on which a behavior-based robot system is built. Each composite behavior includes a behavior coordination operator and a number of behavioral components (primitive behaviors or other composite behaviors).

The power of composite behaviors results from abstraction of behaviors in which high-level behaviors can be built on some simple behaviors, and they can be referred to without needing to know its behavioral components.

A formal expression of a composite behavior is given by

$$\rho = C(G * R) = C(G * B(S)) \tag{2.9}$$

where

$$R=\begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix},\ S=\begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix},\ G=\begin{bmatrix} g_1 & 0 & \cdots & 0 \\ 0 & g_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & g_n \end{bmatrix},\ \text{and}\ B=\begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}$$

In equation (2.9), B represents the n behaviors $(\beta_1,\beta_2,\cdots,\beta_n)$ which constitute the current composite behavior. Also, S denotes a vector of all detectable stimuli relevant for each behavior β_i at time t ; R denotes a vector of responses of n behaviors $(\beta_1,\beta_2,\cdots,\beta_n)$ at time t; G denotes a gain matrix which modifies the response matrix R ; and C denotes the behavior coordination operator of this composite behavior.

There are two types of behavior coordination operators: the competitive operator and the cooperative operator. Figure 2.7 shows the SR diagram of a composite behavior with a competitive operator.

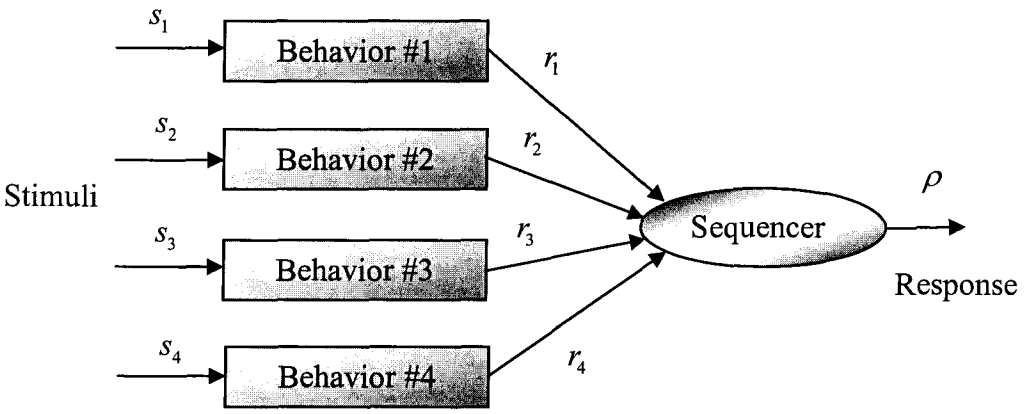
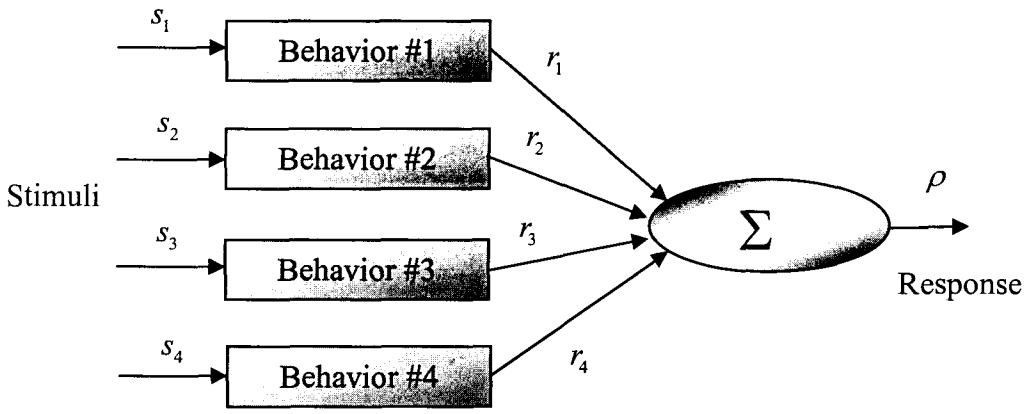


Figure 2.7: The diagram of a composite behavior with a competitive operator.

In Figure 2.7, because multiple behaviors may be activated simultaneously and their responses may conflict with each other, a competitive operator is designed to select the winner of the behaviors, and only the winner's response can be directed to the robot for execution. There are several competitive operators available in literature, such as subsumption-based arbitration, arbitration via action-selection, voting-based arbitration, and so on.

Another type of behavior coordination operator is the cooperative operator, which is shown in Figure 2.8.



2.8: The diagram of a composite behavior with a cooperative operator.

In Figure 2.8, the outputs of multiple behaviors are fused and directed to the robot for execution. It indicates that multiple behaviors are executed by the robot at a time if they do not conflict with each other. The most straightforward cooperative operator is performed through vector addition, which simply adds the outputs of multiple behaviors. The key to a cooperative operator is to design the behaviors so that their responses can be fused. The most popular approach is the potential field based method which defines behavior outputs as positions and orientations of the robot, which can be added easily.

2.6.3 Group Coordination

In order to manage and solve conflicts among robots and make them contribute to common tasks, the activities of the robots have to be carefully coordinated. Group coordination is implemented in the proposed multi-robot architecture through the machine learning unit or the planning unit which is located in its top level. The learning or planning unit continuously decomposes sub-tasks among multiple robots and assigns them to each robot. When a robot receives its sub-task, it will decompose the sub-task further into more detailed composite behaviors in its behavior-based distributed control level and finally convert them into primitive behaviors for execution.

2.6.4 Communication Behaviors

In order to coordinate the activities of the robots, communication behaviors are designed to exchange sensing information, the objectives, and internal states of the robots.

There are two types of communication: explicit communication and implicit communication. Explicit communication concerns sending or receiving information via express data links; for example, a computer network. However, a robot also can indirectly exchange information with its teammates by interacting with the environment or using some sensing approaches. For instance, it can estimate its relative distance from another robot by using its CCD camera or laser distance finder.

2.6.5 Coordinated Object Transportation

In this thesis, a multi-robot coordinated object transportation system is developed. In order to facilitate a robot to complete its sub-tasks as assigned by the learning/planning units in the top level of the proposed multi-robot architecture, a behavior-based distributed control subsystem which implements Figure 2.5 is developed, as shown in Figure 2.9.

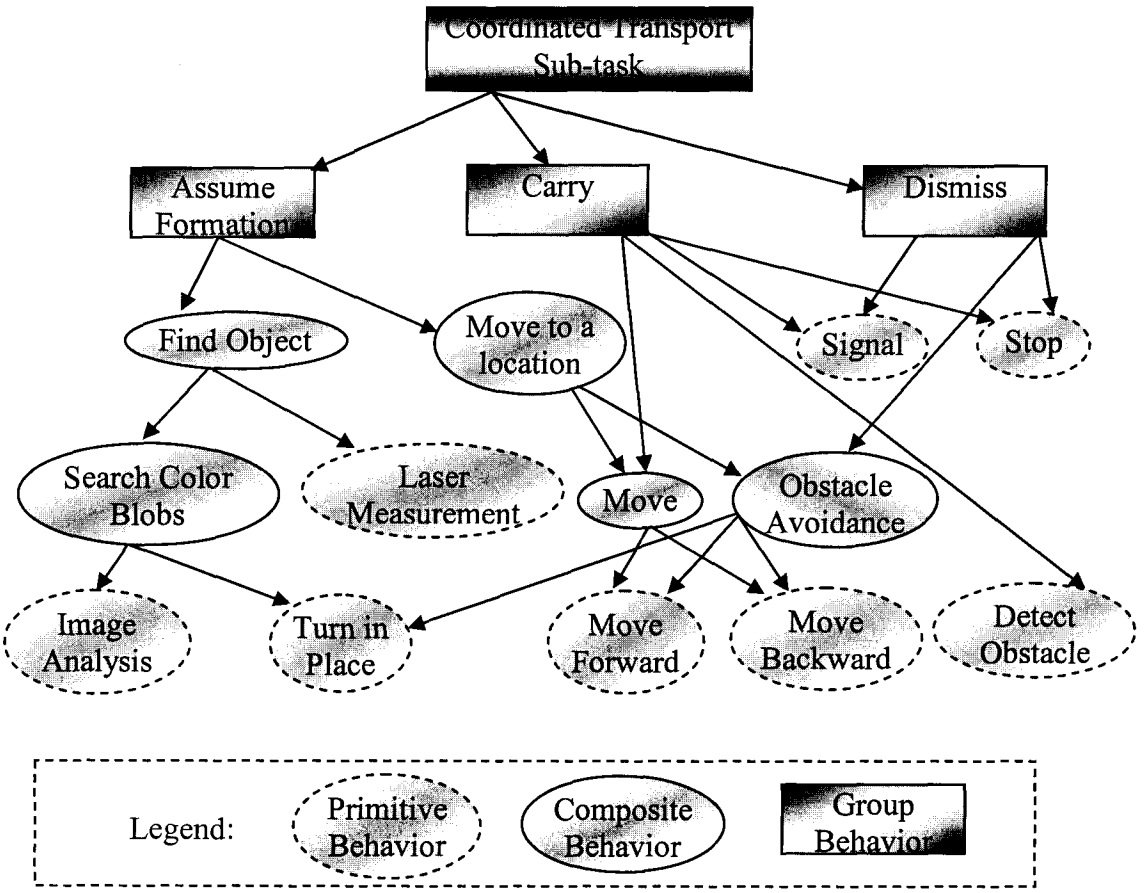


Figure 2.9: Behavioral hierarchy describing the coordinated transport sub-task.

In Figure 2.9, the group behavior of "Assume Formation" is to make the robot move to a designated site around the transported object so as to establish a formation with its peers. The learning/planning unit in the top level of the proposed multi-robot architecture determines the transportation formation of the robots, decomposes it into the specific poses and sites of the robots, and sends downward these sub-tasks to the behavior-control layer for execution. When the "Assume Formation" behavior is activated, it will first activate its child behavior of "Find Object" which attempts to search the transported object and estimates its location and orientation in the environment by suing the CCD camera and the laser distance finder. Accordingly, the "Find Object" behavior will call the "Search Color Blobs" behavior to make the robot turn in place continuously and try to find some predefined color blobs which are attached to the surface of the object of interest, by using its CCD camera, so that it can be distinguished from other objects or obstacles in the environment. When the robot finds the object of interest, the behavior of "Laser Measurement" is activated to estimate the position and orientation of the object with the laser distance finder sensor. Then, the behavior of "Move to a Location" will attempt to move the robot to the designated position as determined by its higher level learning/planning unit. As a robot moves to its goal location, the "Move" behavior and the "Obstacle Avoidance" behavior are activated in turn, depending on the obstacle distribution of the environment.

After all robots reach their designated positions and generate a formation around the object of interest, the "Signal" behavior is waked to start a synchronization process which notifies all robots to activate the "Carry" behavior so that the object is transported. While the robots transport the object, they also monitor whether the object collides with any obstacles in the environment. If the object does collide with an obstacle in the course of transportation, the robots will cancel the "Carry" behavior and report this accident to their learning/planning units in the top level and ask them to re-plan their formation.

In this thesis, Finite State Diagrams (FSA) is used to describe the aggregations and sequences of the behaviors so that the process of behavior activation and transition becomes more explicit. A finite state diagram can be specified by the quadruple (Q, δ, q_0, F) with Q representing all behavioral states; q_0 representing the initial state; F representing all final (termination) states; and δ representing the mapping function

which maps the current behavioral state q and input a into a new behavioral state q', i.e., $q' = \delta(q,a)$. The FSA describing the transitions of the group behaviors in the project of coordinated object transportation is shown in Figure 2.10.

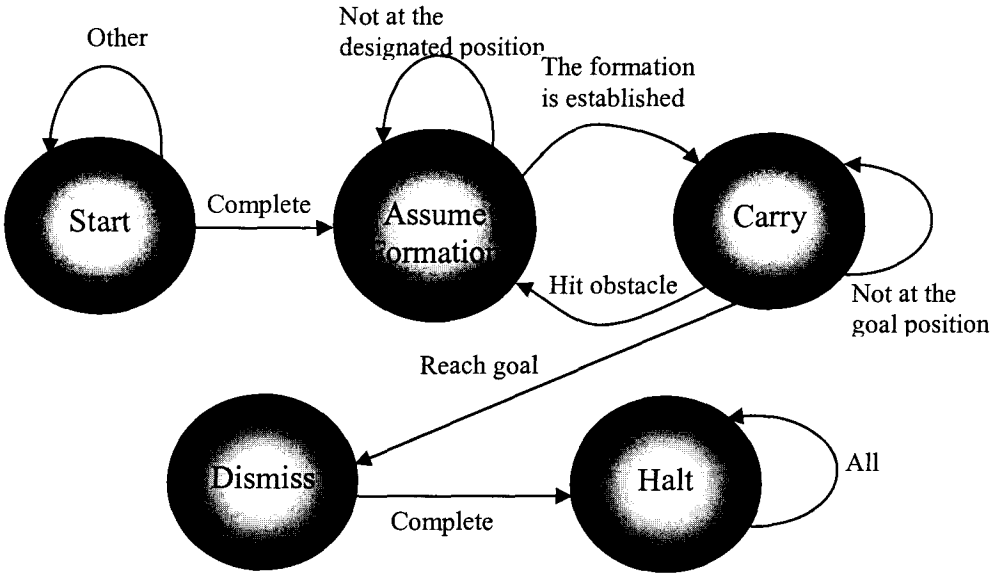


Figure 2.10: A FSA describing the transitions of the group behaviors in the project of coordinated object transportation.

2.7 Low-level Control

The third-level of the proposed multi-robot architecture presented in Figure 2.1 is the low-level "hard" control layer. This layer receives commands such as "Move Forward", "Move Backward" or "Turn in Places for a Certain Degree" from its higher level of "behavior-based control," and exactly controls the physical motion of the robot.

Basically, there are two types of robots employed in multi-robot systems. They are robotic manipulators or mobile robots whose models and corresponding control strategies are quite different. Both types are employed in the present research.

2.7.1 Robotic Manipulators

A robotic manipulator is a robot with multiple links connected with some revolute or prismatic joints which are driven by motors (typically DC motors). The robot is mounted on a fixed base in the environment and the end effector at the other end of the robot can

move in a limited workspace by controlling the rotational angles of its joint motors. Over the past thirty years modeling and control of robotic manipulators has become a mature research field. For example, the dynamic model of a robotic manipulator with n joints can be presented as (Spong et al., 2006)

$$D(q)\ddot{q}+C(q,\dot{q})\dot{q}+g(q)=\tau \tag{2.10}$$

where

$q=(q_1,q_2,\cdots,q_n)^T$ is a vector of generalized coordinates of n joints

and $D(q)=\left[\sum_{i=1}^{n}\{m_i J_{v_i}(q)^T J_{v_i}(q)+J_{w_i}(q)^T R_i(q) I_i R_i(q)^T J_{w_i}(q)\}\right]$ is the $n\times n$ inertia

matrix of the manipulator. The $(k,j)^{th}$ element of the Coriolis matrix $C(q,\dot{q})$ is defined as

$$c_{kj}=\sum_{i=1}^{n}\frac{1}{2}\{\frac{\partial d_{kj}}{\partial q_j}+\frac{\partial d_{ki}}{\partial q_j}-\frac{\partial d_{ij}}{\partial q_k}\}\dot{q}_i$$

and the gravity vector $g(q)$ is given by

$$g(q)=[g_1(q),g_2(q),\cdots,g_n(q)]^T$$

In most cases, the dynamic model given by (2.10) is not used due to its nonlinear and coupled nature, and the independent joint control approach which employs multiple proportional-derivative (PD) controllers to control each joint independently is adequate to meet the control requirements. However, sometimes, nonlinear control approaches based on the dynamic model of (2.10), such as reverse dynamics control, are employed to control the motion of robotic manipulators more accurately, at the expense of increased computational effort.

2.7.2 Mobile Robots

Mobile robots are another class of robots which are used commonly in multi-robot systems. Control issues of mobile robots are quite different from those of fixed robotic manipulators. Figure 2.11 shows a control problem of a mobile robot with two standard wheels (Siegwart and Nourbakhsh, 2004).

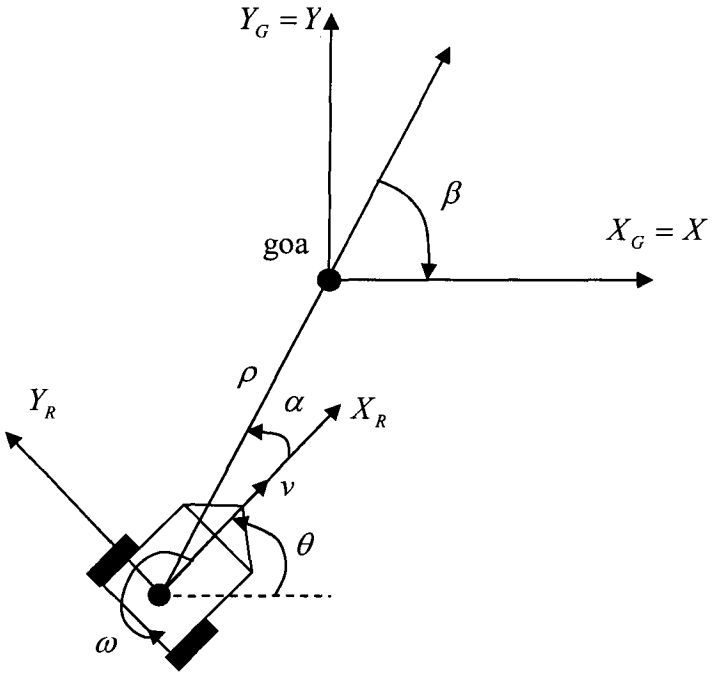


Figure 2.11 Control of a mobile robot.

In Figure 2.11, the axes X_I and Y_I define a global reference frame from the origin $O:\{X_I, Y_I\}$. The coordinate axes $\{X_R, Y_R\}$ define the robot's local reference frame with its origin at point P centered between the two drive wheels, and each wheel is located at a distance l from P. The angular difference between the global and local reference frames is denoted by θ. The diameter of each wheel is r, and the spinning speeds of the two wheel are ω_1 and ω_2. The linear and angular velocities of the robot are v and w, respectively. Given θ, l, v and w, the kinematic model of the mobile robot is presented as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}_I = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{2.11}$$

$$\begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} \frac{r\omega_1}{2} + \frac{r\omega_2}{2} \\ \frac{r\omega_1}{2l} + \frac{-r\omega_2}{2l} \end{bmatrix} \tag{2.12}$$

where $\dot{x}$ and $\dot{y}$ are the linear velocities in the directions of X_I and Y_I, respectively, of the global reference frame.

Given an arbitrary position and orientation of the robot in the global reference frame, the control objective is to drive the robot to a predefined goal position and orientation by controlling the spinning speed of the wheels of the robot. The control block diagram is shown in Figure 2.12.

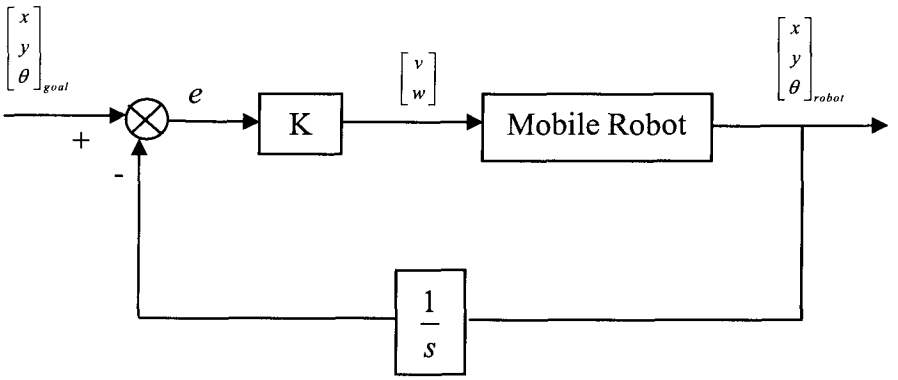


Figure 2.12: The block diagram of motion control of a mobile robot.

Here $[x, y, \theta]^T_{goal}$ and $[x, y, \theta]^T_{robot}$ give position and orientation of the goal and the robot, respectively, in the global reference frame. The task of the controller layout is to find a suitable control gain

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \end{bmatrix} \tag{2.13}$$

such that the control of $v(t)$ and $w(t)$

$$\begin{bmatrix} v(t) \\ w(t) \end{bmatrix} = K \cdot e = K\left(\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{goal} - \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{robot} \right) \tag{2.14}$$

will drive the error e toward zero:

$$\lim_{t \to \infty} e(t) = 0 \tag{2.15}$$

A control law is proposed in (Siegwart and Nourbakhsh, 2004), which may be presented as

$$v = k_\rho \rho \tag{2.16}$$

$$w = k_\alpha \alpha + k_\beta \beta \tag{2.17}$$

where

$$\Delta x = x_{goal} - x_{robot}, and, \Delta y = y_{goal} - y_{robot} \tag{2.18}$$

$$\rho = \sqrt{\Delta x^2 + \Delta y^2} \tag{2.19}$$

$$\alpha = -\theta + a\tan 2(\Delta y, \Delta x) \tag{2.20}$$

$$\beta = -\theta - \alpha \tag{2.21}$$

When the following conditions are met:

$$k_\rho > 0, k_\beta < 0, and\ k_\alpha - k_\rho > 0 \tag{2.22}$$

the closed loop control system in Figure 2.12 will be locally exponentially stable (Siegwart and Nourbakhsh, 2004).

2.8 Distributed Control Issues

The multi-robot architecture proposed in this chapter is a distributed control architecture. It means that there does not exist a leader robot to make decisions for all other robots in the team. Instead, each robot needs to decide its own actions independently by observing the environment or communicating and negotiating with its teammates. The only exception is the local path-planning unit which needs centralized decision-making when the learning and behavior-based multi-robot units are found to being trapped in a local minimum. However, the centralized planning unit works only for a very short period of time. Once the multi-robot system leaves the local minimum, the learning and behavior-based units will takeover the control of the robot again.

Compared with a centralized control architecture, a distributed architecture has many advantages, especially for multi-robot systems. The most beneficial feature of a distributed system is its scalability. For a distributed multi-robot system, it is very easy to scale the robot team without significantly increasing its computational cost. For example, the multi-robot architecture proposed in this chapter can accommodate a robot team with 3 to 20 robots, and the team size can change dynamically if some robot members quit or new members enroll in the team in the course of transportation. However, for a centralized multi-robot system, scalability is a very difficult issue because the computational complexity of the centralized decision-making algorithm increases rapidly with the number of robots. In most of cases, centralized decision-making is an N-P complexity

problem requiring superpolynomial time, which makes the algorithm useless in a real multi-robot system.

Another advantage of a distributed system is its low communication volume. Because there is no centralized leader, the communication bottleneck problem will not appear. Instead, in a centralized multi-robot system, when the number of robots increases, the communication bottleneck will be so serious that the algorithm becomes worthless. In addition, through carefully designing its local behavior rules, a distributed system can complete some cooperative tasks even without communication. It is always desirable for a distributed multi-robot system to work with a limited communication capability because most multi-robot systems need to run in an environment where communication among robots is usually unreliable; for example, on a planetary surface or the bottom of a deep ocean. When communication among robots is not reliable, while a distributed multi-robot system may still work well, a multi-robot system with centralized decision-making becomes weak because the robot members cannot receive commands from their leader in a timely and accurate manner.

The third advantage of a distributed system is its robustness. Decision-making capabilities are spread across the robot members. When one robot in the team fails or quits, other robots still can make decisions for themselves so as to continue their tasks. However, a centralized system is not a robust one. In such a system, because one leader robot makes decisions for all other robots in the team, once the leader robot fails, the whole system will totally fail.

A main disadvantage of a distributed system is its low efficiency, where multiple robots reach their agreements through some negotiation technique such as voting, auction, and so on. These negotiation approaches are usually time-consuming and not efficient. In addition, the distributed planning and learning algorithms are much more difficult and complex than their centralized counterparts. Consequently, compared to a centralized system, the final policy reached by the members in a distributed system is usually not optimal. Therefore, it is still an open question as to how to implement efficient and optimal distributed decision-making, especially distributed planning and distributed learning, for multi-robot systems.

2.9 Summary

In this chapter, a distributed multi-robot architecture, which accommodates distributed machine learning, local path planning, behavior-based distributed control and low level "hard" control, was developed. The objective of the architecture is to support multi-robot systems for carrying out cooperative tasks in a dynamic and unknown environment in an effective and robust manner.

In the beginning of the chapter, several popular multi-robot architectures given in the literature were introduced and compared, and their advantages and disadvantages were highlighted. Basically, there are five architectures that have been employed by the existing multi-robot systems; namely, the behavior-based architecture, the planning-based architecture, the distributed control based architecture, the market-based architecture, and the learning-based architecture. Although each architecture possesses some advantages in solving the multi-robot cooperative problem, all of them are generally weak in supporting a multi-robot system that operates in a dynamic and unknown environment.

In order to facilitate multi-robot systems when operating in a dynamic and unknown environment, which is quite common for multi-robot tasks, a distributed hierarchical multi-robot architecture was developed in this chapter. This architecture is a general platform that consolidates several existing techniques of multi-robot decision-making, and accommodates homogenous or heterogeneous robots in the same framework.

Next, the top level of the proposed architecture was introduced, which includes a machine learning unit, a local path planning unit and an arbitrator. By monitoring whether the multi-robot system is trapped at a local minimum, the arbitrator dynamically selects either the learning-based or the planning-based approach to make decisions for the robots. In particular, the potential-field-based path planning algorithm was introduced.

The second level of the proposed multi-robot architecture constitutes a behavior-based control layer which further decomposes the behaviors of the individual robots. The behavior-based approach has been proved to be robust and effective in the literature. In this thesis, it is combined with the learning and planning units in the top level of the architecture to control the actions of a single robot.

Below the behavior-based control layer there exists the low-level "hard" control layer. The low-level controller in this layer receives its commands from the "primitive

behaviors" in the upper level, and precisely controls the motion or force of a robot. The dynamic or kinematic models of two types of representative robots--robotic manipulators and mobile robots--were presented. In particular, a motion control law for a wheeled mobile robot was introduced.

Finally, some distributed control issues of multi-robot systems were discussed. In the inception of multi-robot systems, centralized architectures were used commonly due to their simplicity. However, centralized architectures caused many problems when the robot team size became large. The advantages of distributed architectures were pointed out and some open questions in this field were identified.

The proposed multi-robot architecture provides a framework and guideline for designing physical multi-robot systems. Its machine-learning based multi-robot decision-making will be presented in more details in Chapter 3 and Chapter 4, and the simulation and experimental results will be given in Chapter 6, which will validate this new multi-robot architecture.

Chapter 3
Machine Learning for Multi-robot Decision-Making

3.1 Overview

Multi-robot decision-making has been a challenging problem because the actions of multiple robots and their interactions have to be considered simultaneously so as to reach a global optimal policy. Intuitively, planning is the suitable approach to deal with this problem. However, it is shown that planning is usually not feasible if the environment changes fast (Arkin, 1998) because a planned optimal policy can become inapplicable due to changes of the environment.

Multi-robot environments, such as planet surfaces and urban environments are typical dynamic and unstructured environments, where the terrain features can change continuously. In addition, even though the physical environment is stationary, from the viewpoint of a single robot, it is still a dynamic one because the other robots in the team take actions in the work environment and change it continuously.

There are two solutions to the challenge of the dynamic environment: the behavior-based approach and the machine learning approach. The behavior-based approach as mentioned in Chapter 2, is known to be more robust than the planning approach in a dynamic environment. By identifying the current world state and tightly coupling it with a specific action, behavior-based systems can quickly adapt to a fast changing environment and show good robustness. However, although the behavior-based approach is quite successful in simple environments, it is shown to be weak in a complex environment with a large number of states. In particular, it is almost impossible for a designer to consider thousands or ten-thousands of world states and arrange the corresponding behavior rules and preferences for those rules. Therefore, the behavior-based approach is sometimes mocked as "the insect-level intelligence" which only can deal with a simple world.

By observing the human capabilities to deal with a dynamic environment, it is not difficult to draw a conclusion that a human employs not only planning or reactive (behavior-based) techniques but also learning techniques to successfully complete tasks in

a dynamic environment. Through learning, a human learns new world states, finds corresponding optimal actions from the past experience, and improves his planning and reactive techniques. In other words, learning makes a human deal with unexpected world states, decreases uncertainties in the environment, and makes his decisions more robust and effective in a dynamic environment.

Accordingly, machine learning has become an important topic in the robotics community. Especially in the multi-robot field, machine learning is regarded as as a new and important trend due to the highly dynamic character of typical multi-robot environments.

Among the existing machine learning approaches, Reinforcement Learning (RL), especially Q-learning, is used most commonly in multi-robot systems due to its simplicity and good real-time performance. Few multi-robot systems employ neural network or neuro-fuzzy approaches because they need complex training and longer computational time.

In this chapter, machine learning is adopted to make decisions for a multi-robot object transportation task. In section 3.2, Markov Decisions Processes (MDP) and the single-agent Q-learning algorithm are introduced. The single agent Q-learning algorithm will converge to an optimal policy in an MDP problem without a transition model, and it will be integrated with Genetic Algorithms (GA) to constitute the centralized decision-making unit in a multi-robot transportation task, as presented in section 3.3. In section 3.4, a distributed version of Q-learning based multi-robot box-pushing is presented and it is validated through computer simulation. In section 3.5, another Q-learning algorithm more suitable for multi-robot tasks, the Team Q-learning algorithm, is applied to multi-robot decision making, and it is compared with the single agent Q-learning algorithm, again using computer simulation.

3.2 Markov Decision Process (MDP) and Reinforcement Learning

Markov Decision Process (MDP) is a popular topic in the artificial intelligence (AI) community. Not only is it important in the theoretical research of AI, it also has some practical applications such as facilitating an intelligent agent to make decisions in a real-world situation. The basic idea of MDP is presented below.

MDP can be defined by the 4-tuple $<S, A, T, R, \beta>$, where

- $S = \{s_1, s_2, \ldots, s_n\}$, is a set of states of the environment
- $A = \{a_1, a_2, \ldots, a_m\}$, is a set of actions available to the robot
- $T : S \times A \rightarrow \prod(S)$, is a transition function, which decides the next environmental state s' when the robot selects an action a_i under the current state s. $\prod(s)$ is the probability distribution over the states.
- $R : S \times A \rightarrow \Re$, is a reward function, which determines the immediate reward when the robot takes an action a_i under the current state s.

MDP describes a sequence of decisions made by an agent or a robot in an environment with many states. At time t, the agent needs to observe the environment and determine its current state $s \in S$, and then select an action $a \in A$ based on its policy π which specifies what the agent should do for any state that it might reach. In addition, there is a reward function which determines the immediate reward when the agent takes an action a under the current state s.

There are two things which make MDP interesting and challenging. The first one is that the subsequent state s' is not deterministic when the agent takes an action a under the current environmental state s. Instead, it has a probability distribution $\prod(S)$ over all the states. This probability distribution is defined by a transition function or transition model $T(s,a,s')$. The second attribute of MDP is that the transitions among states are Markovian; that is, the probability of reaching s' from s depends only on s and not on the history of earlier states.

The performance of an agent policy π is measured by the rewards the agent received when it made a sequence of decisions according to this policy and visited a sequence of states. This measurement is usually represented by a sum of discounted rewards, as given by

$$\left[\sum_{t=0}^{\infty} \beta^t R(s_t) \mid \pi\right] \tag{3.1}$$

where, $0 < \beta \leq 1$ is the discount factor, and $R(s_t)$ is the reward received when the agent visits the state s_t at time t. Because the transitions among states are not deterministic in

view of the probablistic nature of the transition function $T(s,a,s')$, given a policy, the sequence of states visited by the agent each time is not fixed and has a probability distribution. Therefore, an optimal policy π^* is a policy that yields the highest expected sum of the discounted rewards, which is given by

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t=0}^{\infty} \beta^t R(s_t) \mid \pi\right] \tag{3.2}$$

Given an MDP problem, one crucial consideration is how to find all optimal policies (if there exist more than one optimal policy). The value iteration algorithm has been developed to find the solution to this problem (Russel and Norvig, 2003).

In the value iteration algorithm, first the concept of "utility of a state" is defined as

$$U(s) = E\left[\sum_{t=0}^{\infty} \beta^t R(s_t) \mid \pi^*, s_0 = s\right] \tag{3.3}$$

From equation (3.3), the utility of a state s is given by the expected sum of discounted rewards when the agent executes the optimal policy.

If we have the utilities of all states, an agent's decision making (or action selection) will become easy; specifically, to choose the action that maximizes the expected utility of the subsequent state:

$$\pi^*(s) = \arg\max_{a} \sum_{s'} T(s,a,s')U(s') \tag{3.4}$$

However, the problem is how to find the utilities of all states. It is not an easy task. Bellman found that the utility of a state could be divided into two parts: the immediate reward for that state and the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

$$U(s) = R(s) + \beta \max_{a} \sum_{s'} T(s,a,s')U(s') \tag{3.5}$$

Equation (3.5) is the famous Bellman equation. For each state, there is a corresponding Bellman equation. If there are n states totally, then we can have n Bellman equations. The utilities of the n states are found by solving the n Bellman equations. However, the Bellman equation is a nonlinear equation because it includes a "max" operator. The analytical solutions cannot be found using techniques of linear algebra. The only way to find the solutions of n Bellman equations is to employ some iterative techniques. The

value iteration algorithm is such an approach to find the utilities of all states. This algorithm is presented below (Russel and Norvig, 2003).

Function Value-Iteration (mdp, ε) returns a utility function
inputs: *mdp*, an MDP with state S, transition model T, reward function R, discount β, and the maximum error ε allowed in the utility of any state.
local variables: U, U' are the vectors of the utilities for the states in S, initially zero. δ is the maximum change in the utility of any state in an iteration.
repeat
$U \leftarrow U'; \delta \leftarrow 0$
for each state s in S **do**
$U'(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s')U(s')$
If $|U'(s) - U(s)| > \delta$ **then** $\delta \leftarrow |U'(s) - U(s)|$
until $\delta < \varepsilon(1-\beta)/\beta$

Figure 3.1: The value iteration algorithm to calculate the utilities.

3.2.1 Reinforcement Learning

In section 3.2, the value iteration algorithm is introduced to solve MDP problems. It appears that MDP is no longer a difficult issue. However, in a real robot decision-making situation, the value iteration algorithm is not practical because the environment model $T(s,a,s')$ and the reward function $R(s)$ are usually unknown. In other words, it is usually impossible for one to obtain perfect information of the environment model and employ the value iteration algorithm to solve the Markov Decision Process (MDP) problem. Reinforcement Learning is developed to meet this challenge.

In reinforcement learning, through trials of taking actions under different states in the environment, the agent observes a sequence of state transitions and rewards received, which can be used to estimate the environment model and approximate the utilities of the states. Therefore, reinforcement learning is a type of model-free learning, which is its most important advantage.

There are several variants of reinforcement learning among which Q-learning is the most popular one. A typical Q-learning algorithm is presented below.

- For each state $s_i \in (s_1, s_2, \cdots, s_n)$ and action $a_j \in (a_1, a_2, \cdots, a_m)$, initialize the table entry $Q(s_i, a_j)$ to zero.
  Initialize τ to 0.9. Initialize the discount factor $0 < \beta \leq 1$ and the learning rate $0 < \eta \leq 1$.
- Observe the current state s
- Do repeatedly the following:
  - Probabilistically select an action a_k with probability
    $P(a_k) = \frac{e^{Q(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{Q(s,a_l)/\tau}}$, and execute it
  - Receive the immediate reward r
  - Observe the new state s'
  - Update the table entry for $Q(s, a_k)$ as follows:
  - $Q(s,a_k) = (1-\eta)Q(s,a_k) + \eta(r + \beta \max_{a'} Q[s', a'])$
  - $s \leftarrow s'$, $\tau \leftarrow \tau * 0.999$

Figure 3.2: The single-agent Q-learning algorithm.

At the initiation of the Q-learning algorithm, an empty Q-table is set up and all its entries are initialized to zero. The Q-table is a 2D table whose rows represent the environment states and whose columns represent the actions available to the agent (the robot). Here, the value of a Q-table entry, $Q(s_i, a_j)$, represents how desirable it is to take the action a_j under the state s_i, while the utility $U(s_i)$ in section 2.2 represents how appropriate for the agent to be in the state s_i. Parameters such as the learning rate η, the discount factor β and the " temperature " parameter τ, have to be initialized as well.

During operation, the agent observes the environment, determines the current state s, probabilistically selects an action a_k with probability given by (3.6) and executes it.

$$P(a_k) = \frac{e^{Q(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{Q(s,a_l)/\tau}} \tag{3.6}$$

After the agent takes the action a_k, it will receive a reward r from the environment and observe the new environment s'. Based on the information of r and s', the agent will update its Q-table according to

$$Q(s,a_k) = (1-\eta)Q(s,a_k) + \eta(r + \beta \max_{a'} Q[s',a']) \tag{3.7}$$

In this manner the current environment state is transited from s to s'. Based on the new state, the above operations are repeated until the values of the Q-table entries converge.

In the Q-learning algorithm described in Figure 3.2, a $\varepsilon - greedy$ search policy presented in equation (3.6) is used instead of a greedy policy which always selects the action with the maximum Q value. In a $\varepsilon - greedy$ search policy, with probability ε, the agent chooses one action uniformly randomly among all possible actions, and with probability $1-\varepsilon$, the agent chooses the action with a high Q value under the current state. In addition, the probability ε is decreased gradually. The advantage of the $\varepsilon - greedy$ search policy is its balance in exploring unknown states against exploiting known states when the agent attempts to learn its decision-making skills and improve its Q-table. In equation (3.6), the probability ε is controlled through decreasing the " temperature " parameter τ.

3.3 Multi-robot Transportation Using Machine Learning: First Version

An important research topic in multi-agent robotics is multi-robot object transportation. There, several autonomous robots move cooperatively to transport an object to a goal location and orientation in a static or dynamic environment, possibly avoiding fixed or removable obstacles. It is a rather challenging task. For example, in the transportation process, each robot needs to sense any change in the environment, the positions of the obstacles, and the other robots in the neighborhood. Then it needs to communicate with the peers, exchange the sensing information, discuss the cooperation strategy, suggest the obstacle avoidance strategy, plan the moving path, assign or receive the subtasks and roles, and coordinate the actions so as to move the object quickly and successfully to the goal location. Arguably, the success of this task will require the use of a variety of technologies from different fields. As a result, the task of multi-robot transportation is a good

benchmark to assess the effectiveness of a multi-agent architecture, cooperation strategy, sensory fusion, path planning, robot modeling, and force control. Furthermore, the task itself has many practical applications in fields such as space exploration, intelligent manufacturing, deep sea salvage, dealing with accidents in nuclear power plants, and robotic warfare.

In this section, a physical multi-robot system is developed, integrated with machine learning and evolutionary computing, for carrying out object transportation in an unknown environment with simple obstacle distribution. In the multi-agent architecture, evolutionary machine learning is incorporated, enabling the system to operate in a robust, flexible, and autonomous manner. The performance of the developed system is evaluated through computer simulation and laboratory experimentation.

As explained in section 3.1, a learning capability is desirable for a cooperative multi-robot system. It will help the robots to cope with a dynamic or unknown environment, find the optimal cooperation strategy, and make the entire system increasingly flexible and autonomous. Although most of the existing commercial multi-robot systems are controlled remotely by a human, the autonomous performance will be desirable for the next generation of robotic systems. Without a learning capability, it will be quite difficult for a robotic system to become fully autonomous. This provides the motivation for the introduction of machine-learning technologies into a multi-robot system.

The primary objective of the work presented here is to develop a physical multi-robot system, where a group of intelligent robots work cooperatively to transport an object to a goal location and orientation in an unknown and dynamic environment. A schematic representation of a preliminary version of the developed system is shown in Figure 1.1 and it is repeated in Figure 3.3.

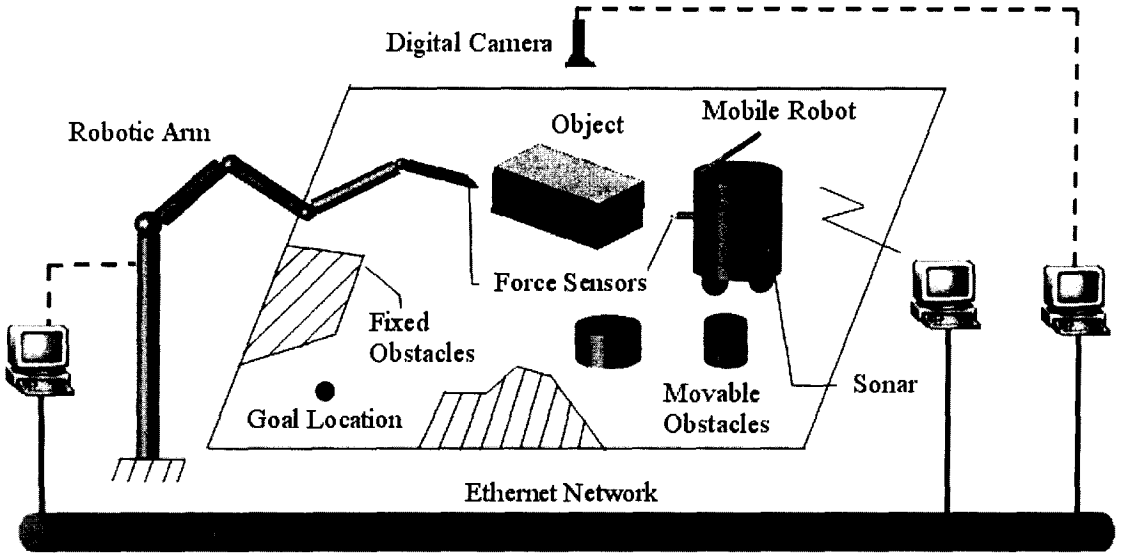


Figure 3.3: The developed multi-robot system.

3.3.1 Multi-agent Infrastructure

The multi-agent architecture as outlined before is used here as the infrastructure to implement cooperative activities between the robots. This infrastructure is schematically shown in Figure 3.4.

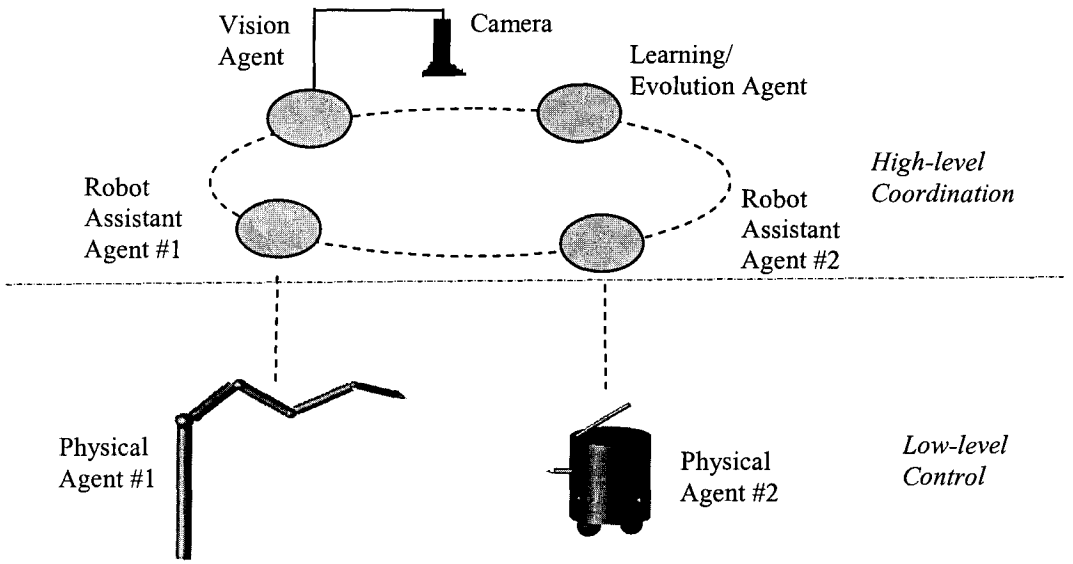


Figure 3.4: The multi-agent architecture used in the developed system.

In Figure 3.4 shows four software agents and two physical agents in the developed architecture, forming the overall multi-agent system. Each agent possesses its own internal

state (intention and belief) and is equipped with independent sensing and decision-making capabilities. They also are able to communicate with each other and exchange information on their environment as acquired through sensors, and the intentions and actions of the peers. Based on the information from their own sensors and their internal states, the agents cooperatively determine a cooperation strategy to transport the object.

In Figure 3.4, the four software agents constitute a high-level coordination subsystem. They will cooperate and coordinate with each other to generate cooperation strategies, and assign subtasks to the two robot assistant agents. In the meantime, the two physical agents (robots) form a low-level subsystem of control and execution, which receives commands from the upper-level subsystem.

There is a vision agent, which is in charge of acquiring and processing the images using a CCD camera. Because the present section focuses on developing a learning-based multi-robot architecture, a simplified computer vision scheme is employed, which uses a global camera and a global localization technology. The vision agent will analyze the acquired images and compute the exact locations and orientations of the robots, the object and the obstacles. All the information is then broadcasted to the other three software agents so that they will immediately know the current world state (the position and orientation messages of the robots, the object and the obstacles; and the environmental map information).

A special learning/evolution agent is used in the architecture to play the combined role of a learner, a monitor and an advisor. The machine-learning algorithm and the decision-making capabilities are embedded in this agent. First, it will collect from other agents the information of the environment, the robots, the object and the obstacles, to form the current world state, which can be shared by its peers. Then it will make decisions on the optimal cooperation strategy based on its own knowledge base so that the common task can be completed effectively and quickly. Finally, it will assign the roles of other agents, and monitor the task schedule.

Each physical agent (robot) is linked to a corresponding assistant agent, which needs to forward the robot position/force information to the other agents in the upper level, and to receive the desired force/trajectory of the robot from the learning/evolution agent and send them to the corresponding robot in the low-level subsystem. All six agents will form a

tightly-coupled multi-agent system to transport the object cooperatively to the goal location while avoiding obstacles.

3.3.2 Cooperation Based on Machine Learning

Learning ability is a desirable feature for a multi-robot transportation system. It will help the robots to cope with dynamic changes in an unknown environment. Although conventional planning algorithms can make the robots move effectively in a static environment, they usually fail in an environment with dynamic obstacles. A multi-robot system with learning capability possesses characteristics of adaptation and self-organization, which will enable it to overcome the difficulties in an unknown environment. In this section, two well-known machine learning approaches, Reinforcement Learning (RL) and Genetic Algorithms (GAs), are integrated into the learning/evolution agent to achieve robust and effective cooperation between robots.

3.3.2.1 Reinforcement learning

A key problem in a multi-robot transportation system is the determination of the cooperation strategy; i.e., finding the optimal amplitudes and locations of the applied forces of the robots so that a net force with maximum amplitude will point toward the goal location, while avoiding any obstacles. Reinforcement Learning (RL) provides a way to determine an appropriate cooperation strategy through experimentation with the environment. A method of reinforcement learning known as the Q-learning algorithm, as given in Figure 3.2, is employed here to find the optimal action policy for each robot.

In the system developed in this section, the world state is defined as the relative position and orientation between the object and the goal location, and the positions of the obstacles within a fixed detection radius around the object. It is shown in Figure 3.5.

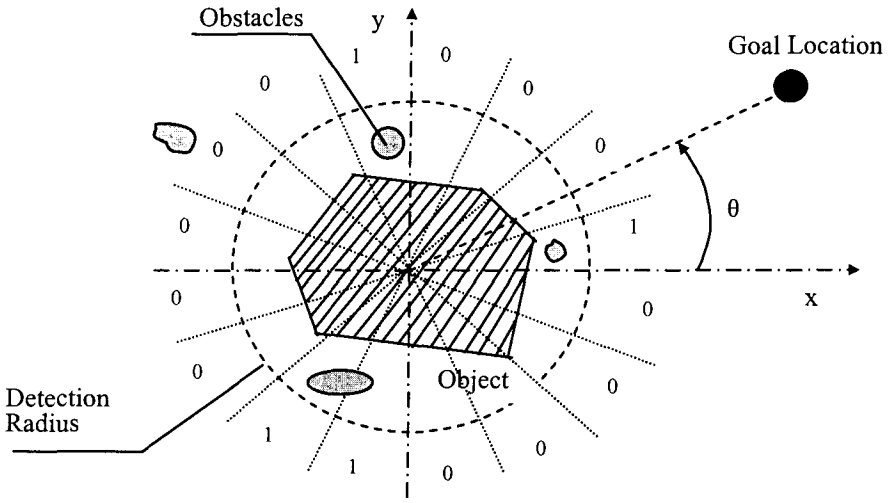


Figure 3.5: The world state representation.

In Figure 3.5, only the obstacles located within the detection radius around the object are detected and considered. The detection circle is evenly divided into 16 sectors and the sectors are represented by a 16-bit binary code. If one obstacle is located in one of the sectors, the corresponding bit of the binary code will become "1." Otherwise it will become "0." Accordingly, this binary code represents the obstacle distribution around the object. The world state s in the Q-learning algorithm is made up of this binary code and the angel θ in Figure 3.5, which describes the relative orientation between the goal location and the object. For example, the world state shown in Figure 3.5 can be represented by the vector $s=[\theta\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0]$.

The cooperation strategies; i.e., the force locations and amplitudes for all robots, which are shown in Figure 3.6, form the set of actions in the Q-learning algorithm.

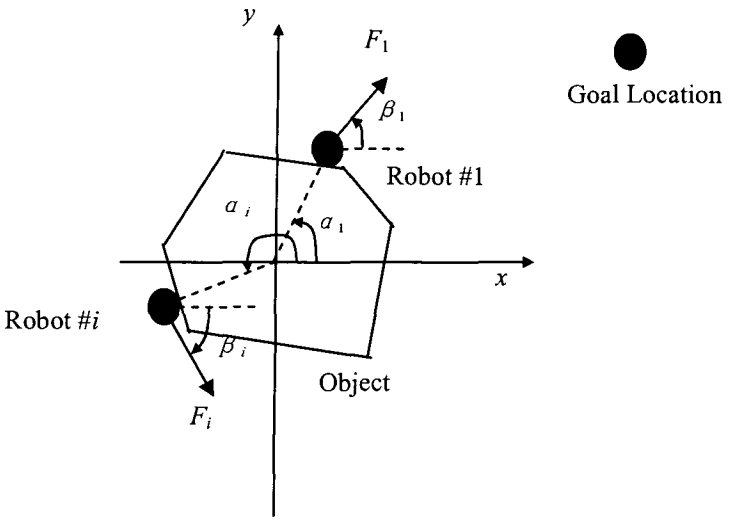


Figure 3.6: Representation of the cooperation strategy (action).

In Figure 3.6, the application point of the force from each robot is represented by the pair of angles *(α, β)*. Therefore, possible actions in the Q-learning algorithm for the robot R_i can be modeled as the set $\Lambda_i = \{(\alpha, \beta, F)_k\}$, $k = 1,2,\ldots,m_i$.

There remains a special action called "*sweeping*" which instructs the robots to remove all the obstacles in the path after establishing that they are capable of carrying out this sweeping task.

The reward r is calculated as

$$r = D_{t_i} - D_{t_{i-1}} \tag{3.8}$$

where, D_{t_i} and $D_{t_{i-1}}$ are the distance between the mass center of the object and the goal location, at times t_i and t_{i-1}, respectively.

3.3.2.2 Genetic Algorithms

Another approach that is used here to find the optimal cooperation strategy is Genetic Algorithms (GAs). By simulating the biological evolution process, GA provides a powerful way to quickly search and determine the optimal individual in a large candidate space.

A typical Genetic Algorithm as found in (Karray and de Silva, 2004; Mitchell, 1997) is given below.

- Initialize p (the number of individuals in the population), η (the fraction of the population to be replaced by crossover at each step), m (the mutation rate), and *fitness threshold.*
- Generate p individuals at random to produce the first generation of population P
- Calculate the *fitness(i)* for each i (individual) in P
- While $(\max_i fitness(i)) < fitness\_threshold$, then repeat:

  Produce a new generation P_s:

  1. Probabilistically select $(1-\eta)p$ members of P to be included in P_s. The selection probability is given by: $\lambda(i) = \frac{Fitness(i)}{\sum_{j=1}^{p} Fitness(j)}$
  2. Execute the Crossover operation to generate $(\eta p)/2$ pairs of offspring and add them to P_s
  3. Mutate m percent of the numbers of P_s
  4. Update $P \leftarrow P_s$
  5. Calculate *Fitness(i)* for each i in P
- Return the individual with the highest fitness in P

Figure 3.7: The Genetic Algorithm.

In this section, the entire candidate space is formed by the "*sweeping*" action and $\Phi = (\alpha_1, \beta_1, F_1, \cdots, \alpha_i, \beta_i, F_i, \cdots \alpha_n, \beta_n, F_n)$ described in Figure 3.6, which is the set of all possible cooperation strategies. This candidate space can be represented by a vector space $\Re^{3n+1}$ which is made up of all possible actions: *(Φ, sweeping).* The Genetic Algorithm given in Figure 3.7 is used to search for the optimal individual vector; i.e., the optimal robot cooperation strategy, in this large vector space.

In the beginning, the "*sweeping*" action and nine individuals randomly selected from the candidate space are used to constitute the first generation of population. Then the

Genetic Algorithm outlined before is used to search for the optimal individual with the maximum fitness value. The fitness is calculated as follows:

$$Fitness = \begin{cases} C & \text{if it is a } \textit{"sweeping"} \text{ action and some removable obstacles are in the path.} \\ 0 & \text{if a } \textit{"sweeping"} \text{ action without removable obstacles in the path.} \\ M & \text{otherwise} \end{cases} \tag{3.9}$$

$$M = \frac{k_1 * |F| + k_2 * (1+\cos\theta)^2 + k_3 / |\Gamma| + k_4 S + k_5 / (1+\cos\tau)^2}{k_1 + k_2 + k_3 + k_4 + k_5} \tag{3.10}$$

where, C is a threshold value which is determined through trials, F is the net force applied by the robots, θ is the angle between the net force vector and the goal location vector which is described in Figure 3.5, Γ is the net torque applied by the robots, S is the area of the polygon formed by the robot locations, τ is the angle between the net force vector and the position vector of its closest obstacle, and k_1 to k_5 are the weights ($k_1 + k_2 + k_3 + k_4 + k_5 = 1$).

The fitness function includes several practical considerations. First, the "*sweeping*" action is always intended for use in the current world state. If a removable obstacle is present in the forward path of the object (it is assumed that, based on its size, color, shape and so on, the agents are able to judge whether the obstacle can be moved away by the robots), the *"sweeping"* action will assume a high fitness value so that the obstacle can be removed immediately. In order for it to always have the opportunity to be selected, the "*sweeping*" action is included in the population of the first generation and it is always copied to the next generation in the evolution process.

Second, a large net force that points to the goal location will earn a high fitness value so that the internal force becomes minimum and the object is transported to the goal location quickly and effectively. Third, a high net torque Γ is not encouraged, because the resulting unnecessary rotation of the object will make it difficult for the robots to handle it, while a small Γ may be encouraged for avoiding the obstacles. Fourth, a large S implies that the spacing of the robots is quite adequate, which will receive a high fitness value, because crowding of the robots will make their operations more difficult and less effective.

Finally, a small τ means that the probability of the presence of an obstacle in the path is high, and this condition should be punished. The weights $k_1 \sim k_5$ are calibrated through experimentation.

In each step of object transportation, the GA is used to search for the optimal cooperation strategy. Then this strategy is broadcasted to the two robot assistant agents by the learning/evolution agent. Next, the robot assistant agents forward the commands to the physical robots. Based on their internal hybrid force/position controllers, the robots then move themselves to the desired positions and transport the object for a specified time period. In the meantime, the current force and position information is sent back to the high-level coordination subsystem via the robot assistant agents for decision-making in the next step. After the object is moved through some distance, the world state would be changed (For example, new obstacles are observed and so on). In this manner, the GA is used again to search for the new optimal cooperation strategy under the new world state. This process will be repeated until the object is successfully moved to the goal location.

The main problem of GA in a physical robotic system is that its real-time performance cannot be guaranteed because of its low convergence speed. However, in the present thesis, this problem is solved partly by using the layered architecture shown in Figure 3.4. In this two-layer multi-agent architecture, the low-level physical agents can continue to execute the old commands before the new commands are given by the agents in the upper level. In other words, the agents in the upper level are not necessary to work at the speed of the physical agents in the lower level. This enables a "low-speed" GA agent to make decisions for a real-time task.

3.3.2.3 Integrating reinforcement learning with Genetic Algorithms

Reinforcement Learning (RL) may face difficulties and generate non-optimal outputs. A typical problem is that "the agent runs the risk of over-committing to actions that are found to have high Q values during the early stages of training, while failing to explore other actions that also have high values" (Mitchell, 1997). Another problem is that RL may not be able to capture the slow changes in the environment. In other words, when the environment changes very slowly, RL will have less opportunity to try the actions under

different world states. Consequently, the knowledge base of RL will be updated very slowly. It means that the learning process will become very slow and the output of RL will not be optimal.

Genetic Algorithms (GAs) can partly overcome these problems. Because of the special evolution operations such as Crossover and Mutation in GAs (Karray and de Silva, 2004), it is possible for GAs to select a new search point in the candidate space, which had not been tried in the past steps. This will provide an opportunity for the actions with low Q values in Reinforcement Learning to demonstrate their capabilities. Moreover, because a GA simulates the biological evolution process, it provides better adaptivity than RL and is able to capture slow changes in the environment.

On the other hand, GAs also benefit from RL because GAs cannot guarantee a precisely optimal output. In particular, in the system developed in the present work, the crowding phenomenon (Mitchell, 1997) was observed, where "very similar individuals take over a large fraction of the population." In addition, GAs are unable to deal with quick changes in the environment because the evolution process usually requires a considerable duration of time. All these problems can greatly slow down the evolution process and make the output of the GAs unacceptable.

However, by integrating GAs with RL, different individuals can be inserted into the population of GA through RL, because RL uses an entirely different mechanism to select the new cooperation strategy. Consequently, it would be possible to disturb the population purity in a GA and resolve the "crowding" problem in part. In addition, RL is able to deal with quick changes in the environment, where GA usually fails.

The combination of learning with evolution, which are two important adaptive mechanisms that exist in nature, gives rise to distinct advantages, as discussed before. In this section, a specific integration scheme of these two capabilities is implemented in the multi-agent object transportation system, which is indicated in Figure 3.8.

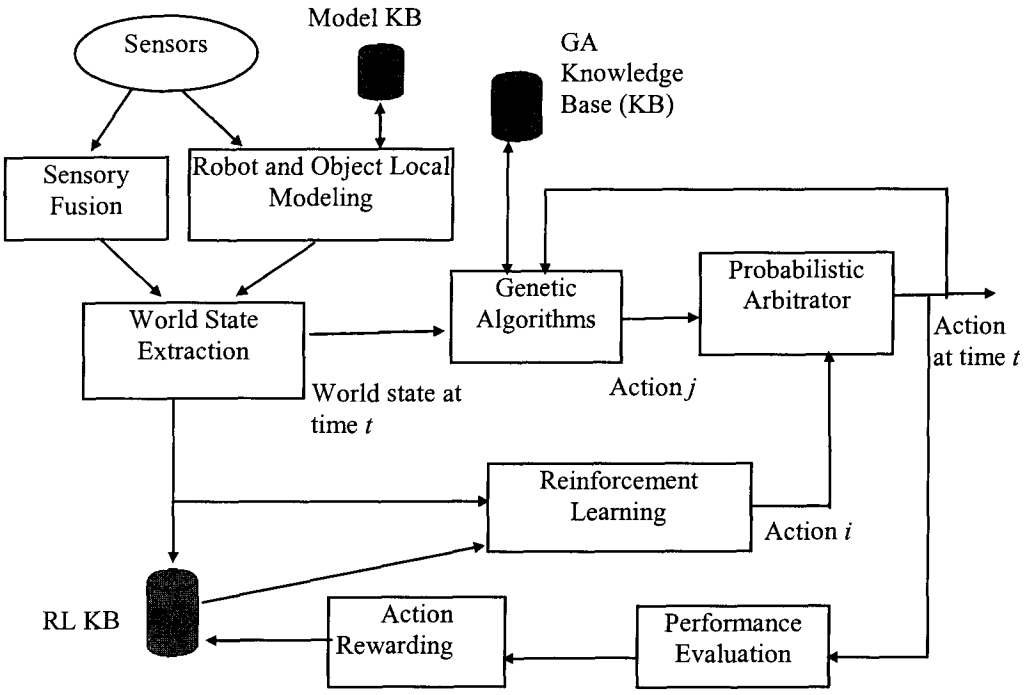


Figure 3.8: The integration scheme of RL and GA.

In Figure 3.8, the sensor data is first sent to a sensory fusion module. This information will form the current positions and orientations of the object, the robots and the obstacles, and the state of the environment. Moreover, there is a model knowledge base in which the geometry and dynamic parameters of the object and the robots are stored. These model parameters are determined off line and stored in the model knowledge base prior to operating the system. When the sensory data is sent to a local modeling module, the module will identify and update the current model parameters by combining the off-line information in the knowledge base and the new sensory data. The information from the sensory fusion module and the model knowledge base is then integrated to form the new world state, as described in Figure 3.5.

The two decision-making mechanisms, Reinforcement Learning and Genetic Algorithm, are implemented based on the algorithms presented in the previous sections. They both are able to determine a robot cooperation strategy according to the current world state. Their outputs are probabilistically selected by an arbitrator, which determines the winning output. In this chapter, it is assumed that the output of the reinforcement learning block is selected by the arbitrator with a probability of 0.7, while the output of the

genetic algorithm block is selected with a probability of 0.3. The output of the arbitrator; i.e., the cooperation strategy, is then broadcasted to the two robot assistant agents by the learning/evolution agent to implement the object transportation task.

3.3.3 Simulation Results

Some simulation work has been carried out to demonstrate the performance of the evolutionary learning mechanism described before. Because the simulation is used to validate the evolutionary learning approach in section 3.3.2, a special case is assumed, which is somewhat different from Figure 3.4. Specifically, it is assumed that there are three autonomous robots, which cooperate to transport a rectangular box (object) to a goal location in a 400*400-unit environment without any obstacles. The box has a width of 60 units and a height of 40 units. Each robot is simplified as a circular point in the environment shown in the subsequent figures. The multi-thread programming of Java language is used to simulate the parallel operations of the three robots.

The present work is applicable not only in object transportation tasks with small displacements, described in Figure 3.4, but also in more challenging cases such as space exploration. For example, consider the case of several robots sent to a planet for exploration of a special ore, where they transport the ore cooperatively into a spacecraft. In this application, it is impossible for a robot or an agent to possess global information. In other words, each robot only possesses the local sensing ability and it will need to communicate with the other robots to establish the global information. In the present simulation, each robot is assumed to be only equipped with local sensing capability. Moreover, the evolutionary learning mechanism is also embedded in each robot so that they can cope with the possible existence of unreliable communication, in the planetary exploration.

Because a robot only possesses local sensing capability and the environment dimension is somewhat large, it will not know where the box is in the beginning. Therefore, a random search strategy is used to find the object in the beginning of the simulation. Specifically, the robots walk randomly to search the object. When one robot reaches a wall or another robot, it simply changes its direction and continues the "wandering state." However, if a robot reaches the box, the "wandering" behavior ceases and the position of the box is

immediately broadcasted to the peers. Then this robot will temporarily take over the "leader" role in the subsequent transportation task. Next, it will use its embedded evolutionary learning mechanism to compute the optimal cooperation strategy and send it out to the other robots. Consequently, the other robots will move themselves to the desired locations and orientations, which are described in the new cooperation strategy sent by the leader robot. This process is shown in Figure 3.9.

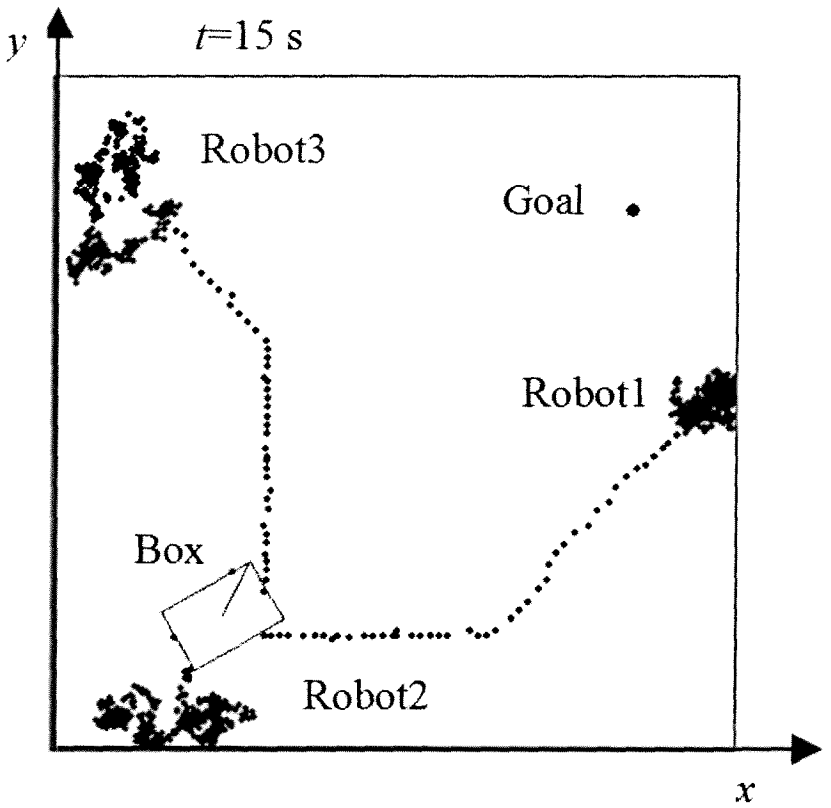


Figure 3.9: The robots move randomly and locate the box.

The leader robot is responsible for synchronizing the forces and motions among the teammates. It will check whether all the robots have moved to the desired position and orientation. If all the robots are ready, the leader robot will send out a "transport" command. Then the robots will exert the desired forces on the box, and the position and orientation of the box will be changed as a result. Next, the leader robot will compute the new world state and generate the new cooperation strategy. This process will be repeated until the box is transported to the goal location and orientation. The overall process, as simulated, is shown in Figure 3.10.

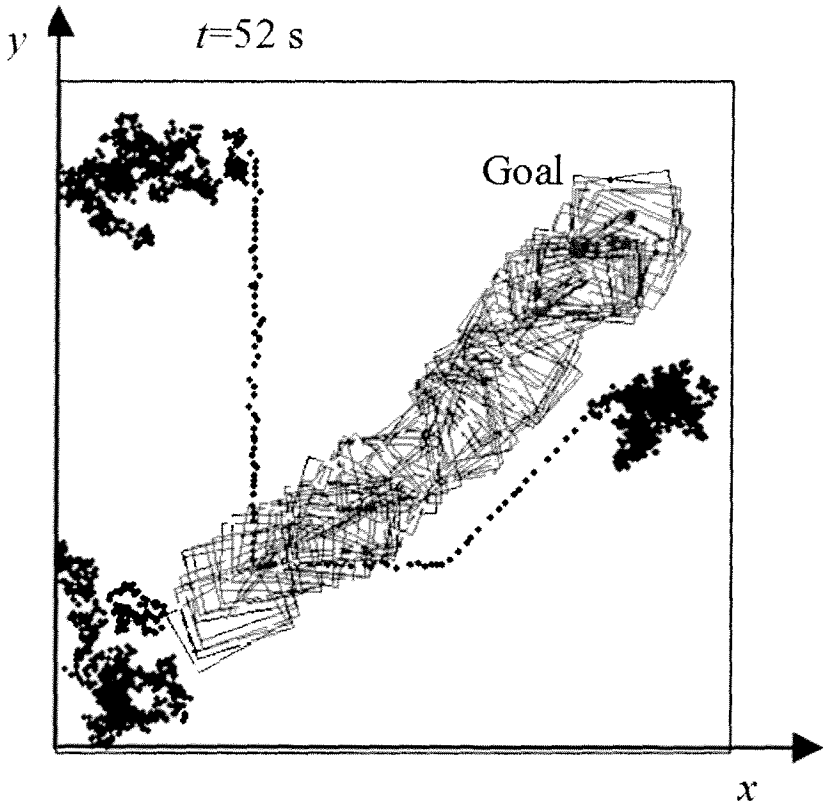


Figure 3.10: The whole transportation process.

Figure 3.11 demonstrates the adaptivity of the evolutionary learning mechanism employed by the robots. In this example, the goal location was suddenly changed from (380,370) to (385,60) at $t = 34s$, while the robots were transporting the box. From Figure 3.11 it is observed that the robots quickly adjusted the cooperation strategy and successfully transported the box to the new goal location.

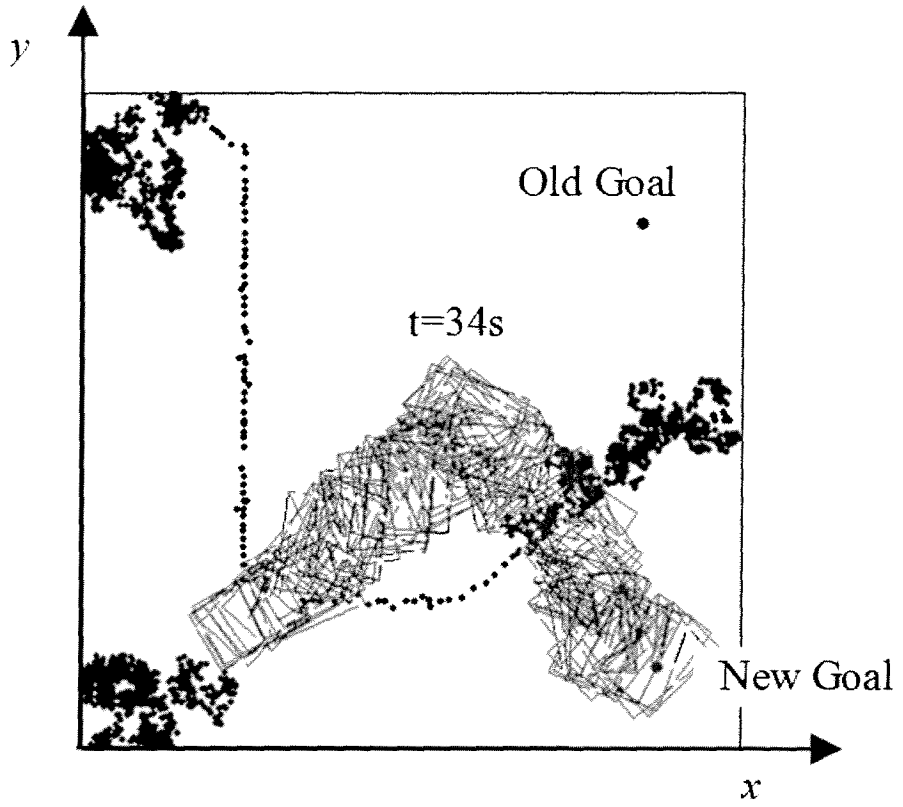
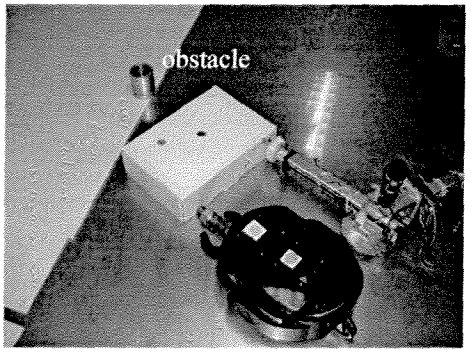


Figure 3.11: The goal location was suddenly changed from (380,370) to (385,60) at t = *34s* while the robots were transporting the box.

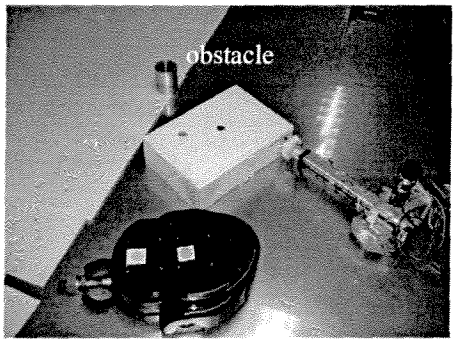
From the simulation results presented here, it is seen that the evolutionary learning mechanism described before is quite effective in meeting its design objective.

3.3.4 Experimentation

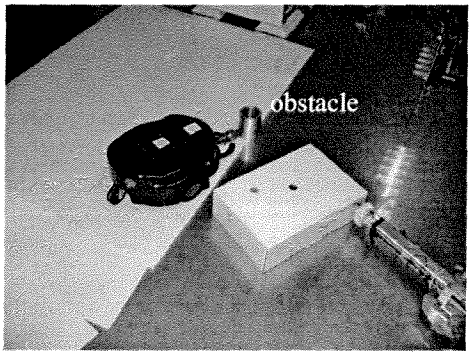
The experiment presented here is designed to test the multi-agent learning strategy described before. Here, a movable obstacle is placed in the path while the robots are cooperatively transporting the box to the goal location. Figure 3.12 shows how the robots handle this problem.



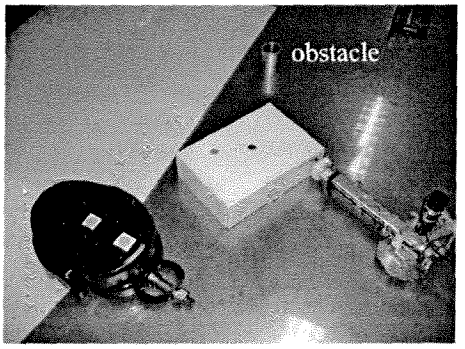
(a) Initial state (Pushing of the box).



(b) Leaving the box.



(c) Removing the obstacle.



(d) Returning.

Figure 3.12: The robots have selected the "sweeping" action to remove a movable obstacle in the path.

From Figure 3.12 it is seen that when the robots detect the existence of a movable obstacle in the forward path with the help of the camera agent, they select the "sweeping" action. Accordingly the mobile robot leaves the box, moves toward the obstacle, and pushes it away from the path. While the mobile robot is sweeping the obstacle, the robotic arm just stays in the same place, waiting for its peer to return. Once the obstacle is swept away, the mobile robot returns to continue the task of box transportation in cooperation with the robotic arm.

This experiment demonstrates that the multi-agent robotic system developed in the present work, equipped with machine learning capability, exhibits good flexibility and adaptivity in a complex environment, while effectively carrying out the intended task.

3.4 Distributed Multi-robot Q-learning in Complex Environments

In section 3.3, machine learning is used to find optimal cooperation policies for a multi-robot transportation task. Although this fist version of learning-based multi-robot transportation system is successful, it has two limitations. First, it is a partly centralized system. In the proposed multi-agent architecture shown in Figure 3.4, a centralized learning/evolution agent makes decisions for all robots (physical agents) in the environment. As stated in section 2.1, this centralized learning/evolution agent will decrease the scalability performance of the system and result in a communication bottleneck problem. When the number of robots is large, the burden of the learning/evolution agent will increase rapidly. In other words, the multi-agent architecture in Figure 3.4 is not suitable for a large multi-robot system with many robots.

Second, a relatively simple environment is assumed in section 3.3. There are few obstacles in the environment and the obstacles are usually stationary. However, in a real multi-robot environment, the obstacle distribution will be more complex. In particular, there may be many obstacles which may appear and disappear randomly. In such a complex environment, a multi-robot system is expected to still complete its task in a robust manner.

In this section, a multi-robot transportation system equipped with distributed Q-learning is studied, which works in a dynamic and unknown environment with complex obstacle distribution. In this more challenging environment, the Q-learning algorithm will still show good adaptivity and robustness.

3.4.1 Distributed Q-learning

There are two options for extending single-agent Reinforcement Learning to the multi-robot field. The first one is to simply regard the multi-robot learning as a combination of multiple single-agent learning processes across the team members. In this case, each robot is equipped with the reinforcement learning technique, and learns their skills as if the other robots do not exist in the environment. This type of reinforcement learning is called "distributed learning." The main advantage of "distributed learning" is its simplicity (Yang and Gu, 2004). Every robot just needs to monitor its own actions and the corresponding rewards, and it does not need to care about the behaviors of the peer robots.

As a result, the learning space is small and the computational cost is low. However, directly extending the single-agent reinforcement learning algorithm to the multi-robot domain violates the MDP (Markov Decision Processes) assumption, the theoretical basis of reinforcement learning, which assumes that the agent's environment is stationary and does not contain any other active agents. Although some research has shown that "distributed learning" is effective in some cases, more investigation is needed on this aspect.

The second one is the SG-based (Stochastic Game based) Reinforcement Learning. In that extension, each robot not only monitors its own actions but also observes the actions of the peer robots in the same environment. It believes that the rewards and the environmental changes are caused by the joint actions of all the robots. This type of reinforcement learning is called "SG-based learning." Several types of SG-based learning algorithms have been developed in the multi-agent systems area, such as MiniMax-Q learning, Nash-Q learning, and Friend-or-Foe-Q learning. With a solid theoretical basis, these algorithms are shown to be effective in several multi-agent/ multi-robot simulations and experiments. However, they introduce an extensive learning space (state/action space) and complex calculations in the Reinforcement Learning algorithms because each robot has to observe not only its own actions but also the actions of its peers. The extensive learning space is absolutely a nightmare for researchers in the Reinforcement Learning (RL) field, especially in the multi-robot RL field where the real-time performance is important.

It follows that, "distributed learning" still makes sense in the multi-robot learning domain because of its simplicity. Yang and Gu (2004) indicates that this type of learning may work in a purely cooperative multi-robot task, and Littman (2001) theoretically proves its convergence under some conditions. Next, "distributed Q-learning" is applied to a multi-robot box-pushing task in a dynamic and unknown environment, which is a challenging and important sub-field in the multi-robot domain. Effectiveness, robustness and adaptivity of this learning-based multi-robot system will be shown through computer simulation, and some issues of "distributed learning" will be discussed.

3.4.2 Task Description

Multi-robot box-pushing is a long-standing yet difficult problem in the multi-robot domain. In this task, multiple autonomous robots have to cooperatively push a box to a goal location and orientation in a dynamic and unknown environment. The box is big and heavy enough so that no single robot can complete this task alone. In addition, each robot only possesses local sensing capability and the obstacles may occur and disappear randomly in the path, so the robots cannot plan the trajectory in advance. In such a dynamic and unknown environment with a continuous space which is presented in Figure 3.13, any static planning technology is bound to fail. Although dynamic planning (Miyata et al., 2002) may be useful to complete the task, it will fail if the environment changes quickly and the calculation speed of the robots is not sufficiently fast.

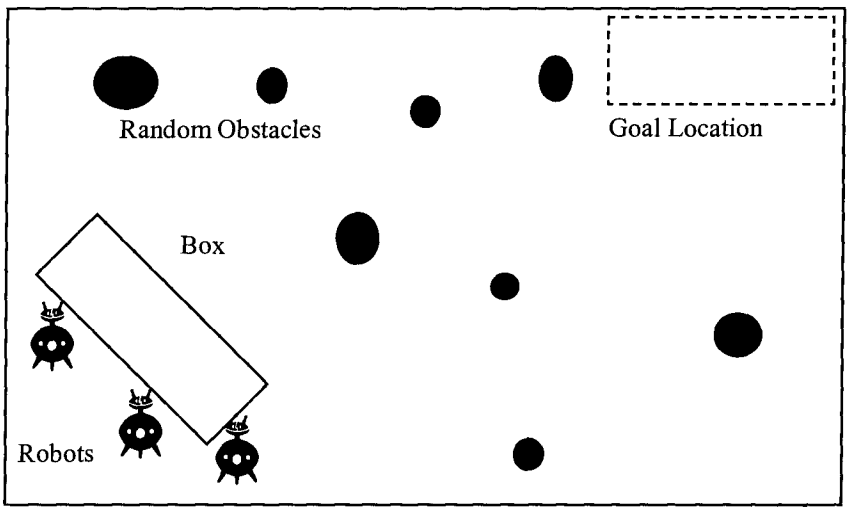


Figure 3.13: Description of a multi-robot box-pushing task.

Because the learning-based architecture may be a better solution to this challenging problem than a static/dynamic planning architecture, next, the single-agent Q-learning described in Figure 3.2 is extended to this multi-robot box-pushing problem.

3.4.3 Multi-robot Box-pushing With Distributed Q-learning

In this section, single-agent Q-learning is directly extended to the multi-robot box-pushing problem. The basic idea is as follows. Each robot is equipped with single-agent Q-learning. When the robots begin to transport the box, each robot identifies the current environmental state, and then independently employs the Q-learning algorithm in Figure

3.2 to select the "optimal" action to push the box. After the actions are taken, each robot independently observes the new environmental state and receives its direct reward *r*. The robots then update their own Q-tables with the reward and the new state information, and continue with the next step of box-pushing. There are no communications among the robots except the synchronization signal. Each robot independently maintains its own Q-table and makes its decisions. Therefore, it is a fully distributed learning-based decision-making system. In other words, the multi-robot box-pushing problem is solved with a purely learning-based architecture instead of complex static or dynamic planning approaches.

The main challenge of such a "distributed reinforcement learning" system is that it violates the assumption of stationary environment in the MDP problem. While a robot is pushing the box, its peer robots are also pushing it. It means that the environment is no longer stationary from the viewpoint of this robot and Q-learning may not converge to the optimal policy. In addition, the random obstacles in the environment can make the case far worse. However, the simulation results in the following section will show that the task still can be completed at the cost of slight reduction in efficiency or increase in the required number of pushing steps.

3.4.3.1 The environmental states

Because the environmental space in the multi-robot box-pushing problem is continuous, in order to employ the Q-learning algorithm, the continuous space is converted into a set of finite states. In particular, each environmental state is coded into a 13-bit binary code which is shown in Figure 3.14.

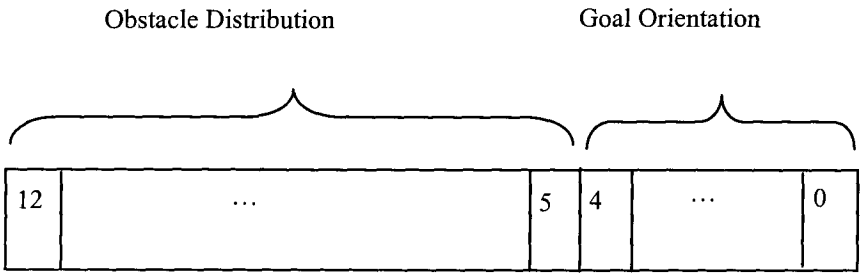


Figure 3.14: The binary coding of the environmental states.

In Figure 3.14, the higher 8 bits (from bit 5 to bit 12) in the code represent the current obstacle distribution around the box within the detection radius. The lower 5 bits (from bit 0 to bit 4) represent the orientation of the goal relative to the current box pose. The coding rule is explained in Figure 3.15.

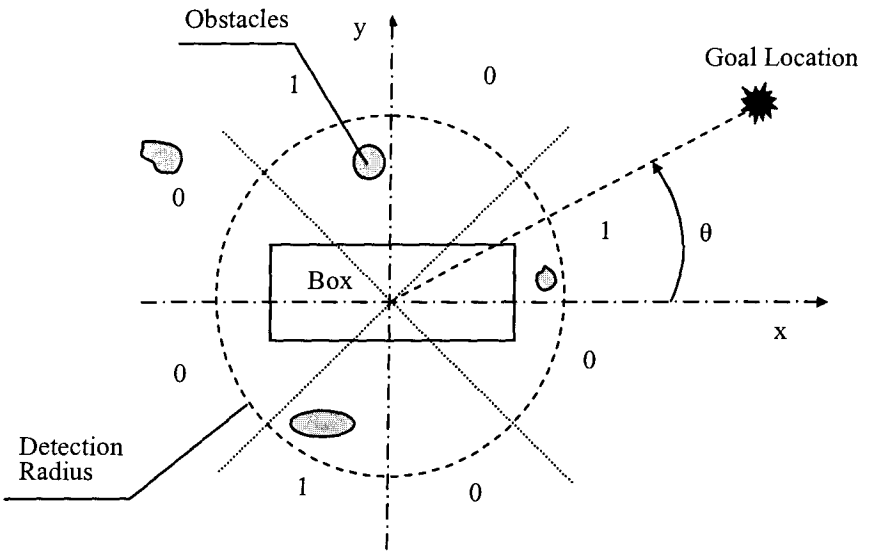


Figure 3.15: Coding rule of the environmental states.

In Figure 3.15, it is assumed that the robots only possess local sensing abilities. Consequently, only those obstacles within the detection radius can be detected and coded into the higher 8-bits of the environmental state. In order to describe the obstacle distribution in Figure 3.15, the detection circle is divided into 8 equal sectors. Each sector corresponds to one of the higher 8 bits of the environmental state. If there is at least one obstacle detected in this sector, the corresponding bit in the state will become "1." Otherwise, it will become "0." For example, in Figure 3.15, the higher 8-bits of the environmental state is "10100100" which indicates the current obstacle distribution around the box within the detection radius.

The lower 5 bits of the environmental state indicate the orientation of the goal relative to the current box pose. The orientation angle θ of the goal in Figure 3.15 is used to calculate the lower 5-bits of the state using

$$State_{bit(0\sim4)} = INT(\theta/(360.0/32)) \tag{3.11}$$

where, *INT()* is a function to covert a float value into an integer.

3.4.3.2 Robot actions

In a physical world, a robot usually can push a box at any contact point. It means that there are infinite actions available to a robot to push the box. However, in this section, because it is focused on extending the single-agent reinforcement learning algorithm to the multi-robot domain, the action space is simplified and only six actions are assumed to be available to each robot. The available actions ($a_1 \sim a_6$) are presented in Figure 3.16.

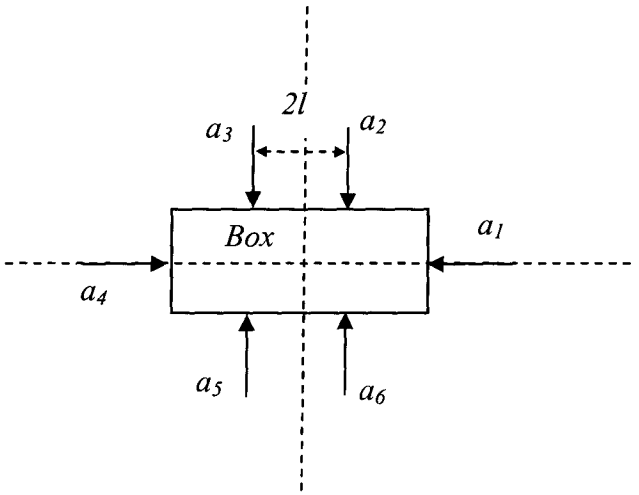


Figure 3.16: Pushing actions available to each robot.

3.4.3.3 Box dynamics

Because this section focuses on extending the single-agent reinforcement learning algorithm to the multi-robot box-pushing problem, the box dynamics is simplified in the simulation system. It is assumed that the box displacement is proportional to the net force exerted by the robots, and the rotational angle of the box is also proportional to the net torque exerted by the robots. In Figure 3.17, the dynamics of the box is illustrated, where the box is being pushed by three mobile robots.

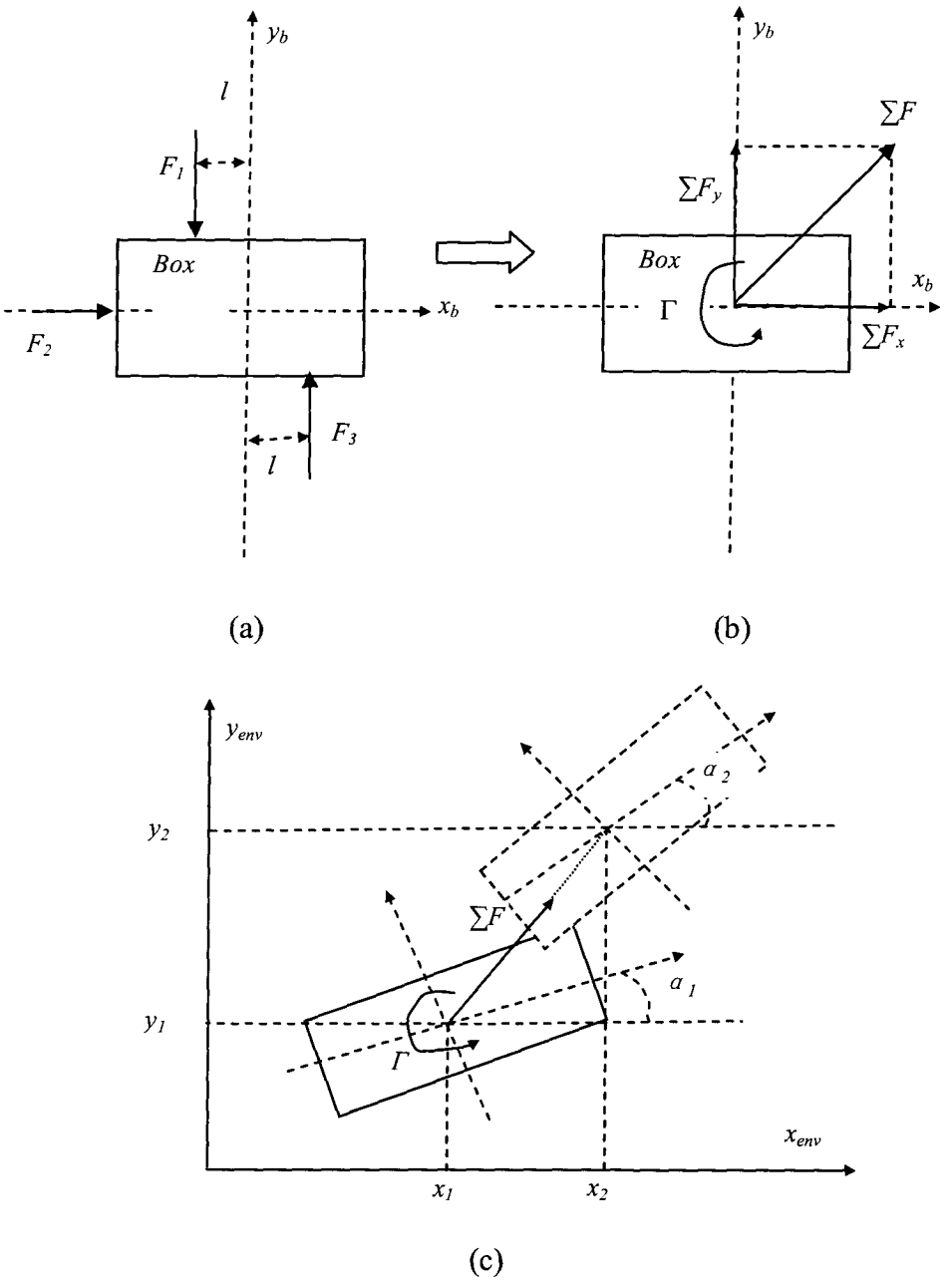


Figure 3.17: The dynamics of the box pushed by three robots.

In Figure 3.17, three robots push the box and $F_1 \sim F_3$ are their pushing forces. The box dynamics is described with the following equations:

$$x_2 = x_1 + c_f \left|\sum F\right| \cos(\gamma + \alpha_1) \tag{3.12}$$

$$y_2 = y_1 + c_f \left|\sum F\right| \sin(\gamma + \alpha_1) \tag{3.13}$$

$$\alpha_2 = \alpha_1 + c_t \Gamma \tag{3.14}$$

where, $\sum F$ is the net force, Γ is the net torque, c_f and c_t are coefficients, and $\gamma = arctg2(|\sum F_y| / |\sum F_x|)$.

3.4.3.4 Reward function

A common payoff reward strategy is employed in this section. Specifically, each robot gets the same reward after it pushes the box for a short period of time Δt. The reward consists of the following three parts:

(1) The reward for pushing the box toward the goal location

$$R_{distace} = (Dist_{old} - Dist_{new}) c_d \tag{3.15}$$

where, $Dist_{old}$ and $Dist_{new}$ are the distances between the goal location and the box center before and after taking the pushing actions, and c_d is a coefficient.

(2) The reward for the rotational behavior of the box

$$R_{rotation} = \cos(\alpha_2 - \alpha_1) - 0.9 \tag{3.16}$$

where, $(\alpha_2 - \alpha_1)$ is the rotational angle of the box when it is pushed by the robots for a period of time Δt. While a small rotation is encouraged, a large rotation will be punished because it will make handling the box difficult for the robots.

(3) The reward for obstacle avoidance

$$R_{obstacle} = \begin{cases} = 1.0 \quad if\ no\ obstacle \\ = 1.5(\sin(\psi / 2) - 0.3) \end{cases} \tag{3.17}$$

where, ψ is the angle between the direction of the net force and the direction of the obstacle closest to the net force.

Finally, the total reward is calculated as.

$$R = w_1 R_{distance} + w_2 R_{rotation} + w_3 R_{obstacle} \tag{3.18}$$

where, $w_1 \sim w_3$ are the weights and $w_1 + w_2 + w_3 = 1.0$.

3.4.3.5 Random action

Reinforcement Learning may face difficulties in the action selection process. For example, "the agent runs the risk that it will over-commit to actions that are found to have

high $\hat{Q}$ values during the early stages of training, while failing to explore other actions that also have high values." One solution to this problem is to select actions with the GLIE (Greedy in the Limit with Infinite Exploration) policy (Littman, 2001a). However, here a simpler strategy instead of the GLIE policy is used. The strategy is to let the robots randomly select their actions at a low level of probability. In other words, most of the time, the robots take their actions as determined by the Q-learning algorithm, while they do select their actions randomly at a low level of probability.

3.4.4 Simulation Results

A simulation system is developed with the Java language to simulate a learning-based multi-robot box-pushing system, which has been described in the foregoing sections. The dimensions of the environment are 950×700 units and the dimensions of the box are 120×80 units. There are three robots, denoted by circles with a radius of 2 units in the subsequent figures, which push the box to the goal location. The box is big and heavy so that no individual robot can handle it without the help of its peers. The Java multi-thread programming technique is used to implement the parallel operations of the robots.

When the simulation system starts to run, based on the current environmental state, the robots independently select their actions using the single-agent Q-learning algorithm. Then they begin to push box after synchronization. The box is moved for a short period of time Δt, and the reward is collected and distributed equally among the robot. Based on the reward and the new environmental state, the robots independently update their own Q-tables. Then they start the next step of pushing, and so on, until the goal location is reached or the total step number exceeds the maximum limit.

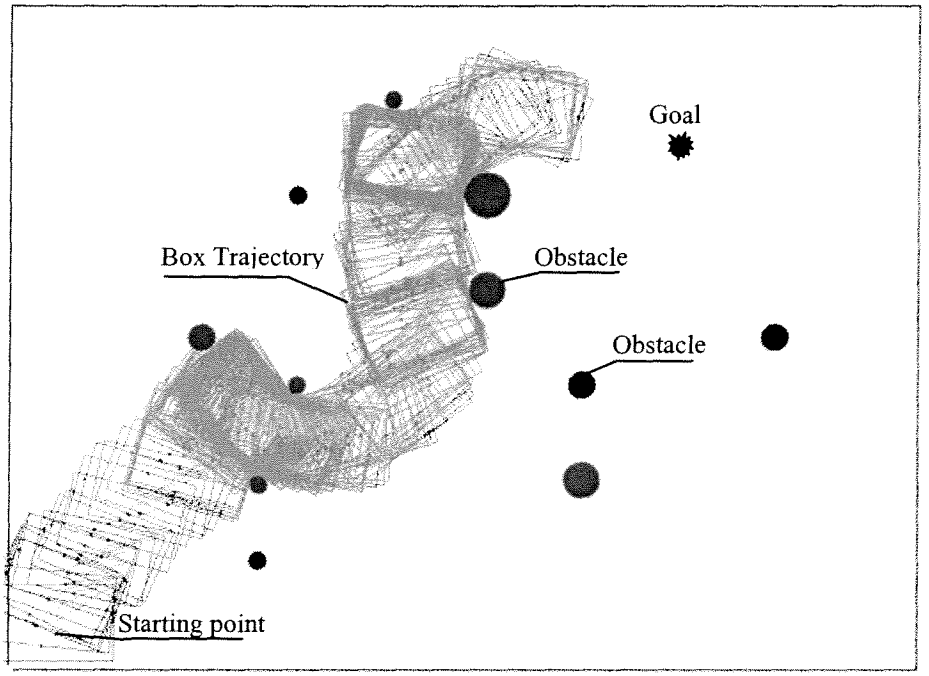
When the box reaches the goal location or the total step number in one round exceeds the maximum allowed step number, this round of box-pushing is completed and a new round of box-pushing is started. In addition, every time the next round of box-pushing is started, the locations of the box, the obstacles, and the goal are selected randomly. However, the Q-table of each robot is inherited from the last round. When the total number of rounds reaches the desired number, the simulation system is stopped.

Parameters of the simulation system are: β =0.9, $\tau = 0.9$, $\eta = 0.8$, $c_f = 0.2, c_t = 0.02, c_d = 0.5, w_1 = 0.7, w_2 = 0.05,$ $w_3 = 0.25, l = 10$, the probability of

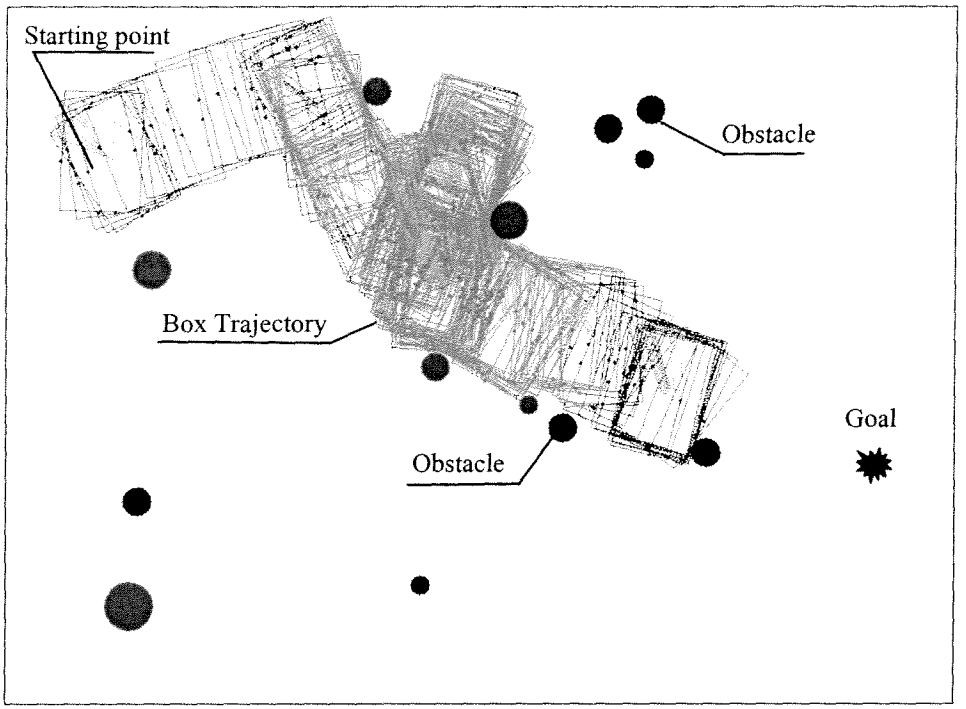
selecting random actions is 0.2, the total number of obstacles in the environment is 11, and the maximum number of steps per round is 6,000.

3.4.4.1 Box-pushing simulation results

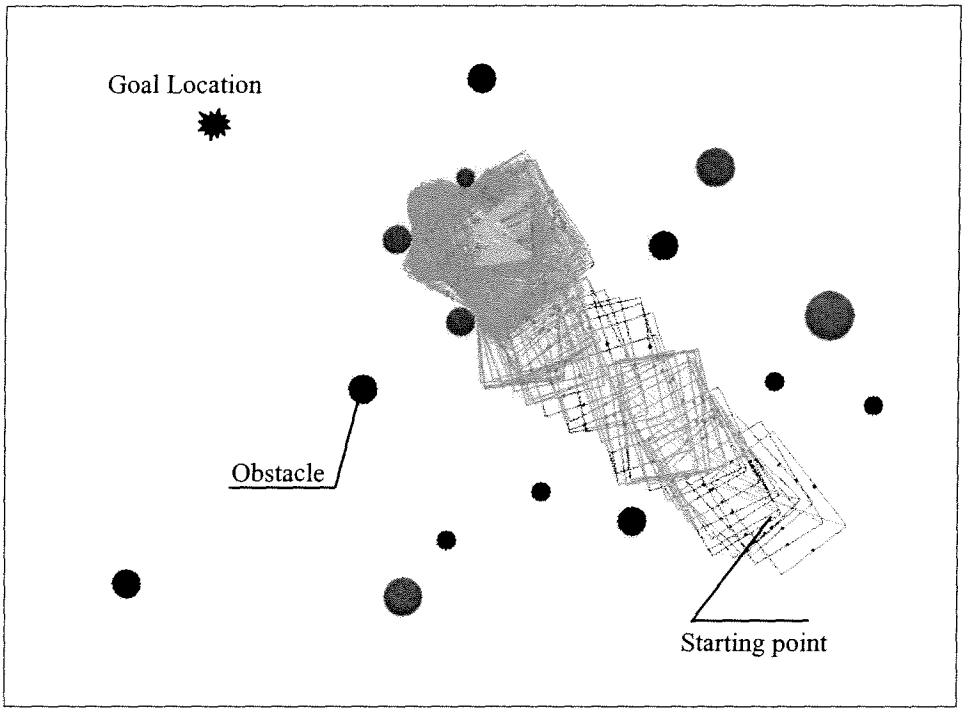
Three different box-pushing snapshots under different obstacle distributions are presented in Figure 3.18 where three robots represented by very small circles are located on the sides of the box. The eleven big circles in Figure 3.18 represent the obstacles in the environment.



(a) A successful box-pushing task.



(b) Another successful box-pushing task with a different obstacle distribution



(c) A failed box-pushing task because of a local maximum

Figure 3.18: Three box-pushing snapshots under different obstacle distributions and goal locations.

In Figure 3.18 (a), the three robots first cooperatively push the box upwards to approach the goal location in the top right corner. However, the box reaches an obstacle in the path. After several failed trials, the robots select to slightly push the box downwards to avoid the obstacle. Then they push the box upwards again to approach the goal location. However, before the box reaches the goal location, it reaches another obstacle. The robots again adapt their actions to the new states and avoid the obstacle. Finally, the box is successfully pushed to the goal location. Figure 3.18(b) presents another successful box-pushing task with a different obstacle distribution.

Figure 3.18(a)-(b) demonstrates that the single-agent reinforcement learning algorithm can be extended to the multi-robot box-pushing domain in an effective and successful manner. Even though each robot is only equipped with a simple single-agent Q-learning algorithm, and no communications exist among robots except the synchronization signal, the robots still can cooperatively push the box to the goal location in an unknown environment with a random obstacle distribution. In the process of pushing and obstacle avoidance, the robots demonstrate a flexible and adaptive action selection strategy in a complex and unknown environment. In addition, the system employs a fully distributed multi-robot decision-making architecture with local sensing capability and limited communications (only the synchronization signal). Consequently, it is robust and scalable, and may be easily employed in various complex environments and applications.

It is also observed that the learning algorithm is not perfect in every aspect. It can suffer from the lack of local sensing and the presence of a local maximum. A typical case is shown in Figure 3.18(c), where the robots are trapped in a local maximum and they cannot avoid the obstacles to reach the goal. The problem can be solved partly by increasing the sensing radius. However, that will lead to rapid expansion of the learning space, which makes the algorithm ineffective.

3.4.4.2 Effectiveness of single-agent reinforcement learning in a multi-robot cooperative task

In general, in a multi-robot environment, because a robot's reward is affected by the actions of its peers, the assumption of a stationary environment in the single-agent Q-

learning algorithm is violated. This means the Q-learning algorithm makes little sense in a multi-robot environment. However, Yang and Gu (2004) suggested that it was possible for single-agent Q-learning to learn the coordination among robots in a purely cooperative multi-robot task. With the multi-robot box-pushing simulations presented in this section, this observation is verified and supported. Furthermore, in the following figures, some probability analysis results are employed to show why the single-agent Q-Learning can work in a multi-robot cooperative task.

It should be noted that the Q-learning algorithm presented in this chapter is different from Littman's algorithm (Littman, 2001a). The latter is a type of "SG-based" Q-learning, and requires a robot to monitor the actions of all peer robots. Consequently, it also suffers due to the extensive learning space problem, highlighting the difference between Littman's algorithm and the "distributed learning" algorithm discussed in the present section.

In the computer simulations, the robots are allowed to push the box continuously for 100,000 rounds to gain experience and update their respective Q-tables. Then one environmental state is selected randomly for analysis, whose state value is binary "1000000" or decimal "64," and its Q values corresponding to the six action candidates are examined. According to the coding rule of the environmental states in section 3.4.3.1, the binary value of "1000000" represents an environmental state, which is shown in Figure 3.19.

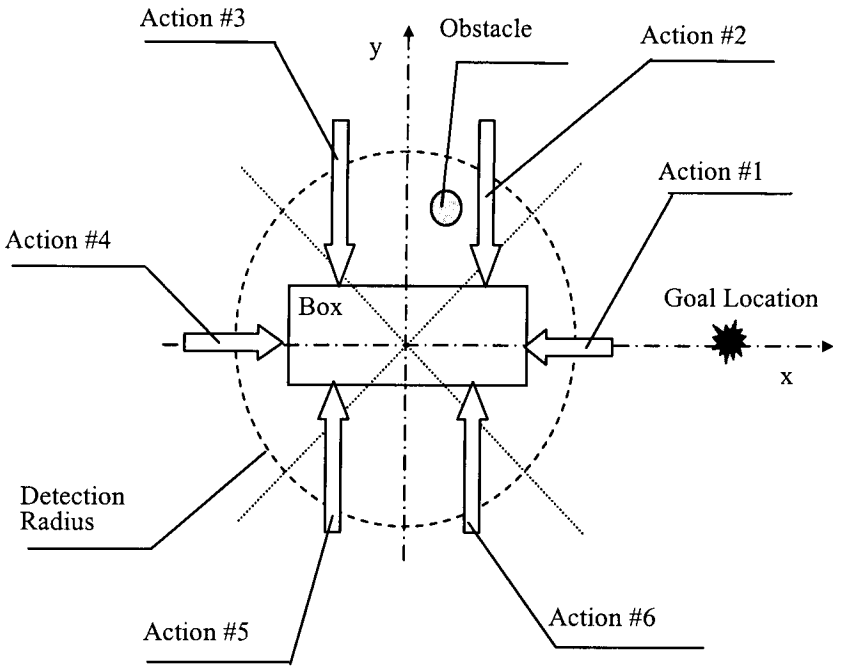


Figure 3.19: The environmental state with binary value "1000000" and six pushing actions available to the robots.

The Q values under the state of "1000000" in 100,000 rounds of box-pushing are recorded and shown in Figure 3.20, and the probability density estimate of these Q values is given in Figure 3.21.

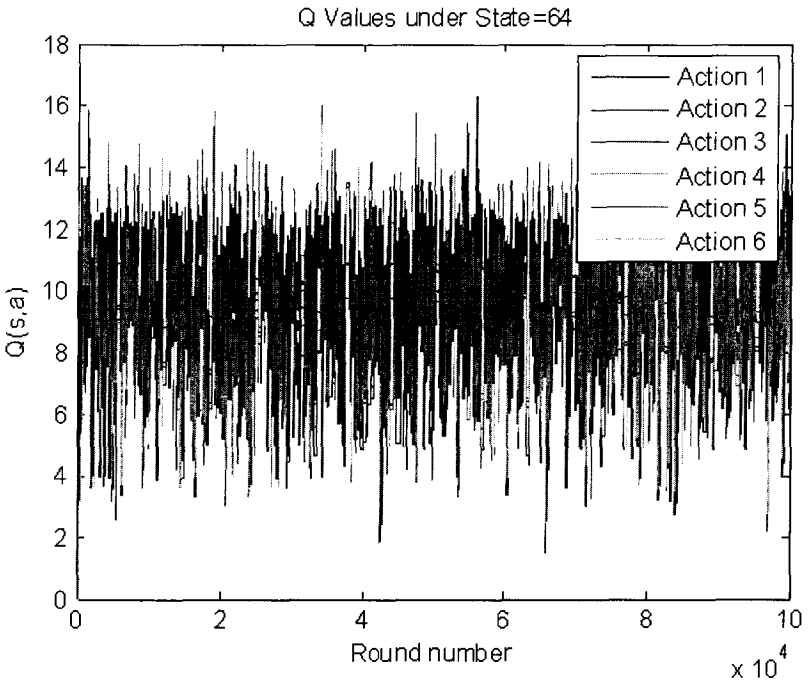


Figure 3.20 The Q values under the state of "1000000" in 100,000 rounds of box-pushing.

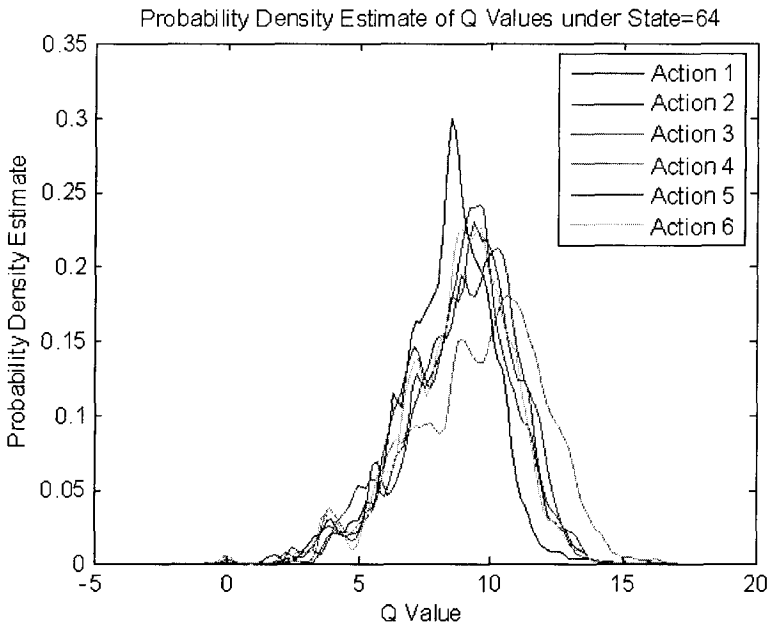


Figure 3.21: Probability density estimate of Q values under state of "1000000" in 100,000 rounds of box-pushing.

It is known that if the environment is stationary, the Q values in a single-robot system will usually converge to some values with probability one (Yang and Gu, 2004). However,

in the multi-robot box-pushing task presented in this chapter, from Figure 3.20, it is seen that the Q values increase rapidly from the initial values of zero to some positive values in a very early stage. Then they begin to fluctuate in a bounded area with an upper bound of 16 and a lower bound of 2. From Figure 3.20 and Figure 3.21, it is rather difficult to find which action is selected at higher probability under the state of "1000000." The Q values in Figure 3.20 seem to fluctuate randomly and all the curves overlap. Therefore, it appears that the robots do not learn any cooperative strategy in the training process and only randomly select their actions. This phenomenon conflicts with the results in Figure 3.18.

In order to reveal the action selection strategy learned by the robots, the action selection probability in each round is calculated using the equation $P(a_k)=\frac{e^{\bar{Q}(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{\bar{Q}(s,a_l)/\tau}}$, and the probability density estimate of the action selection probability is determined with the sample of 100,000 rounds of box-pushing. The result is presented in Figure 3.22.

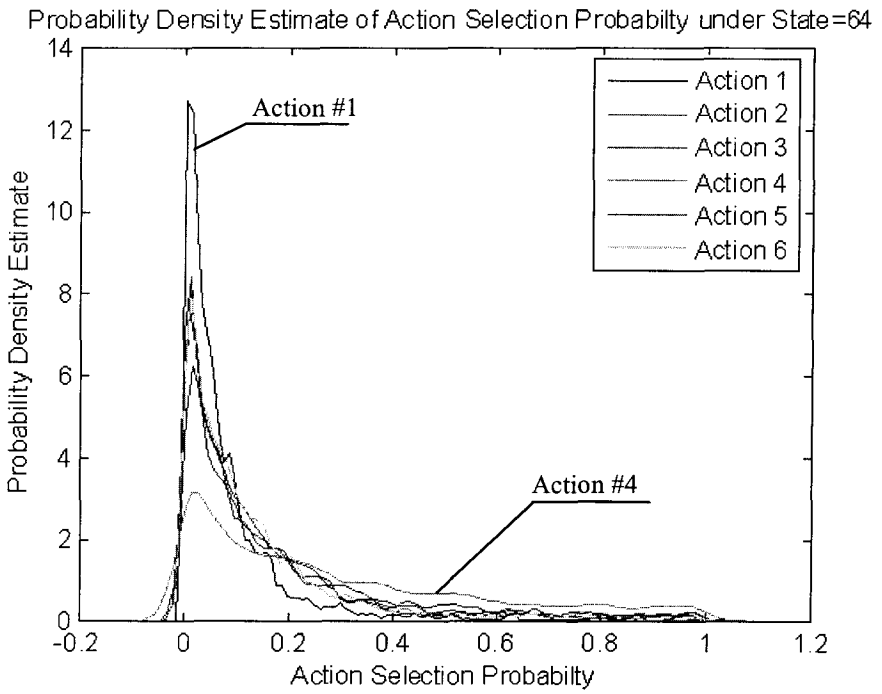


Figure 3.22: Probability density estimate of action selection probability under the state of "1000000" in 100,000 rounds of box-pushing.

Figure 3.22 gives the probability density estimate of action selection probability under a special state in 100,000 rounds of box-pushing. From Figure 3.22 it is seen that there is a high probability that the robots select action #1 in a low probability, and there is a lower probability that the robots select action #4 in a low probability. In other words, action #1 is selected by the robots in low probability under the environmental state of "1000000" and action #4 is selected by the robots in higher probability under this state. From Figure 3.19 it is observed that action #1 pushes the box away from the goal location, and action #4 pushes the box towards the goal without colliding with the obstacle. By comparing Figure 3.22 with Figure 3.19, it is easy to conclude that the robots learn the correct action selection strategy when they cooperatively push the box to the goal location. This explains why the robots in Figure 3.18 can cooperate to push the box to the goal successfully in a complex and dynamic environment with random obstacle distributions.

3.5 Team Q-learning Based Multi-robot Cooperative Transportation

In section 3.4, the single-agent Q-learning algorithm is directly extended to the multi-robot domain for decision-making. Although some positive results were obtained, such extensions do not have a solid theoretical foundation because they violate the static environment assumption in single-agent Q-learning. In the multi-robot domain, the environment is no longer static because multiple robots may change it simultaneously (Yang and Gu, 2004). Therefore, robots may be confused if the single-agent Q-learning algorithm is directly employed in a multi-robot environment. This kind of confusion can be observed in Figure 3.20 where no action prevails over other actions under a specific state.

It appears important for a multi-agent Q-learning algorithm to model a dynamic environment instead of a static environment in the single-agent Q-learning algorithm. Several types of multi-agent reinforcement algorithms, such as MiniMax Q-learning and Nash Q-learning, have been developed by the multi-agent community (Yang and Gu, 2004; Littman, 1994). All of them assume a dynamic environment and are proved to converge to optimal policies under certain conditions. These algorithms enjoy a good theoretical foundation and are preferable in multi-agent tasks over the simple extension of single-agent reinforcement learning. In particular, Littman suggests using the team Q-learning

algorithm for purely cooperative games, which is much simpler than the Nash Q-learning algorithm but has comparable performance (Littman, 2001a).

In this section, the team Q-learning algorithm is applied to a multi-robot box-pushing task in an unknown environment with a complex obstacle distribution. A fully distributed multi-robot system will be developed, which is equipped with learning-based decision-making and local sensing capabilities. Using computer simulations, it will be studied how the learning parameters such as random action probability, detection radius, and the number of robots affect the performance of the team Q-learning algorithm in a multi-robot task.

3.5.1 Stochastic Games and the Team Q-learning Algorithm

Most multi-agent reinforcement learning algorithms are based on the framework of Stochastic Games (SGs) (Yang and Gu, 2004). A learning framework of SGs is defined by the tuple $<S, A_1, \ldots, A_n, T, R_1, \ldots, R_n, \beta>$, where

- S is a finite set of states and n is the number of agents.
- $A_1, \ldots, A_n$ are the corresponding finite sets of the actions available to each agent.
- $T: S \times A_1 \times A_2 \times \ldots \times A_n \rightarrow S$, is a transition function which maps the current state to the next state.
- $R_i : S \times A_1 \times A_2 \times \ldots \times A_n \rightarrow \Re$, is a reward function which describes the rewards received by the i_{th} agent.
- $0 < \beta < 1$ is the discount factor.

In a SGs framework, the joint actions of the agents determine the next state and the rewards received by each agent. Based on the interaction history, each agent tries to find the optimal action-selection policy that maximizes the expected sum of the discount rewards in the future. For a purely cooperative game in which each agent shares the same goal and receives the same payoff as its peers, Littman suggests the team Q-learning algorithm, which is a simplified version of the Nash Q-learning algorithm, to find the optimal policy (Littman, 2001a). The team Q-learning algorithm is given below

- For each state $s_i \in S$ and joint action $(a_1, \ldots, a_n) \in (A_1, \ldots, A_n)$, initialize the table

entry $\hat{Q}(s_i, a_1, ..., a_n)$ to zero

- Observe the current state s
- Do repeatedly the following:
  - Let $(a_1^*, ..., a_n^*) = \arg\max_{(a_1, ..., a_n)} Q(s, a_1, ..., a_n)$. Execute the joint action $(a_1^*, ..., a_n^*)$.
  - Receive an immediate reward r
  - Observe the new state s'
  - Update the table entry for $\hat{Q}(s, a_1^*, ..., a_n^*)$ as follows:

    $$\hat{Q}(s, a_1^*, ..., a_n^*) = (1-\varepsilon)\hat{Q}(s, a_1^*, ..., a_n^*) + \varepsilon(r + \beta \max_{a_1, ..., a_n} \hat{Q}[s', a_1, ..., a_n])$$

    where, $0 \le \varepsilon < 1$ is the learning rate
  - $s \leftarrow s'$

After a sufficient number of iterations, the $\hat{Q}(s_i, a_1, ..., a_n)$ will converge to the optimal value. In a purely cooperative game, compared with the Nash Q-learning algorithm, the team Q-learning algorithm requires less computational resources, and exhibits comparable performance (Littman, 2001a).

The standard team Q-learning algorithm uses the greedy policy to choose actions (Littman, 2001a). However, this policy may explore too few states to converge to the optimal policy. Therefore, in this section, a modified GLIE policy (Greedy in the Limit with Infinite Exploration) as given below is used to guarantee the team Q-learning algorithm to converge to optimal policies (Littman, 2001a):

$$P_{random} = \begin{cases} c_r / n(s) & if\ c_r / n(s) > P_{threshold} \\ P_{threshold} & if\ c_r / n(s) \le P_{threshold} \end{cases} \tag{3.19}$$

where, P_{random} is the probability of selecting random actions, $n(s)$ is the number of times state s has been encountered so far during learning, $0 < c_r < 1$ is a constant, and $P_{threshold}$ is the threshold value of P_{random}.

3.5.2 Simulation Results

A simulation system is developed with the Java language to simulate a multi-robot box-pushing system using the team Q-learning algorithm. In this simulation system, the

representation of environmental states, robot actions and the reward function of the team Q-learning algorithm are the same as those in the single-agent Q-learning algorithm, as described in section 3.4.3. The dimensions of the environment are 950×700 units and the dimensions of the box are 120×80 units. There are three robots, denoted by circles with radius 2, in the subsequent figures, which push the box to the goal location. Parameters of the simulation system are β=0.9, $\varepsilon = 0.8$, $c_f = 0.2, c_t = 0.02, c_d = 0.5, w_1 = 0.7, w_2 = 0.05$, $w_3 = 0.25, l = 10$, $c_r = 0.9$, $P_{threshold} = 0.1$, the detection radius is 72, the maximum number of steps per round of box-pushing (from the beginning position to the goal position) is 5,000, and the total number of rounds in the training stage is 10,000. Figure 3.23 presents a round of box-pushing after training.

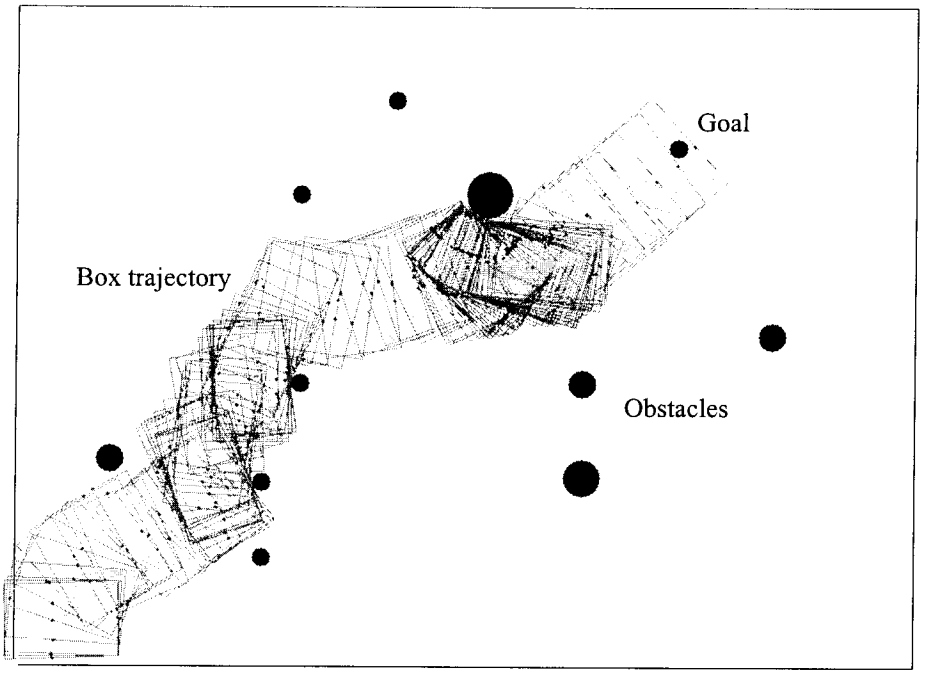


Figure 3.23: The box trajectory in a round of box-pushing.

The rewards received by the robots are shown in Figure 3.24 and the Q values learned by the robots after 10,000 cycles of training are shown in Figure 3.25.

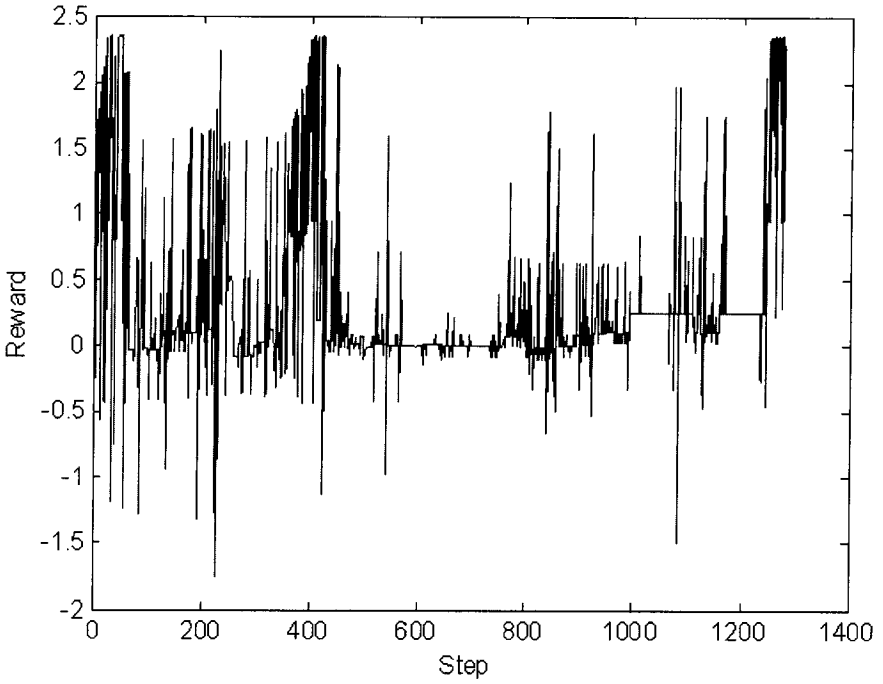


Figure 3.24: The rewards received by the robots in a round of box-pushing.

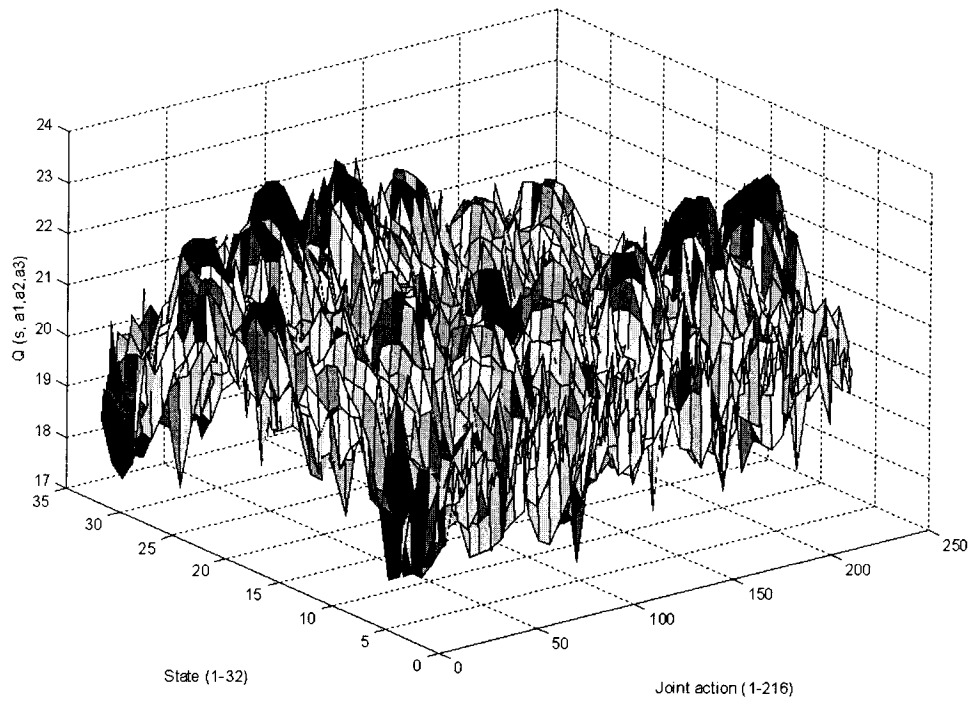


Figure 3.25: The surface of the Q-table (the states from 1 to 32) after 10,000 training cycles.

From Figure 3.23 to Figure 3.25, it may be concluded that the team Q-learning algorithm can be effective in a multi-robot box-pushing task. As shown in Figure 3.23, the box was successfully pushed to the goal by the robots in an environment with complex obstacle distribution. The rewards shown in Figure 3.24 indicate that the robots received positive rewards most of the time although sometimes they received negative values for colliding with the obstacles. The robots quickly adapted to new states with a different obstacle distribution and pushed the box to the goal without any centralized strategies. The Q-table surface in Figure 3.25 also indicates that the robots learned and increased the Q values in 10,000 rounds of training.

3.5.3 Impact of Learning Parameters

In this section, the impact of several learning parameters on the team performance will be assessed. These parameters are the random action probability, the detection radius, the number of training rounds, the pushing forces of the robots, and the number of robots. Prior to assessing the impact of the learning parameters, the robots are given 10,000 rounds of training in box-pushing. Then, based on the knowledge acquired in the training stage, the robots push the box for 1,000 rounds and record the number of steps in each round, to assess the performance of the team Q-learning algorithm, based on the number of steps per round.

3.5.3.1 Impact of training

The impact of training on the team performance is assessed in Figure 3.26.

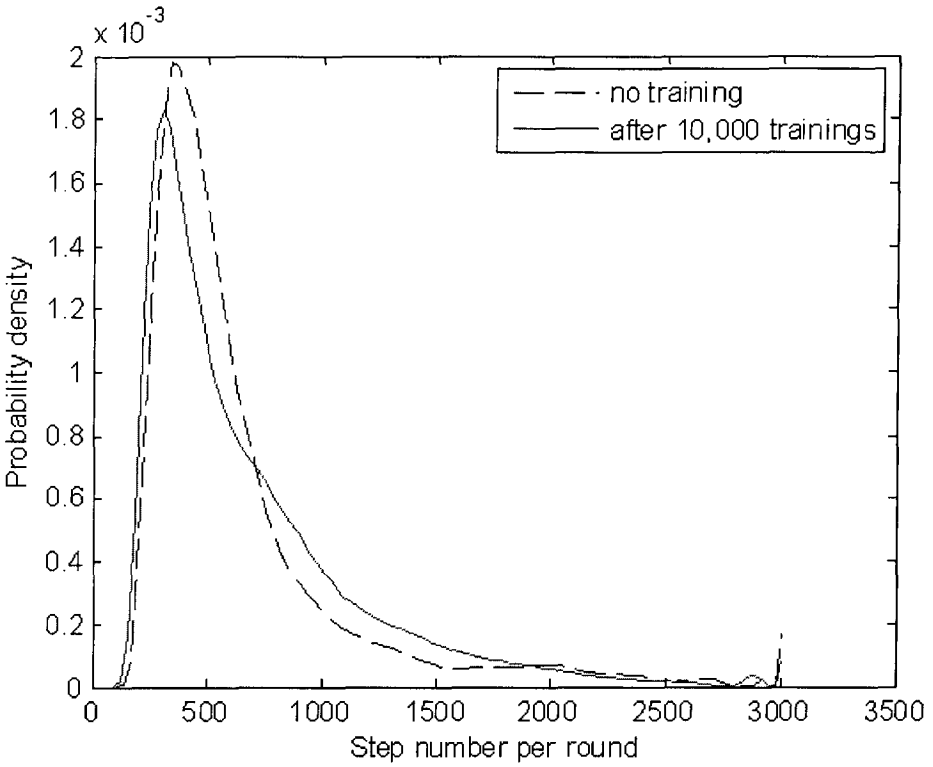


Figure 3.26: The impact of training.

In Figure 3.26, two cases are compared. The first one corresponds to "no training" in which the robots pushed for 1,000 rounds without any preliminary training. In the second case, denoted by "after 10,000 trainings," the robots first pushed the box for 10,000 rounds to gain experience and then pushed the box for further 1,000 rounds to assess the team performance. The probability distributions of the number of steps per round in the two cases are shown in Figure 3.26. They indicate that training is helpful but it has only a small influence on the team performance.

3.5.3.2 Random action probability.

In equation (3.19), the modified GLIE policy is used to determine the random action probability of the robots so that the team Q-learning algorithm explores an adequate number of states to converge to the optimal policy. Here, the impact of the random actions is assessed. Three different lower bounds of the random action probability ($P_{threshold}$ *=0, 0.3 and 0.6)* are employed by the robots to push the box, and the results are presented in Figure 3.27.

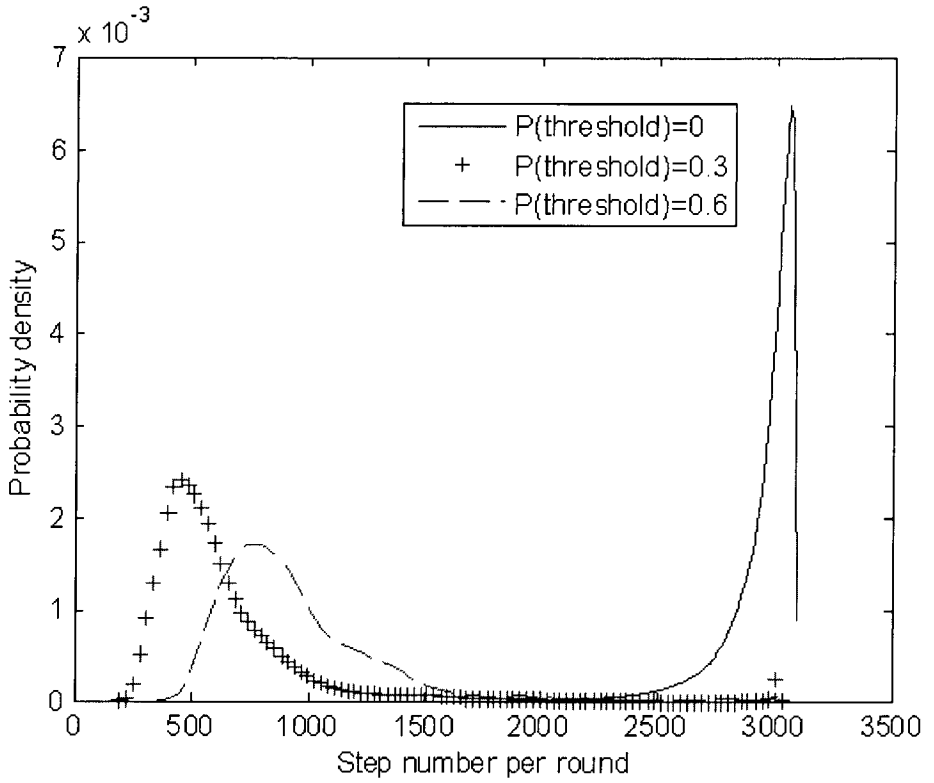


Figure 3.27: Impact of $P_{threshold}$ (Lower bound of the random action probability).

Figure 3.27 illustrates that random actions are very important for the team Q-learning algorithm. When $P_{threshold}$ is equal to zero, the number of steps per round is 3,000 at a high probability. Because the maximum number of steps per round is set to 3,000 in the simulation program, it means that the robots cannot complete the task in the desired number of steps at a high probability when $P_{threshold}$ is equal to zero. In other words, the standard team Q-learning algorithm with GLIE policy (i.e., $P_{threshold}$=0) cannot work in an environment with complex obstacle distribution because it cannot escape from a local minimum. When $P_{threshold}$ increases to 0.3, the number of steps per round decreases to 500 at all probabilities. It means that the robots push the box to the goal more quickly. However, when $P_{threshold}$ increases to 0.6, the number of steps per round increases to 700 at all probabilities because too many random actions make the robots fail to commit themselves to the goal and reduce the team performance.

3.5.3.3 Detection radius

In the developed multi-robot box-pushing system, each robot only possesses the local sensing capability. Only the obstacles within the detection radius can be sensed by the robots. Here, the impact of the detection radius on the team performance is assessed, as presented in Figure 3.28.

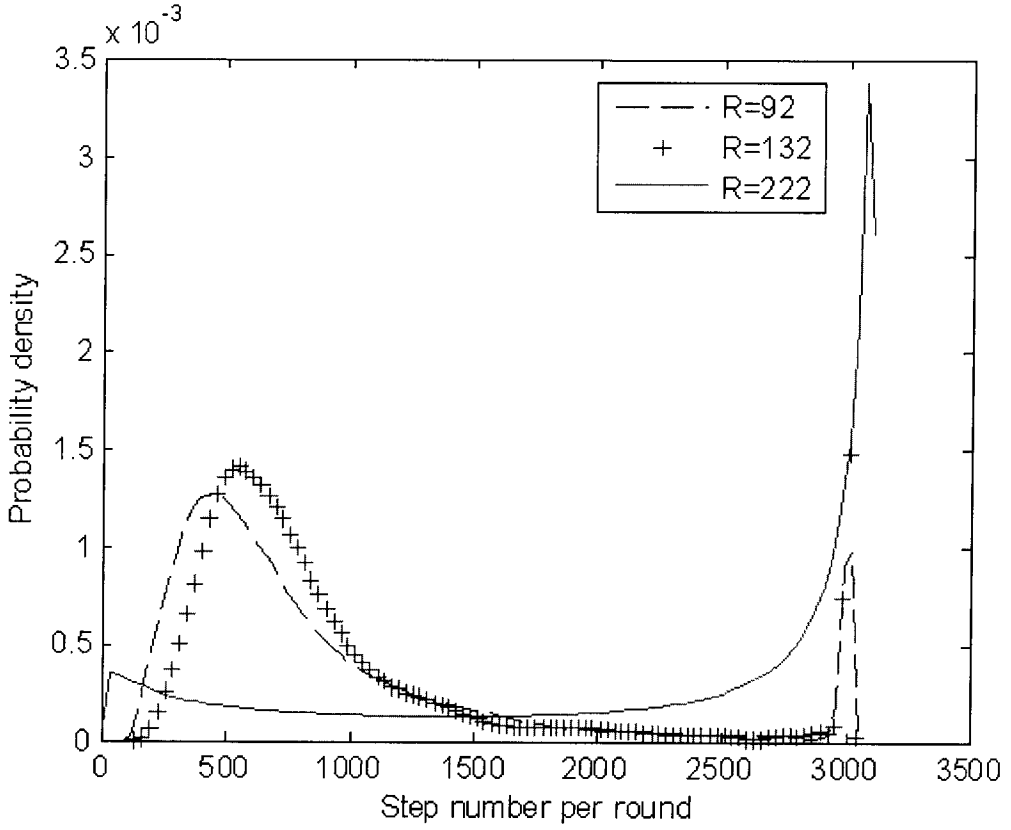


Figure 3.28: The impact of the detection radius.

In Figure 3.28, three multi-robot box pushing systems with different detection radii (*R=92, R=132, and R=222*) are compared. Although a bigger detection radius helps the robots to find more local obstacles, Figure 3.28 indicates that team performance degrades when the detection radius increases.

3.5.3.4 Pushing force

In the previous sections, a team of homogenous robots each possessing the same capabilities is assumed. In particular, each robot has the same pushing force ($F_i = 10, i = 1,2,3$). However, here a team of robots with different pushing forces are

assessed, as presented in Figure 3.29.

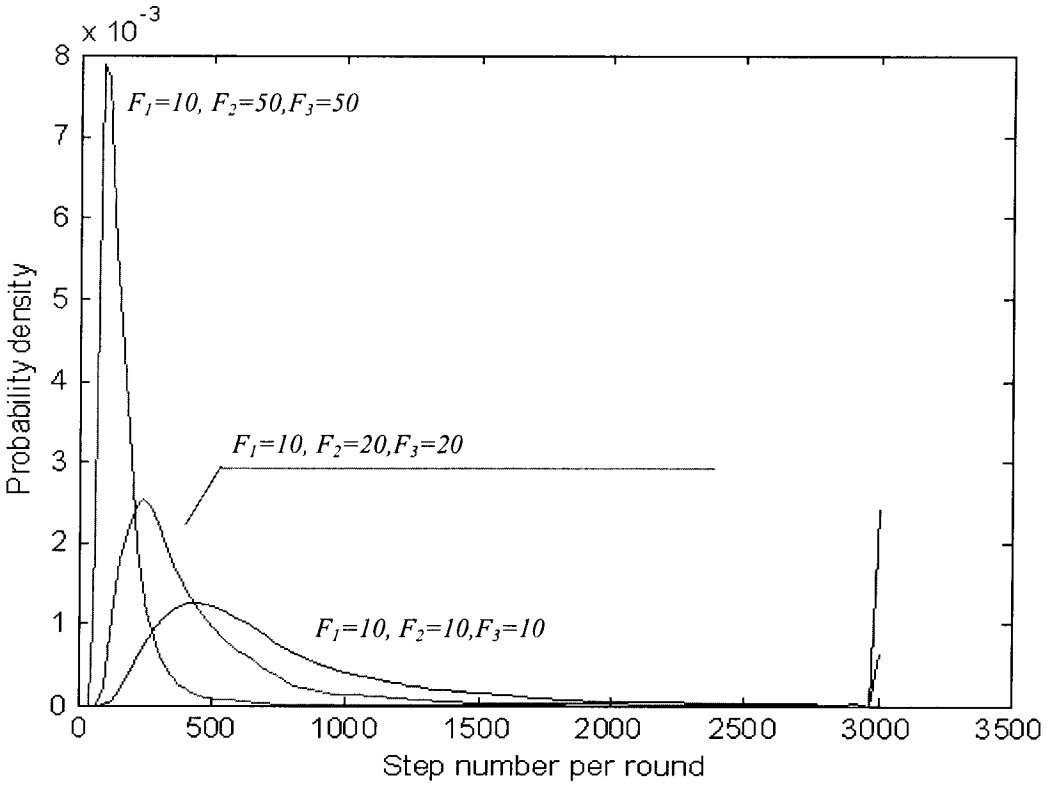
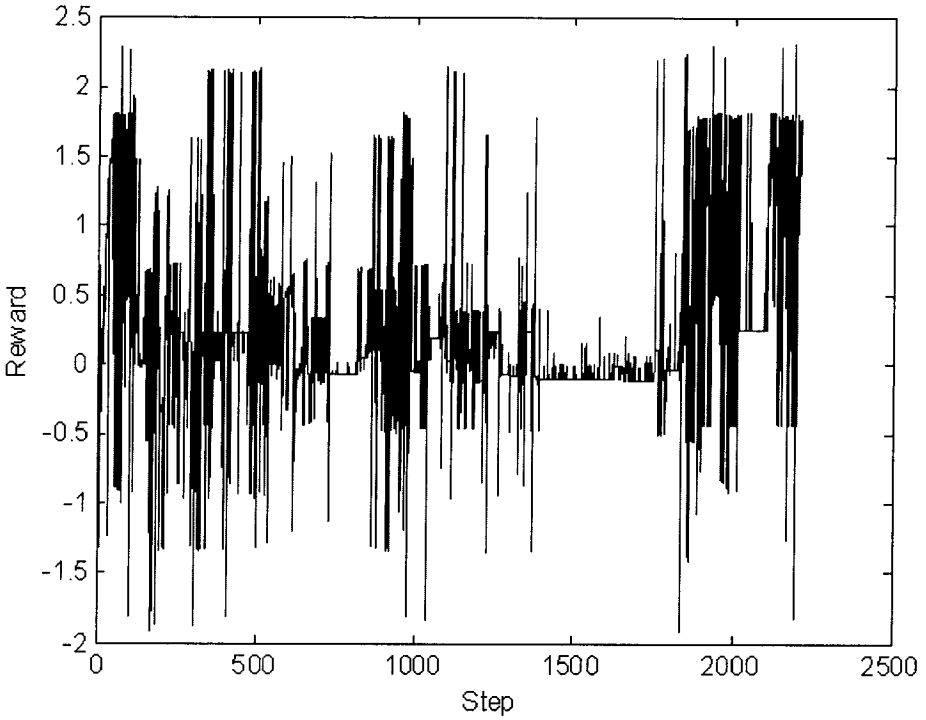


Figure 3.29: The impact of the pushing force.

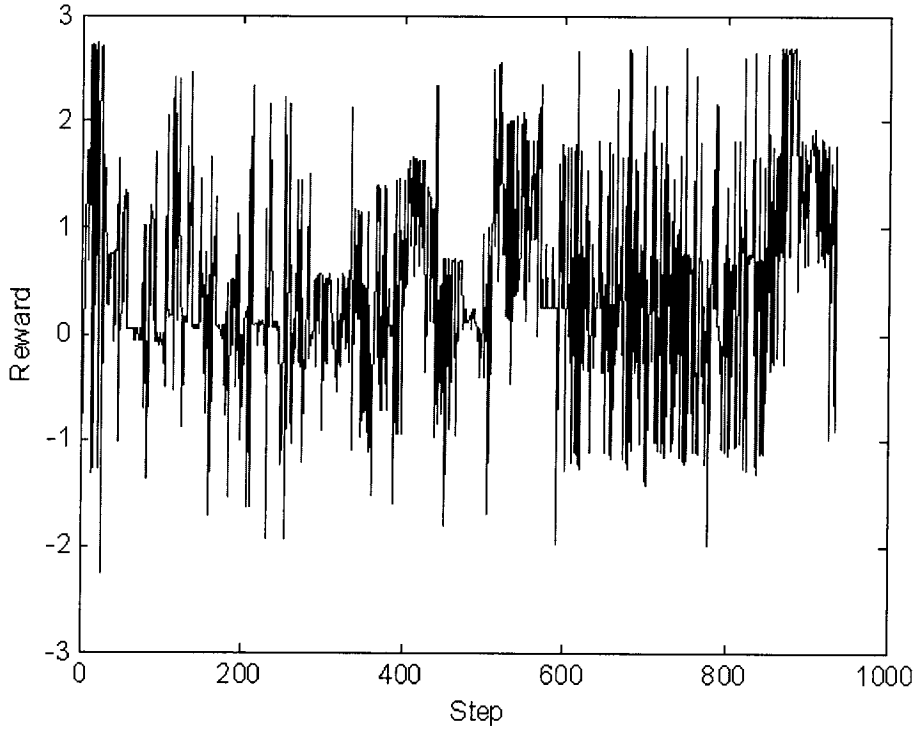
Figure 3.29 indicates that a bigger pushing force helps to improve the team performance because it enables the robots to move the box more quickly and makes it easy to escape from a local minimum. However, Figure 3.29 does not indicate that a bigger pushing force will result in excessive box rotations, which can make it difficult for the robots to handle the box.

3.5.3.5 The impact of the number of robots and the disadvantage of team Q-learning

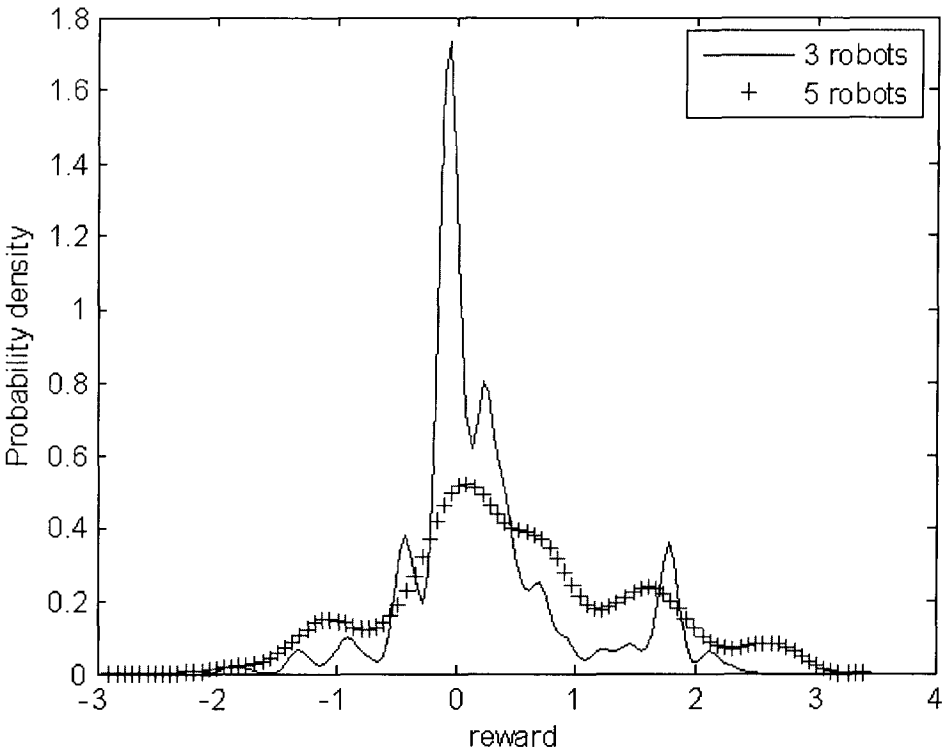
In the previous sections, three robots were used to push the box. Now, five robots are assumed to complete the same task and compare its team performance with that of the three-robot team. The rewards received by the robots in a round of box-pushing are shown in Figure 3.30.


2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Reward
0
500
1000
1500
2000
2500
Step

(a)


3
2
1
0
-1
-2
-3
Reward
0
200
400
600
800
1000
Step

(b)



(c)

Figure 3.30: The rewards received by the robots in a round of box-pushing. (a) The rewards received by the three-robot team. (b) The rewards received by the five-robot team. (c) The probability density of the rewards.

Figure 3.30 indicates that the five-robot team does not do a better job than the three-robot team. In particular, the former does not get more rewards than the latter. Because the net force is bigger in the five-robot team, if the members cooperate correctly, the number of steps per round should be much smaller, as indicated in Figure 3.30. However, this team performance is improved at a cost of much heavier computational burden.

The main disadvantage of the team Q-learning algorithm is its extensive learning space (state/action space). When the number of robots increases, the learning space grows quickly. For example, in this section, there are three robots and 8,192 environmental states, and each robot can select one of six available actions. Then the Q-table size is 8192×6^3 ($\approx$1.7M). However, if the number of robots increases to 10, the Q-table size will increase to 8192×6^{10} ($\approx$495G) which will require an extensive memory space in the host

computer. Clearly, it is very difficult and even impossible to operate and maintain such a huge memory space in a computer. Therefore, when the number of robots is more than 10, the team Q-learning algorithm can become entirely ineffective. In this case, directly extending the single-agent Q-learning algorithm to the multi-robot domain becomes an appropriate alternative because its learning space is fixed.

In this section, the team Q-learning algorithm has been assessed in a multi-robot box-pushing task. As a type of multi-agent reinforcement learning algorithm, the team Q-learning algorithm possesses a good theoretical foundation. Hence, it is preferable over the single-agent Q-learning algorithm for multi-robot problems. The algorithm of team Q-learning has been presented in this section and a multi-robot box-pushing simulation system developed to assess the algorithm. In addition, the impact of the learning parameters has been studied. The simulation results show that the team Q-learning algorithm is successful in a multi-robot box-pushing system with distributed decision-making, local sensing capabilities and an unknown environment containing obstacles. The main disadvantage of the team Q-learning algorithm is that it only can work in a small robot team, typically with 2 to 5 members.

3.5.4 Comparison of Team Q-learning with Multi-robot Distributed Q-learning

In section 3.4, the single-agent Q-learning (or distributed Q-learning) has been directly extended to a multi-robot box-pushing task. Now, the team Q-learning algorithm is compared with distributed Q-learning, using the same benchmark problem: the multi-robot box-pushing project described in Figure 3.13. Both Q-learning algorithms employ the same representation of the environmental states, robot actions, and reward function, as presented in section 3.4.3. A simulation system is developed to simulate the learning-based multi-robot system, and compare the performances of the two Q-learning algorithms. The maximum number of steps per round in the simulation is 5,000, and the total number of rounds in the training stage is 10,000. The simulation results are presented below.

3.5.4.1 Simulation results without training

The simulation results of the two Q-learning algorithms are presented in Figure 3.31. Note that the robots in Figure 3.31 do not receive any training before they begin to push

the box. Accordingly, all the robots are novice and they do not have any experience in box pushing.

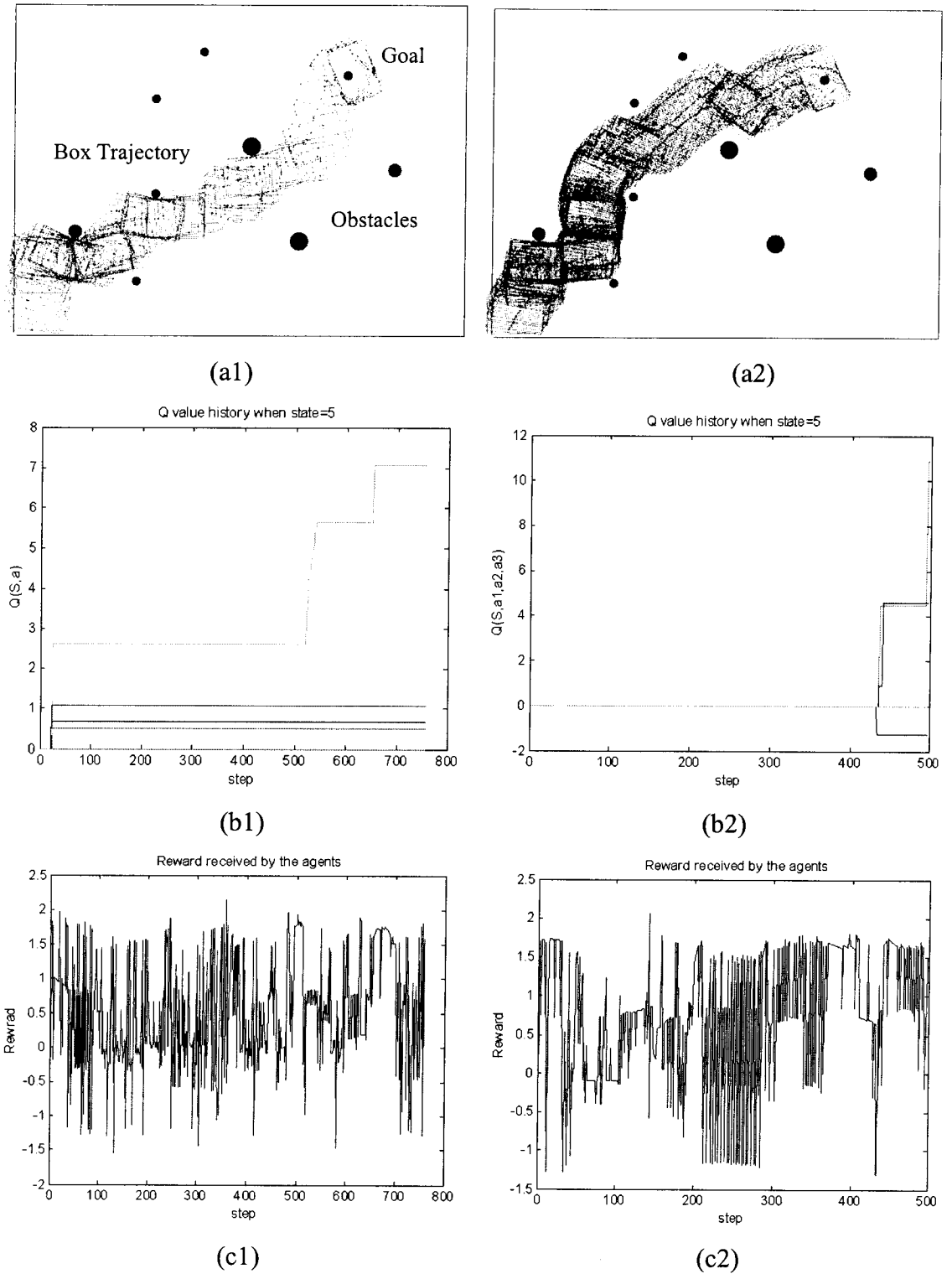


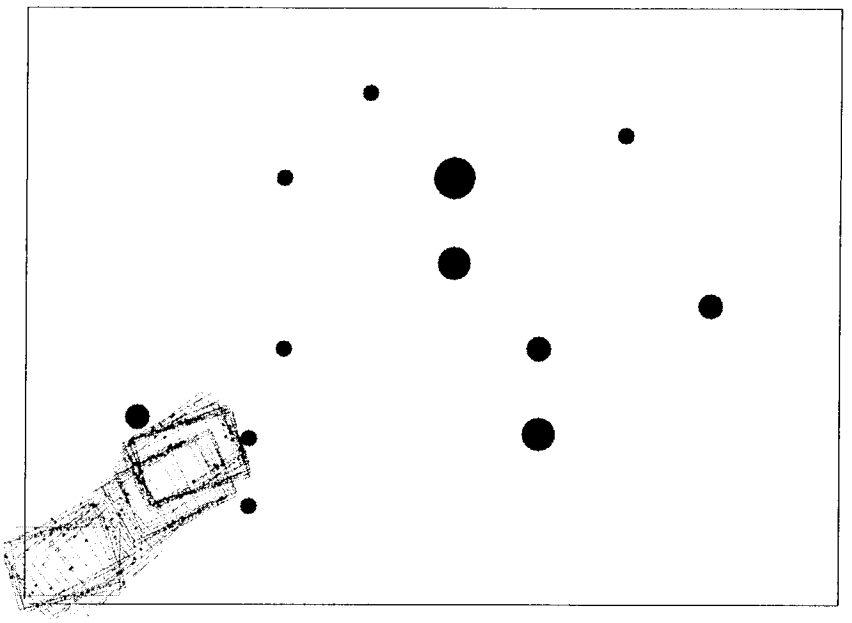
Figure 3.31: Simulation results with the single-agent Q-learning algorithm and the team

Q-learning algorithm: (a1) The box-pushing process with the single-agent Q-learning algorithm; (a2) The box-pushing process with the team Q-learning algorithm; (b1) The Q value history (state=5) in the single-agent Q-learning algorithm; (b2) The Q value history (state=5) in the team Q-learning algorithm; (c1) The received reward in the single-agent Q-learning algorithm; (c2) The received reward in the team Q-learning algorithm.

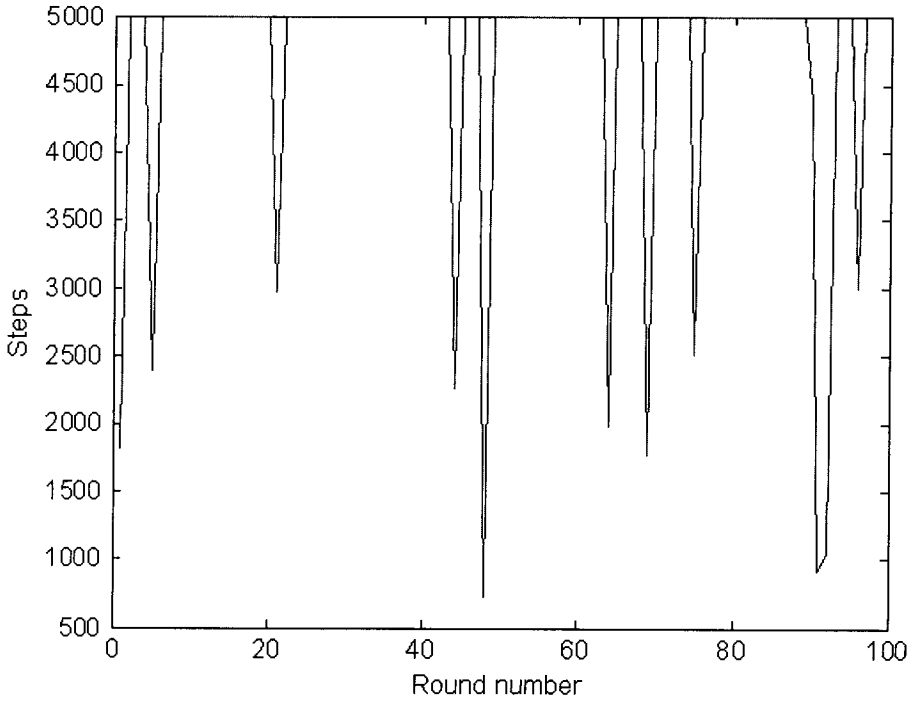
From Figure 3.31 (a1) and (b1), it is seen that both Q-learning algorithms can successfully push the box to the goal location in an unknown environment. Because each robot independently makes decisions and chooses its actions, it is a fully distributed multi-robot system. From Figure 3.31 (a2) and (b2), it is seen that the Q values converge to some fixed values. In Figure 3.31 (a3) and (b3), it is observed that the rewards received by the robots tend to fluctuate. This fluctuation results from reaching unknown obstacles during the pushing process, or disturbance from the actions of its peers. However, most of the time, the robots receive positive rewards.

3.5.4.2 Comparison between two Q-learning algorithms

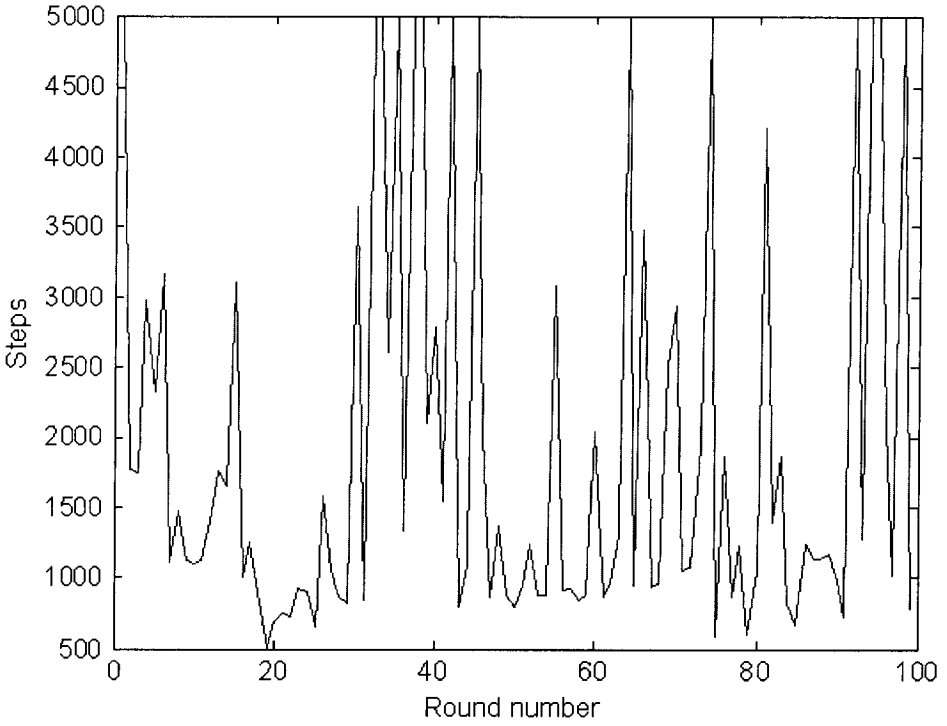
In Figure 3.32, the performance of the single-agent Q-learning algorithm before and after training is compared. Here, the performance index is taken as the reciprocal of the total number of steps per round. All simulations are based on the same complex environment with eleven obstacles, as shown in Figure 3.32 (a).



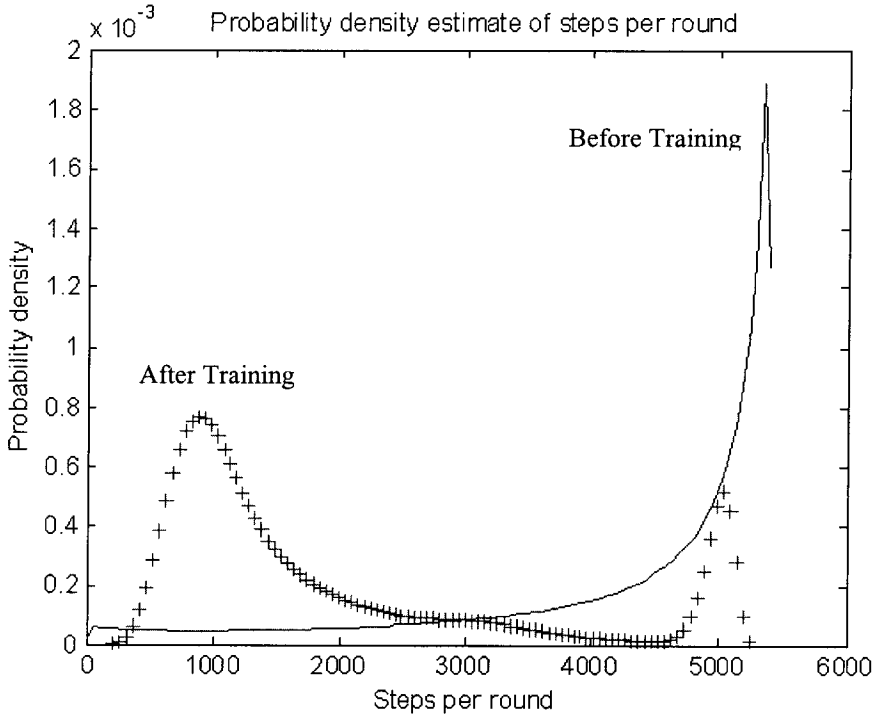
(a) The environment with 11 obstacles.



(b) The total steps per round BEFORE training.



(c) The total steps per round AFTER training.

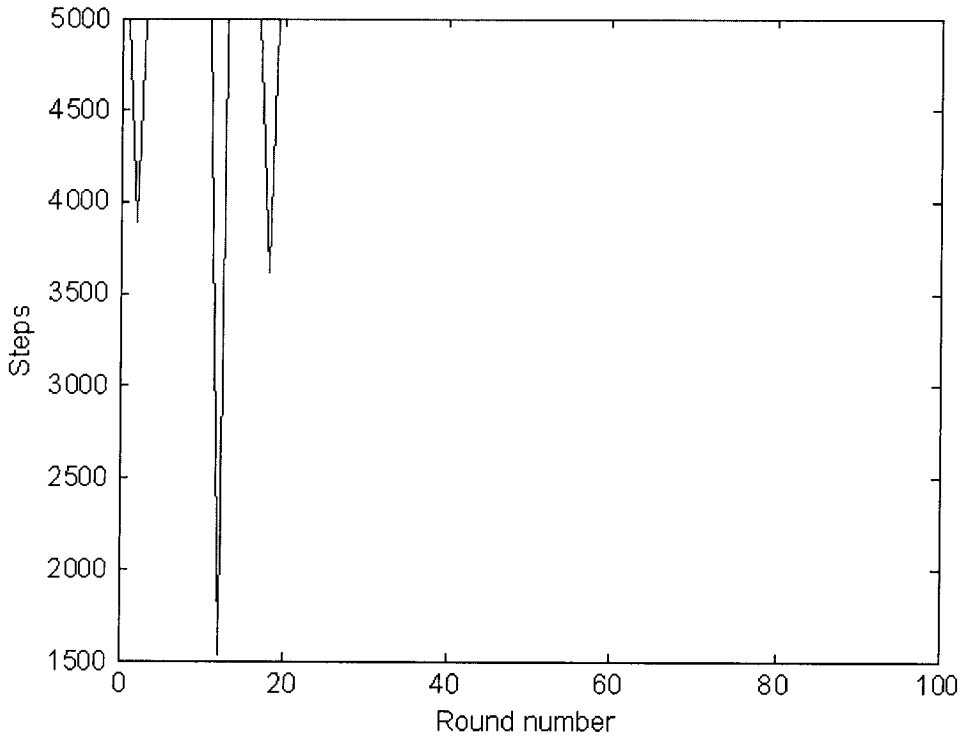


(d) Probability density estimate of steps per round

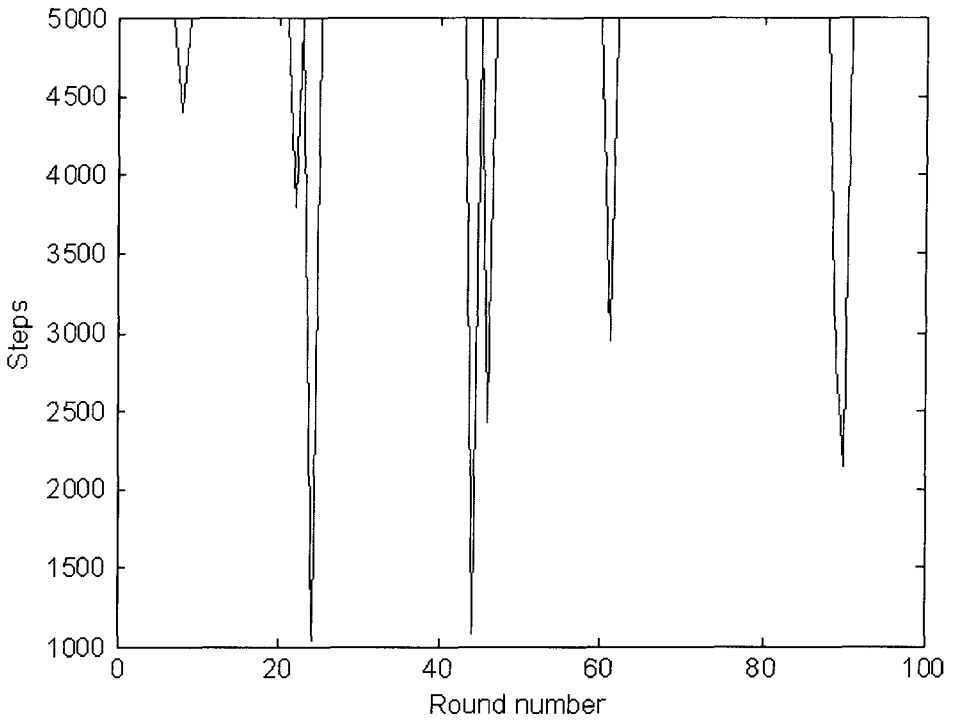
Figure 3.32: Performance index comparison of the single-agent Q-learning algorithm before and after the training stage.

From Figure 3.32 it is seen that the performance index is significantly improved in the training stage. After training, the total number of steps per round is decreased from 5,000 to 1,000. Because the maximum number of steps is set at 5,000 for a round of box-pushing, a total number of steps of 5,000 means that the robots fail to push the box to the goal location within the desired number of steps in that round. Figure 3.32 indicates that the success rate of box pushing greatly increases after training because the robots learn and improve the box-pushing policy through experience.

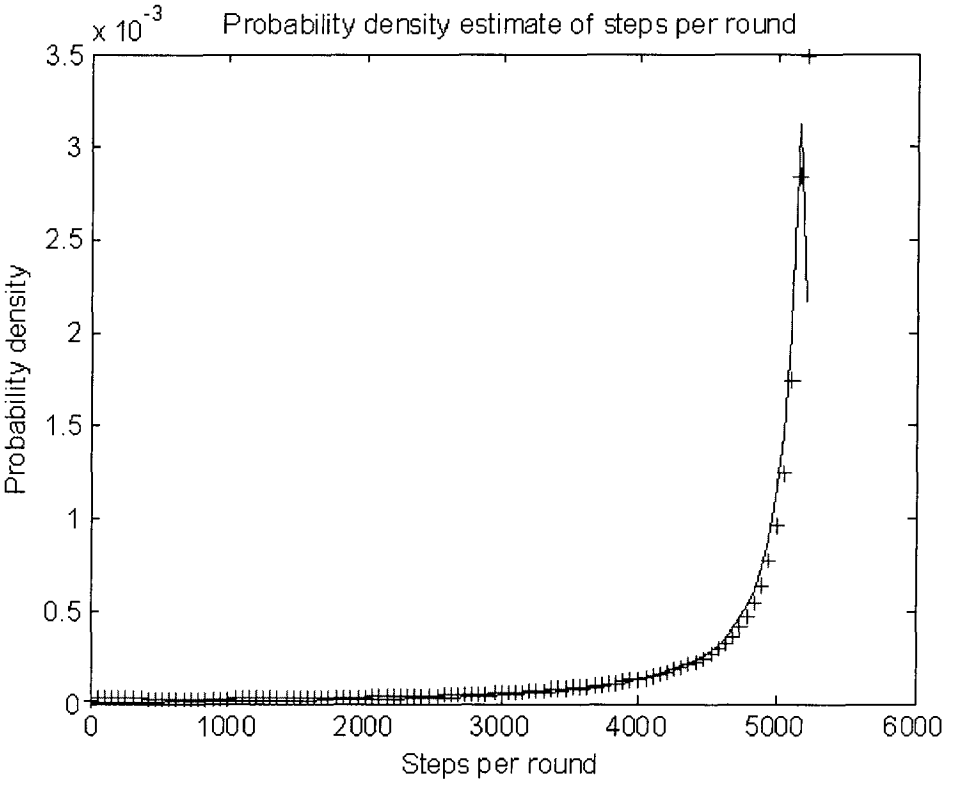
Similarly, the performance of the team Q-learning algorithm before and after training is compared in Figure 3.33. The simulations are based on the same environment as in Figure 3.32 (a).



(a)


5000
4500
4000
3500
3000
2500
2000
1500
1000
Steps
0
20
40
60
80
100
Round number

(b)


x 10^-3
Probability density estimate of steps per round
3.5
3
2.5
2
1.5
1
0.5
0
Probability density
0
1000
2000
3000
4000
5000
6000
Steps per round

(c)

Figure 3.33: Performance index comparison of the Team Q-learning algorithm before and after the training stage. (a) The total number of steps per round BEFORE training. (b) The total number of steps per round AFTER training. (c) Probability density estimate of steps per round before and after training.

From Figure 3.33, the following conclusion may be drawn: Training is not required in the team Q-learning algorithm. The robots do not learn any useful policy for pushing the box in the training stage. It appears that the team Q-learning algorithm does not help the robots to improve their skills using the experience received during the training stage. The total number of steps per round has a high probability index of 5,000 before and after training. It means that the robots always fail to push the box to the goal location in the desired number of steps even after training.

By comparing Figure 3.32 (d) with Figure 3.33 (c), one observes a further interesting result: The single-agent Q-learning algorithm seems to do a better job than the team Q-learning algorithm even though the latter has a stronger theoretical foundation. A possible explanation for this phenomenon is that some random actions are necessary to solve the local minimum problem in a complex and unknown environment for a multi-robot box-pushing task. Although the team Q-learning algorithm with GLIE policy does take random actions, the probability of random action is still too low. Therefore, the team Q-learning algorithm can be easily trapped in a local maximum.

The single-agent Q-learning algorithm is different since a robot/agent does not observe the actions of its peers when it makes decisions. The peer actions are regarded as a type of random disturbance or random "noise" in the environment. When this type of random "noise" is large, it degrades the team performance. However, if the random "noise" is small, it can help the robot team to escape from a local maximum. That is why the single-agent Q-learning algorithm does a better job than the team Q-learning algorithm in the present simulations.

There are two possible ways to improve the performance of the team Q-learning algorithm in an unknown environment with a complex distribution of obstacles. First is to increase the probability of random actions, which will overcome the local minimum

problem. Second is to increase the number of training rounds so that the robot can explore enough states and converge to the optimal policy.

In the present section, the performance of two types of Q-learning algorithms has been comparatively studied in the context of multi-robot box pushing. First, the single-agent Q-learning algorithm was directly extended to the multi-robot domain. Such extensions do not possess a solid theoretical foundation because they violate the static environment assumption of the algorithm. Second, the team Q-learning algorithm was introduced into the multi-robot box-pushing field, which had been specifically designed to solve the purely cooperative stochastic game problem. This algorithm was shown to converge to the optimal policy if an adequate number of explorations were executed. Both algorithms were implemented in a multi-robot box-pushing system. The simulation results showed that both algorithms were effective in a simple environment without obstacles or with a very few obstacles. However, in a complex environment with many obstacles, the simulation results showed that the single-agent Q-learning algorithm did a better job than the team Q-learning algorithm. It is believed that more random disturbances in the former helped it to escape from local minimum.

3.6 Summary

This chapter addressed the application of machine learning to multi-robot decision making. Multi-robot systems face some special challenges; for example, dynamic and unknown environments, which the traditional AI technologies cannot effectively deal with. Therefore, it is important to integrate machine learning with the conventional planning or behavior-based decision-making techniques to help multi-robot systems work in a complex and unknown environment, in a robust and effective manner.

Reinforcement learning is commonly used to make decisions for a single robot due to its good real-time performance and guaranteed convergence in a static environment. In this chapter, first, the single-agent reinforcement learning was directly extended to multi-robot decision-making. The basic idea of the single-agent Q-learning and its framework of Markov Decision Processes were introduced in section 3.2. The first version of learning-based multi-robot transportation was developed in section 3.3 to validate the feasibility of the proposed algorithm. A centralized decision-making architecture was assumed for the

multi-robot system.

In section 3.4, a more challenging multi-robot environment was assumed, which is dynamic and unknown and has more complex obstacle distribution. For such an environment, a distributed multi-robot box-pushing system based on the single-agent Q-learning algorithm was developed. The simulation results showed that the robots still could complete the task well albeit at a slower speed.

In order to overcome the static environment assumption in the single-agent Q-learning algorithm, the team Q-learning algorithm which inherently models the dynamic environment was employed to build a multi-robot decision-making subsystem. The performances of the two Q-learning algorithms were compared and analyzed on the same platform of multi-robot box-pushing.

The simulation results show that both Q-learning algorithms have disadvantages when they are used to make decisions in a multi-robot system, even though they can help robots make good decisions and properly complete tasks in most of cases. Because it violates the assumption of a static environment, directly applying the single-agent Q-learning algorithm to the multi-robot domain will result in very slow convergence speed, and there is no guarantee that the algorithm will converge to the optimal policies. On the other hand, the team Q-learning algorithm will generate an extensive learning space which makes the algorithm infeasible when the number of robots is large. It indicates that a new Q-learning algorithm that is suitable for multi-robot decision-making is required, by integrating the advantages of both single-agent Q-learning and team Q-learning.

Chapter 4
Sequential Q-learning with Kalman Filtering (SQKF)

4.1 Overview

Learning is important for cooperative multi-robot systems whose environments are usually dynamic and unknown. In chapter 3, two Q-learning algorithms are employed to find optimal action policies for the members of a multi-robot system. Although both Q-learning algorithms can help robots find good cooperation policies, they also have some serious disadvantages in a multi-robot environment. Furthermore, directly extending the single-agent Q-learning algorithm to the multi-robot domain violates its assumption of static environment, and as a result the values of the Q-table cannot converge. Although the robots still can find some good policies for cooperation, the performance of the whole team can be degraded when that approach is used. On the other hand, the team Q-learning algorithm models a dynamic environment inherently and is clearly more suitable for multi-robot systems, but it requires an extensive learning space (state/action space) and this space increases rapidly with the number of robots in the system. It is very difficult for a robot to manipulate and manage an extensive learning space in real-time. This is exacerbated by the fact that all states of this extensive space should be visited to guarantee that the team Q-learning algorithm converges to its optimal policies, which is quite difficult to achieve in a real multi-robot system.

These observations indicate that it is important to develop a new Q-learning algorithm which is suitable for real-time operation of cooperative multi-robot systems. In this chapter, a modified Q-learning algorithm termed Sequential Q-learning with Kalman Filtering (SQKF) is developed, which takes inspiration from human cooperation in performing heavy-duty manual tasks, to meet the main challenges of Q-learning in the multi-robot domain. The SQKF algorithm is presented in section 4.2, and it is validated through computer simulation in section 4.3.

4.2 Sequential Q-learning with Kalman Filtering

The Sequential Q-learning algorithm with Kalman Filtering (SQKF) is specially designed to cope with various challenges which the single-agent Q-learning algorithm and the team Q-learning algorithm face in a multi-robot environment. The SQKF algorithm, which is inspired by human cooperation in executing heavy-duty manual tasks, has two parts: Sequential Q-learning and Kalman Filtering based Reward Estimation, which are described next.

4.2.1 Sequential Q-learning

The sequential Q-learning algorithm may be summarized as follows:

- *Assume that there are n robots,* $R_1, R_2, \ldots, R_n$, *which are arranged in a special sequence. The subscripts represent their positions in this sequence.*
- $\Lambda_1, \Lambda_2, \ldots, \Lambda_n$ *are the corresponding action sets available for the robots. In particular, the robot* R_i *has* m_i *actions available for execution as given by* $R_i : \Lambda_i = (a_1^i, a_2^i, \ldots, a_{m_i}^i)$
- $Q_1(s,a), Q_2(s,a), \ldots, Q_n(s,a)$ *are the corresponding Q tables for the robots. All entries in the Q tables are initialized to zero.*
- *Initialize* τ *to 0.99.*
- Ψ *is a set including all actions selected by the robots thus far, and* ϕ *represents the empty set.*
- *Do repeatedly the following:*
  - *Initialize* $\Psi = \phi$
  - *Observe the current world state s*
  - *For (i=1 to n)*
    1) *Generate the currently available action set* $\Delta_i = (\Lambda_i - (\Lambda_i \bigcap \Psi))$
    2) *The robot* R_i *selects the action* $a_j^i \in \Delta_i$ *with probability*

$$P(a_j^i) = \frac{e^{Q_i(s,a_j^i)}}{\sum_{r=1}^{k} e^{Q_i(s,a_r^i)}}, \tag{4.1}$$

where $a_r^i \in \Delta_i (r = 1,2,\cdots,k)$ *and k is the size of the set* Δ_i

3) *Add action* a_j^i *to the set* Ψ

- *End For*

- *For each robot* $R_i (i = 1,2,\ldots,n)$, *execute the corresponding selected action* a_j^i
- *Receive an immediate reward r*
- *Observe the new state* s'
- *For each robot* $R_i (i = 1,2,\ldots,n)$, *update its table entry for* $Q_i(s,a_j^1,a_j^2,\ldots,a_j^i)$ *as follows:*

$$Q_i(s,a_j^1,a_j^2,\ldots,a_j^i) = (1-\varepsilon)Q_i(s,a_j^1,a_j^2,\ldots,a_j^i) + \varepsilon(r + \mu \max_{a^1,a^2,\ldots,a^i} Q_i[s',a^1,a^2,\ldots,a^i]) \quad (4.2)$$

where $0 < \varepsilon < 1$ *is the learning rate and* $0 < \mu < 1$ *is the discount rate.*

- $s \leftarrow s'$, $\tau \leftarrow \tau * 0.999$

The basic idea of the Sequential Q-learning algorithm comes from a strategy that is typically employed by humans when they cooperatively push a large and heavy object to a goal location. In transporting the object, the group members typically do not select their pushing locations and forces concurrently. Instead, one of them will select his pushing location and apply a force first. Then, by observing the first person's action, the second person will select his pushing action (i.e., a cooperative strategy) accordingly. Next, the third person will determine his action by observing the actions of the first two people. In this manner, when all the people in the group have determined their actions, they will execute the overall pushing actions through simple synchronization. This cooperative strategy is known to work well in manual tasks. By taking inspiration from human cooperation, the same strategy is used here to develop the Sequential Q-learning algorithm for multi-robot transportation tasks.

In the Sequential Q-learning algorithm, in each step of decision making, the robots do not select their actions simultaneously. Instead, they select their actions one by one according to a pre-defined sequence. Before a robot selects its action, it observes the nature of actions that have been chosen by the robots who precede it in the sequence. By

not selecting exactly the same actions as those of the preceding robots and instead by taking a complementary cooperative action, this robot is able to successfully solve the behavior conflict problem and promote effective cooperation with its teammates.

The benefits of the Sequential Q-learning algorithm are obvious. Because each robot observes the actions of the robots preceding it in the sequence before it makes its own decision, the Sequential Q-learning algorithm is likely to avoid conflicts and promote complementary actions, resulting in more effective cooperation and faster convergence than a single-agent Q-learning algorithm in a multi-robot cooperative task. Furthermore, because a robot does not need to observe the actions of all its teammates, the Sequential Q-learning algorithm results in a significantly smaller learning space (or Q table) than that for the team Q-learning algorithm. In addition, by selecting their actions according to a particular pre-defined sequence, the robots avoid selecting the same action which would result in behavior conflicts and cooperation failure in multi-robot cooperative transportation tasks.

4.2.2 Kalman Filtering Based Reward Estimation

When the conventional Q-learning algorithms (single-agent Q-learning or team Q-learning) are employed in a multi-robot environment, there are several factors that confuse or even fail the learning agents. First, as argued before, a multi-robot environment is essentially dynamic. In this dynamic environment the learning agent finds it very difficult to assess all the rewards received and update its learning process effectively.

Second, usually a multi-robot environment is only partially observed. Due to limitations in the sensor performance, a robot or an agent sometimes cannot see the regions of the environment that are obstructed or far away from it. Consequently, it only observes a part of the world. The feature of partial observation makes it difficult for a learning agent to assess an action under a specific state. In particular, the agent may be confused when an action receives different rewards under the "same" world state.

Third, the credit assignment is rather difficult in multi-robot learning. For example, when multiple robots cooperatively push a box to a goal location, how to split the observed global reward among the robots (e.g., how to assign a lazy or disruptive robot a negative reward and a hard-working robot a positive value) is an important issue. In the

single-agent Q-learning or team Q-learning algorithm, the global reward is directly assigned to each robot. It means that each robot receives the same reward regardless of whether it makes a positive contribution to the collective task. Due to this global reward policy, the simulation results in Chapter 3 have shown that the learning agents can converge slowly or even become confused in a multi-robot cooperative task. It follows that updating the Q-values with a global reward is not proper in a dynamic and unknown environment. How to estimate the real reward of each robot from the global reward signal has become a key topic in multi-robot Q-learning.

The issue of Kalman filtering based reward estimation was first studied by Chang and Kaelbling (2003). In their paper, they proposed to model the effects of unobservable states on the global reward as a random white noise process, and employed a Kalman filter to remove the noise and continuously extract the real reward from the global reward. While their simulation results validated this approach, they had made some crucial assumptions to simplify their model, which degraded its value to some degree. Among those assumptions, most importantly, they model the effect of unobservable states on the global reward as a zero-mean Gaussian noise process, which may be not correct in a real multi-robot system. Another limitation of their approach is the necessity to guess a variance value σ_w of the random noise process before the algorithm is executed. Therefore, some improvements are needed before the approach can be applied to a real multi-robot system.

In this section, a modified approach to estimate real rewards from global rewards, which is based on the original approach given in (Chang and Kaelbling, 2003), is proposed. A more general noise process is assumed and a method to estimate and update the parameters of the noise process is included in the algorithm to improve its performance. In addition, the approach of Kalman fitering based reward estimation is integrated into the sequential Q-learning algorithm presented in the previous section, to promote cooperation among robots and speed up the convergence process.

4.2.2.1 The system model

In the approach of Kalman filtering based reward estimation, the global reward received by a learning agent is thought to be the sum of its real reward and a random noise signal caused by changes in the environment and unobservable world states. In particular,

if the agent is in the world state i at time t and it receives a global reward g_t, then it can be expressed as

$$g_t = r(i)_t + b_t \tag{4.3}$$

where $r(i)_t$ is the real reward the agent receives in state i at time t, and b_t is an additive noise process which models the effect of the unobservable states on the global reward and evolves according to

$$b_{t+1} = b_t + z_t, \quad z_t \sim N(\mu, \sigma_w^2) \tag{4.4}$$

Here z_t is a Gaussian random variable with mean μ and variance σ_w^2. Based on these assumptions, the system model may be presented as

$$\begin{cases} x_t = Ax_{t-1} + w_t, & w_t \sim N(\Delta, \Sigma_1) \\ g_t = Cx_t + v_t, & v_t \sim N(0, \Sigma_2) \end{cases} \tag{4.5}$$

where, $x_t = \begin{pmatrix} r(1)_t \\ r(2)_t \\ \vdots \\ r(|s|)_t \\ b_t \end{pmatrix}_{(|s|+1)\times 1}$ is the state vector, $|s|$ is the total number of world states, w_t is the system noise having a Guassian distribution with mean $\Delta = \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \mu \end{Bmatrix}_{(|s|+1)\times 1}$ and covariance matrix $\Sigma_1 = \begin{pmatrix} 0 & \cdots & 0 & 0 \\ \vdots & \ddots & & \vdots \\ 0 & & 0 & 0 \\ 0 & \cdots & 0 & \sigma_w^2 \end{pmatrix}_{(|s|+1)\times(|s|+1)}$, and v_t is the observation error which is a zero-mean Gaussian white noise.

In addition, Σ_2 is set to zero because no observation error is assumed. The system matrix is $A = I$, and the observation matrix is $C = \begin{pmatrix} 0 & \cdots & 0 & 1_i & 0 & \cdots & 0 & 1 \end{pmatrix}_{1\times(|s|+1)}$ where the 1_i occurs in the i^{th} position when state i is observed.

4.2.2.2 Kalman filtering algorithm

Kalman filter is a powerful tool to estimate states of a linear system with Guassian noise. Here, the Kalman filtering algorithm is employed to dynamically estimate the real rewards and process noise through observing global rewards received by the learning agent. Then, the estimated real reward $r(i)_t$ in state i at time t instead of the global reward g_t will be used to update the Q-table. The standard Kalman filtering algorithm is based on a system model with zero-mean Gaussian white noise. In order to employ the standard Kalman filtering algorithm, the system model in equation (4.5) is transformed to

$$\begin{cases} x_t = Ax_{t-1} + Bu + \varepsilon_t, & \varepsilon_t \sim N(0,\Sigma_1) \\ g_t = Cx_t + v_t, & v_t \sim N(0,\Sigma_2) \end{cases} \tag{4.6}$$

where, $B = \begin{pmatrix} 0 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & 1 \end{pmatrix}_{(|s|+1)\times(|s|+1)}$, $u = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \mu \end{pmatrix}_{(|s|+1)\times 1}$, ε_t is a zero-mean Gaussian white noise signal with covariance matrix Σ_1, and $\Sigma_2 = 0$.

Based on the model given by (4.6), the Kalman filtering algorithm is presented below.

- Initialize $x_0 = (0,\cdots,0)^T$, the covariance matrix $P_0 = I$, $u = \begin{pmatrix} 0 & \cdots & 0 & 0 \end{pmatrix}^T$, $\sigma_w^{\ 2} = 0.1$, and $t = 1$.
- While (true)
  - From current state i, select an action a with Q-learning and execute it, observe a new state k, and receive a global reward g_t.
  - Update the estimate $\hat{x}_t$ and its covariance matrix $\hat{P}_t$ according to

    $\hat{x}_t = Ax_{t-1} + Bu$

    $\hat{P}_t = AP_{t-1}A^T + \Sigma_1$
  - These priori estimates are updated using the current observation g_t.

    $C_t = (0,\cdots,1_i,0,\cdots,0,1)$, whose i^{th} element is 1.

$$K_t = \hat{P}_t C^T_t (C_t \hat{P}_t C^T_t)^{-1}$$

$$x_t = \hat{x}_t + K_t(g_t - C_t \hat{x}_t)$$

$$P_t = (I - K_t C_t)\hat{P}_t$$

- Replace the global reward r in equation (4.2) with $x_t(i)$ to update the Q-table.
- Re-estimate the mean μ and variance σ_w^2 of the noise process with the history of b_t (i.e., $x_{t-width}(|s|+1), \cdots, x_t(|s|+1)$), and update u and Σ_1.
- $t \leftarrow t+1, \quad i \leftarrow k$

- End While

4.2.2.3 Online parameter estimation of the noise process

In the approach proposed by Chang and Kaelbling (2003), a value of σ_w^2, the covariance of the process noise, has to be guessed before the Kalman filtering algorithm is executed. However, in a real multi-robot task, it is usually very difficult to guess this covariance value. In this section, an online estimation method is presented to estimate the covariance σ_w^2 with the history of b_t (i.e., $x_t(|s|+1)$). Moreover, the mean μ is estimated online because it is not assumed to be zero in the system model presented in equation (4.6). The estimation method is given below.

- Initialize $\mu_0 = 0$, σ_{w0}^2=0.1, and $t = 0$.
- Execute the Kalman filtering algorithm for n ($n > 200$) iterations with constant μ_0 and σ_{w0}^2, and record the historical values of $x_{t+1}(|s|+1), x_{t+2}(|s|+1), \cdots, x_{t+n}(|s|+1)$.
- While (true)
  - Estimate the mean and covariance as follows:

$$\mu_t = \frac{1}{n-1}\sum_{i=2}^{n}(x_{t+i}(|s|+1) - x_{t+i-1}(|s|+1))$$

$$\sigma_{wt}^2 = \frac{1}{n-1}\sum_{i=2}^{n}(x_{t+i}(|s|+1) - x_{t+i-1}(|s|+1) - \mu_t)^2$$

  - Execute the Kalman filtering algorithm with μ_t and σ_{wt}^2, and record the value of $x_{t+n+1}(|s|+1)$.

- $t \leftarrow t+1$

- End While

4.3 Simulation Results

In order to validate the SQKF algorithm developed in this chapter, a multi-robot box pushing simulation system is used as shown in Figure 4.1.

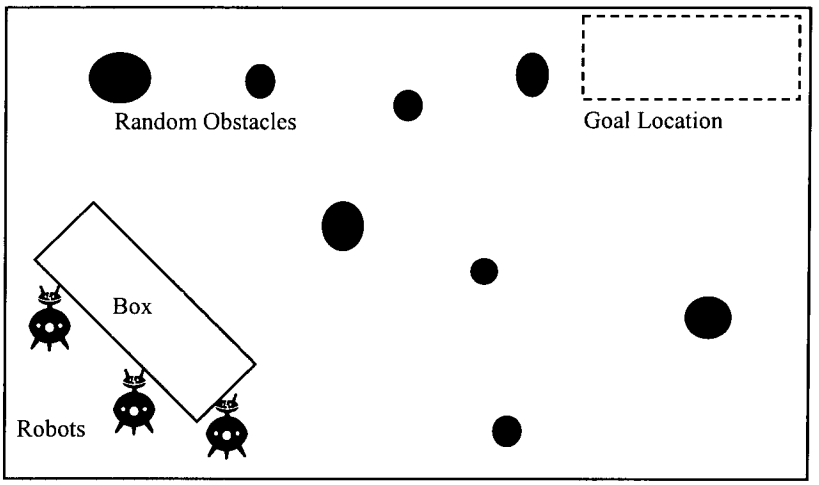


Figure 4.1: The simulated multi-robot box-pushing system.

Here, three autonomous robots attempt to cooperatively push a box to a goal location and orientation in an unknown and dynamic environment. The box is sufficiently big and heavy so that no single robot can complete the task alone. In addition, each robot only possesses local sensing capability and the obstacles may occur and disappear randomly in the path, so the robots are unable to plan the trajectory in advance. In such an unknown and dynamic environment in continuous space, a static planning approach will fail.

A system is developed using the Java language to simulate the multi-robot box-pushing system equipped with the proposed SQKF algorithm presented in the previous sections. The dimensions of the environment are 950×700 units and the dimensions of the box are 120×80 units. There are three robots, denoted by circles with a radius of 2 in the subsequent figures, which push the box to the goal location. The Java RMI (Remote Method Invocation) is used to implement a fully distributed multi-robot simulation system.

The definitions of the environmental states, robot actions and reward functions of the Q-learning algorithm are given in Chapter 3. Without any prior knowledge, the three mobile robots employ the SQKF learning algorithm to find good cooperation strategies so that the box is pushed to the goal location as quickly as possible. A successful example of box-pushing is shown in Figure 4.2.

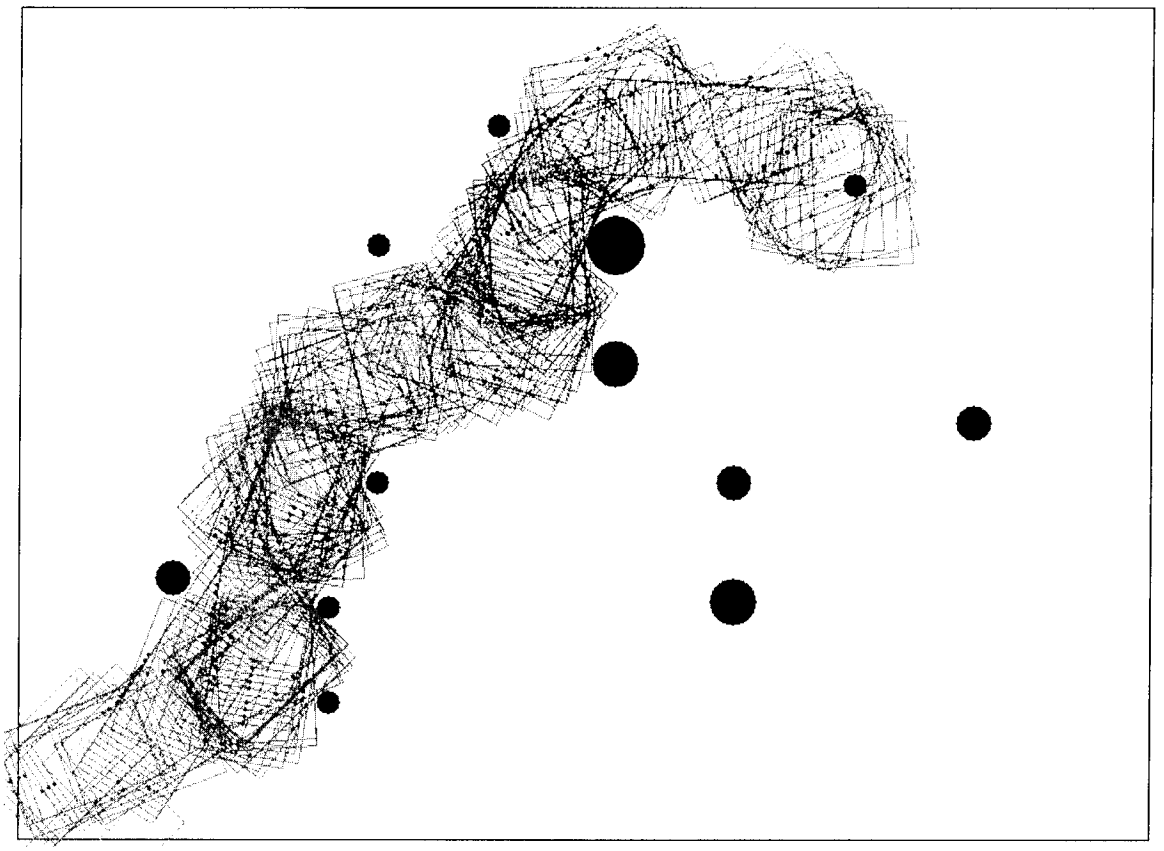


Figure 4.2: A successful example of multi-robot box pushing with the new SQKF algorithm.

Figure 4.3 shows the effect of training on the number of steps per round of box-pushing (The robots pushing the box from the start position to the goal position is counted as a round).

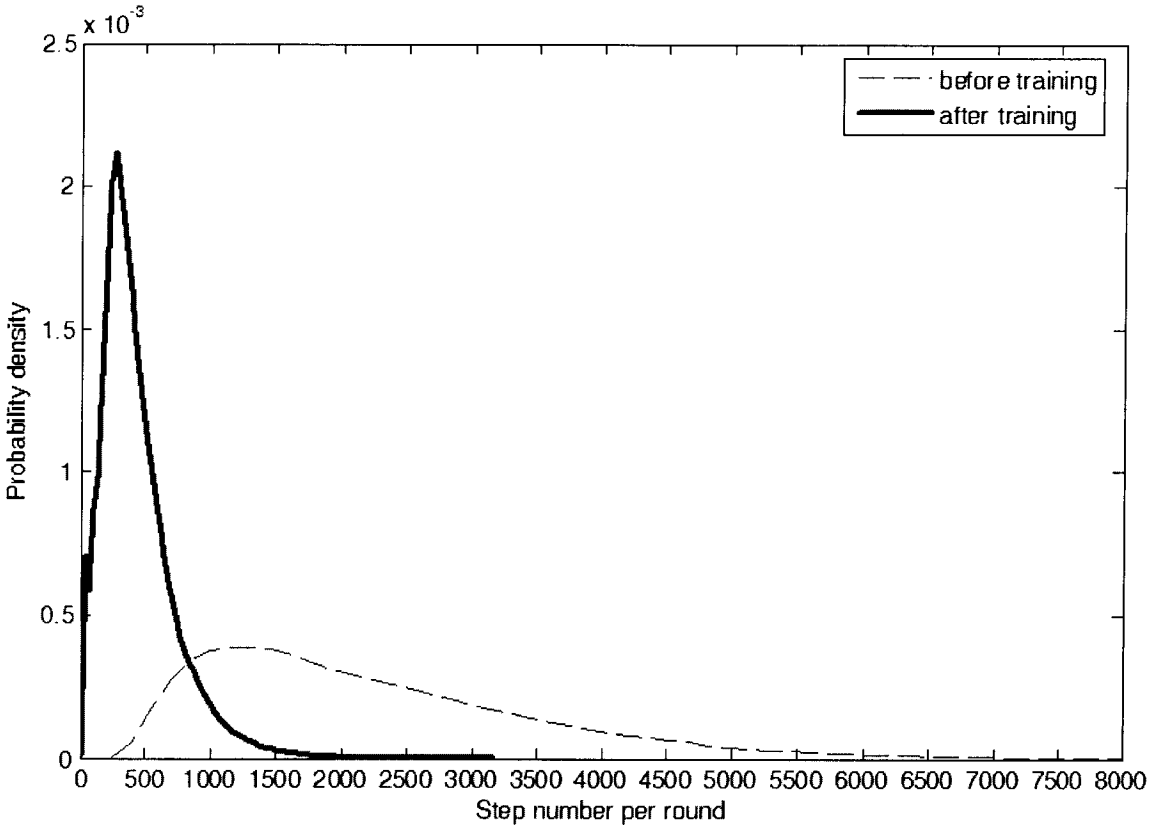


Figure 4.3: Probability density estimate of the number of steps per round before and after training.

In Figure 4.3, 500 rounds of box pushing are first executed, the number of steps per round is recorded, and its probability density is estimated based on the 500 samples. When each round of box pushing is started, the Q-table entries of each robot is filled with zeros which means the robots do not have any prior knowledge of pushing the box. Then, the robots are trained to improve their Q tables by pushing the box for 10,000 rounds continuously. After training, another 500 rounds of box pushing are executed, the number of steps per round is recorded, and its probability density is estimated. Similarly, every time a round of box-pushing is started, the Q-table of each robot is reloaded with the Q-table learned in the training stage. Figure 4.3 shows that the robots spent around 1,500 steps to push the box from the starting position to the goal location before the training stage, while they only spent approximately 300 steps to complete the same task after training, which demonstrates the effectiveness of learning. In addition, the variation of the number of steps per round after training is much smaller than that before training.

Therefore, it is clear that the robots have learned good cooperative policies and improved their performance significantly in the training stage.

Besides the number of steps per round, the average global reward per round also increases after training. Figure 4.4 shows the effect of training on the global reward.

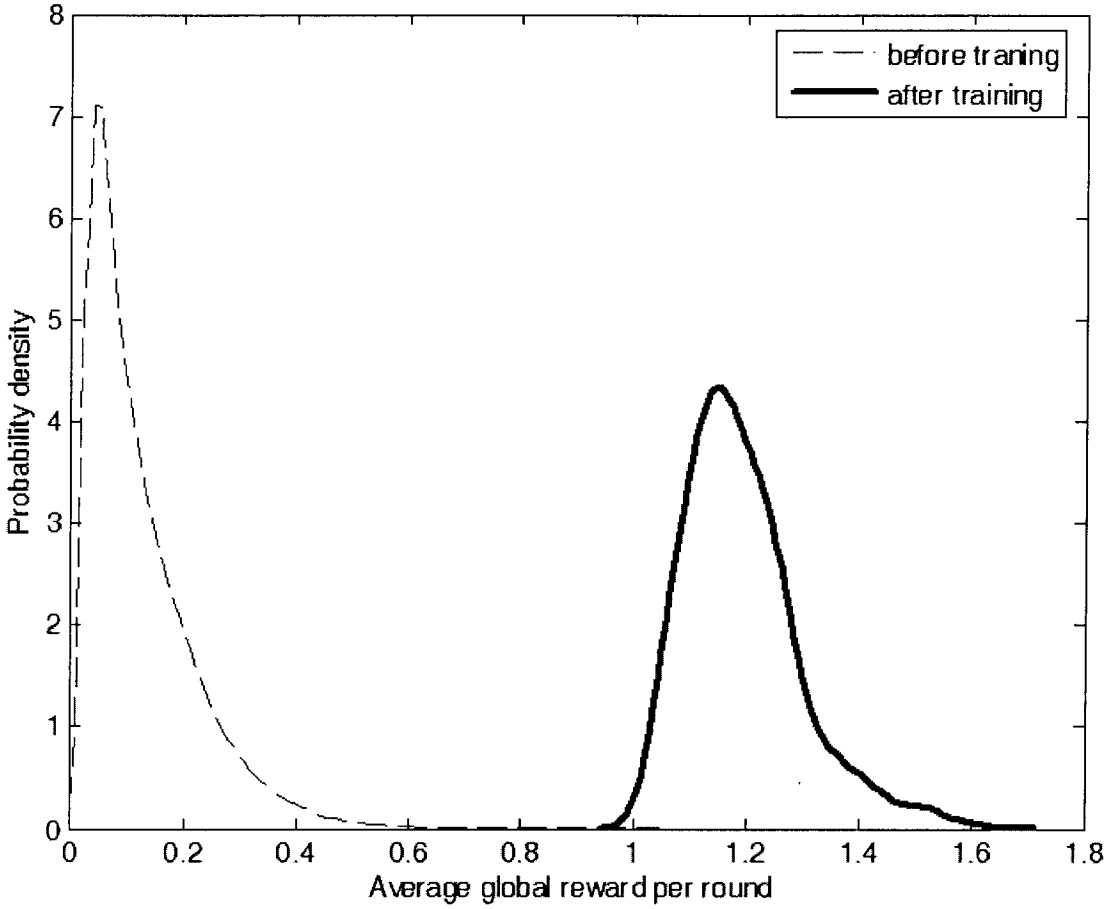


Figure 4.4: Probability density estimate of the average global reward per round before and after training.

The average global reward per round is obtained by dividing the sum of global rewards received by the number of steps in a round of box-pushing. Figure 4.4 shows the probability density estimates of the average global reward per round before and after training. Both probability density curves are obtained by analyzing 500 samples of data. From Figure 4.4, it is clear that the average global reward is approximately 0.05 before training while it becomes 1.15 after training. Because a bigger average global reward indicates that the robots employed a better cooperation strategy in this round of box-

pushing, training/learning is effectively used in the new SQKF algorithm to improve its performance and help the robots learn good cooperation policies in a multi-robot box-pushing task.

Figure 4.5 shows the history of number of steps per round in 1,000 rounds of continuous box-pushing.

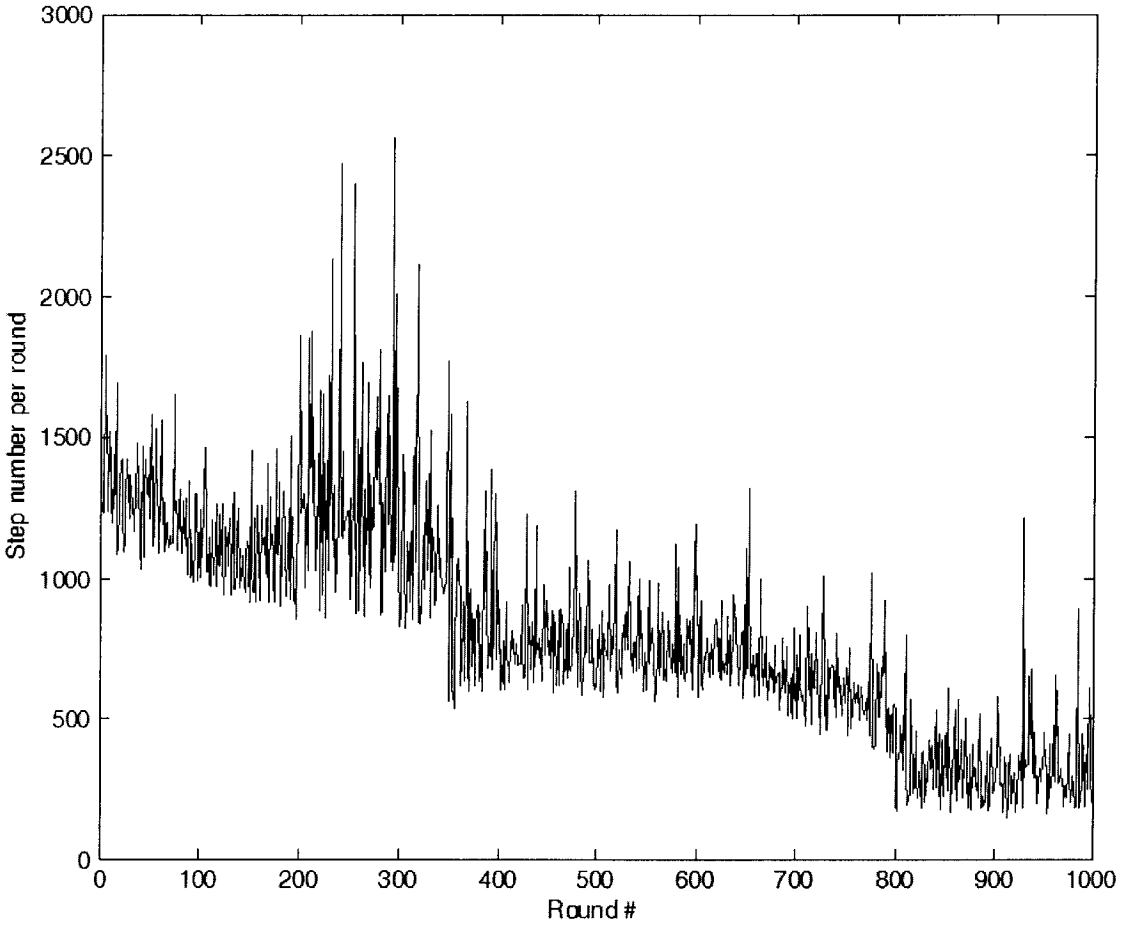


Figure 4.5: The number of steps per round in 1,000 rounds of continuous box-pushing.

Here, because in the beginning the robots do not have any prior knowledge of pushing the box, they take approximately 1,300 steps to complete the task in the first 20 rounds (i.e., cooperatively pushing the box from the start position to the goal location). After the robots explore most world states and learn good cooperation strategies through improving their Q-tables with the experience obtained in the previous box-pushing operations, the number of steps per round is decreased subsequently. From the 800^{th} round, the robots

take only approximately 400 steps to complete the same task. It means that the robots have learned correct strategies to cooperatively push the box in a faster manner and more successfully avoid obstacles in the environment.

The above observation is also validated by Figure 4.6 which shows the history of the average global reward per round in 1,000 rounds of continuous box-pushing.

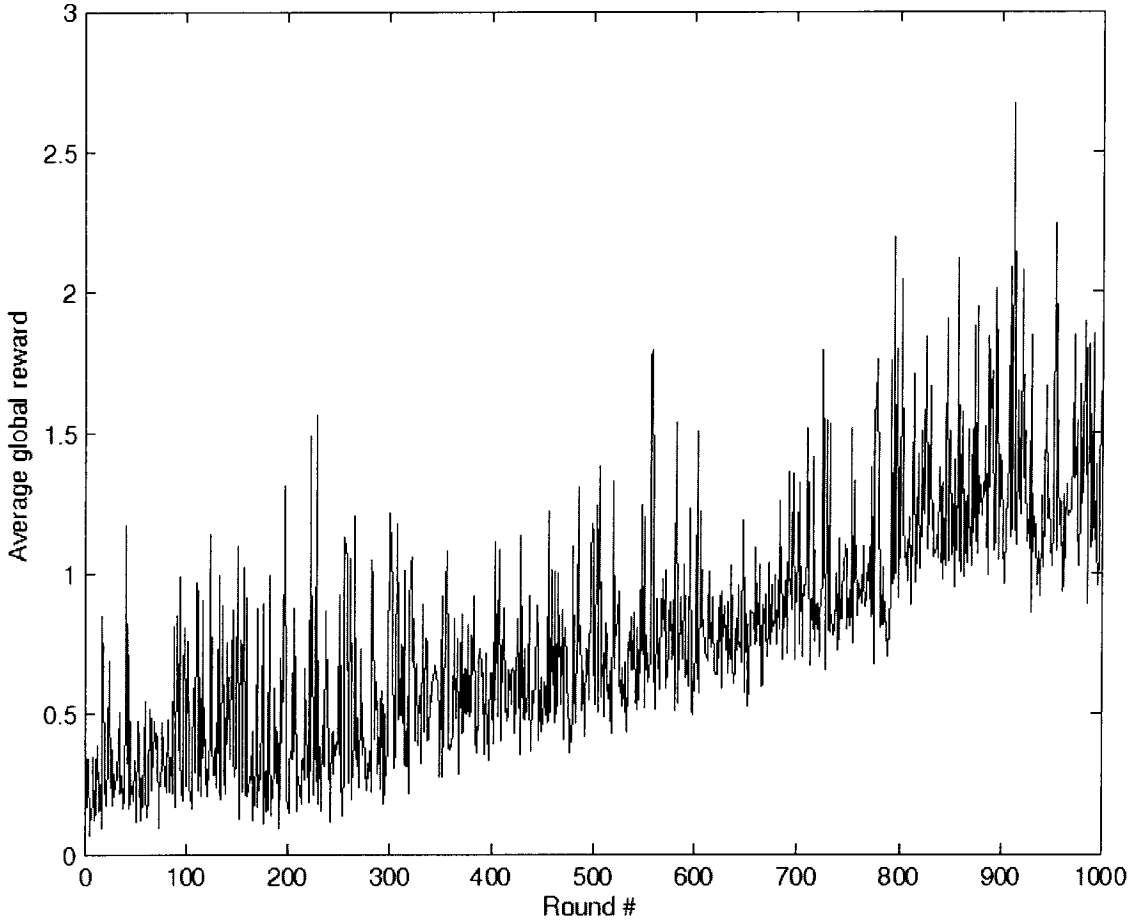


Figure 4.6: The average global reward per round in 1,000 rounds of continuous box-pushing.

From Figure 4.6, it is clear that the average global reward per round is increased when the robots obtained more experience of box-pushing. It increases from approximately 0.3 to 1.4 in 1,000 rounds of continuous box-pushing. Because a bigger average global reward indicates that the robots have employed good cooperation strategies to move the box with low probability of colliding with obstacles, Figure 4.6 confirms that the new SQKF

algorithm is effective in helping the robots to improve their performance in carrying out the cooperative task.

The global rewards within one round are observed in order to understand the effect of training on the performance of the robots. Figure 4.7 shows the global rewards received by the robots in a round of box-pushing before training, while Figure 4.8 shows the probability density estimate of the global rewards.

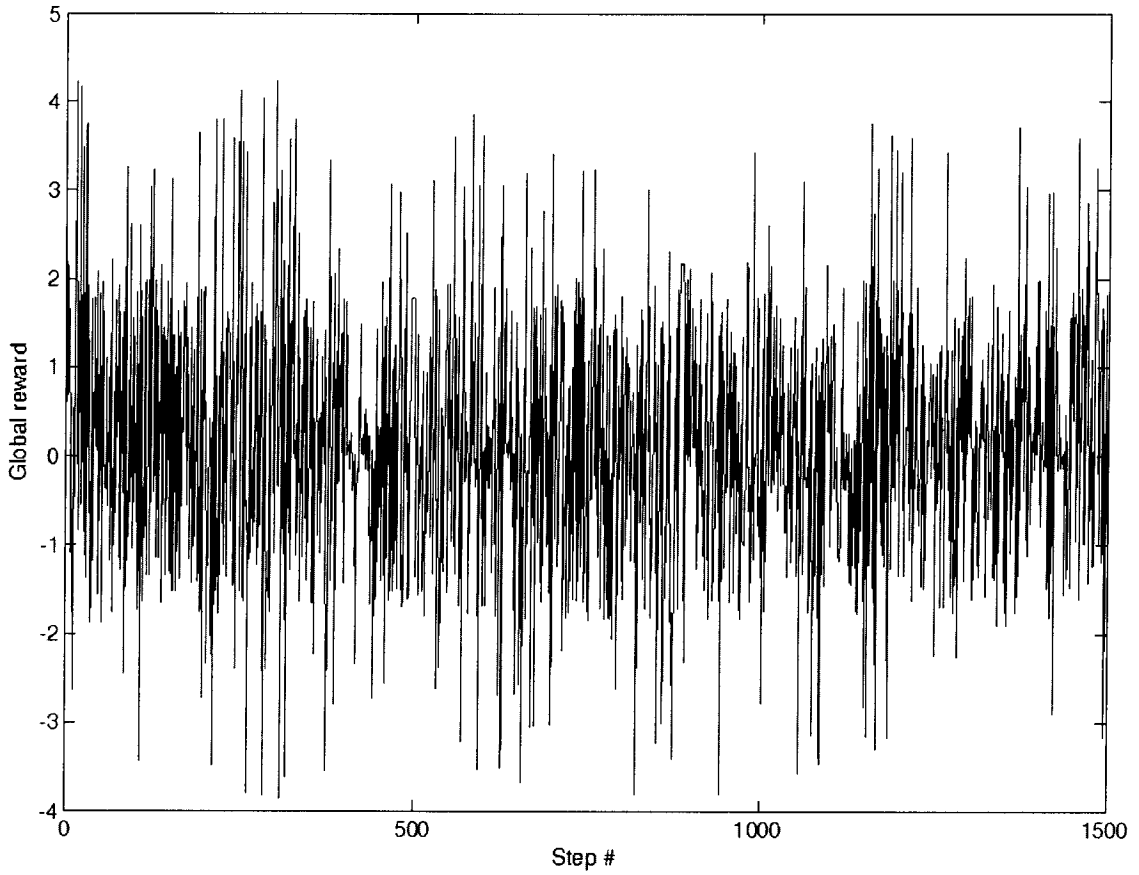


Figure 4.7: The global rewards received by the robots in a round of box-pushing before training.

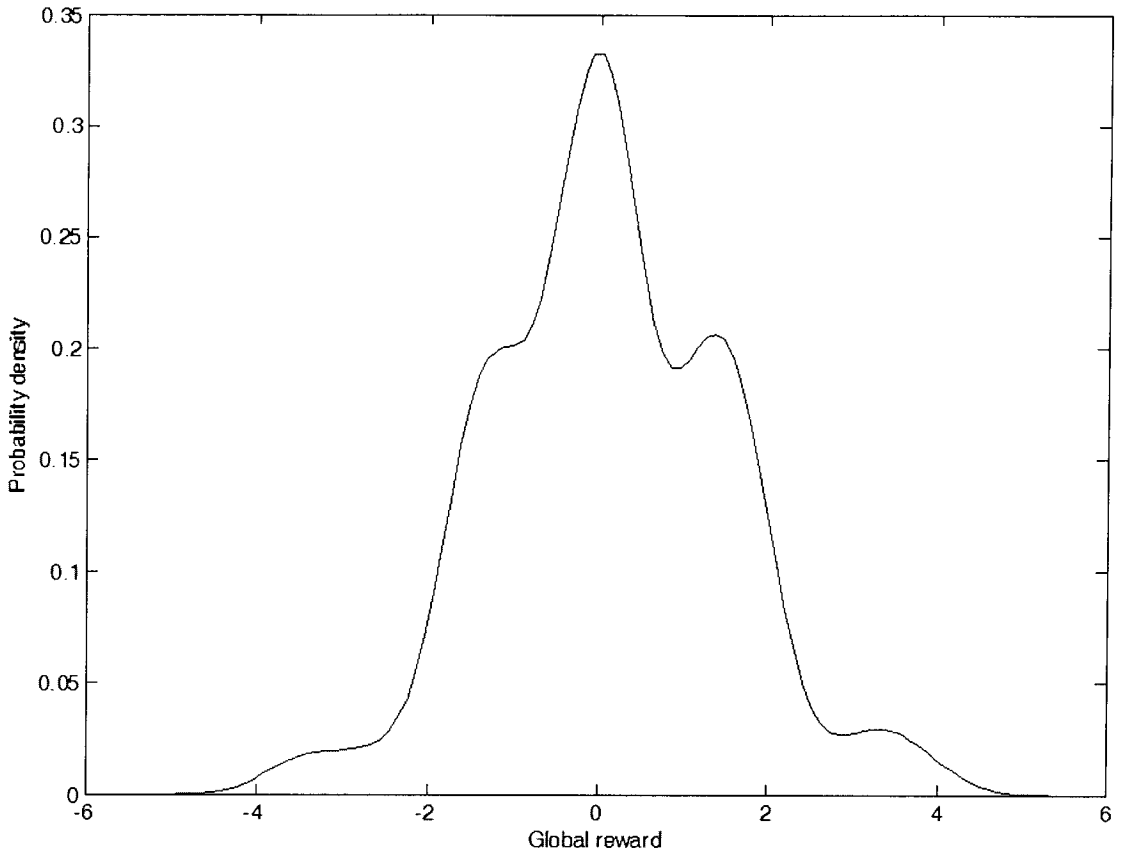


Figure 4.8: The probability density estimate of the global rewards in Figure 4.7.

Figure 4.7 shows that the robots take 1,500 steps to cooperatively push the box to the goal location while avoiding the obstacles in the environment. The global rewards are very random in the course of transportation. Figure 4.8 describes the statistical nature of the global rewards received when the box is moved from the start position to the goal location. The average value of the global rewards is approximately 0 and the rewards fluctuate from -4 to +4. Figure 4.8 and Figure 4.7 indicate that the robots receive both positive and negative global rewards in a round of box-pushing before training. It means that at this stage the robots are still exploring the world states and collecting experience to improve their Q-tables.

Figure 4.9 show the global rewards received in a round of box-pushing after the training stage, and the corresponding statistical result is shown in Figure 4.10.

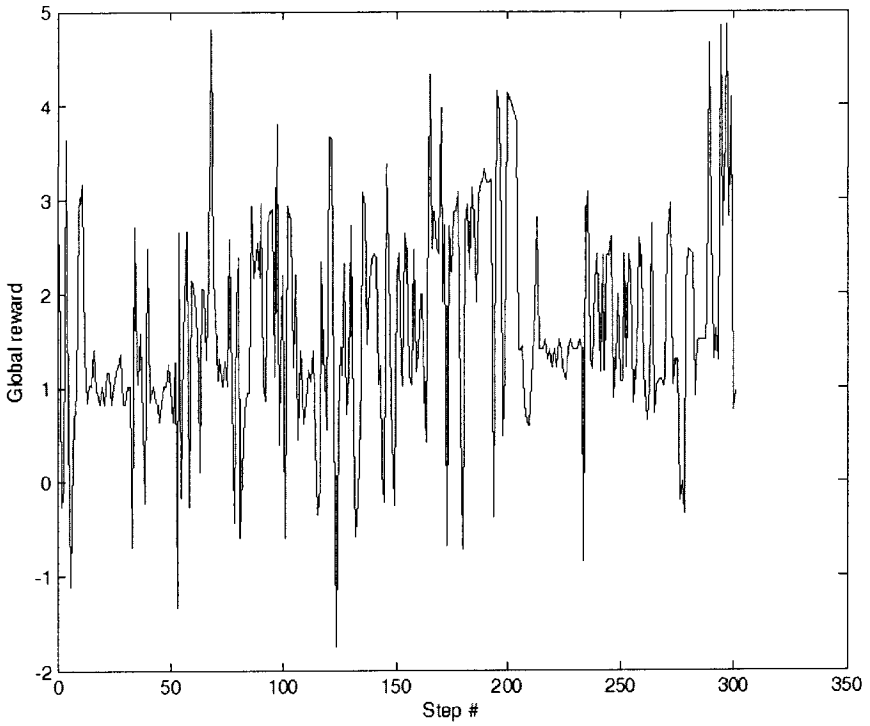


Figure 4.9: The global rewards received by the robots in a round of box-pushing after training.

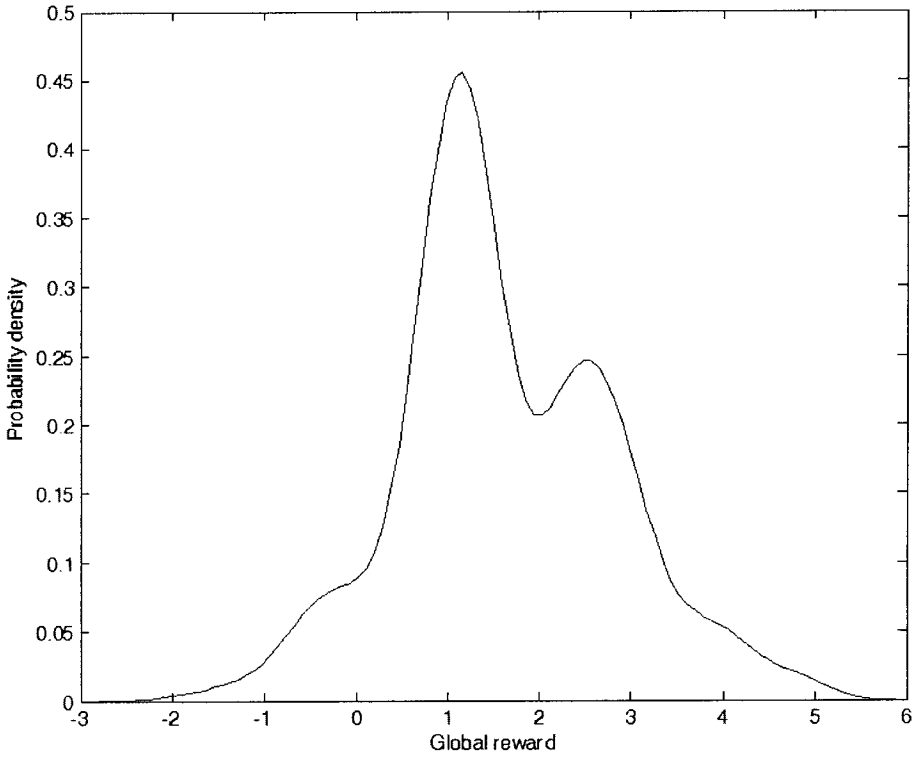


Figure 4.10:.The probability density estimate of the global rewards in Figure 4.9.

From Figure 4.9 and Figure 4.10 it is clear that the global rewards are increased significantly after training, thereby decreasing the number of steps to 300, which means the robots have leaned good cooperation strategies in the training stage and improved their performance significantly. In particular, now the average value of the global rewards is approximately 1.5. By comparing Figure 4.7 and Figure 4.8 with Figure 4.9 and Figure 4.10, it is convincing that the new SQKF algorithm can improve the performance of the robots to cooperatively transport the box through training.

Next the performance of the new SQKF algorithm is compared with that of a conventional single-agent Q-learning algorithm for the same multi-robot task. The comparison results are presented in Figure 4.11 and Figure 4.12.

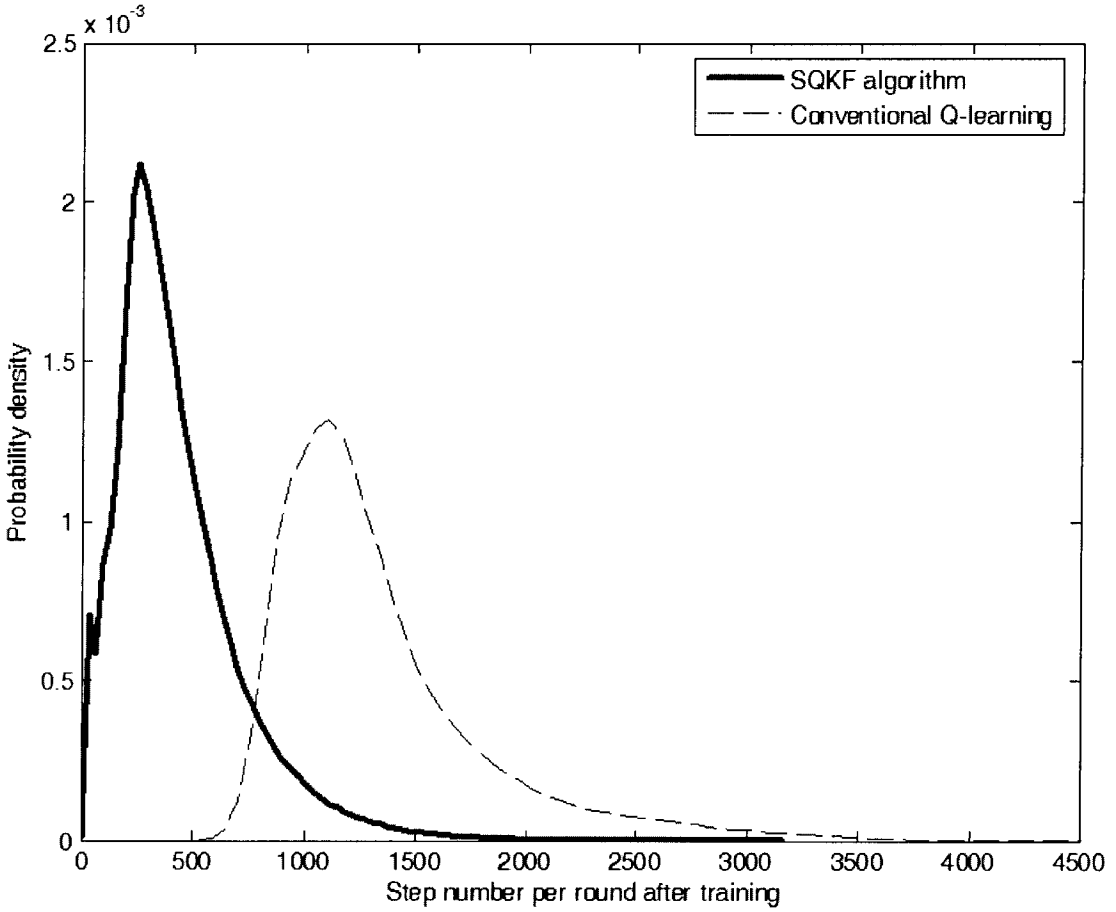


Figure 4.11: Probability density estimate of number of steps per round in 500 rounds of box-pushing.

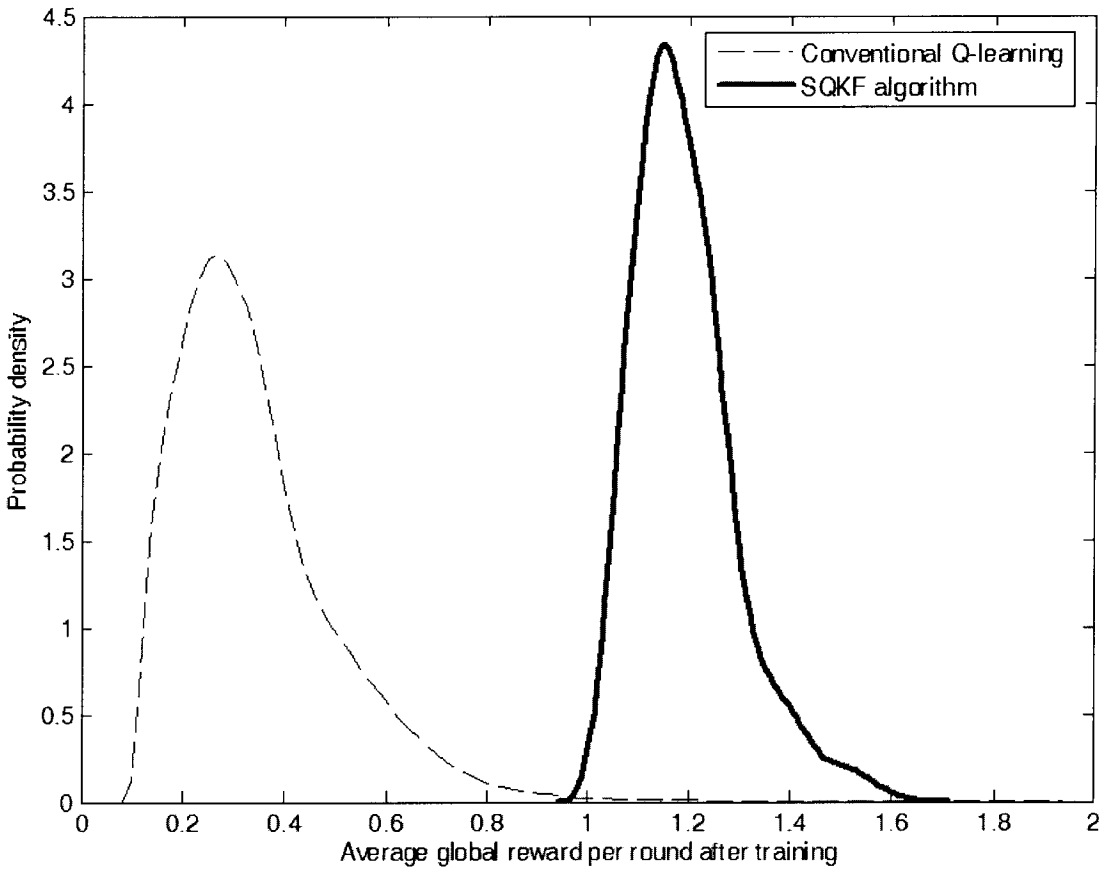


Figure 4.12: Probability density estimate of average global reward per round in 500 rounds of box-pushing.

For both learning algorithms, the robots are trained for 10,000 rounds of box-pushing first to improve their Q-tables. Then another 500 rounds of box-pushing are executed, and the number of steps and the average global reward per round are recorded for statistical analysis. From Figure 4.11 and Figure 4.12, it is clear that the new SQKF algorithm has better performance than the conventional Q-learning algorithm in the multi-robot box-pushing project. In particular, the new SQKF algorithm takes an average of 350 steps to complete the task while the conventional Q-learning algorithm takes almost 1,200 steps to complete the same task. Similarly, the new algorithm receives a higher average of global rewards than the conventional Q-learning algorithm.

4.4 Summary

Most multi-robot operating environments, such as planet surfaces, are dynamic and unstructured where the terrain features and the obstacle distribution change with time. Furthermore, even if the external physical environment is stationary, the overall system structure is still dynamic from the viewpoint of a single robot because other robots may take actions thereby changing the environment of that particular robot continuously. The environmental dynamics makes multi-robot decision-making quite complex, and the traditional task planning becomes ineffective because a planned optimal policy can become inappropriate a few minutes later due to changes in the environment.

By observing human capabilities of dealing with a dynamic environment, it is easy to conclude that a human employs not only planning or reactive (behavior-based) techniques but also learning techniques to successfully complete a task in such an environment. Through learning, a human learns new world states, finds optimal actions under these states, and improves his planning and reactive techniques continuously. Learning enables him to deal with unexpected world states, decrease uncertainties in the environment, and make his decision-making capability more robust in a dynamic environment.

Therefore, machine learning has become important in multi-robot applications. Among the existing machine learning approaches, Reinforcement Learning, especially Q-learning, is used most commonly in multi-robot systems due to its simplicity and good real-time performance. Chapter 3 addressed how to employ two Q-learning algorithms, the single-agent Q-learning and the team Q-learning, to find optimal cooperative strategies for a multi-robot box-pushing task. While they help improve the performance of the robots in a cooperative task, there are some serious disadvantages in both Q-learning algorithms. In particular, the single-agent Q-learning has a slow convergence speed when applied to a multi-robot task due to the presence of dynamic environment, and the learning space of the team Q-learning algorithm can become too extensive to be practical in a real multi-robot application. Consequently, it has become important to develop a new reinforcement learning algorithm that is suitable for multi-robot decision-making.

In this chapter, a modified Q-learning algorithm suitable for multi-robot decision making was developed. By taking inspiration from the execution of heavy-duty object moving tasks by a team of humans, through arranging the robots to learn in a sequential

manner and employing Kalman filtering to estimate real rewards and the environmental noise, the developed learning algorithm showed better performance than the conventional single-agent Q-learning algorithm in a simulated multi-robot box-pushing project.

Chapter 5

Computer Vision in Multi-robot Cooperative Control

5.1 Overview

Multi-robot systems have become a promising area in robotics research (Cao et al., 1997). In such a system, several autonomous robots cooperatively work to complete a common task. Due to the advantages of robustness, flexible configuration, potential high efficiency and low cost, and close similarity to human-society intelligence, multi-robot systems have attracted researchers in the robotics community.

In a multi-robot system, it is important for a robot to know the latest poses (positions/orientations) of other robots and potential obstacles in the environment so as to make rational decisions. There are many means to measure the pose of a robot or an obstacle; for example, using sonar or laser distance finders. However, most multi-robot systems employ digital cameras for this purpose. The main reasons are: (1) A digital image provides a rich source of information on multiple moving objects (robots and obstacles) simultaneously in the operating environment, (2) The vast progress in computer vision research in recent years has made it possible to build fast and accurate vision subsystems at low cost, (3) Use of cameras by robots to observe and understand their operating environment is as "natural" as the use of eyes by the humans to observe the world.

Some work has been done to monitor multiple moving objects in a dynamic environment by using computer vision. Stone and Veloso (1998) studied a multi-robot soccer system. In their work, they used a global camera to monitor the positions of the robots and objects in the game. Veeraraghavan et al. (2003) developed a computer vision algorithm to track the vehicle motion at a traffic intersection. A multilevel approach using a Kalman filter was presented by them for tracking the vehicles and pedestrians at an intersection. The approach combined low-level image-based blob tracking with high-level Kalman filtering for position and shape estimation. Maurin et al. (2005) presented a

vision-based system for monitoring crowded urban scenes. Their approach combined an effective detection scheme based on optical flow and background removal that could monitor many moving objects simultaneously. Kalman filtering integrated with statistical methods was used in their approach. Chen et al. (2005) presented a framework for spatiotemporal vehicle tracking using unsupervised learning-based segmentation and object tracking. In their work, an adaptive background learning and subtraction method was applied to two real-life traffic video sequences in order to obtain accurate spatiotemporal information on vehicle objects

Nevertheless, significant difficulties exist in employing the conventional computer vision algorithms in a multi-robot system. The first difficulty results from the real-time constraints in the specific system. In a multi-robot system, multiple mobile robots move quickly in an unpredicted manner, and the vision system needs to capture their positions and orientations within a very short time. Consequently, it is not practical to employ an effective yet time-consuming vision algorithm. The conventional algorithms, such as those used by others in the work outlined above, are too complex for meeting real-time constraints in a multi-robot system. The second difficulty arises when multiple mobile robots move in a large area that has different levels of illumination. Multi-robot systems usually work in large and unstructured environments with uneven lighting conditions. The robots usually move into and move out of sub-areas having different brightness levels. In order to track the robots effectively in such an environment, the vision system must be robust with respect to different illumination conditions. However, most of existing algorithms do not consider this problem.

A fast color-blob tracking algorithm has been developed (Wang and de Silva, 2006d) for the multi-robot projects which are carried out in the Industrial Automation Laboratory (de Silva, 2005; Karray and de Silva, 2004) of the University of British Columbia (UBC). The algorithm meets the real-time constraints in our multi-robot projects, and can effectively track the positions and orientations of the robots when they move quickly in an unstructured environment with uneven illumination. In the next section, the multi-robot transportation system in the laboratory is introduced. In Section 5.3, the developed computer vision algorithm is presented. In Section 5.4, a representative experimental result obtained through the developed approach is presented. Based on the proposed color-

blob tracking algorithm, a multi-robot route planning approach is designed and presented in Section 5.5. The conclusions are given in Section 5.6.

5.2 Multi-robot Transportation System

The objective of our multi-robot transportation project in the Industrial Automation Laboratory (IAL), is to develop a physical system where a group of intelligent robots work cooperatively to transport an object to a goal location and orientation in an unknown and unstructured dynamic environment. Obstacles may be present and even appear randomly during the transportation process. Robot control, sensing, multi-agent technologies, and machine learning are integrated into the developed physical platform to cope with the main challenges of the problem. A schematic representation of the initial version of the developed system is shown in Figure 1.1 and is repeated in Figure 5.1. The latest system in the laboratory uses state-of-the art mobile robots together with fixed-base articulated robots (one state-of-the art commercial robot and two multi-modular robot prototypes developed in our laboratory).

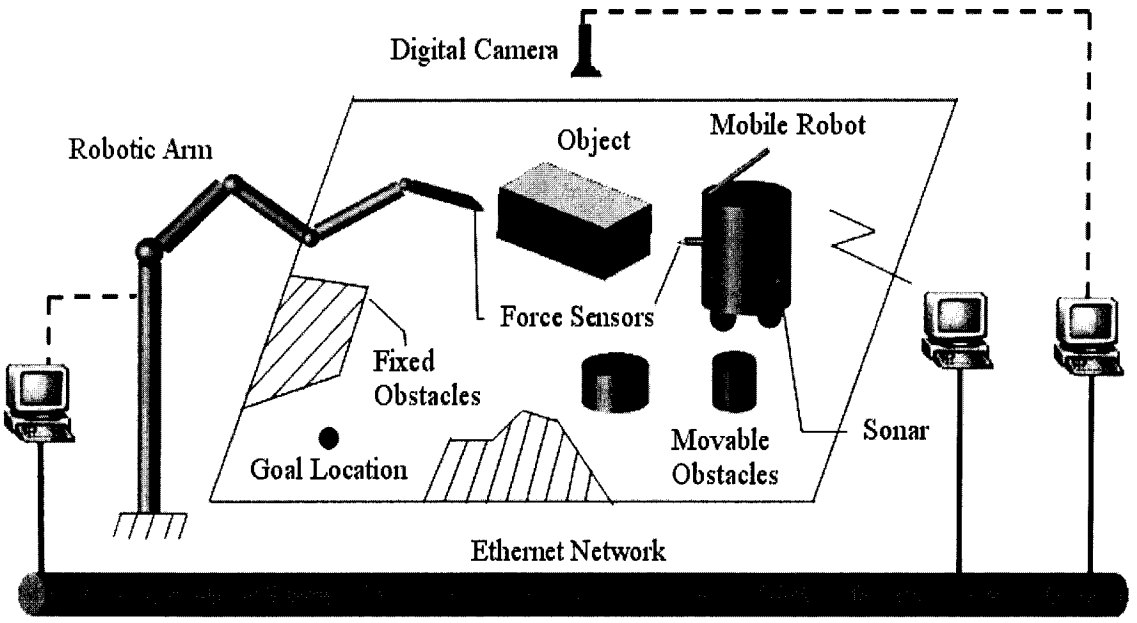


Figure 5.1: The developed physical multi-robot system.

In this first prototype there is one fixed robot and one mobile robot, which cooperatively transport an object to a goal location. During the course of the transportation,

the robots have to negotiate the cooperation strategy and decide on the optimal location and magnitude of the individual forces that would be applied by them so that the object would be transported quickly and effectively while avoiding any obstacles that might be present in the path. In some special cases, they also need to consider whether to move an obstacle out of the way rather than negotiating around it. Other considerations such as the level of energy utilization and damage mitigation for the transported object may have to be taken into account as well. A global camera is used to monitor and measure the current location and orientation of the object. The environment is dynamic and unpredictable with some movable obstacles which may appear randomly, and some fixed obstacles. The robots, the camera, and the sensors are separately linked to their host computers, which are connected through a local communication network, to implement complex controls and machine intelligence.

A robotic arm and a mobile robot are used in the system shown in Figure 5.1. The system provides an opportunity to observe the cooperation between two robots with different dynamics and obtain useful information on the system behavior. Moreover, with the ability of exact localization of the robotic arm and the capability of broad ground coverage of the mobile robot, it is possible to integrate different advantages of the two robots in a complementary manner, so as to improve the effectiveness of the overall system.

The vision subsystem is rather crucial in the current project. It is expected to effectively and accurately track the poses (positions and orientations) of the robots and the object in real-time. Several color-blobs with different colors are used to identify the poses of the robots and the object, thereby simplifying the programming requirements. Two difficulties arise in this vision subsystem. The first one is its speed of response, which is subjected to real-time constraints. In order to simultaneously find the current poses of multiple robots and the object, all the image processing work, such as image acquisition, pre-processing, searching the positions of the color-blobs, and coordinate transformation, must be completed within 500 ms so that the decision-making subsystem can effectively track and update the new world model and make correct decisions. The second difficulty is caused by the uneven lighting in the environment, which is unavoidable in a natural environment. In our projects, it is found that uneven lighting greatly degrades the

performance of the vision subsystem, and it represents a primary disturbance in detecting the positions of the color blobs. Because uneven illumination is inevitable in a large physical environment, it is imperative to develop a computer vision algorithm that is robust to uneven lighting conditions.

5.3 The Fast Color-blob Tracking Algorithm

In order to cope with the two main problems in our multi-robot project, as mentioned in the previous section, a fast color-blob tracking algorithm is developed. This algorithm is presented in Figure 5.2 and its steps are described in detail next.

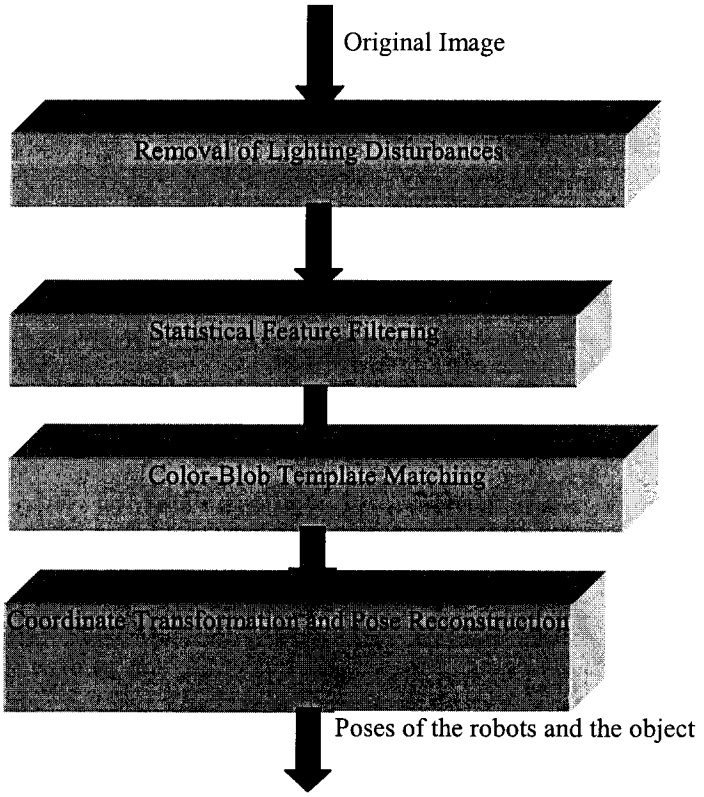


Figure 5.2: The color-blob tracking algorithm.

5.3.1 Removal of the Lighting Disturbances

As discussed before, uneven illumination in the operating environment greatly degrades the performance of the color-blob tracking algorithm. In the first step of the present algorithm, the disturbances caused by uneven illuminations are removed by transforming the original image from the RGB (Red, Blue, and Green) color space to the

HSI (Hue, Saturation, and Intensity) color space and removing its Saturation and Intensity components.

When the image is captured by the camera, it is usually represented in the RGB color space. Each color corresponds to a point in the RGB space; i.e., any color is composed of Red, Green, and Blue with different intensity levels. Although the RGB color space is a good tool to generate color, it is not particularly suitable for describing color. The HSI color space is another well-known color model, which is effectively employed in the digital image processing field (Gonzalez and Woods, 2002). Its most important advantage is that it corresponds closely with the way humans describe and interpret color. Another advantage is that it successfully decouples the color and the gray-level information in an image.

In this section, in order to remove the disturbances caused by uneven illumination in the environment, the original color image is transformed from the RGB color space to the HSI color space. Then the Saturation and the Intensity components in the image are deleted and only the Hue component is retained. Because the Hue component in the image represents the color attribute, the Saturation component represents the amount of white light that is added into a monochromatic wave of light, and the Intensity component represents brightness, by separating the Hue component from the Saturation and the Intensity components, the disturbances arising from uneven illumination in the environment are effectively removed.

Equations (5.1)-(5.3) indicate how to convert a color image from the RGB color space to the HSI color space (Gonzalez and Woods, 2002).

$$H = \begin{cases} \theta & \text{if } B \le G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{5.1}$$

with

$$\theta = \cos^{-1}\left\{ \frac{0.5\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{1/2}} \right\}$$

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)] \tag{5.2}$$

$$I = \frac{1}{3}(R+G+B) \tag{5.3}$$

5.3.2 Statistical Feature Filtering

In this step, a type of statistical feature filtering is employed to remove the colors which are not related to the color blobs of interest in the image so that the succeeding algorithm will be able to easily determine the positions of these color blobs.

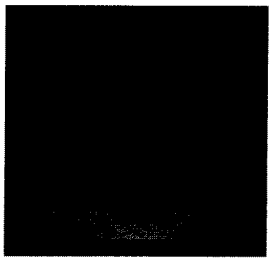


Figure 5.3: A sample image of a color blob.

A sample image of a color blob is shown in Figure 5.3. It is assumed that there are $n \times n$ pixels in this image and $h(i,i)$ represents the hue value of the pixel (i, j). The average hue value $\overline{h}$ and the standard deviation σ of this sample color blob can be calculated as follows:

$$\overline{h} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} h(i,j)}{n^2} \tag{5.4}$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left[h(i,j) - \overline{h}\right]^2}{n^2 - 1}} \tag{5.5}$$

Given k color blobs in the image, and their corresponding average hue values $\overline{h_1}, \cdots \overline{h_k}$ and standard deviations $\sigma_1, \cdots, \sigma_k$, the statistical filtering algorithm that is presented next is executed.

For each pixel in the original image

If *its hue value is within the set of*

$\{h \mid \overline{h_1} - 1.2\sigma_1 \le h \le \overline{h_1} + 1.2\sigma_1$ *or*

$\overline{h_2} - 1.2\sigma_2 \le h \le \overline{h_2} + 1.2\sigma_2$

or $\cdots \overline{h_i} - 1.2\sigma_i \le h \le \overline{h_i} + 1.2\sigma_i$

or $\cdots$ *or* $\overline{h_k} - 1.2\sigma_k \le h \le \overline{h_k} + 1.2\sigma_k)$

it will not be changed

Else

Its hue values will be replaced with the value of 0.

5.3.3 Color-blob Template Matching

After statistical feature filtering, apart from the regions of color blobs there still exist very small regions of pixels which are caused by unknown disturbances that arise when an image is captured by a camera. A color-blob template matching algorithm is developed in order to remove these small regions of pixels.

A color-blob template is a matrix of hue values of a given color blob. The corresponding $m \times m$ matrix is generated off line from the image of the color blob, and is used to search its position in the entire image of the environment which is captured by the global camera. Given a $n \times n$ hue matrix H of the environment image, and a $m \times m$ template matrix T, the following color-blob template matching algorithm is executed:

- *Initialize min-distance=100000, blob_pos=(1,1)*
- *For each element* $H(i, j)$ *in the matrix of* H,

  distance=

  $$\sqrt{\frac{\sum_{k=1}^{m}\sum_{l=1}^{m}(H(i+k-1, j+l-1)-T(k,l))^2}{m^2}}$$

  if (distance < min-distance) then

  - *min-distance=distance*
  - *update: blob_pos=(i,j)*
- *Output blob_pos*

5.3.4 Coordinate Transformation and Pose Reconstruction

Once the positions of all color blobs in the image are obtained, it is necessary to transform their coordinates in the mage to those in the real world. Because the positions are defined in a 2-D plane, this transformation is straightforward. Similarly, given all color blob positions in the real world, it is easy to reconstruct the poses of all robots and the

object in the environment. Usually, two blobs with different colors are needed to determine the position and orientation of a robot or an object. By checking the corresponding color blobs of each robot or object, the poses of all robots and the object can be reconstructed rather quickly.

5.4. Experimental Result

5.4.1 The Test-bed

A test bed is developed for multi-robot projects in the Industrial Automation Laboratory (IAL), as shown in Figure 5.4. It includes a 4-joint 2D robotic arm, an Amigo™ mobile robot, a JVC™ color CCD (charge-coupled device) camera, two Transducer Techniques™ force sensors, a rectangular box (object), an indexing table, a ServoToGo™ motion control board, a Matrox™ frame grabber board, a CISCO™ wireless LAN access point and a Pentium IV™ computer with 512M RAM. The robotic arm is particularly designed to move an object on a 2-D surface. Its four joints (two arm joints and two wrist joints) are equipped with optical encoders to accurately detect the joint motions. One force sensor is mounted in its end effector. The encoder and force data are acquired by the host computer via the motion control board.

The mobile robot is a two-wheel-driven intelligent machine with two 500-tick encoders, 8 sonar units, wireless LAN communication ability and a force sensor in its end effecter. The mobile robot is connected to the host computer with the wireless LAN via a CISCO™ wireless Ethernet access point equipment. It can wander within a radius of 50 meters without any communication interruption.

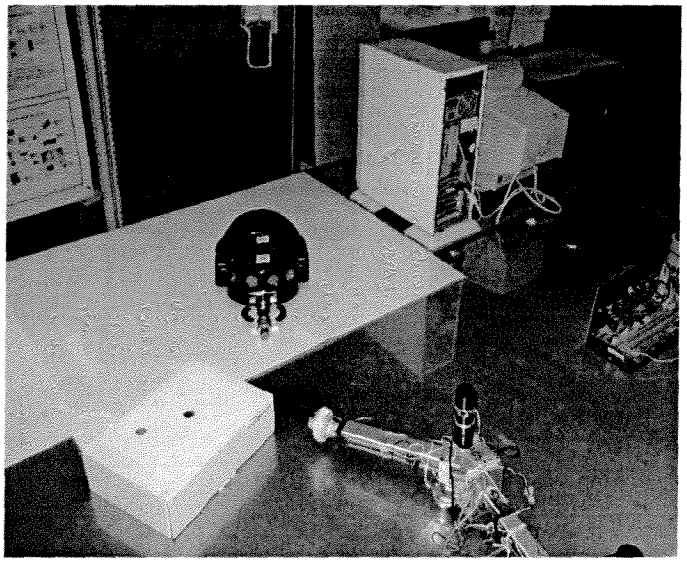


Figure 5.4: The experimental system in IAL at UBC.

A color CCD camera with a resolution of 768×492 pixels is used to monitor the positions and orientations of the box, the obstacles, and the mobile robot. The vision images are captured into the computer by a frame grabber board at a sampling rate of 30 MHz.

5.4.2 Experimental Result

In order to speed up the image processing task and reduce the computational load of the vision subsystem, the mobile robot and the box are marked with color blobs (a purple and a green blob on the mobile robot; an orange and a blue blob on the box), which represent their current poses, as shown in Figure 5.5. The poses of the mobile robot and the box are determined by quickly localizing the four color blobs from the image, followed by the necessary coordinate transformation operations.

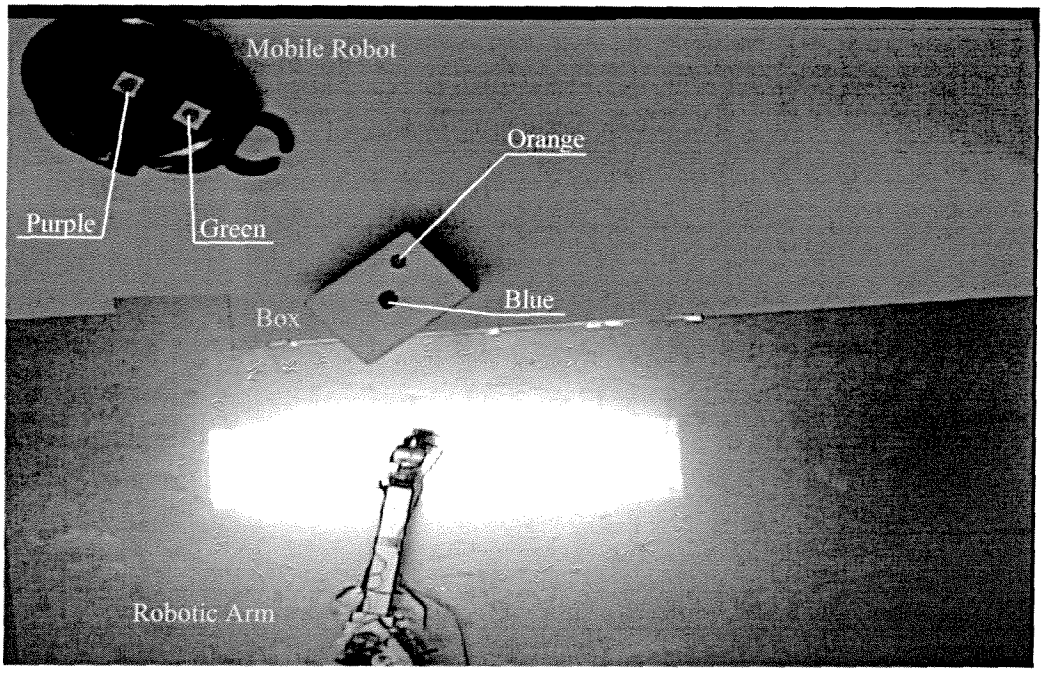


Figure 5.5: An original image captured by the CCD camera (Note the color blobs on the mobile robot and the box.)

The first step of the developed approach is to convert the original image from the RGB color space to the HSI color space so as to remove the disturbances caused by uneven lighting conditions. The result of this processing step is shown in Figure 5.6. By comparing Figure 5.5 with Figure 5.6, it is noted that the influence of uneven lighting is effectively removed and only the hue of the image is retained.

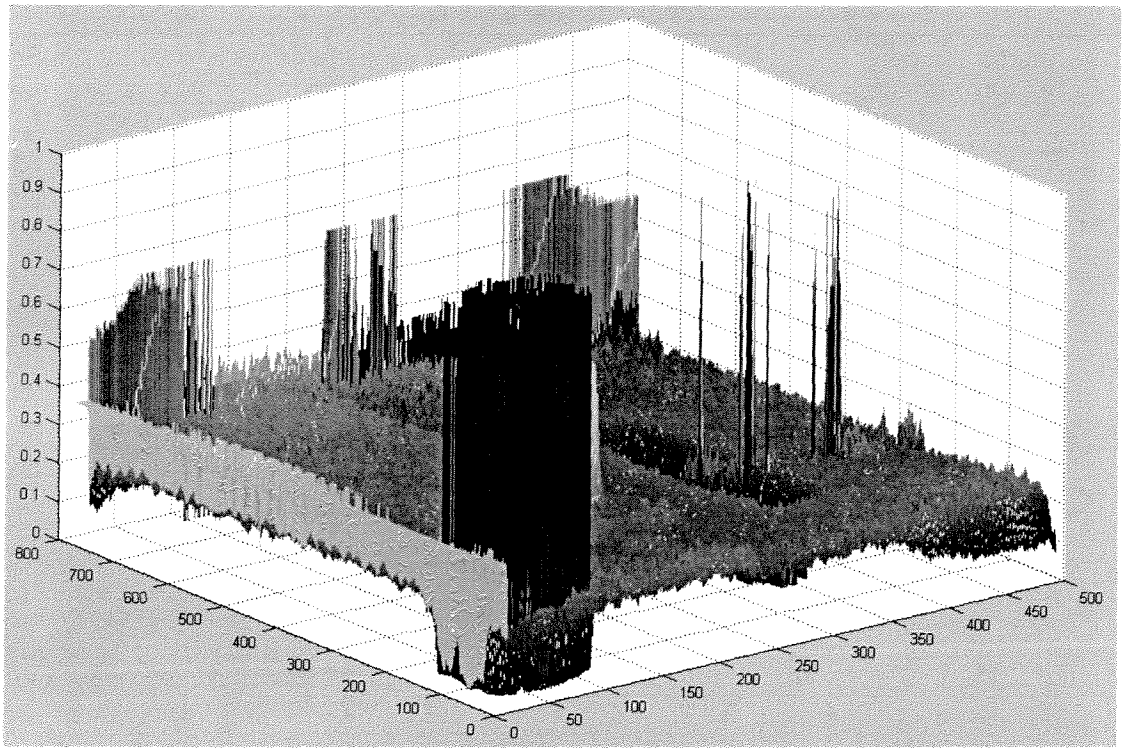


Figure 5.6 The hue component of the original color image in the HSI color space.

The second step is statistical feature filtering, which separates the color blobs of interest from the background. In Figure 5.5, there are four types of sample hue (purple, green, orange, and blue), which have to be detected from an image. First, the sample hue values are extracted from the region of interest. For example, the *4*4=16* pixels, which are around the center of the purple blob on the mobile robot in Figure 5.5, are extracted. Then the average h_p and the standard deviation σ_p are computed from the *16* sample data values, using the equations (5.4)-(5.5) in Section 5.3. Here, the subscript p denotes the color "purple." Next, $h_p \pm 1.2 * \sigma_p$ is considered as the range of "purple hue," which is of interest here. Through similar operations, the statistical results of $h_g \pm 1.2 * \sigma_g$, $h_o \pm 1.2 * \sigma_o$, and $h_b \pm 1.2 * \sigma_b$ are obtained as well. In this experiment, the hue parameters of the four color blobs are as follows:

$$h_p = 0.93948, \sigma_p = 0.00655$$

$$h_b = 0.54805\ , \sigma_b = 0.03896$$

$$h_g = 0.35959, \sigma_g = 0.00527$$

$h_o = 0.02253, \sigma_o = 0.001$

The hue component image in Figure 5.6 is filtered pixel by pixel according to the algorithm described in Section 5.3.2. Only the pixels with hue values falling in the range $h_i \pm 1.2 * \sigma_i$ (i=r, g, o, b) are retained. The filtering result obtained in this manner is shown in Figure 5.7.

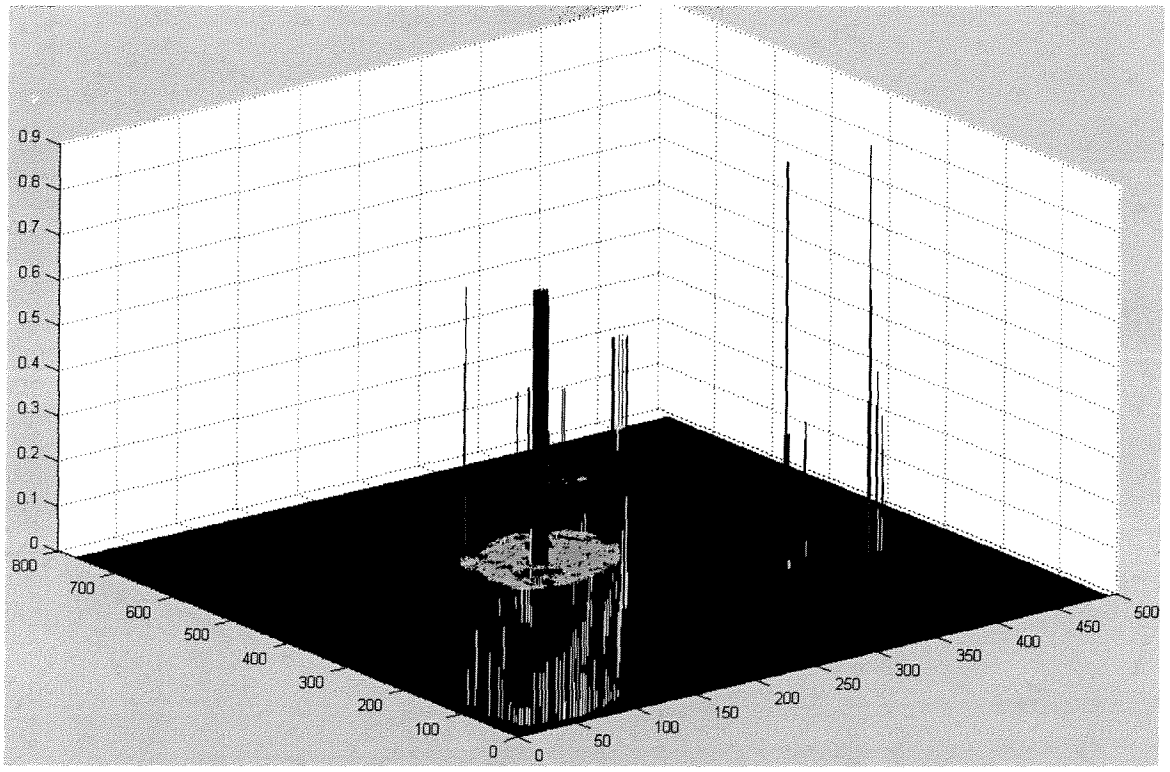


Figure 5.7: The hue component image after the background is filtered.

From Figure 5.7 it is clear that the background of the image is virtually removed and only the four types of sample hue (purple, green, orange and blue) are retained, which are distinguished with the distinct values 0.3, 0.5, 0.7 and 0.9. However, because the hue of the mobile robot is quite similar to the orange blob and there still exist some small regions of disturbance, the positions of the color blobs of interest are not very clear in Figure 5.7.

The third step is to use the template matching method described in Section 5.3.3 and some known space information (for example, the distance between the blue and the purple blobs is always greater than the distance between the blue and the orange blobs) to remove the small unwanted regions and the mobile robot hue in Figure 5.7. The template matrices of four color blobs are generated off line from their original images, which are presented

below.

$$T_{purple} = \begin{bmatrix} 0.95144 & 0.93723 & 0.93841 & 0.94077 \\ 0.95047 & 0.93540 & 0.93663 & 0.93973 \\ 0.94558 & 0.92946 & 0.93457 & 0.93535 \\ 0.94960 & 0.93283 & 0.93496 & 0.93924 \end{bmatrix}$$

$$T_{blue} = \begin{bmatrix} 0.53261 & 0.54085 & 0.50000 & 0.55731 \\ 0.53026 & 0.54336 & 0.52108 & 0.53860 \\ 0.58946 & 0.60163 & 0.62418 & 0.61359 \\ 0.50949 & 0.53026 & 0.50000 & 0.53612 \end{bmatrix} +$$

$$T_{green} = \begin{bmatrix} 0.36359 & 0.35129 & 0.36415 & 0.35065 \\ 0.36787 & 0.36254 & 0.36306 & 0.35478 \\ 0.36301 & 0.35636 & 0.36244 & 0.35636 \\ 0.36595 & 0.35819 & 0.35960 & 0.35364 \end{bmatrix}$$

$$T_{orange} = \begin{bmatrix} 0.02253 & 0.02252 & 0.02253 & 0.02253 \\ 0.02253 & 0.02253 & 0.02253 & 0.02253 \\ 0.02253 & 0.02253 & 0.02253 & 0.02253 \\ 0.02253 & 0.02253 & 0.02253 & 0.02253 \end{bmatrix}$$

The processing result is shown in Figure 5.8.

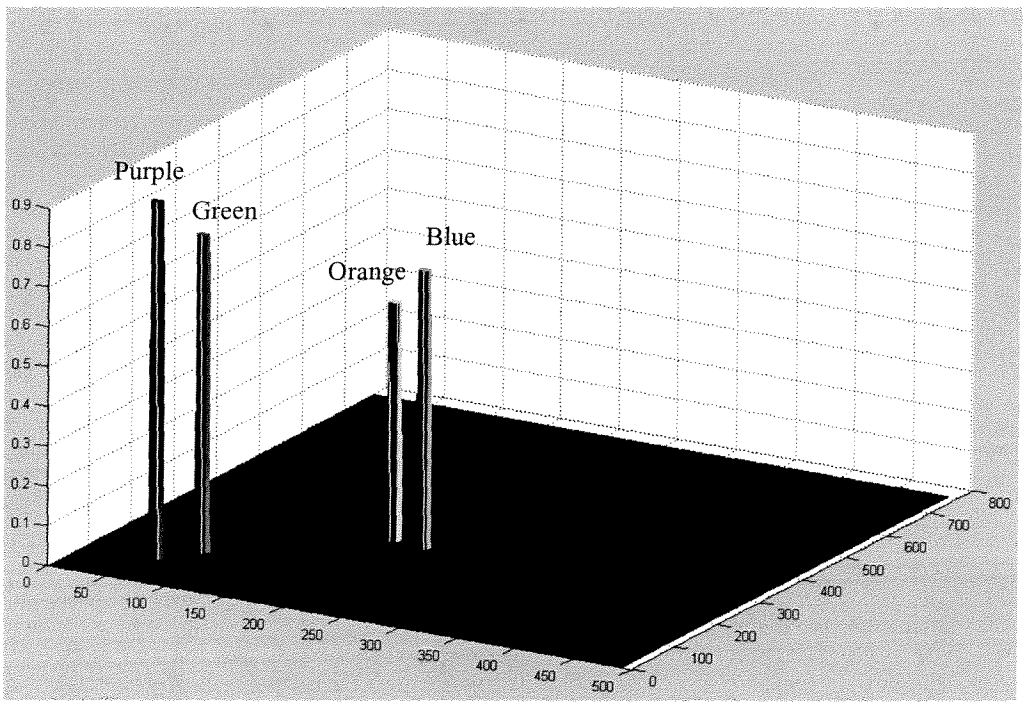


Figure 5.8: The result after filtering Figure 5.7 using template matching.

From Figure 5.8 it is observed that the positions of the four color blobs are accurately established. Through coordinate transformation operations, the positions and orientations of the mobile robot and the box are reconstructed, as shown in Figure 5.9.

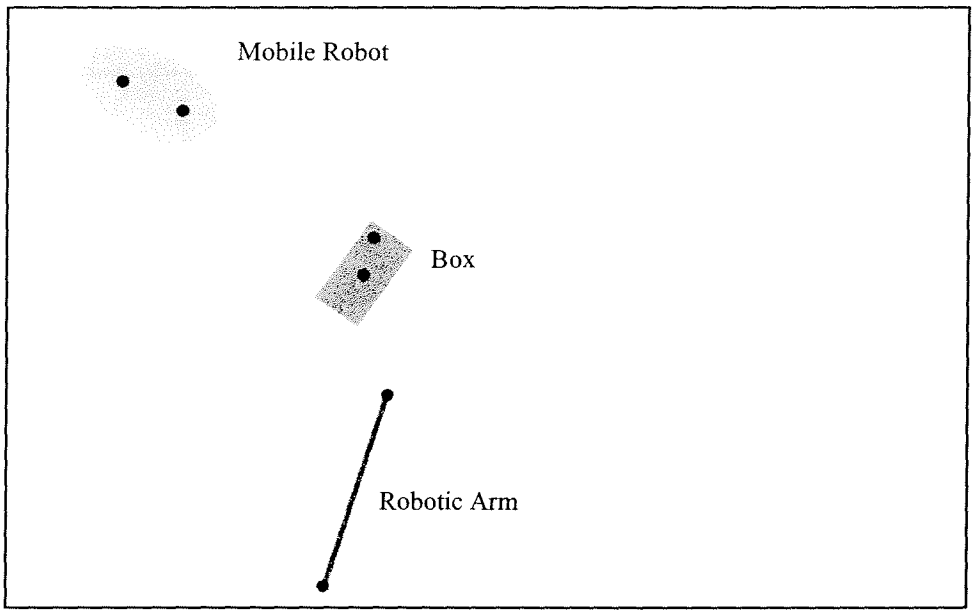


Figure 5.9: The reconstructed result of the poses of the robots and the box.

Because the image processing scheme presented here does not require too many complex operations and since some operations may be carried out off line, the total time of acquiring and processing an image is found to be less than 500 ms, which meets the requirement of the transportation task in the present application.

5.5 Multi-robot Route Planning

The positions and orientations of multiple robots and the object can be tracked simultaneously using the color-blob tracking algorithm presented in this chapter. Consequently, a potential-field-based multi-robot route planning algorithm can be designed.

A main challenge in the multi-robot route planning is the avoidance of collisions with other robots and objects. Finding an optimal and collision-free route in the presence of multiple robots is rather difficult and requires complex and time-consuming computation.

Instead, a simplified multi-robot route-planning approach is employed in this chapter, which is summarized below.

1) Before carrying out route planning, each robot broadcasts its current position and the destination position to all other robots in the team.
2) Each robot broadcasts the positions of obstacles detected by its sensors to its team members.
3) Each robot fuses the obstacle information received from its peers and its own sensors, and computes a global obstacle distribution.
4) Each robot computes the moving distance between its current position and its destination position. Based on the distance information of all robots, a robot queue is generated by sorting the distances in the ascending order. In this queue, the robot with a smaller moving distance will be placed farther ahead.
5) Based on the global obstacle distribution information, each robot computes its route from its current position to its destination position, using the potential-field-based path-planning approach. When computing the route of a robot, the other robots are regarded as special obstacles whose positions are determined as follows: If a teammate is ahead of the current robot in the queue mentioned in step 4, this teammate is regarded as an obstacle whose position is the same as its destination position. If a teammate is behind the current robot in the queue, this teammate is regarded as an obstacle whose position is the same as its current position. In this manner all the robots compute their routes at the same time.
6) After the routs of all the robots are computed in step 5, they will move to the destinations one by one according to these routes. When one robot is in motion, the other robots will remain stationary. In particular, the robot in the first place in the queue will move first, until it reaches its destination positions, and during this period the other robots will stay in their current positions. After the first robot reaches its destination, the second robot in the queue will begin to move towards its destination. This procedure will be repeated until all the robots reach their destinations.

Through the strategy of "Plan the routes simultaneously and move at different times," the multi-robot route planning approach presented here avoids complex optimal path-planning algorithms. As a result it possesses superior real-time performance in practical

multi-robot tasks. The main drawback is that the robots are not allowed to move at the same time. In some multi-robot tasks, as in multi-robot transportation, where the speed may not be a critical specification, the present multi-robot route-planning algorithm can be quite advantageous.

5.6 Summary

In this chapter, a fast and robust color-blob tracking algorithm was presented for use in a developed prototype multi-robot transportation system. In order to remove the disturbances caused by uneven lighting conditions, the original RGB image was converted into the HSI color space and only the Hue component information was retained. Methods of statistical feature filtering and template matching were developed to remove the background and reconstruct the current world state. Because the algorithm did not involve a high computing load, real-time performance could be achieved. A simplified multi-robot route-planning algorithm was developed, which uses the information acquired by the color-blob algorithm. The performance of the method was studied using an experimental robotic object transportation task.

Chapter 6
Experimentation in Distributed Multi-robot Cooperative Transportation

6.1 Overview

Multi-robot cooperative transportation has been an important subject of activity in the robotics community. A typical task of a multi-robot system is where two or more autonomous robots attempt to cooperatively transport an object of interest to a goal location in a static or dynamic environment. There is both theoretical and practical significance to address issues of multi-robot transportation. First, a multi-robot system provides a good benchmark to validate and evaluate various artificial intelligence (AI) techniques such as machine learning, and planning or reactive approaches in a multi-agent environment. The traditional single-agent AI technologies face special challenges in a multi-robot or multi-agent task. These challenges include task assignment, dynamic environment, and distributed planning. New AI approaches such as multi-agent learning, and planning techniques are in development, which will deal with important issues that arise in a multi-agent environment. Multi-robot transportation provides an ideal platform to test and validate these new techniques. Second, multi-robot transportation itself has many practical applications in industrial projects and scientific explorations. It is desirable to have multiple autonomous robots transport an object of interest in a dangerous or hazardous environment, rather than employing human operators for it.

In this chapter, the techniques developed in the previous chapters, such as the hierarchical multi-robot architecture, the SQKF algorithm, and the multi-robot color-blob tracking algorithm, are integrated to implement a physical multi-robot transportation system running in a dynamic and unknown environment. These techniques are tested and validated with the physical robots in our laboratory (IAL). In addition, a hybrid

force/position control strategy suitable for multi-robot transportation is developed and tested using the physical robots.

6.2 Test bed

An experimental system has been developed in the Industrial Automation Laboratory (IAL) at The University of British Columbia to implement a physical multi-robot object transportation system. An overview of this system is presented in Figure 6.1.

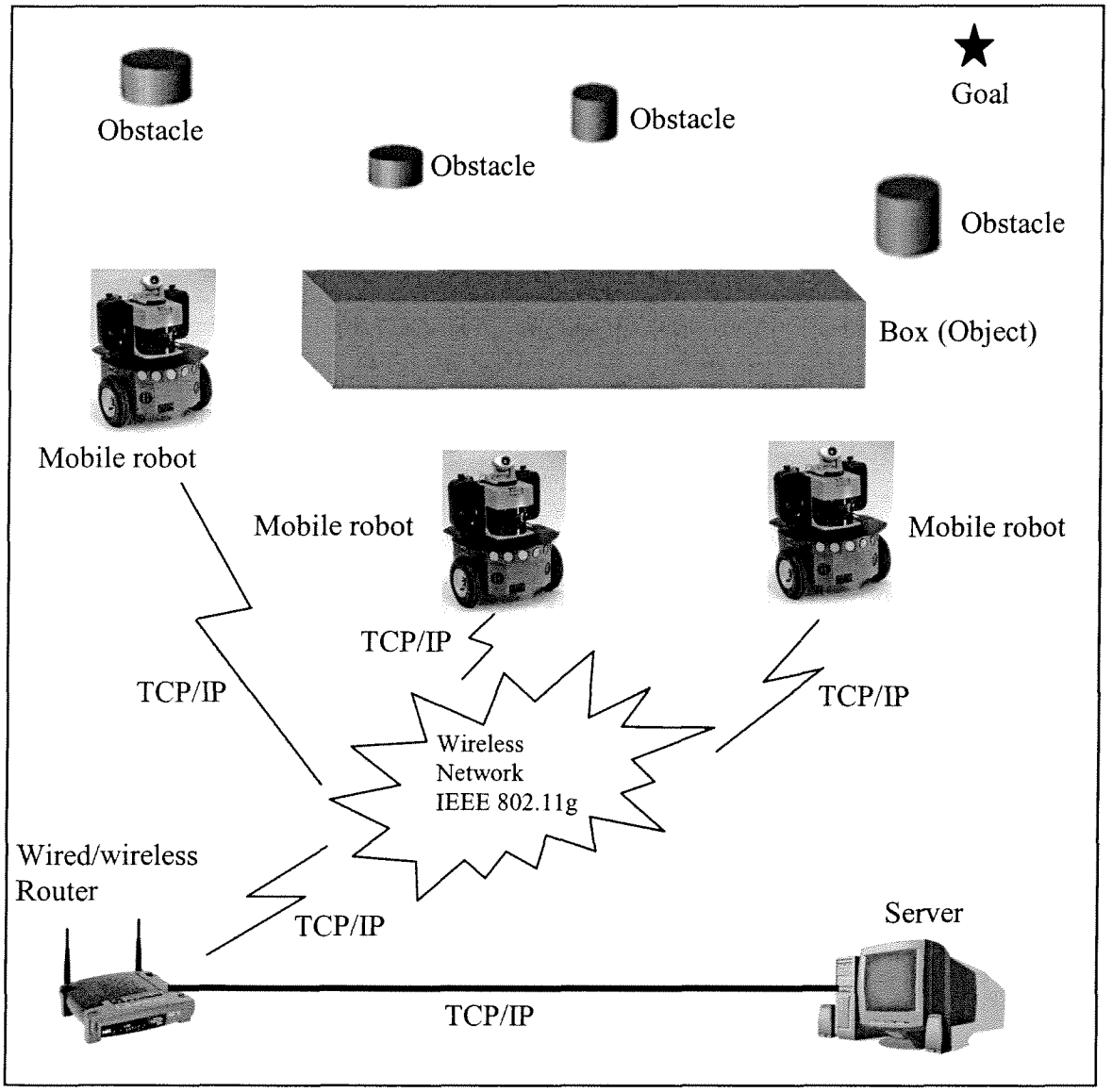


Figure 6.1: The multi-robot object transportation system developed in IAL.

In the developed system, three autonomous mobile robots are employed to transport a long box to a goal location on the lab floor. In order to focus on the techniques developed in the previous chapters, a simple global coordinate system is employed. Each robot is informed its initial position and orientation in this global coordinate system when the multi-robot system begins to operate, and it estimates its latest position and orientation by recording and analyzing the data of the encoders mounted in its wheels and the data of its compass sensor while it moves in the environment. It is also able to inquire the current positions and orientations of its peer robots via the wireless network. In addition, the robots are assumed to know the global coordinates of the goal location in advance.

However, the box and the obstacles are placed on the floor in a random manner. Therefore the robots have to search and estimate the poses of the box and the obstacles in the environment by fusing the sensory data from its sonar, laser distance finder and CCD camera. Furthermore, the sensors are assumed to only detect objects within a radius of 1.5 meters, and a robot is able to exchange its sensory information with its peers via wireless communication to establish a larger local world state. In essence, this is a typical local sensing system and the robots are able to know only a local part of the whole environment.

There are different color blobs on the four lateral sides of the box so that a robot can estimate the orientation and position of the box by identifying the color blobs with its own CCD cameras and fusing this information with the data from its sonar and laser distance finder. If an object without any color blobs is detected, it will be regarded as an obstacle in the environment. The box with the color blobs is shown in Figure 6.2.

(a) A big blue blob and a small blue blob on the same surface of the box.

(b) A blue blob on one side and two green blobs on another side.

Figure 6.2: The color blobs used to identify the orientation of the box.

Each robot makes decisions independently by itself. However, before it makes a decision, it needs to exchange information with its peers so as to form a cooperation strategy, as presented in Chapter 4.

There is a computer server in the developed system, which is used to synchronize the actions of the robots. However, it does not serve as a centralized decision maker because the system is fully distributed where each robot makes decisions independently.

The mobile robots are manufactured by MobileRobots Inc. (formerly ActivMedia Robotics Company) which is a main player in the mobile robot market. In this project, one four-wheel driven Pioneer^{TM} 3-AT robot and two two-wheel driven Pioneer^{TM} 3-DX robots are used. They represent an agile, versatile, and intelligent mobile robotic platform with high-performance current management to provide power when it is needed. Built on a core client-server model, the P3-DX/AT robots contain an embedded Pentium III computer, opening the way for onboard vision processing, Ethernet-based communications, laser sensing, and other autonomous functions. The P3 robots store up to 252 watt-hours of hot-swappable batteries. They are equipped with a ring of 8 forward sonar and a ring of 8 rear sonar. Their powerful motors and 19cm wheels can reach speeds of 1.6 meters per second and carry a payload of up to 23 kg. In order to maintain accurate dead reckoning data at these speeds, the Pioneer robots use 500-tick encoders. Its sensing moves far beyond the ordinary with laser-based navigation, bumpers, gripper, vision,

compass and a rapidly growing suite of other options. The appearance of the P3-DX robot and P3-AT robot is shown in Figure 6.3 and Figure 6.4.

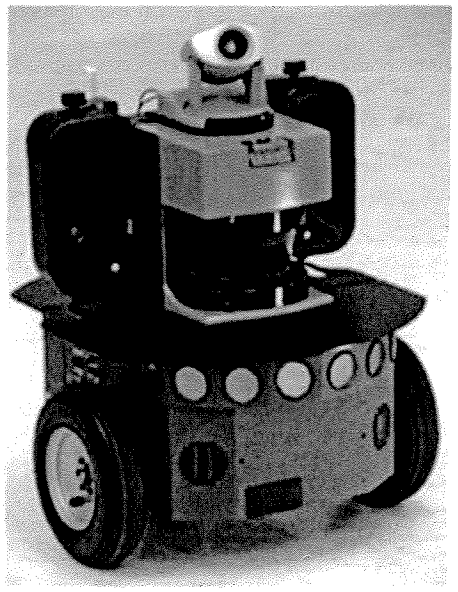


Figure 6.3: The Pioneer™ 3-DX mobile robot.

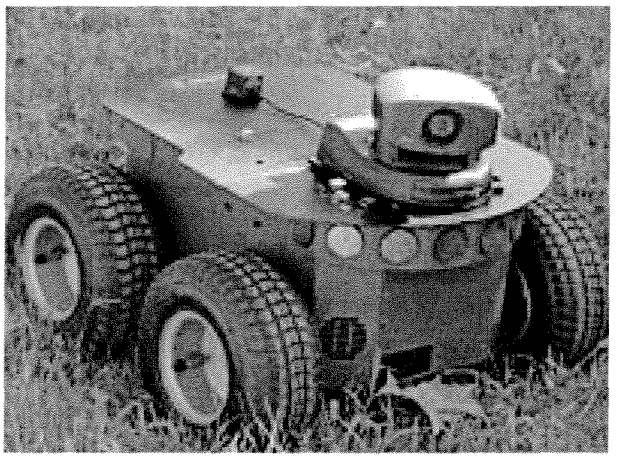


Figure 6.4: The Pioneer™ 3-AT mobile robot.

The P3-DX/AT robots with the ARIA software have the ability to:

- Wander randomly
- Drive under control of software, keys or joystick
- Plan paths with gradient navigation
- Display a map of its sonar and/or laser readings
- Localize using sonar and laser distance finder

- Communicate sensor and control information relating sonar, motor encoder, motor controls, user I/O, and battery charge data
- Provide C/C++/JAVA development platform.
- Simulate behaviors offline with the simulator that accompanies each development environment

With its Laser Mapping & Navigation System and MobileEyes, the robot can map buildings and constantly update its position within a few cm while traveling within mapped areas. With appropriate accessories, a user can view the view of the robot remotely, speak, play and hear audio and send the robot on patrol.

In summary, the Pioneer-3 DX or AT robot is an all-purpose base, used for research and applications involving mapping, teleoperation, localization, monitoring, reconnaissance, vision, manipulation, cooperation and other behaviors.

6.3 JAVA RMI

The multi-robot system developed in this chapter is fully distributed. Each robot identifies the current world state and selects the corresponding action independently. In order to find good cooperation strategies with other robots and complete a common task quickly and effectively, each robot has to observe the environmental states and actions of other robots frequently through fusing its sensory data and communicating with its teammates. Therefore, frequent communication takes place among the robots.

In the developed multi-robot transportation system as presented in Figure 6.1, the IEEE 802.11g wireless TCP/IP protocol provides the communication infrastructure for the robots. While the TCP/IP protocol is a reliable and convenient communication tool, it only works in a low level of the communication hierarchy, which only transfers binary data packages. However, the information which the robots in a multi-robot system may need to exchange, can include data with high-level structures like an action list and world states. As a result, the designer of a multi-robot system has to design his own high-level communication protocol that can run on the TCP/IP protocol so that the robots can exchange information effectively.

A considerable amount of energy and time is required to design a customized high-level communication protocol. It is not sensible for the designer of a multi-robot system to

develop such a customized communication protocol. In this thesis, this problem is solved by using the JAVA RMI technology (Horstmann and Cornell, 2005).

JAVA RMI (Remote Method Invocation) is the JAVA implementation of "Distributed Computing." Its basic idea is the use of a family of objects which can be located anywhere in the network and collaborate with each other through the standard TCP/IP communication protocol. The concept of "distributed computing" or "distributed objects" is proposed to solve critical problems in the traditional client/server computing model. In the client/server model, if a client wants to request data from server, it has to first convert the request into a specific data format which is suitable for transmission via the network. When the server receives this data package, it parses the requested data format, finds the response data in its database, converts it into the desired format, and send it to the client. When the client receives the response data, it also needs to parse the data format and show the result to the users. The traditional client/server mode requires users to design their own data transmission format and develop the corresponding conversion program, which increases the programming burden of the user and degrades the performance of compatibility.

The idea of JAVA RMI is to let a client call a method of a remote object that runs in another computer in the network. The JAVA RMI will automatically convert the request and response data into an appropriate transmission format and administer the communication process for the users. Its principle is presented in Figure 6.5.

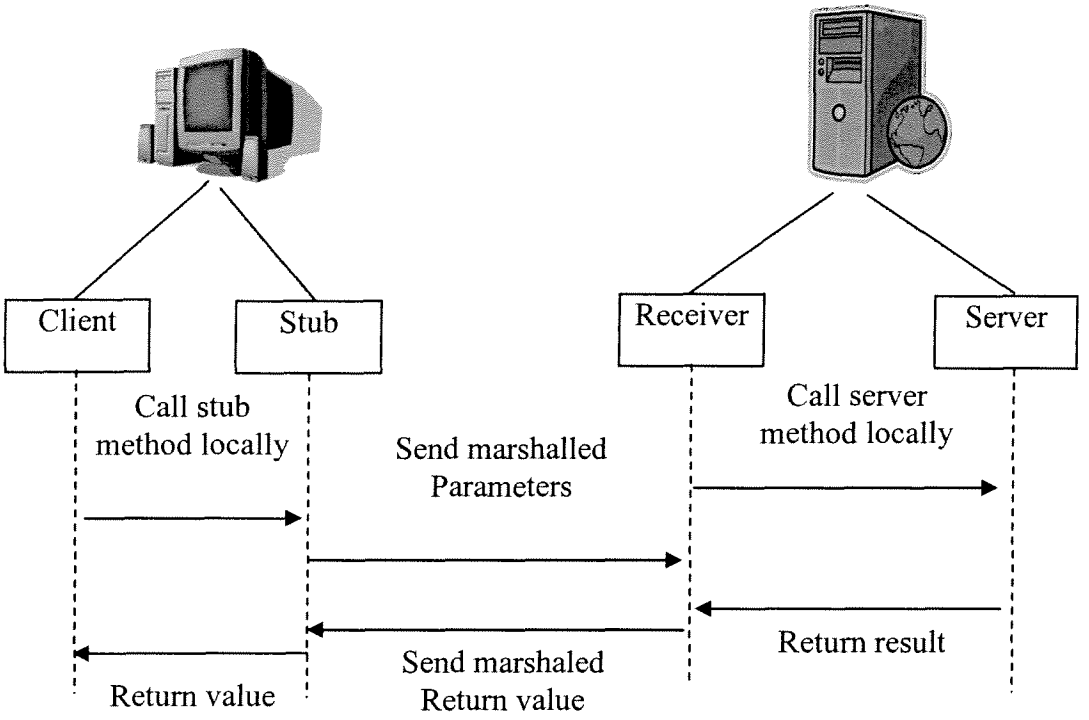


Figure 6.5: JAVA RMI distributed computing

In Figure 6.5, when a client wants to call a method of a remote object that runs on the server, a proxy object termed stub will be called automatically. The stub object resides on the client machine and it packages the parameters used in the remote method into a data package which is encoded with a device-independent protocol. This encoding process is called parameter marshalling whose purpose is to convert the data into a specific format suitable for communication over the network.

When the server receives the data package sent by the stub object on the client machine, it will parse the package, find the object of interest on the server machine and pass the parameters to its corresponding method. The return value of the method will be packaged by the server and sent to the client. When the stub object on the client machine receives the return data package from the server, it will parse it, extract the return data and pass the data to the client object locally.

The whole communication process is based on the TCP/IP protocol, and the data conversion and transmission operations are administered by JAVA RMI automatically. With the help of JAVA RMI, a client can call a remote method on the server machine as if it were a method of a local object.

6.3.1 Remote Interface

In order to implement the JAVA RMI, some programming rules have to be established. First, a remote interface has to be set up, which is used to declare which remote methods can be invoked by the client. A representative remote interface of the developed multi-robot project is presented below.

```
public interface robotServerInterface extends Remote{
   void receiveRobotActionList(Vector actionList, boolean isInitialAction) throws
      RemoteException;
   objPose getPose() throws RemoteException;
   boolean awakeMe()throws RemoteException;
   boolean isSleeping() throws RemoteException;
}
```

This JAVA interface declares that there are 4 remote methods of the robot server which can be called by clients (other robots in the same environment). A robot can call these methods to collect the information of another robot, such as its current action list, current pose, or its sleeping status. In addition, it can wake another robot by calling the remote method of *awakeme()*.

The remote method information is shared by both the client and the server, so the remote interface resides simultaneously on both machines.

6.3.2 Server Side

On the server machine, an object is created which implements the remote methods revoked by the client. A program script of the server object is shown below.

```
public class robotServer extends UnicastRemoteObject
        implements robotServerInterface{
        ...
   public robotServer(robot r)  throws RemoteException {
        theRobot=r;
   }
   ...
    public objPose getPose()  throws RemoteException{
        Lock rlock = rwlock.readLock();
        rlock.lock();
        try{
          return theRobot.getPose();
        }
        finally{
           rlock.unlock();
        }
     }
      ...
}
```

In addition, the server has to register the remote objects with the bootstrap registry service so that the client can locate them and download their stub objects correctly. The registration codes are as follows:

```
String robotName="Robot #2";
robotServer rServer = new robotServer(myRobot);
System.out.println("Binding robot server implementations to registry...");
Context namingContext = new InitialContext();
namingContext.bind("rmi:"+robotName, rServer);
System.out.println("The robot server is ready for calling");
```

6.3.3 Client Side

In order to revoke the remote methods on the server machine, a client program first looks for the remote objects on the server with a known IP address. This process is described below.

```
...
try{
    String url="rmi://142.103.17.138/";
    Context namingContext = new InitialContext();
    robotServerInterface rServer= (robotServerInterface)
        namingContext.lookup(url+"Robot #2");
    rServer.receiveRobotActionList(actionList,true);
    ...
}
catch (Exception e) {
    e.printStackTrace();
}
...
```

Furthermore, a security manager should be loaded by the JAVA RMI to prevent a malicious program from attacking the system. The program script is presented below.

```
...
try
{
  System.setProperty("java.security.policy", "client.policy");
  System.setSecurityManager(new RMISecurityManager());
}
catch (Exception e)
{
  e.printStackTrace();
}
...
```

This script will load the JAVA security policy file: client.policy, which is stored in the same directory as the client program. The security policy is used to restrict the port range and related operations. In the developed multi-robot transportation project, the content of the security policy file is as follows:

```
grant
{
  permission java.net.SocketPermission
    "*:1024-65535", "connect,accept";
  permission java.net.SocketPermission
    "localhost:80", "connect";
  permission java.io.FilePermission
    ".\\-", "read,write,delete";
};
```

6.4 A Physical Multi-robot Transportation System

A physical multi-robot transportation system has been developed in the Industrial Automation Laboratory at The University of British Columbia, which integrates various advanced approaches developed in chapters 2 through 5. This is a distributed multi-robot system with local sensing capability. An experiment is carried out to validate these approaches in a real environment in the presence of sensor noise.

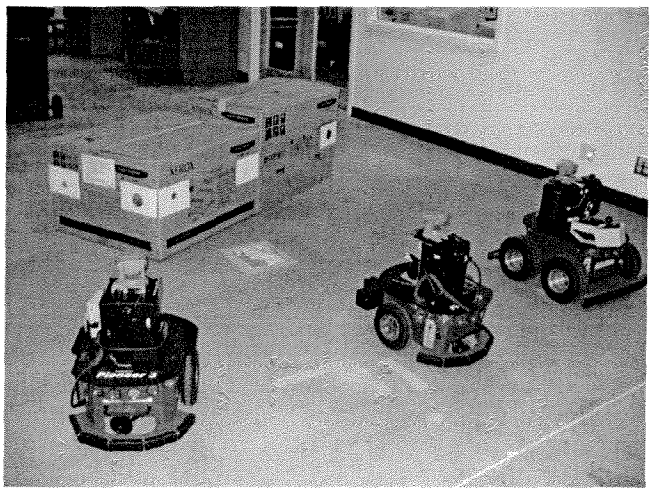
In this experiment, to test the cooperative transportation capability of the developed system, three mobile robots equipped with the SQKF algorithm are employed to transport a big box from one end of the environment to the other one.

Before the experiment is carried out, the simulation system described in the previous chapters is employed to train the robots so that their Q-tables or knowledge bases are improved. After 10,000 rounds of simulated box-pushing, the Q-tables of three simulated robots are exported to the three physical robots to complete a physical cooperative box-pushing task.

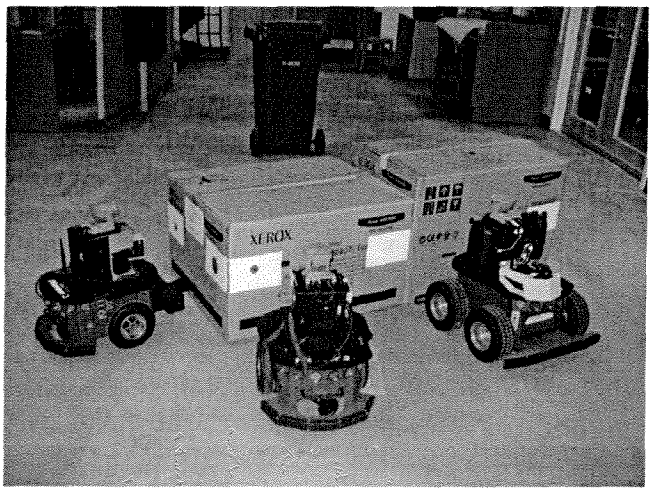
A different policy from the simulation system is employed in the experimental system developed here. In the simulation system described in Chapter 4, after each step of box-pushing, the robots will identify the new world state and select the corresponding actions

with the SQKF algorithm. However, it is time-consuming and not practical for the robots in a real experimental system to change their actions (pushing positions) frequently. Instead, a new policy is employed in the experimental system, where the robots will continue with the actions selected in the previous step unless the reward earned in the previous step is lower than a predefined threshold value. This policy avoids changing the actions of the robots very frequently, which will be expensive and time-consuming for a physical multi-robot box-pushing system.

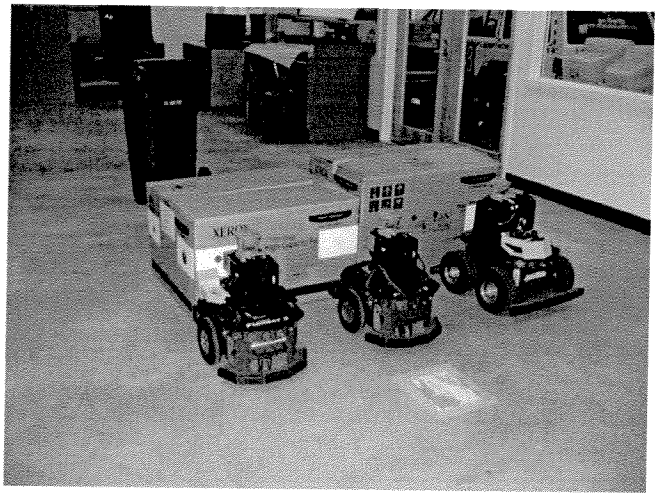
The experimental results of cooperatively transporting a box in a real environment are presented in Figure 6.6.



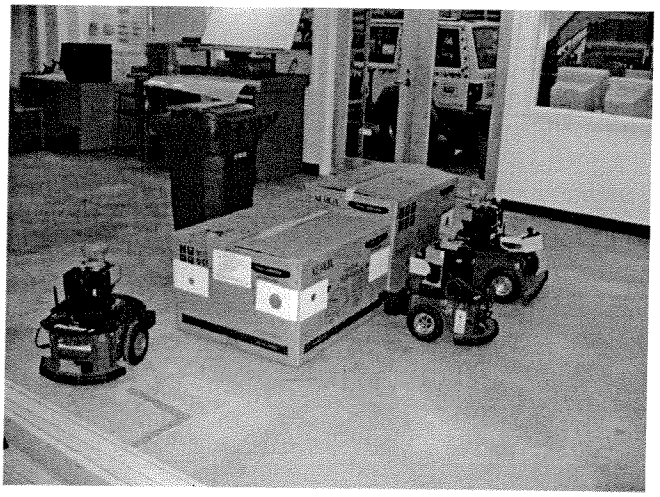
(a)



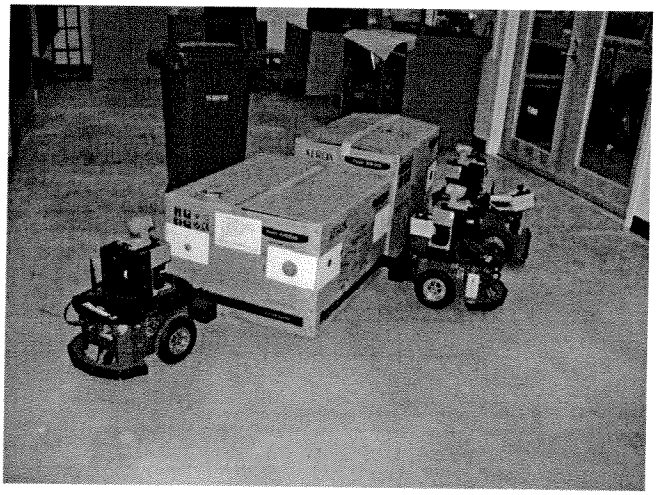
(b)



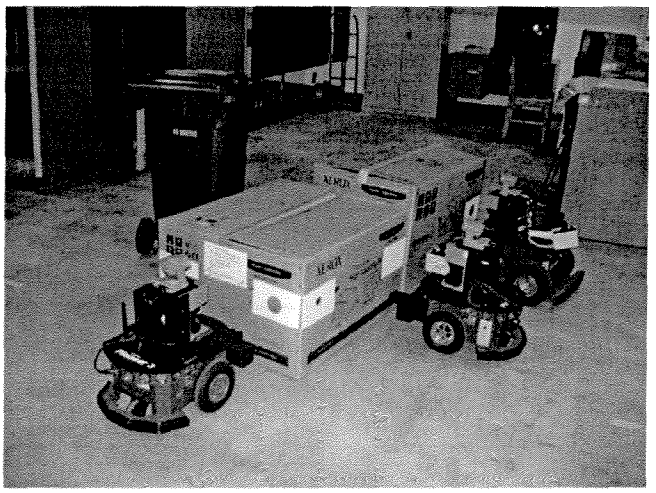
(c)



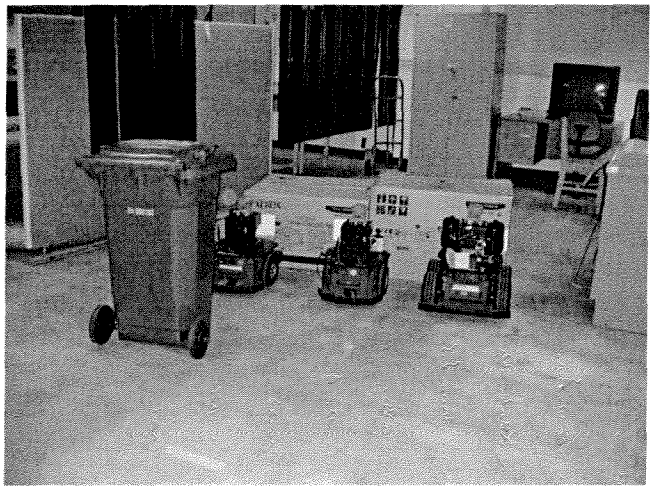
(d)



(e)



(f)



(g)

Figure 6.6: Multi-robot cooperative transportation.

In Figure 6.6 (a), a big box is placed on the floor, which is within the detection radius of the sensors of the mobile robots. The three robots are informed about their initial positions and orientations in the global coordinate system before they begin to work. When the system starts to operate, each robot uses its CCD camera to search and identify the color blobs on the box surface and to estimate the orientation of the box relative to the current pose of the robot. By fusing this relative orientation of the box and the sensory data from its laser distance finder, the robot can estimate the position and orientation of the box in the global coordinate system. If one robot cannot detect the box with its laser or vision sensors, it will communicate with other robots to request the position and

orientation information of the box. If no robot finds the box, they will wander in the environment until one of them detects the box.

At the same time, each robot scans its local environment with the laser distance finder. If an object is detected which is not the box or one of the peer robots, the object will be regarded as an obstacle.

By fusing the information of the box pose with the information of local obstacle distribution, a local world state is established by the robot and the optimal action under this state is selected using its Q-table.

Figure 6.6 (b) shows how the robots push the box with the selected actions, and Figure 6.6 (c) shows that the robots have changed to another formation so that the box is pushed with the biggest net force.

In Figure 6.6 (c), the robots find out there exists an obstacle (the blue garbage bin) in the path, which was not detected before by their sensors due to the limited detection radius. In order to determine the position and area of the obstacle, Figure 6.6 (d) shows that one of the robots moves close to the obstacle and measures the distance between the obstacle and itself. The obstacle position estimated by this robot is sent to its two peers so that they can re-compute their local world states and select the corresponding actions to adapt to the new local world.

Figure 6.6 (e) and (f) show that the robots have changed their formation to adapt to the new world state. Here they attempt to change the orientation of the box in order to avoid the obstacle.

Figure 6.6 (g) shows that the robots have avoided the obstacle successfully and restored to the formation which generates the biggest net pushing force.

From Figure 6.6 (a)-(g), it is observed that the learning based multi-robot system is quite effective in carrying out a cooperative transportation task in an environment with unknown obstacles. The learned Q-tables in the training stage help the robots select good cooperation strategies in a robust and effective manner.

6.5 Force/position Hybrid Control

Force and position control of the robots is quite valuable in a physical multi-robot transportation system. Without proper control of the forces and positions of the robots, the

cooperation strategy described in the previous chapters may not be effective. The hybrid force/position control scheme shown in Figure 6.7 is implemented in the low-level control layer in the multi-robot system of the present work.

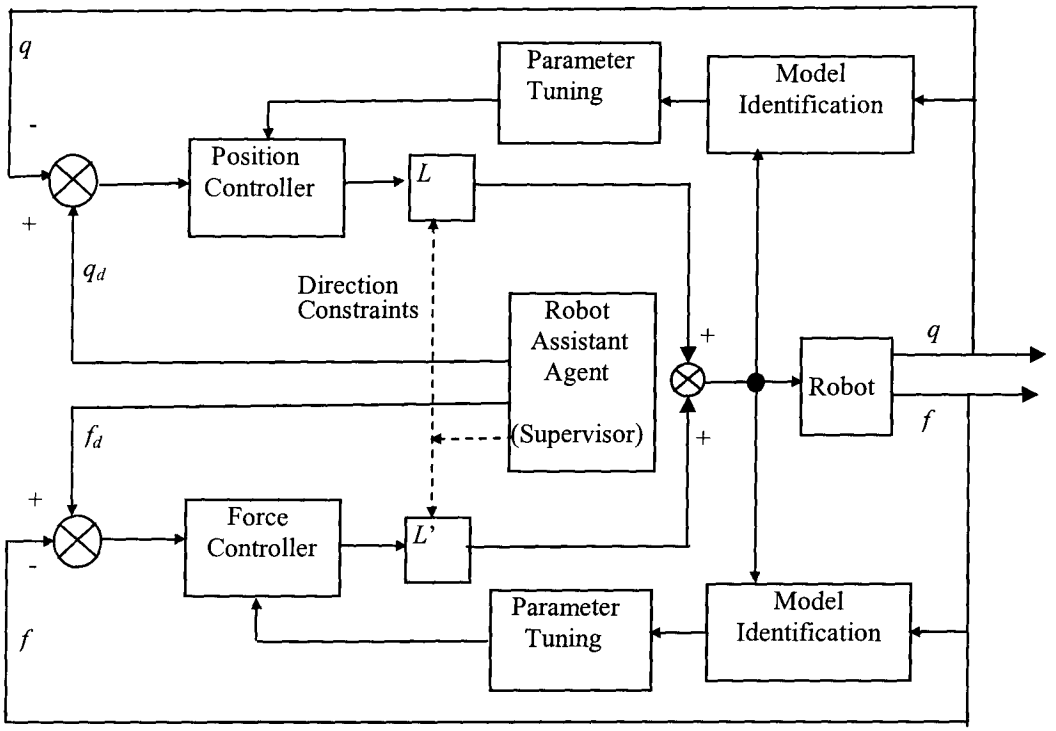


Figure 6.7: The robot hybrid force/position control scheme.

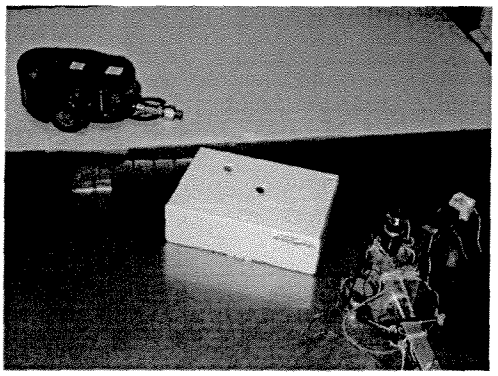
In Figure 6.7, two sets of control modes, position control and force control, are integrated into the robot control system. Here L and L' are the direction constraint matrices (Craig, 2005), which, respectively, determine in which directions the force control mode is used and in which directions the position control mode is employed. They are diagonal matrices with ones and zeros on the diagonal. For a diagonal element of "1" in L, there is a "0" as the corresponding diagonal element of L', indicating that position control is in effect. On the other hand, if a "0" is present on the diagonal of L, then "1" will be present as the corresponding element of L', indicating that force control is in effect. Note that L and L' are set by the robot assistant agent in the upper level. Moreover, the robot assistant agent will provide the desired forces and positions for the low-level control, based on the current cooperation strategy.

In addition, an adaptive control mechanism (Astrom and Wittenmark, 1994) is

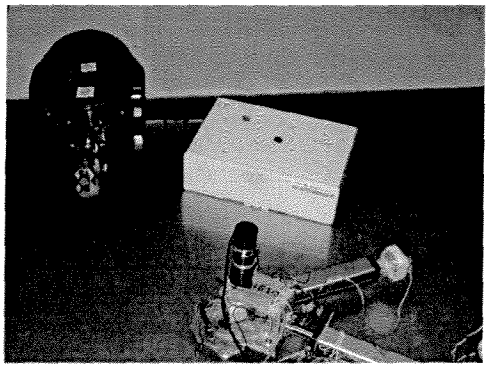
integrated into the control scheme. In each control loop, a dynamic model of the robot is identified on line and this information is used to tune the controller parameters.

Finally, a two-step control strategy is used in the practical control process. When the robot assistant agent receives a new cooperation strategy from the high-level decision-making subsystem, it computes the desired force and position of the corresponding robot. As the upper level supervisor for the low-level control subsystem, the agent then sends commands to the low-level position controller to initiate the first control step, which is to move the robots to the desired position and orientation. Next, the agent starts the second control step, which requires the position controller to continue the position control strategy, while instructing the force controller to execute the desired force control strategy in a specific direction in order for the robot to exert a constant pushing force on the object.

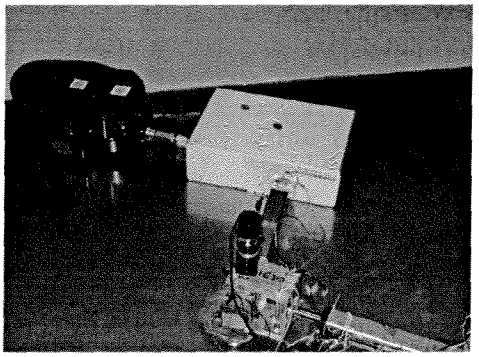
An experiment is developed to validate this force/position hybrid control scheme, where the two robots have to first move to a specified location with a specified orientation, and then cooperatively transport the box for *8* seconds while maintaining a constant contact force. This process is shown in Figure 6.8 (a)-(c).



(a) The initial state



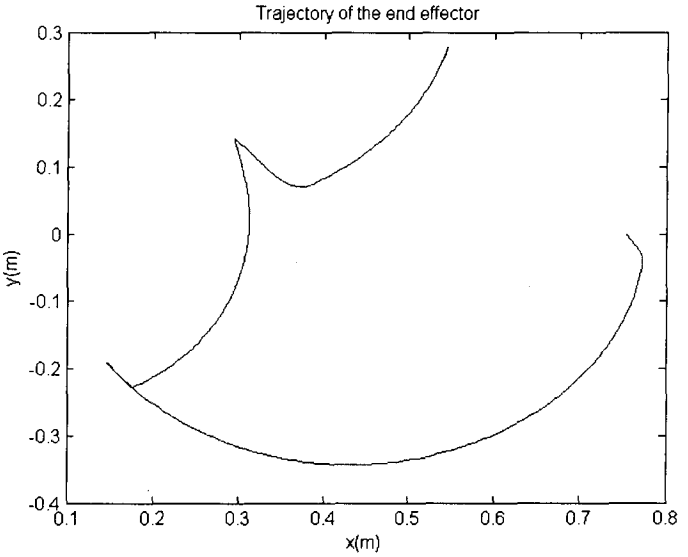
(b) Moving to the desired position



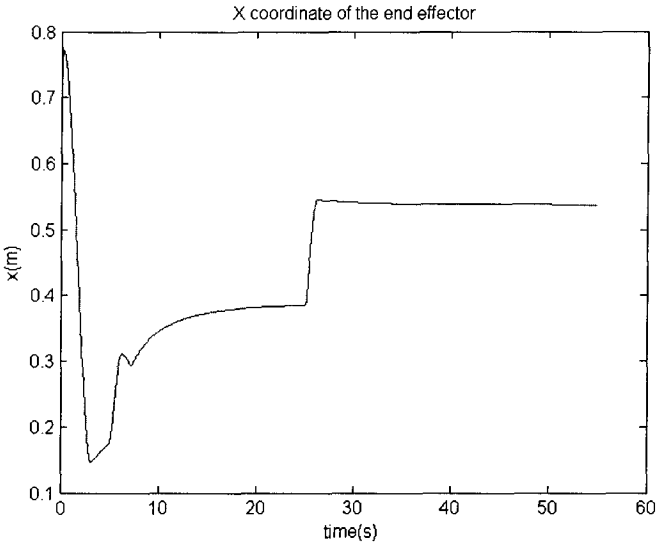
(c)Pushing of the box

Figure 6.8: The robots move to the desired position and push the box under hybrid force/position control.

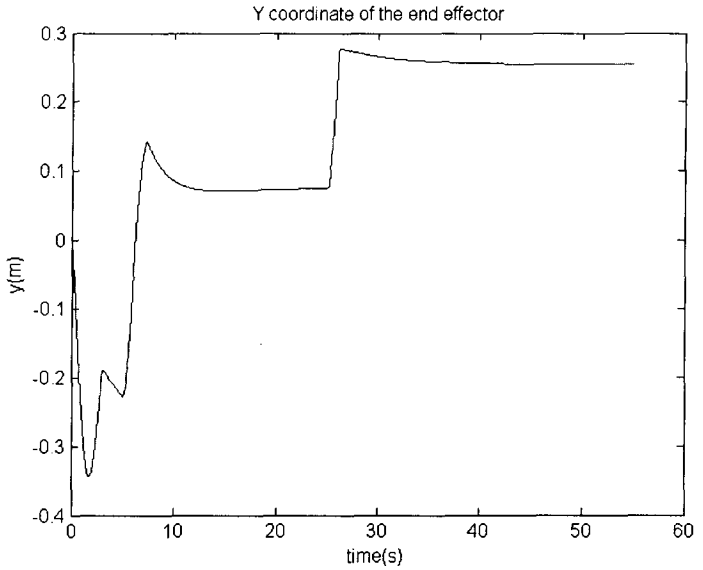
According to the control scheme given in Figure 6.7, the process is divided into two stages: position control and hybrid force/position control. The former is used to move the robots to a desired position with a desired orientation. The latter is employed to exert a constant force in a specific direction while the position is kept unchanged in the other directions. Figure 6.9 (a)-(d) shows the control process of the robotic arm in the first stage.



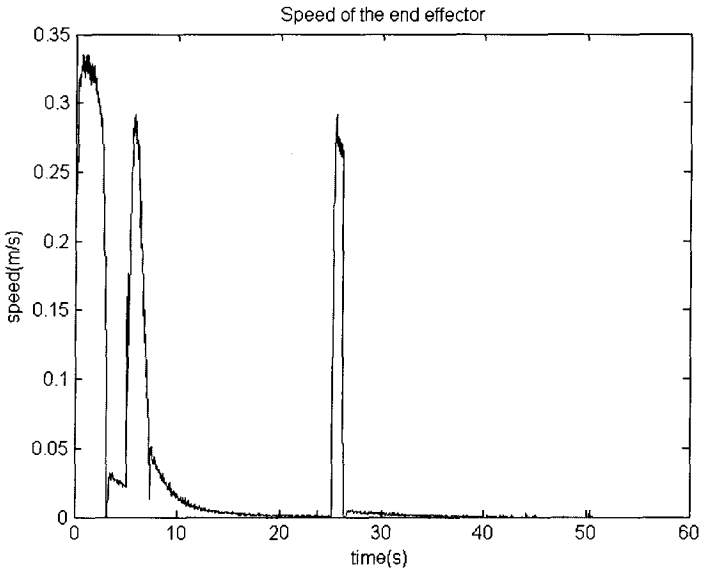
(a) The trajectory of the end effector.



(b) The x coordinate of the end effector.



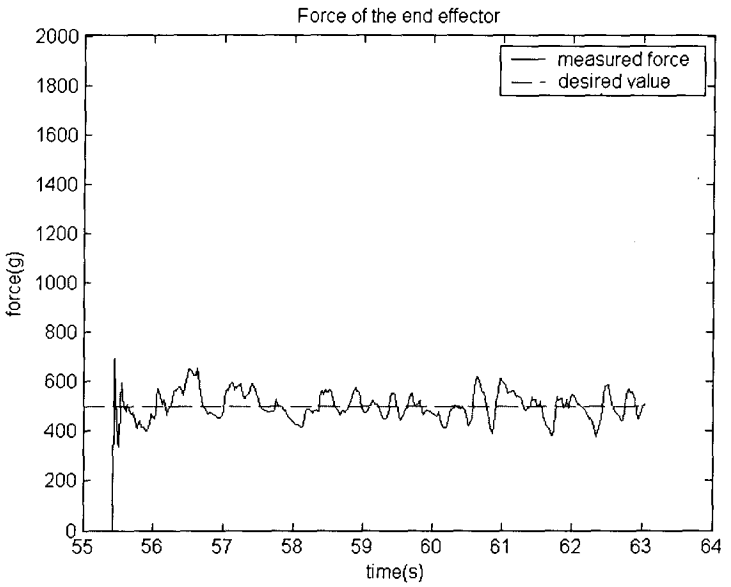
(c) The y coordinate of the end effector.



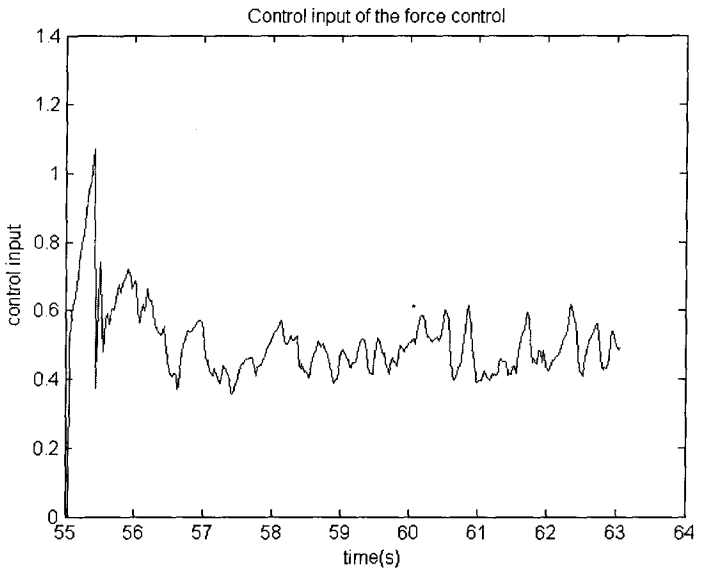
(d) The speed of the end effector.

Figure 6.9: Position control of the robotic arm.

Figure 6.10 (a)-(b) shows the force curve and the control input of the robotic arm under the hybrid force/position scheme in the second stage.



(a) The force on the end effector.



(b) The control input of the joint.

Figure 6.10: The force control of the robotic arm.

From Figures 6.8-6.10 it is observed that the hybrid force/position control scheme and the two-step control strategy, which are presented in this chapter, perform properly in the experiments.

6.6 Summary

In this chapter, a physical multi-robot transportation system is implemented in the Industrial Automation Laboratory (IAL) at the University of British Columbia, which integrated the approaches developed in the previous chapters. In this system, three mobile robots were employed to push a big box to a goal location in the laboratory environment. There were three challenges to reach the goal in the developed system. First, each robot only possesses local sensing capability which means they have to make decisions while they do not possess knowledge of the global environment. Second, there exist sensor noise and uncertainty of actions in a physical system, which do not exist in a simulation system. These constraints degrade the performance of the robots. Finally, the environment is dynamic due to the presence of random obstacles. All these represent important challenges which a multi-robot system has to meet in a real natural environment.

The distributed computing technology, JAVA RMI, was introduced into the developed multi-robot system, which enabled the robots to cooperate in an easy and efficient manner. It was demonstrated through experiments in this chapter that JAVA RMI was a powerful tool to implement distributed multi-robot systems.

An experiment was carried out to evaluate the performance of the developed multi-robot system. In this experiment, cooperative robotic transportation of a box in an environment with unknown obstacles was carried out. The experimental results showed that the developed multi-robot system was able to work well in a realistic environment. It can be concluded that the approaches developed in chapters 2 through 5 are helpful and effective in enable multi-robot systems to operate in a dynamic and unknown environment.

A force/position hybrid control scheme was proposed in this chapter, which is important in multi-robot transportation systems. The developed scheme was evaluated using a physical experiment involving a robotic arm pushing a box on a 2D surface. Good performance was achieved by using the hybrid force/position control strategy.

Chapter 7
Conclusions

7.1 Primary Contributions

Multi-robot systems have to undergo many improvements before they can be used in real-life applications. For example, there are enormous challenges to develop a multi-robot system which can cooperatively transport an object of interest in a dynamic and unknown environment like the Mars surface. In this thesis, several techniques have been developed and integrated to support the operation of multi-robot transportation systems in complex and unstructured dynamic environments with unknown terrains. In particular the thesis has made contributions with respect to self-deterministic learning, robustness, action optimization, and real-time operation of a cooperating team of robots in such environments. Basically, the contributions of the thesis can be classified into four main areas as follows.

First, a general distributed multi-robot architecture was developed in Chapter 2. This hierarchical architecture integrated several techniques from artificial intelligence (AI) so that multi-robot systems could operate in a robust and effective manner in a dynamic and unknown environment. In its top level, a machine learning unit and a local path planning unit were combined to establish good cooperation strategies for the robots while they attempt to transport an object cooperatively. A machine learning unit with effective training was used to select proper actions for the robots so that they could complete a common task quickly and effectively. If a local minimum of decision-making was detected, a local path planning unit would be temporarily activated by an arbitrator to select a local transportation path for the object in order for the robots to escape the local minimum. In the middle level of the architecture, the behavior-based approach, which has been proved to be very effective for single robot systems, was employed to decompose the abstract behavior sent by its upper level into more detailed primitive behaviors so that the detailed control requirements could be generated and accepted by the lower level controller. For the bottom level of the architecture, a low-level controller was designed to

receive control tasks from the behavior-based layer and control the motion or force of the robot. This architecture also included a communication module so that any robot could easily exchange information with other robots using standard network protocols. By combining the learning, planning and behavior-based approaches, and carefully designing the coordination mechanism among them, the developed multi-robot architecture was found to be more flexible and powerful than a traditional two-layer architecture based on a single AI technique.

Second, machine learning was employed to find optimal cooperation strategies for multi-robot systems so that they could operate in a dynamic and unknown environment. For the first prototype system, a centralized learning agent was proposed, where a Q-learning unit and a Genetic Algorithm unit were integrated to search the optimal cooperation strategies for all robots in the environment. While the approach was found to be successful for a robot team with relatively few members, it faced challenges with respect to scalability, computational complexity and communication bottleneck when the number of robots increased. In order to meet these challenges, a distributed Q-learning multi-robot transportation system was developed. By directly extending single-agent Q-learning to the multi-robot domain and carefully designing the reward function, the developed system was able to demonstrate good adaptivity and effectiveness in a complex environment with multiple obstacles. However, due to lack of knowledge of the actions of the peer robots, the single-agent Q-learning algorithm was found to converge very slowly in a multi-robot environment. In order to improve the convergence speed, a team Q-learning algorithm was developed for use in the same multi-robot task. An interesting conclusion was drawn by comparing the simulation results of the team Q-learning algorithm with those of the single-agent Q-learning algorithm. Specifically, single-agent Q-learning showed a better performance (as measured by the average number of steps required in a round of box-pushing) than team Q-learning even though the latter is usually thought to be more efficient. Furthermore, the drawback of a large learning space in the team Q-learning algorithm was demonstrated through computer simulation.

In order to overcome various disadvantages of the traditional single-agent Q-learning algorithm and the team Q-learning algorithm, a modified Q-learning algorithm termed Sequential Q-learning with Kalman Filtering (SQKF) was developed in this thesis. By

enabling the robots to learn in a predefined sequence and employing a Kalman filter to extract the real rewards of a robot dynamically, the new SQKF algorithm was able to overcome several major shortcomings in the single-agent Q-learning algorithm or the team Q-learning algorithm, and consequently more suitable for multi-robot tasks. The simulation results showed that the SQKF algorithm needed less time to complete a multi-robot transportation task and received better reward than the traditional Q-learning algorithms.

Third, a fast computer vision algorithm was developed to track multiple color-blobs simultaneously so that the orientations and positions of the robots could be determined in a multi-robot transportation task. This algorithm was found to meet two challenges in carrying out the specific task: the changing illumination levels in the work environment and the real-time operating requirement of the algorithm. By extracting the hue information in the image and combining it with some statistical techniques and the available domain knowledge, the algorithm was able to effectively and quickly track multiple moving color blobs simultaneously in an environment with uneven illumination.

Finally, a physical multi-robot transportation project was developed in the laboratory to implement and validate the approaches developed in the thesis. The physical system faced more challenges than the computer simulation system; in particular, sensor noise, wheel slip, real-time operation, and motion constraints. In addition, in order to implement effective communication among multiple robots and facilitate proper execution of the new SQKF algorithm, the JAVA RMI technology was incorporated into the system. The experimental results showed that the physical system was able to operate effectively and robustly in a dynamic physical environment with obstacles.

7.2 Limitations and Suggested Future Research

Although the developed multi-robot transportation system has demonstrated quite good performance both in computer simulation and physical experimentation, there are some areas that need improvement. Some directions for further research are indicated next.

7.2.1 Improvement of the Model and Algorithm of SQKF

The SQKF algorithm developed in this thesis assumes a linear model for use in the Kalman filtering algorithm, which may not be true in a real multi-robot system. In practice, there is a high possibility that the model is nonlinear. If the model nonlinearity is significant, the standard linear Kalman filtering algorithm may not estimate the reward values correctly, thereby making the SQKF algorithm ineffective. Therefore, more work may be needed to identify a suitable nonlinear model for the SQKF algorithm. Furthermore, with a nonlinear model one may have to incorporate advanced filtering algorithms, such as the Extended Kalman Filter or the Unscented Kalman Filter, in extending the SQKF algorithm.

7.2.2 GA-based Learning Space Compression

In this thesis, when applying the Q-learning algorithm to the multi-robot domain, the resulting large learning space was not dealt with rigorously. When the number of robots in the environment increases, the resulting extensive learning space will make the Q-learning algorithm ineffective.

One possible solution to this problem would be to integrate a Genetic Algorithm (GA) with the Q-learning algorithm. For example, when a step of Q-learning is completed, by searching an optimal sub-space in the Q-table, the GA algorithm may be able to set up a new learning sub-space for the Q-learning algorithm which is much smaller than the original Q-table. The next step of Q-learning would be carried out using this smaller new sub-space. The challenge is how to determine which sub-space in the Q-table is "optimal" and how to deal with any "unexpected" world states that might be encountered by the robots, which were not included into the "optimal" sub-space.

7.2.3 Local Modeling and Low-level Control

Most multi-robot systems have focused on high-level decision-making and ignored low-level control of multiple robots. Although, some work has been done in this thesis, there is room to improve the control methodology in the context of multi-robot cooperation.

The control needs in a multi-robot system are different from those in a single robot system. Local modeling may be a possible solution to the low-level control problem in multi-robot systems.. The so-called local modeling will be such that one robot not only collects information from its own sensors but also exchanges information with other robots so that a local kinematic or dynamic model can be established to facilitate the control task. The model may include the information of the formation and capabilities of the robots, their current states and actions, the shared environment, and so on. Based on the local model, an advanced force/position controller may be developed to meet the control requirements communicated by the high-level decision-making sub-system.

7.2.4 Integrating Planning and Learning

Planning and learning are both important for multi-robot systems when operate in an unknown and dynamic environment. In this thesis, a type of switching strategy between learning and planning was developed. In particular, a local path planning unit was temporarily activated when a local minimum of the machine learning approach was encountered. However, a more advanced integration scheme would be useful so as to make use of planning in a multi-robot task.

7.3 Summary

New developments in multi-robot systems will endow the next generation of robotic systems with more powerful capabilities. It has been an important objective to allow multiple robots to autonomously carry out such tasks as cooperative object transportation or robotic soccer, in dynamic, unknown, and unstructured environments. In this thesis, several key approaches were developed toward achieving this goal. It is believed that no single AI technique can meet this objective. Instead, learning, reactive (behavior-based) paradigm, planning, optimization, and low-level control approaches should be integrated, and new approaches should be developed on that basis so that an autonomous multi-robot system will be able to operate in a dynamic and unknown environment in an effective and robust manner. Some significant work has been carried out in this thesis towards this general objective, specifically with regard to self-deterministic learning, improved robustness, optimization of robotic action, and real-time operation.

Bibliography

Alenya, G., Escoda, J., Martinez, A.B. and Torras, C., "Using laser and vision to locate a robot in an industrial environment: a practical experience," *Proceedings of IEEE International Conference on Robotics and Automation*, Barcelona, Spain, pp. 3528-3533, April, 2005.

Alpaydin, E., *Introduction to Machine Learning*, The MIT Press, Cambridge, MA, 2004.

Arai, T., Pagello, E. and Parker, L.E., "Advances in multirobot systems," *IEEE Trans. on Robotics and Automation*, Vol.18, No. 5, pp. 655-661, 2002.

Arkin, R.C., *Behavior-Based Robotics*, The MIT Press, Cambridge, MA, 1998.

Asada, M., Uchibe, E. and Hosoda, K., "Cooperative behavior acquisition for mobile robots in dynamically changing real world via reinforcement learning and development," *Artificial Intelligence*, Vol. 110, No. 2, pp. 275-292, 1999.

Asensio, J.R., Montiel, J.M.M. and Montano, L., "Goal directed reactive robot navigation with relocation using laser and vision," *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 4, pp. 2905-2910, May, Detroit, MI, 1999.

Astrom, K.J. and Wittenmark, B., *Adaptive Control*, Second Edition, Addison-Wesley Publishing Company, Menlo Park, CA,, 1994.

Balch, T. and Arkin, R.C., "Behavior-based formation control for multirobot teams," *IEEE Trans. on Robotics and Automation*, Vol. 14, No.6, pp. 926-939, 1998.

Bellman, R.E., *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, 1957.

Bouloubasis, A.K., McKee, G.T. and Schenker, P.S., "A behaviour-based manipulator for multi-robot transport tasks," *Proceedings of IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 2287-2292, 2003.

Camacho, D., Fernandez, F. and Rodelgo, M.A., "Roboskeleton: an architecture for coordinating robot soccer agents," *Engineering Application of Artificial Intelligence*, Vol. 19, No. 2, pp. 179-188, 2006.

Cao, Y.U., Fukunaga, A.S. and Kahng, A.B., "Cooperative mobile robotics: antecedents and directions," *Autonomous Robots*, Vol. 4, No. 1, pp. 7-27, 1997.

Cao, Z., Tan, M., Li, L. Gu, N. and Wang, S., "Cooperative hunting by distributed mobile robots based on local interaction," *IEEE Transactions on Robotics*, Vol. 22, No. 2, pp. 403-407, 2006.

Casella, G. and Berger, R.L., *Statistical Inference*, Second Edition, Thomson Learning, Belmont, CA, 2002.

Chaimowicz, L., Kumar, V. and Campos, M.F.M., "A paradigm for dynamic coordination of multiple robots," *Autonomous Robots*, Vol. 17, No. 1, pp. 7-21, 2004.

Chang, Y.-H., Ho, T., and Kaelbling, L.P., "All learning is local: multi-agent learning in global reward games," *Proceedings of Neural Information Processing Systems (NIPS)*, Vancouver, BC, Canada, 2003.

Chen, S., Shyu, M., Peeta, S. and Zhang, C., "Spatiotemporal vehicle tracking," *IEEE Robotics and Automation Magazine*, Vol. 12, No. 1, pp.50-58, 2005.

Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L.E. and Thrun, S., *Principles of Robot Motion*, The MIT Press, Cambridge, MA, 2005.

Chui, C.K. and Chen G., *Kalman Filtering with Real-Time Applications*, Third Edition, Springer, Berkeley, CA, 1999.

Craig, J.J., *Introduction to Robotics: Mechanics and Control*, Third Edition, Pearson Prentice Hall, Upper Saddle River, NJ, 2005.

Dahl, T.S., Mataric, M.J. and Sukhatme, G.S., "Multi-robot task-allocation through vacancy chains," *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 2293-2298, 2003.

Dahl, T.S., Mataric, M.J. and Sukhatme, G.S., "Adaptive spatio-temporal organization in groups of robots," *Proc. of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Lausanne, Switzerland, pp. 1044-1049, 2004.

De Silva, C.W, *MECHATRONICS—An Integrated Approach*, Taylor & Francis/CRC Press, Boca Raton, FL, 2005.

Deng, L.Y., Tang, N.C., Lee, D., Wang, C.T. and Lu, M.C., "Vision based adaptive traffic signal control system development," *Proceedings of 19th International Conference on Advanced Information Networking and Applications*, Taipei, Taiwan, pp. 385-388, 2005.

Dudek, G., Jenkin, M. and Milios, E., "A taxonomy of multirobot systems," (Chapter 1), Editor: Balch, T. and Parker, L.E., *Robot Teams: From Diversity to Polymorphism*, AK Peters, Ltd., Natick, Massachusetts, 2002.

Elahibakhsh, A.H., Ahmadabadi, M.N., Janabi-Sharifi, F. and Araabi, B.N., "Distributed form closure for convex planar objects through reinforcement learning with local

information," *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, pp. 3170-3175, 2004.

Emery, R. and Balch, T., "Behavior-based control of a non-holonomic robot in pushing tasks," *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, pp. 2381-2388, 2001.

Farinelli, A., Iocchi, L. and Nardi, D., "Multirobot systems: a classification focused on coordination," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 34, No. 5, pp. 2015-2028, 2004.

Ferch, M. and Zhang, J., "Learning cooperative grasping with the graph representation of a state-action space," *Robotics and Autonomous Systems*, Vol. 38, No. 3-4. pp.183-195, 2002.

Fernandez, F. and Borrajo, D., "A reinforcement learning algorithm in cooperative multi-robot domains," *Journal of Intelligent and Robotic Systems*, Vol. 43, No. 2-4, pp. 161-174, 2005.

Frontoni, E. and Zingaretti, P., "A vision based algorithm for active robot localization," *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation,* pp. 347-352, June, Espoo, Finland, 2005.

Gerkey, B.P. and Mataric, M.J., "Pusher-watcher: an approach to fault-tolerant tightly-coupled robot coordination," *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, pp. 464-469, 2002a.

Gerkey, B.P., and Mataric, M.J., "Sold!: auction methods for multirobot coordination," *IEEE Trans. on Robotics and Automation*, Vol. 18, No. 5, pp. 758-768, 2002b.

Gerkey, B.P., and Mataric, M.J., "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, Vol. 23, No. 9, pp. 939-954, 2004.

Goldberg, D. and Mataric, M.J., "Design and evaluation of robust behavior-based controllers," (Chapter 11), Editor: Balch, T. and Parker, L.E., *Robot Teams: From Diversity to Polymorphism*, AK Peters, Ltd., Natick, Massachusetts, 2002.

Gonzalez, R.C. and Woods, R.E., *Digital Image Processing*, Prentice Hall, Upper Saddle River, NJ, 2002.

Gustafson, S. and Gustafson, D.A., "Issues in the scaling of multi-robot systems for general problem solving," *Autonomous Robots*, Vol. 20, No. 2, pp. 125-136, 2006.

Gopalakrishnan, A., Greene, S. and Sekmen A., "Vision-based mobile robot learning and navigation," *Proceedings of IEEE International Workshop on Robot and Human Interactive Communication*, Nashville, TN, pp. 48-53, 2005.

Hajjdiab, H. and Laganiere, R., "Vision-based multi-robot simultaneous localization and mapping," *Proceedings of 1st Canadian Conference on Computer and Robot Vision*, pp. 155-162, May, London, ON, Canada, 2004.

Hastie, T., Tibshirani, R. and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer-Verlag, New York, 2001.

Horstmann, C.S. and Cornell, G., *Core Java 2, Volume II – Advanced Features*, Seventh Edition, Pearson Education, Inc., Upper Saddle River, NJ, 2005.

Hu, J. and Wellman, M.P., "Multiagent reinforcement learning: theoretical framework and an algorithm," *Proc. Of 15th International Conf. on Machine Learning*, San Francisco, CA, pp. 242-250, 1998.

Huntsberger, T., Pirjanian, P., Trebi-Ollennu, A., Nayar, H.D., Aghazarian, H, Ganino, A.J., Garrett, M., Joshi, S.S. and Schenker, P.S., "Campout: a control architecture for tightly coupled coordination of multi-robot systems for planetary surface exploration," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 33, No. 5, pp. 550-559, 2003.

Ishiwaka, Y., Sato, T. and Kakazu, Y., "An approach to the pursuit problem on a heterogeneous multiagent system using reinforcement learning," *Robotics and Autonomous Systems*, Vol. 43, No. 4, pp. 245-256, 2003.

Inoue, Y., Tohge, T. and Iba, H., "Object transportation by two humanoid robots using cooperative learning," *Proc. of the 2004 Congress on Evolutionary Computation*, Portland, OR, pp. 1201-1207, 2004.

Ito, K. and Gofuku, A., "Hybrid autonomous control for multi mobile robots," *Advanced Robotics*, Vol. 18, No. 1, pp. 83-99, 2004.

Jones, C. and Mataric, M.J., "Automatic synthesis of communication-based coordinated multi-robot systems," *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, pp. 381-387, 2004a.

Jones, C. and Mataric, M.J., "Utilizing internal state in multi-robot coordination tasks," *Proceedings of Nineteenth National Conference on Artificial Intelligence (AAAI-2004)*, San Jose, CA, pp. 958-959, 2004b.

Kalman, R.E., "A new approach to linear filtering and prediction problems," *Transactions of ASME - Journal of Basic Engineering*, Vol. 82 (Series D), pp. 35-45, 1960.

Kapetanakis, S. and Kudenko, D., "Reinforcement learning of coordination in cooperative multi-agent systems," *Proceedings of Eighteenth national conference on Artificial intelligence*, Edmonton, Alberta, Canada, pp. 326-331, 2002.

Karray, F.O. and de Silva, C.W., *Soft Computing and Intelligent Systems Design*, Addison Wesley/Pearson, New York, NY, 2004.

Kato, K., Ishiguro, H. and Barth, M., "Identification and localization of multiple robots using omnidirectional vision sensors," *Electronics and Communications in Japan, Part II: Electronic*, Vol. 86, No. 6, pp. 1270-1278, 2003.

Kawakami, K., Ohkura, K. and Ueda, K., "Reinforcement learning approach to cooperation problem in a homogeneous robot group," *IEEE International Symposium on Industrial Electronics*, Pusan, Korea, pp. 423-428, 2001.

Konidaris, G.D. and Hayes, G.M., "An architecture for behavior-based reinforcement learning," *Adaptive Behavior*, Vol. 13, No. 1, pp. 5-32, 2005.

Kovac, K., Zivkovic, I. and Basic, B.D., "Simulation of multi-robot reinforcement learning for box-pushing problem," *Proceedings of the Mediterranean Electrotechnical Conference – MELECON*, Dubrovnik, Croatia, pp. 603-606, 2004.

Kube, C.R. and Bonabeau, E., "Cooperative transport by ants and robots," *Robotics and Autonomous Systems*, Vol. 30, No. 1, pp. 85-101, 2000.

Kumar, M. and Garg, D.P., "Sensor-based estimation and control of forces and moments in multiple cooperative robots," *Journal of Dynamic Systems, Measurement, and Control, Transactions of the ASME*, Vol. 126, No. 2, pp. 276-283, 2004.

Kume, Y., Hirata, Y., Wang, Z. and Kosuge, K., "Decentralized control of multiple mobile manipulators handling a single object in coordination," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, EPFL, Lausanne, Switzerland, pp. 2758-2763, 2002.

Lerman, K., Jones, K., Galstyan, A. and Mataric, M.J., "Analysis of dynamic task allocation in multi-robot systems," *International Journal of Robotics Research*, Vol. 25, No. 3, p 225-241, 2006.

Liberty, J., *Teach Yourself C++ in 21 Days*, Third Edition, Sams Publishing, Indianapolis IN, 1999.

Littman, M.L., "Markov games as a framework for multi-agent reinforcement learning," *Proc. of the Eleventh International Conference on Machine Learning (ML-94)*, New Brunswick, NJ, pp.157-163, 1994.

Littman, M.L., "Value-function reinforcement learning in Markov games," *Journal of Cognitive Systems Research*, Vol. 2001, No. 1, pp.1-12, 2001a.

Littman, M.L., "Friend-or-Foe Q-learning in general-sum games," *Proc. 18th International Conf. on Machine Learning*, San Francisco, CA, pp. 322-328, 2001b.

Liu, J. and Wu, J., *Multi-agent Robotic Systems*, CRC Press, Boca Raton, FL, 2001.

Marshall, J.A., Fung, T., Broucke, M.E., D'eleuterio, G.M.T., Francis, B.A., "Experiments in multirobot coordination," *Robotics and Autonomous Systems*, Vol. 54, No. 3, pp. 265-275, 2006.

Martinson, E., Stoytchev, A. and Arkin, R.C., "Robot behavioral selection using Q-learning," *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems*, EPFL, Lausanne, Switzerland, pp. 970-977, 2002.

Martison, E. and Arkin, R.C., "Learning to role-switch in multi-robot systems," *Proc. of IEEE International Conference on Robotics & Automation*, Taibei, Taiwan, pp. 2727-2734, 2003.

Mataric, M.J., Nillsson, M., and Simsarian, K.T., "Cooperative multi-robot object-pushing," *Proc. of IEEE/RSJ Int. Conf. on Human Robot Interaction and Cooperative Robots,* Pittsburgh, PA, pp. 556-561, 1995.

Mataric, M.J., "Reinforcement learning in the multi-robot domain," *Autonomous Robots*, Vol. 1997, No. 4, pp.73-83, 1997.

Maurin, B., Masoud, O. and Papanikolopoulos, N. P., "Tracking all traffic: computer vision algorithms for monitoring vehicles, individuals, and crowds," *IEEE Robotics and Automation Magazine*, Vol. 12, No. 1, pp. 29-36, 2005.

Mitchell, T.M., *Machine Learning*, McGraw-Hill Companies, Inc., New York, NY, 1997.

Miyata, N., Ota, J., Arai T. and Asama, H. "Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment," *IEEE Trans. on Robotics and Automation*, Vol. 18, No. 5, pp. 769-780, 2002.

Murarka, A., Modayil, J. and Kuipers, B., "Building local safety maps for a wheelchair robot using vision and lasers," *Proceedings of the 3rd Canadian Conference on Computer and Robot Vision*, Quebec City, QC, Canada, pp. 25-32, 2006.

Nolfi, S. and Floreano, D., "Learning and evolution," *Autonomous Robots*, Vol. 7, No. 1, pp. 89-113, 1999.

Ogata, K., *Modern Control Engineering*, Third Edition, Prentice Hall, Inc., Upper Saddle River, NJ, 1998.

Ortin, D., Neira, J. and Montiel, J.M.M., "Relocation using laser and vision," *Proceedings of IEEE International Conference on Robotics and Automation*, New Orleans, LA, pp. 1505-1510, 2004.

Panait, L. and Luke, S., "Cooperative multi-agent learning: the state of the art," *Autonomous Agents and Multi-Agent Systems*, Vol. 11, No. 3, pp. 387-434, 2005.

Park, K., Kim, Y. and Kim, J., "Modular Q-learning based multi-agent cooperation for robot soccer," *Robotics and Autonomous Systems*, Vol. 35, No. 2, pp. 109-122, 2001.

Parker, L.E., "ALLIANCE: an architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, Vol. 14, No. 2, pp. 220-240, 1998.

Parker, L.E., "Lifelong adaptation in heterogeneous multi-robot teams: response to continual variation in individual robot performance," *Autonomous Robots*, Vol. 2000, No. 8, pp. 239-267, 2000a.

Parker, L.E., "Current state of the art in distributed autonomous mobile robotics," *Distributed Autonomous Robotic Systems 4*, Springer-Verlag, Tokyo, pp. 3-12, 2000b.

Parker, L.E., Touzet, C. and Fernandez, F., "Techniques for learning in multirobot teams," (Chapter 7), Editor: Balch, T. and Parker, L.E., *Robot Teams: From Diversity to Polymorphism*, AK Peters, Ltd., Natick, Massachusetts, 2002.

Pereira, G., Campos, M. and Kumar, V., "Decentralized algorithms for multi-robot manipulation via caging," *The International Journal of Robotics Research*, Vol. 23, No. 7-8, pp. 783-795, 2004.

Pham, D.T., and Awadalla, M.H., "Neuro-fuzzy based adaptive co-operative mobile robots," *Proceedings of the 2002 28th Annual Conference of the IEEE Industrial Electronics Society*, Sevilla, Spain, pp. 2962-2967, 2002.

Pimentel, B.S., Pereira, G.A.S. and Campos, M.M.F.M., "On the development of cooperative behavior-based mobile manipulators," *Proceedings of the International Conference on Autonomous Agents*, Bologna, Italy, pp. 234-239, 2002.

Rabie, T., Abdulhai, B., Shalaby, A. and El-Rabbany, A., "Mobile active-vision traffic surveillance system for urban networks," *Computer-Aided Civil and Infrastructure Engineering*, Vol. 20, No. 4, pp. 231-241, 2005.

Rimon, E. and Koditschek, D.E., "Exact robot navigation using artificial potential functions," *IEEE Trans. on Robotics and Automation*, Vol. 8, No. 5, pp. 501-518, 1992.

Rocha, R., Dias, J. and Carvalho, A., "Cooperative multi-robot systems: A study of vision-based 3-D mapping using information theory," *Robotics and Autonomous Systems*, Vol. 53, No. 3-4, p 282-311, 2005.

Rosen, K.H., *Discrete Mathematics and Its Applications*, Fifth Edition, McGraw-Hill Companies, Inc., New York, NY, 2003.

Ross, S.M., *Introduction to Probability Models,* 8th Edition, Elsevier, Singapore, 2006.

Rus, D., Donald, B., and Jennings, J., "Moving furniture with teams of autonomous robots," *Proc. of IEEE/RSJ International Conference on Human Robot Interaction and Cooperative Robots*, Pittsburgh, PA, pp. 235-242, 1995.

Russell S. and Norvig P., *Artificial Intelligence: A Modern Approach*, Second Edition, Pearson Education, Inc., Upper Saddle River, NJ, 2003.

Sanchez-Pena, R. and Sznaier, M., *Robust Systems Theory and Applications*, John Wiley & Sons, Inc., New York, NY, 1998.

Sardag, A. and Akin, H.L., "ARKAQ-Learning: Autonomous state space segmentation and policy generation," *Lecture Notes in Computer Science*, Vol. 3733, LNCS, pp. 512-523, Springer Verlag, New York, NY, 2005.

Schenker, P.S., Huntsberger, T.L., Pirjanian, P., Trebi-Ollennu, A., Das, H., Joshi, S., Aghazarian, H., Ganino, A.J., Kennedy, B.A. and Garrett, M.S, "Robot work crews for planetary outposts: Close cooperation and coordination of multiple mobile robots," *Proceedings of SPIE - The International Society for Optical Engineering*, Boston, MA, pp. 210-220, 2000.

Schenker, P.L., Huntsberger, T., Pirjanian, P., Baumgartner, E.T. and Tunstel, E., "Planetary rover developments supporting MARs exploration, sample return and future human-robotic colonization," *Autonomous Robots*, Vol. 14, No. 2-3, pp. 103-126, 2003.

Sen, S., "Evolution and learning in multiagent systems," *International Journal of Human-Computer Studies*, Vol. 48, No. 1, pp. 1-7, 1998.

Shoham, Y., Powers, R. and Grenager, T., "Multi-agent reinforcement learning: a critical survey," *Technical report, Stanford University*, Stanford, CA, 2003.

Siegwart, R. and Nourbakhsh, I.R., *Introduction to Autonomous Mobile Robots*, The MIT Press, Cambridge, MA, 2004.

Siyal, M.Y., "A novel image processing based approach for real-time road traffic applications," *Proceedings of 2004 IEEE Region 10 Conference: Analog and Digital Techniques in Electrical Engineering*, Chiang Mai, Thailand, pp. 447-450, 2004.

Spong, M.W., Hutchinson, S., and Vidyasagar, M., *Robot Modeling and Control*, John Wiley & Sons, Inc., Hoboken, NJ, 2006.

Stone, P. and Veloso, M., "Layered approach to learning client behaviors in the ROBOCUP soccer server," *Applied Artificial Intelligence*, Vol. 12, No. 203, pp. 165-188, 1998.

Stone, P. and Veloso, M., "Multiagent systems: a survey from a machine learning perspective," *Autonomous Robots*, Vol. 8, No. 3, pp. 345-383, 2000.

Stone, P., Sridharan, M., Stronger, D., et al., "From pixels to multi-robot decision-making: A study in uncertainty," *Robotics and Autonomous Systems*, Vol. 54, No. 11, pp. 933-943, 2006.

Stroupe, A., Huntsberger, T., Okon, A, Aghazarian, H. and Robinson, M, "Behavior-based multi-robot collaboration for autonomous construction tasks," *Proceedings of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmont, Alberta, Canada, pp. 1989-1994, 2005.

Stroupe, A., Okon, A., Robinson, M., Huntsberger, T., Aghazarian, H. and Baumgartner, E., "Sustainable cooperative robotic technologies for human and robotic outpost infrastructure construction and maintenance," *Autonomous Robots*, Vol. 20, No. 2, pp. 113-123, 2006.

Sugar, T.G., Desai, J.P. Kumar, V. and Ostrowski, J.P., "Coordination of multiple mobile manipulators," *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, Seoul, Korea, pp. 3022-3027, 2001.

Sugar, T.G. and Kumar, V., "Control of cooperating mobile manipulators," *IEEE Trans. on Robotics and Automation,* Vol.18, No. 1, pp. 94-103, 2002.

Sutton, R.S. and Barto, A.G., *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge, MA, 1998.

Svinin, M.M., Kojiama, F., Katada, Y. and Ueda, K., "Initial experiments on reinforcement learning control of cooperative manipulations," *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 2000.

Takahashi, S. and Ghosh, B.K., "Parameter identification of a moving object based on sensor fusion," *Proceedings of IEEE International Conference on Mechatronics and Automation*, Niagara Falls, ON, Canada, pp. 171-176, 2005.

Tangamchit, P., Dolan, J.M. and Khosla, P.K., "Crucial factors affecting decentralized multirobot learning in an object manipulation task," *Proceedings of IEEE International Conference on Intelligent Robots and System*, Las Vegas, NV, pp. 2023-2028, 2003.

Taylor, M.E. and Stone, P., "Behavior transfer for value-function-based reinforcement learning," *Proceedings of the 4th International Conference on Autonomous Agents and Multi agent Systems*, Utrecht, Netherlands, pp. 201-207, 2005.

Thrun, S., Burgard, W. and Fox, D., *Probabilistic Robotics*, The MIT Press, Cambridge, MA, 2005.

Tomono, M., "Environment modeling by a mobile robot with a laser range finder and a monocular," *Proceedings of 2005 IEEE Workshop on Advanced Robotics and its Social Impacts*, Nagoya, Japan, pp. 133-138, 2005.

Trebi-Ollennu, A., Nayar, H.D., Aghazarian, H., Ganino, A., Pirjanian, P., Kennedy, B., Huntsberger, T. and Schenker, P., "Mars rover pair cooperatively transporting a long payload," *Proceedings of IEEE International Conference on Robotics and Automation*, Washington, DC, pp. 3136-3141, 2002.

Veeraraghavan, H., Masoud, O. and Papanikolopoulos, N.P., "Computer vision algorithms for intersection monitoring," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 4, No. 2, pp. 78-89, 2003.

Wang, Y. and de Silva, C.W., "An object transportation system with multiple robots and machine learning," *Proc. of American Control Conference (ACC 2005)*, Portland, OR, pp.1371-1376, 2005a.

Wang, Y. and de Silva, C.W., "A fast computer vision algorithm for multi-robot cooperative control," *Proceedings of International Symposium on Collaborative Research in Applied Science*, pp. 189-194, Vancouver, BC, Canada, October, 2005b.

Wang, Y. and de Silva, C.W., "Extend single-agent reinforcement learning approach to a multi-robot cooperative task in an unknown dynamic environment," *Proceedings of IEEE 2006 International Joint Conference on Neural Networks (IJCNN)*, Vancouver, BC, Canada, pp. 10098-10104, 2006a.

Wang, Y. and de Silva, C.W., "Multi-robot box-pushing: single-agent Q-Learning vs. team Q-Learning," *Proceedings of 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Beijing, China, pp. 3694-3699, 2006b.

Wang, Y. and de Silva, C.W., "Cooperative transportation by multiple robots with machine Learning, " *Proceedings of 2006 IEEE Congress on Evolutionary Computation (CEC),* Vancouver, BC, Canada, pp. 10407-10413, 2006c.

Wang, Y. and de Silva, C.W, "A fast and robust algorithm for color-blob tracking in multi-robot coordinated tasks," *International Journal of Information Acquisition*, Vol. 3, No. 3, pp. 191-200, 2006d.

Wang, Y. and de Silva, C.W, "A machine learning approach to multi-robot coordination," *Engineering Applications of Artificial Intelligence (Elsevier),* 2007a (In Press).

Wang, Y. and de Silva, C.W, "A modified Q-learning algorithm for multi-robot decision-making," *Proceedings of ASME International Mechanical Engineering Congress and Exposition (IMECE 2007)*, Seattle, WA, 2007b (In Press).

Wang, Y. and de Silva, C.W, "Assess team Q-learning algorithm in a purely cooperative multi-robot task," *Proceedings of ASME International Mechanical Engineering Congress and Exposition (IMECE 2007)*, Seattle, WA, 2007c (In Press).

Wang, Z., Kumar, V., Hirata, Y. and Kosuge, K., "A strategy and a fast testing algorithm for object caging by multiple cooperative robots," *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, pp. 2275-2280, 2003a.

Wang Z., Nakano, E. and Takahashi, T., "Solving function distribution and behavior design problem for cooperative object handling by multiple mobile robots," *IEEE Trans. On Systems, Man, and Cybernetics – Part A: Systems and Humans*, Vol. 33, No. 5, pp. 537-549, 2003b.

Wang Z., Hirata, Y. and Kosuge, K., "Control multiple mobile robots for object caging and manipulation," *Proceedings of the 2003 IEEE/RSJ International Conference On Intelligent Robots and Systems*, Las Vegas, Nevada, pp. 1751-1756, 2003c.

Wang, Z., Takano, Y., Hirata, Y., and Kosuge, K., "A pushing leader based decentralized control method for cooperative object transportation," *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, pp. 1035-1040, 2004a.

Wang, Z., Hirata, Y. and Kosuge, K., "Control a rigid caging formation for cooperative object transportation by multiple mobile robots," *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, New Orleans, LA, pp. 1580-1585, 2004b.

Wang, Z., Hirata, Y., and Kosuge, K., "An algorithm for testing object caging condition by multiple mobile robots," *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Edmonton, Alberta, Canada, pp. 2664-2669, 2005.

Weiss, G., *Multiagent Systems*, The MIT Press, Cambridge, MA, 1999.

Wooldridge, M., *An Introduction to Multiagent Systems*, John Wiley & Sons, LTD, Hoboken, NJ, 2002.

Yamada, S. and Saito, J., "Adaptive action selection without explicit communication for multi-robot box-pushing," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 31, No. 3, pp. 398-404, 2001.

Yamashita, A., Arai, T., Ota, J. and Asama, H., "Motion planning of multiple mobile robots for cooperative manipulation and transportation," *IEEE Trans. On Robotics and Automation*, Vol. 19, No. 2, pp. 223-237, 2003.

Yang, E., and Gu, D., "Multiagent reinforcement learning for multi-robot systems: a survey," *Technical Report, http://robotics.usc.edu/~maja/teaching/cs584/papers/yang04multiagent.pdf*, 2004.

Zaerpoora, A., Ahmadabadia, M.N., Barunia, M.R. and Wang, Z., "Distributed object transportation on a desired path based on constrain and move strategy," *Robotics and Autonomous Systems*, Vol. 50, No. 2-3, pp. 115-128, 2005.

Zlot, R. and Stenz, A., "Market-based multirobot coordination for complex tasks," *The International Journal of Robotics Research*, Vol. 25, No. 1, pp. 73-101, 2006.

Appendix

JAVA Documents of the Developed System

1. Summary

Package SQKF

Interface Summary	
envServerInterface	The interface for remote product objects.
robotServerInterface	The interface for remote product objects.

Class Summary	
robot	
robotServer	
box	
env	
envServer	This is the implementation class for the remote product objects.
goal	
main	
objPose	
obstacle	
painter	
point	
robotData	

2. Class box

SQKF
Class box

```
java.lang.Object
  └ SQKF.box
```

```
public class box
extends java.lang.Object
```

Constructor Summary

box(env e, java.util.Vector oGroup, objPose p, goal g)
Creates a new instance of box

Method Summary

void	**boxResponse**()
objPose	**getBoxPose**()
double	**getDetectionRadius**()
double	**getGlobalReward**()
int	**getHeight**()
int	**getWidth**()
void	**setBoxPose**(objPose bPose)
void	**setEnvServer**(envServer s)

Constructor Details

1) box

```
public box(env e,
           java.util.Vector oGroup,
           objPose p,
           goal g)
```

Creates a new instance of box

Method Details

2) setEnvServer

```
public void setEnvServer(envServer s)
```

3) getBoxPose

```
public objPose getBoxPose()
```

4) setBoxPose

```
public void setBoxPose(objPose bPose)
```

5) getWidth

```
public int getWidth()
```

6) getHeight

```
public int getHeight()
```

---

7) getDetectionRadius

```
public double getDetectionRadius()
```

---

8) boxResponse

```
public void boxResponse()
```

---

9) getGlobalReward

```
public double getGlobalReward()
```

3. Class env

SQKF
Class env

```
java.lang.Object
  └ SQKF.env
```

```
public class env
extends java.lang.Object
```

Field Summary

double	**height**
double	**width**

Constructor Summary

env(double wid, double hei)

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Details

10) width

```
public double width
```

11) height

```
public double height
```

Constructor Details

12) env

```
public env(double wid,
           double hei)
```

4. Class envServer

SQKF

Class envServer

```
java.lang.Object
  └java.rmi.server.RemoteObject
      └java.rmi.server.RemoteServer
          └java.rmi.server.UnicastRemoteObject
              └SQKF.envServer
```

All Implemented Interfaces:

java.io.Serializable, java.rmi.Remote, envServerInterface

```
public class envServer
extends java.rmi.server.UnicastRemoteObject
implements envServerInterface
```

This is the implementation class for the remote product objects.

See Also:

Serialized Form

Field Summary

Fields inherited from class java.rmi.server.RemoteObject

ref

Constructor Summary

envServer(goal g, box b, java.util.Vector oGroup) Creates a new instance of envServer

Method Summary

int	**addRobot**(robotData r)

boolean	**allRobotReady**()
void	**awakeAllRobots**()
void	**awakeRobot**(int id)
void	**clearAllRobotReady**()
point	**getBoxDimension**()
objPose	**getBoxPose**()
double	**getDetectionRadius**()
double	**getGlobalReward**()
point	**getGoal**()
java.util.Vector	**getObstacleGroup**()
java.util.Vector	**getRobotDataList**()
java.util.Vector	**getRobotPoseList**()
java.util.Vector	**getRobotStatusList**()
java.util.Vector	**getRobotUrlList**()
void	**setBoxPose**(objPose bPose)
void	**setRobotAction**(int id, int action)
void	**setRobotPose**(int id, objPose p)
void	**setRobotReady**(int id)
void	**setRobotStatus**(int id, boolean s)

Constructor Details

13) envServer

public **envServer**(goal g,
box b,
java.util.Vector oGroup)

throws java.rmi.RemoteException

Creates a new instance of envServer

Throws:

java.rmi.RemoteException

Method Details

14) addRobot

```
public int addRobot(robotData r)
             throws java.rmi.RemoteException
```

Specified by:

addRobot in interface envServerInterface

Throws:

java.rmi.RemoteException

---

15) getGoal

```
public point getGoal()
             throws java.rmi.RemoteException
```

Specified by:

getGoal in interface envServerInterface

Throws:

java.rmi.RemoteException

---

16) getObstacleGroup

```
public java.util.Vector getObstacleGroup()
                                  throws java.rmi.RemoteException
```

Specified by:

getObstacleGroup in interface envServerInterface

Throws:

java.rmi.RemoteException

---

17) getBoxPose

```
public objPose getBoxPose()
                  throws java.rmi.RemoteException
```

Specified by:

getBoxPose in interface envServerInterface

Throws:

java.rmi.RemoteException

---

18) getBoxDimension

```
public point getBoxDimension()
                    throws java.rmi.RemoteException
```

Specified by:

getBoxDimension in interface envServerInterface

Throws:

java.rmi.RemoteException

---

19) getRobotUrlList

```
public java.util.Vector getRobotUrlList()
                                  throws java.rmi.RemoteException
```

Specified by:

getRobotUrlList in interface envServerInterface

Throws:
java.rmi.RemoteException

---

20) getRobotStatusList

```
public java.util.Vector getRobotStatusList()
                                    throws java.rmi.RemoteException
```

Specified by:

getRobotStatusList in interface envServerInterface

Throws:

java.rmi.RemoteException

---

21) getGlobalReward

```
public double getGlobalReward()
                        throws java.rmi.RemoteException
```

Specified by:

getGlobalReward in interface envServerInterface

Throws:

java.rmi.RemoteException

---

22) getDetectionRadius

```
public double getDetectionRadius()
                          throws java.rmi.RemoteException
```

Specified by:

getDetectionRadius in interface envServerInterface

Throws:

java.rmi.RemoteException

---

23) getRobotPoseList

```
public java.util.Vector getRobotPoseList()
                                  throws java.rmi.RemoteException
```

Specified by:

getRobotPoseList in interface envServerInterface

Throws:

java.rmi.RemoteException

---

24) getRobotDataList

```
public java.util.Vector getRobotDataList()
```

---

25) awakeAllRobots

```
public void awakeAllRobots()
```

---

26) awakeRobot

```
public void awakeRobot(int id)
```

---

27) allRobotReady

```
public boolean allRobotReady()
```

---

28) setRobotAction

```
public void setRobotAction(int id,
                           int action)
                    throws java.rmi.RemoteException
```

Specified by:

setRobotAction in interface envServerInterface

Throws:

java.rmi.RemoteException

---

29) setRobotReady

```
public void setRobotReady(int id)
                    throws java.rmi.RemoteException
```

Specified by:

setRobotReady in interface envServerInterface

Throws:

java.rmi.RemoteException

---

30) setBoxPose

```
public void setBoxPose(objPose bPose)
                 throws java.rmi.RemoteException
```

Specified by:

setBoxPose in interface envServerInterface

Throws:

java.rmi.RemoteException

---

31) clearAllRobotReady

```
public void clearAllRobotReady()
                        throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

32) setRobotPose

```
public void setRobotPose(int id,
                         objPose p)
                  throws java.rmi.RemoteException
```

Specified by:

setRobotPose in interface envServerInterface

Throws:

java.rmi.RemoteException

---

33) setRobotStatus

```
public void setRobotStatus(int id,
                           boolean s)
                    throws java.rmi.RemoteException
```

Specified by:

setRobotStatus in interface envServerInterface

Throws:

java.rmi.RemoteException

5. Interface envServer

SQKF
Interface envServerInterface

All Superinterfaces:

java.rmi.Remote

All Known Implementing Classes:

envServer

```
public interface envServerInterface
extends java.rmi.Remote
```

The interface for remote product objects.

Method Summary

int	**addRobot**(robotData r)
point	**getBoxDimension**()
objPose	**getBoxPose**()
double	**getDetectionRadius**()
double	**getGlobalReward**()
point	**getGoal**()
java.util.Vector	**getObstacleGroup**()
java.util.Vector	**getRobotPoseList**()
java.util.Vector	**getRobotStatusList**()
java.util.Vector	**getRobotUrlList**()
void	**setBoxPose**(objPose boxPose)
void	**setRobotAction**(int id, int action)
void	**setRobotPose**(int id, objPose p)
void	**setRobotReady**(int id)
void	**setRobotStatus**(int id, boolean s)

Method Details

34) addRobot

```
int addRobot(robotData r)
             throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

35) getGoal

```
point getGoal()
              throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

36) getObstacleGroup

```
java.util.Vector getObstacleGroup()
                                    throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

37) getBoxPose

```
objPose getBoxPose()
                    throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

38) getBoxDimension

```
point getBoxDimension()
                      throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

39) getRobotUrlList

```
java.util.Vector getRobotUrlList()
                                  throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

40) getGlobalReward

```
double getGlobalReward()
                       throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

41) getDetectionRadius

```
double getDetectionRadius()
                          throws java.rmi.RemoteException
```

Throws:

java.rmi.RemoteException

---

42) getRobotPoseList

```
java.util.Vector getRobotPoseList()
                                   throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

43) getRobotStatusList

```
java.util.Vector getRobotStatusList()
                                    throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

44) setRobotAction

```
void setRobotAction(int id,
                    int action)
                    throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

45) setRobotReady

```
void setRobotReady(int id)
                   throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

46) setRobotPose

```
void setRobotPose(int id,
                  objPose p)
                  throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

47) setRobotStatus

```
void setRobotStatus(int id,
                    boolean s)
                    throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

---

48) setBoxPose

```
void setBoxPose(objPose boxPose)
                throws java.rmi.RemoteException
```

Throws:
java.rmi.RemoteException

6. Class goal

SQKF
Class goal

```
java.lang.Object
  └ SQKF.goal
```

```
public class goal
extends java.lang.Object
```

Field Summary

point	**position**

Constructor Summary

goal(double x, double y)
Creates a new instance of goal

Method Summary

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Details

49) position

public point **position**

Constructor Details

50) goal

public **goal**(double x,
double y)
Creates a new instance of goal

7. Class main

SQKF
Class main

java.lang.Object
└ java.awt.Component
└ java.awt.Container
└ javax.swing.JComponent
└ javax.swing.JPanel
└ **SQKF.main**

All Implemented Interfaces:

java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible

---

public class **main**
extends javax.swing.JPanel

See Also:

Field Summary

Constructor Summary

main()

Method Summary

void	**clearCanvas**()
static void	**main**(java.lang.String[] args)
void	**paintComponent**(java.awt.Graphics g)
static double	**randomNum**(double a1, double a2)

Constructor Details

51) main

public **main**()

Method Details

52) paintComponent

public void **paintComponent**(java.awt.Graphics g)

Overrides:

paintComponent in class javax.swing.JComponent

---

53) clearCanvas

public void **clearCanvas**()

---

54) main

public static void **main**(java.lang.String[] args)
throws java.lang.Exception

Throws:

java.lang.Exception

---

55) randomNum

public static double **randomNum**(double a1,
double a2)

8. Class objPose

SQKF
Class objPose

```
java.lang.Object
  └ SQKF.objPose
```

All Implemented Interfaces:

java.io.Serializable

```
public class objPose
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Field Summary

double	**angle**
point	**center** Creates a new instance of objPose

Constructor Summary

objPose()
Creates a new instance of objPose

Field Details

56) center

```
public point center
```

Creates a new instance of objPose

---

57) angle

```
public double angle
```

Constructor Details

58) objPose

```
public objPose()
```

Creates a new instance of objPose

9. Class obstacle

SQKF
Class obstacle

```
java.lang.Object
  └ SQKF.obstacle
```

All Implemented Interfaces:

java.io.Serializable

```
public class obstacle
extends java.lang.Object
implements java.io.Serializable
```

See Also:

Serialized Form

Field Summary

point	**position**
double	**r**

Constructor Summary

obstacle(point p, double r1)

Creates a new instance of obstacle

Field Details

59) position

```
public point position
```

---

60) r

```
public double r
```

Constructor Details

61) obstacle

```
public obstacle(point p,
                double r1)
```

Creates a new instance of obstacle

10. Class robotServer

SQKF

Class robotServer

```
java.lang.Object
  └java.rmi.server.RemoteObject
      └java.rmi.server.RemoteServer
          └java.rmi.server.UnicastRemoteObject
              └SQKF.robotServer
```

All Implemented Interfaces:

java.io.Serializable, java.rmi.Remote, robotServerInterface

```
public class robotServer
```

```
extends java.rmi.server.UnicastRemoteObject
implements robotServerInterface
```

Field Summary

Fields inherited from class java.rmi.server.RemoteObject

`ref`

Constructor Summary

`robotServer``(robot r)`

Creates a new instance of robotServer

Method Summary

boolean	**`awakeMe`**`()`
objPose	**`getPose`**`()`
boolean	**`isSleeping`**`()`
void	**`receiveRobotActionList`**`(java.util.Vector actionList, boolean isInitialAction)`

Constructor Details

62) robotServer

```
public robotServer(robot r)
            throws java.rmi.RemoteException
```

Creates a new instance of robotServer

Throws:

`java.rmi.RemoteException`

Method Details

63) receiveRobotActionList

```
public void receiveRobotActionList(java.util.Vector actionList,
                                   boolean isInitialAction)
                            throws java.rmi.RemoteException
```

Specified by:

`receiveRobotActionList` in interface `robotServerInterface`

Throws:

`java.rmi.RemoteException`

---

64) getPose

```
public objPose getPose()
                throws java.rmi.RemoteException
```

Specified by:

getPose in interface robotServerInterface

Throws:

java.rmi.RemoteException

---

65) awakeMe

```
public boolean awakeMe()
                throws java.rmi.RemoteException
```

Specified by:

awakeMe in interface robotServerInterface

Throws:

java.rmi.RemoteException

---

66) isSleeping

```
public boolean isSleeping()
                    throws java.rmi.RemoteException
```

Specified by:

isSleeping in interface robotServerInterface

Throws:

java.rmi.RemoteException

11. Class robot

SQKF
Class robot

```
java.lang.Object
  └java.lang.Thread
      └SQKF.robot
```

All Implemented Interfaces:

java.lang.Runnable

```
public class robot
extends java.lang.Thread
```

Nested Class Summary

Nested classes/interfaces inherited from class java.lang.Thread
java.lang.Thread.State, java.lang.Thread.UncaughtExceptionHandler

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

robot(double fValue, java.lang.String sUrl, java.lang.String mUrl, objPose mPose)

Creates a new instance of robot

Method Summary

boolean	**amSleeping**()
void	**awakeMe**()
void	**becomeWaiting**(int timeout)
void	**frame2To1**(point p1, point p2, point origin, double theta)
objPose	**getPose**()
static void	**main**(java.lang.String[] args)
void	**modiRobotActionList**(java.util.Vector actionList, boolean isInitialAction)
void	**run**()

Constructor Details

67) robot

```
public robot(double fValue,
             java.lang.String sUrl,
             java.lang.String mUrl,
             objPose mPose)
```

Creates a new instance of robot

Method Details

68) main

```
public static void main(java.lang.String[] args)
```

69) run

```
public void run()
```

Specified by:

run in interface java.lang.Runnable

Overrides:

run in class java.lang.Thread

70) becomeWaiting

```
public void becomeWaiting(int timeout)
```

---

71) awakeMe

```
public void awakeMe()
```

---

72) amSleeping

```
public boolean amSleeping()
```

---

73) modiRobotActionList

```
public void modiRobotActionList(java.util.Vector actionList,
                                boolean isInitialAction)
```

---

74) frame2To1

```
public void frame2To1(point p1,
                      point p2,
                      point origin,
                      double theta)
```

75) getPose

```
public objPose getPose()
```