MACHINE LEARNING-BASED MULTI-ROBOT COOPERATIVE TRANSPORTATION OF OBJECTS

by

Pallege Gamini Dilupa Siriwardana

B.Sc., University of Peradeniya, Sri Lanka, 2003
B.Tech., The Open University of Sri Lanka, 2004

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

MASTER OF APPLIED SCIENCE

in

The Faculty of Graduate Studies

(Mechanical Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)

May 2009

Abstract

Multi robot cooperative transportation is an important research area in the multi robot domain. In the process of object transportation, several autonomous robots navigate cooperatively in either a static or a dynamic environment to transport an object to a goal location and orientation. The environment may consist of both fixed and removable obstacles and it will be subject to uncertainty and unforeseen changes within the environment. Furthermore, more than one robot may be required in a cooperative mode for handling heavy and large objects. These are some of the challenges addressed in the present thesis.

This thesis develops a machine learning approach and investigates relevant research issues for object transportation utilizing cooperative and autonomous multiple mobile robots. It makes significant original contributions in distributed multi robot coordination and self deterministic learning for robot decision making, and comes up with an optimal solution to the action selection conflicts of the robots in the cooperative system. This will help to improve the real time performance and robustness of the system. Also, the thesis develops a new method for object and obstacle identification in complex environments using a laser range finder, which is more realistic than the available methods. A new algorithm for object pose estimation algorithm is developed, enabling a robot to identify the objects and obstacles in a multi-robot environment by utilizing the laser range finder and color blob tracking.

The thesis develops a fully distributed hierarchical multi-robot architecture for enhanced coordination among robots in a dynamic and unknown environment. It strives to improve the real time performance and robustness. The system consists with three layers. By combining two popular artificial intelligence (AI) techniques such as learning and behavior based decision making, the developed architecture is expected to facilitate effective autonomous operation of cooperative multi-robot systems in a dynamically changing, unstructured, and unknown environment.

Machine learning, is integrated into the developed multi-robot system for decision making during the transportation under uncertainty and unforeseen changes within the environment. For this purpose the conventional algorithm of Q learning known as single-agent Q learning algorithm improved to form the “modified Q learning algorithm”, which provides efficient state identification and optimally resolves action selection conflicts in multi robot learning.

A C++ based simulation system and a physical multi-robot experimental system are developed in the laboratory to transport an object of interest to a goal location in a dynamic and unknown environment with a complex obstacle distribution. The approaches developed in this thesis are implemented in the prototype system and validated through computer simulation and experimentation. The results validate the developed techniques.

TABLE OF CONTENTS

List of Tables

List of Figures

Nomenclature

Notations

$S = \{s_1, s_2, \ldots, s_n\}$	Set of states in the environment
$A = \{a_1, a_2, \ldots, a_m\}$	Set of actions available to a robot
$T(s,a,s')$	Environment model, which decides the next environmental state s' when robot selects action a_i under current state s.
$\Pi(s)$	Probability distribution over the states
$T : S \times A \to \pi(s)$	Transition function or transition model
t	time
$R : S \times A \to \Re$	Reward function. This gives the immediate reward when robot takes action a_i under current state s
π	Policy with which a robot selects its actions
$R_{(s)}$	Received Reward
β	Discount factor, which varies from "0" to "1"
π^*	Optimal policy, which yields the highest expected sum of the discounted rewards
$U(s)$	Utility of a state
$\pi^*_{(s)}$	Utility of the subsequent state
ε	Maximum error allowed in the utility of any state
δ	Maximum change in the utility of any state in an iteration
U, U'	Vectors of the utilities for the states in s
$Q(s_i, a_j)$	An entry in the Q-table
η	Learning rate
r	Direct reward
τ	" Temperature " parameter (see Chapter 3)
ϕ	Empty set
Ψ	A set including all actions selected by the robots

θ	Orientation angle
$\sum F$	Net force applied by the robots
Γ	Net torque applied by the robots
c_f, c_t, c_d	Coefficients
Δt	Short period of time
D_{old}	Distance between the goal location and the object center before taking the pushing action
D_{new}	Distance between the goal location and the object center after taking the pushing action
T	Sampling time
Λ_i	Action set of the i^{th} robot
Υ	Set including the currently available actions

Abbreviations

MDP	Markov Decision Process
RL	Reinforcement Learning
MQLA	Modified Q Learning Algorithm
AI	Artificial Intelligence
LAN	Local Area Network
TCP/IP	Transmission Control Protocol/ Internet Protocol
MRS	Multi-Robot System
CCD	Charge Couple Device
GPS	Global Positioning System
FSA	Finite State Diagram
VLSI	Very Large Scale Integration
ACTS	Advanced Color Tracking System

Acknowledgements

A number of people contributed to this effort intellectually, personally and financially. First and foremost, I wish to thank Prof. Clarence W. de Silva, Director of the Industrial Automation Laboratory, in view of number of reasons. He was my research supervisor, and his dedication and commitment towards the student community is appreciated by all of us who were his students. Had it not been for Professor de Silva's considerable persuasive skills, boundless patience, great-heartedness, keen guidance and unbelievable human qualities; this particular research perhaps would never have come to fruition.

I take this opportunity to thank Dr. Ying Wang, The post-doctoral research scientist in the Industrial Automation Laboratory, who is a scholar of multi-robotic systems. Whenever I had a problem in multi robotic systems, he assisted me along the correct path and I am privileged and honored to have worked with him.

I must thank Mr. Howard Hu, a Research Assistant in the Industrial Automation laboratory, for his constant support and constructive remarks, especially in debugging the computer programs.

I wish to express my sincere gratitude to Prof. Mu Chiao and Dr. Farbod Khoshnoud of SOFTEK, who served as members of my research committee.

Furthermore, I wish to thank The Open University of Sri Lanka and the Asian development Bank funded Distance Education Modernization Project for the financial assistance in part. Major financial assistance, for my Research Assistantship, and for equipment and other resources for my research, was provided by research grants held by Prof. de Silva, notably: the National Sciences and Engineering Research Council (NSERC) of Canada Special Research Opportunity (SRO) and Discovery grants; Tier 1 Canada Research Chair (CRC); Canada Foundation for Innovation (CFI); and British Columbia Knowledge Development Fund (BCKDF). I also acknowledge the support extended by my Sri Lankan friends, Mr. Anjana Punchihewa, Mr. Eranda Harinath and Mr. Lalinda Weerasekara for their encouragements and support in different ways.

I take this opportunity to thank the members of the Industrial Automation Laboratory; especially Prof. Lalith Gamage, Mr. Tahir Khan, Mr. Mohammed Alrasheed, Ms. Madalina Wierzbicki, Mr. Roland Haoxiang Lang, Mr. Bhenam Razavi, Mr. Srinivas Raman, Mr. Guan-Lu Zhang, Mr. Arunasiri Liyanage, Mr. Ramon Campos and Prof. Qingshan Zeng for their assistance, guidance and encouragements throughout the research project. Last but not least, I wish to thank my family for their constant support, especially my wife Lelka Samanmalee, my daughter Viduni Siriwardana, my mother, my father and my sister. Without their unwavering support, tolerance of my idiosyncrasies and abiding confidence, this research never would have been possible.

Chapter 1

Introduction

The field of Multi-Robot Systems emerged as a major area of robotics research in the late 1980's. Since then, the robotics community has investigated a number of important issues of cooperative multiple mobile robots. Prior to that most research was concentrated on either single robots or distributed problem-solving systems that did not involve multi-robot cooperation. Multi-robot systems (MRSs) are becoming increasingly significant in industrial, commercial and scientific applications. The number of robots used in these applications is also increasing.

An MRS can be defined as a set of autonomous robots interacting in the same work environment in fairly complex ways to carry out a common task. It should be able to operate in dynamic environments, where uncertainty and unforeseen changes can arise due to the presence of moving robots, other agents and both stationary and moving obstacles. Such a system may consist of complex mobile robotic platforms equipped with sophisticated sensors and actuators, which may be required to execute complex tasks.

The advantages of a multi-robot system (MRS) over a single-robot system are indicated now. Multiple robots have the potential to accomplish a given task more efficiently because the capabilities more diverse, flexible, and distributed. Multiple robots can localize themselves faster and more accurately by exchanging information among themselves about their positions and orientations (pose). An MRS possesses a greater fault tolerance than a powerful and expensive single robot, particular because if one robot in the MRS fails or malfunctions there will be other robots to assume its tasks. In summary, an MRS will have a greater range of task capabilities, greater efficiency, increased system performance, fault tolerance and robustness, lower economic cost, ease of development, distributed sensing and action, inherent parallelism in comparison to a single robot of comparable capability.

In the MRS domain, the robotics community began to investigate major challenges such as robot synchronization, determination of a robot coordination strategy and identification of complex work environments. Also, they have investigated the potential for using MRS research to provide insight into social and life sciences involving groups of biological entities.

Multi-robot object transportation is an important research subject in the multi-robot domain. In an object transportation process, several autonomous robots navigate cooperatively in either a static or a dynamic environment to transport an object to a goal location and orientation. The environment may consist of fixed and movable obstacles and may be subject to uncertainty and unforeseen changes within the work environment. A single robot may not be able to handle heavy and large objects alone. Underlying challenges can be overcome by multiple mobile robots. A primary concern associated with multi-robot application is the method of enabling individual robots to autonomously adapt their behaviors to meet the dynamic changes in their task environment. For this, each robot needs to communication with its peers, exchange sensor information, formulate the coordination strategy and obstacle avoidance scheme, plan the robot trajectory, and assign and receive subtasks and roles for carrying out the required task quickly and successfully. Due to main challenges inherent in various research fields, multi-robot object transportation is a good benchmark application to assess the effectiveness of multi-robot control and coordination strategies. Therefore control and coordination methods for multi-robot systems have been investigated by various researchers and multi robot object transportation problem has become a theme of research and development.

1.1 Project Description

A main objective of the overall project of "Multi-Robot Autonomous Assembly System" in our laboratory (Industrial Automation Laboratory) is to develop a physical multi-robot system with object manipulation, transportation and assembly capabilities. First, a group of intelligent autonomous robots cooperatively search and identify any useful parts within the work environment. Then the identified parts are individually or cooperatively transported to a designated place and assembled into a useful device. Finally, the built device is transported individually or cooperatively to a known location in an unknown and dynamic environment. During this process, the obstacles may be static or even appear randomly during the operation. The overall project representation is shown in Figure 1.1.

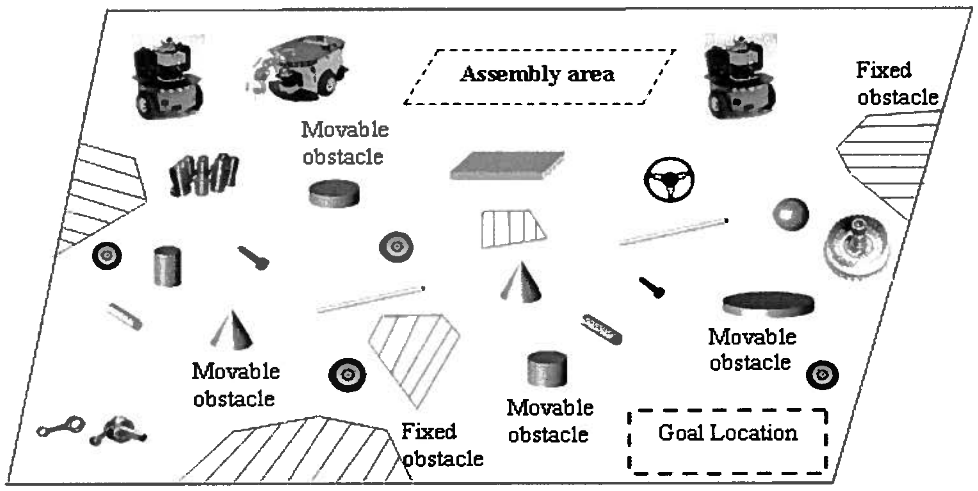


Figure 1.1: The overall project representation.

The overall project consists of three main stages:

1. Multi-Robot Foraging: Multiple mobile robots autonomously search the environment for useful objects/parts. The identified useful parts are transported by individual robots or cooperating multiple robots to a designated place.

2. Multi-Robot Assembly: Robots autonomously determine the assembly sequence for constructing a useful device, properly handle and manipulate the parts, and assemble the device.

3. Multi-Robot Transportation: Tow or more robots cooperatively transport the built device to the goal location.

A major challenge of the overall project is the transportation of an object of arbitrary size and shape - a key issue in the multi-robot domain. In the stage of transporting components for assembly at a designated location the robots may have to work in a complex dynamic environment. The stages of multi-robot forging and transportation need to use cooperative object transportation technologies to accomplish the overall task. This thesis mainly concerns object transportation technologies and the development of a physical multi-robot system to transport an object of arbitrary size and shape.

1.2 Objective of the Research

The physical multi-robot object transportation system, as developed through the research reported in the present thesis, has the ability to navigate an object to a predetermined location though a dynamic and unknown environment with complicated obstacle distribution. The primary research objectives of the thesis are listed below.

- Develop a fully distributed physical multi-robot system for operation in a dynamic and unknown environment with complicated obstacle distribution.
- Improve an available single-agent machine learning algorithm so as to enable a multi-robot system to deal with the uncertainty and unforeseen changes in the environment including changes of the robots.
- Develop a technique to identify obstacles within a dynamic and unknown environment.
- Develop a technique for pose estimation in a task of object transportation, to more realistically estimate the pose of the object of interest.
- Evaluate the performance attributes such as coordination, self-learning and adaptation, speed, and accuracy of the developed multi-robot system and its underlying methodologies.

1.3 Problem Definition

Development of a physical multi-robot system is an important objective of the present work. This system consists of a group of intelligent autonomous robots, which have the ability to work cooperatively for transporting an object to a goal location and orientation. The work environment is considered unknown and dynamic, and obstacles may be present or even appear randomly during the transportation process. Multi-robot coordination and machine learning are integrated into the developed physical system to cope with the challenges of the problem. A schematic representation of the developed system is shown in below Figure 1.2.

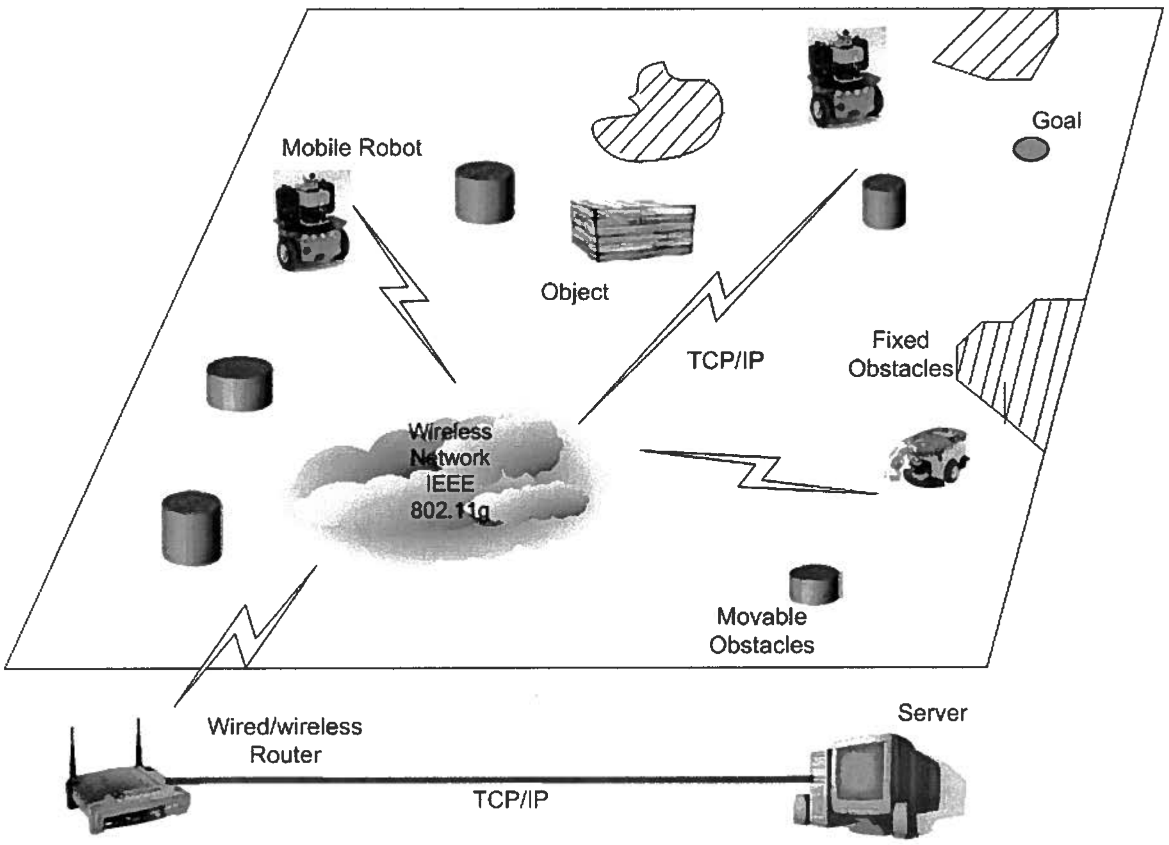


Figure 1.2: Schematic representation of the developed system.

Three or more autonomous mobile robots are present in the system, which enable the cooperative transportation of an object to a goal location. During the course of the transportation, they have to interact with each other to establish the strategy of coordination and cooperation. The feasible and suitable locations and points of action have to be established for each robot assisting in the cooperative transportation. The object has to be transported quickly and effectively while avoiding obstacles that may be present during the transportation process. Furthermore, the system will have to avoid damage to the transported object. In the developed system, charge coupled device (CCD) cameras, laser range finders and sonars are used to monitor and measure the current location and orientation of an object. The dynamic and unpredictable environment contains both fixed objects and also movable obstacles, which may appear randomly. The robots and sensory system are separately linked to their host computers, which are connected though a local network to implement machine intelligence and system control.

It is important to consider three major challenges in the development of the multi-robot system. The first one is the dynamic and unpredictable environment. In particular, dynamic obstacles may be scattered arbitrarily within the working environment. The obstacles may appear and disappear randomly during the transportation process. In addition, there will be multiple robots working parallel. While one robot makes decisions, some other robots may have changed the environment according to their behaviors. This also makes the environment dynamic from the view of a single robot. As a result, the process of decision making becomes rather complex for the robots within the environment. Challenge is to make decisions in real time and learn to identify the conditions of the surrounding environment from the local viewpoint of the robots.

The second challenge results from the localized sensing capabilities of the robots. In the present project it is assumed that each robot is capable of detecting an object or obstacle within a limited detection radius. The robots possess local sensing capabilities only, and consequently the global environment is generally unknown to them. A robot will not have a good knowledge about the location or the shape of an object until it moves sufficiently close to the object. The unknown environment is another major challenge in the multi-robot domain, because the robots do not have a complete understanding about the environment in advance, partly due to the unfamiliarity and partly due to the dynamic nature of the environment. Thus the system will not be able to utilize the traditional planning methodologies for the decision making process.

The third challenge is the lack of reliable communication among robots, where the disturbances are represented by white noise. Due to unreliable communication, the robots may end up selecting a wrong coordination strategy. Improvement of the robustness of multi-robot decision-making in an environment with unreliable communication is also a challenging issue in the multi-robot domain.

Normally, the environment of a multi-robot system is dynamic and unknown. In such a situation, the communication of robots is generally inconsistent. In this thesis, several approaches are introduced to meet these challenges in multi-robot systems in enabling them to work effectively in an unstructured environment.

1.4 Related Work

A significant amount of work has been performed by the robotics community in the past 10 years in multi-robot coordination and learning for decision making, particularly to support effective and robust operation of multi-robot systems in both simple and dynamic environments. This section presents some relevant work in this context.

Cao et al. (2006) presented a multi-robot hunting task in an unknown environment. They proposed a distributed control approach involving local interactions with respect to local coordinate systems. Proposed approach can cope with the cumulative errors of wheels of the robots and also unreliable communication networks. Computer simulation validated their approach.

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

Stone et al. (2006) examined the issue of uncertainty in multi-robot systems. They identified several sources of uncertainty and proposed a method for reducing the uncertainty and making decisions in the face of uncertainty. Their method enables effective planning under uncertainty for robots engaged in goal orientated behavior within a dynamic and complex environment. The scaling issues of multi-robot systems are presented by Gustafson et al. (2006). They used several abstract models to elucidate that it was counterproductive to increase the number of robots or the sensor strength in a multi-robot system.

Mataric et al. (2006) presented a mathematical model for dynamic task allocation in multi-robot systems, based on stochastic processes. They made an assumption for a large-scale multi-robot system with local sensing, behavior based controller, and distributed task allocation capabilities. Through storing the history of the environment examined in the internal state of a robot, they tried to cope with the issues of local sensing and indirect communication in multi-robot systems. Their model was demonstrated for a multi-robot foraging task.

In another paper, the same authors (Gerkey and Mataric, 2004) developed a taxonomy 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 introduced method, they compared six typical multi-robot architectures with their MRTA approaches.

Many valuable surveys have been conducted on multi-robot systems. Farinelli et al. (2004) completed a survey on multi-robot systems and proposed a new taxonomy for multi-robot coordination. Multi-robot tasks were classified 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.

In a survey carried out on existing multi-robot architectures, Parker (2000) pointed out several challenges in typical multi-robot tasks. Specially, she categorized the available architectures into three multi robot architectures as: general architecture, specialized architecture and hybrid architecture. In addition, she classified three typical multi-robot tasks and their main challenges. For multi-robot object transportation, a major challenge is uneven terrain; for multi-robot learning, a major challenge is how to assign credits among robots; and for multi-robot motion coordination, the key challenge is computational complexity.

There are a number of surveys that review Multi-Robot Learning (MRL) or Multi-Agent Learning (MAL). For example, the survey presented by Yang and Gu (2004) identified that scaling an individual reinforcement learning algorithm to multi-robot systems would violate its Markov decision process assumption, which is the weakness of many available multi-robot reinforcement learning approaches. They also classified four frameworks of multi-agent reinforcement learning: the stochastic games (SGs)-based framework, the fictitious play framework, the Bayesian framework, and the policy iteration framework. Furthermore, they categorized multi-agent reinforcement algorithms for possible application to multi-robot systems. They argued that there were four major challenges in applying reinforcement learning to multi-robot systems: the Markov decision assumption, continuous spaces, uncertainties, and incomplete information. Three future directions were pointed out as well: the behavior based approach with reinforcement learning, fuzzy approximation functions, and continuous reinforcement learning.

Panait and Luke (2005) presented a survey on multi-agent learning for cooperation and they discussed the two main challenges in cooperative multi-agent learning; namely, the large learning space and the dynamic environment. Furthermore, they classified two types of learning: concurrent learning and team learning. The major issues in the area of concurrent learning were discussed as: credit assignment, learning dynamics, teammate modeling and relationship between learning and communication.

1.4.1 Multi robot Coordination

To improve a mechanism for coordinating multiple robots in the execution of cooperative tasks has been a challenging objective in multi-robot systems. Multi-robot coordination has been carried out according to a developed multi-robot architecture. Multi-robot architectures seek to improve the coordination mechanism for the object transportation process. Primarily five types of multi robot coordination approaches are briefly discussed now; namely, the behavior-based approach, the planning-based approach, the market-based approach, the distributed control-based approach, and the learning-based approach.

1.4.1.1 Behavior-based approach for coordination

Parker (2000) introduced a distributed and behavior-based multi-robot architecture called L-ALLIANCE. It employs the concepts "Impatient" and "Acquiesce" to dynamically stimulate behaviors for coordination among robots. Moreover, by evaluating the performance of the teammates and dynamic changes in the environment, L-ALLIANCE autonomously updates its parameters to adjust to those changes. Introduced architecture was validated in a box pushing task with heterogeneous mobile robots.

The behavior based multi-robot coordination, CAMPOUT, has been introduced in the Robot Construction Crew (RCC) project developed in the Jet Propulsion Laboratory (JPL) of NASA. CAMPOUT is a fully distributed, behavior based multi-robot architecture and it uses leader follower distributed coordination strategy. CAMOUT was validated using a multi-robot transportation task in an autonomous deployment project on a planetary surface. In their latest work they introduced a multi-robot autonomous construction task based on the CAMPOUT architecture.

Mataric et al. (2002) developed a behavior based approach for a distributed multi-robot system. They developed controllers which were robust to noise and robot failures. Three versions of the task were evaluated in a spatio-temporal context using interference, time to completion, and distance traveled as the main investigative parameters.

Konidaris and Hayes (2005) proposed to integrate the behavior-based approach with reinforcement learning for multi-robot coordination. Furthermore, they suggested to use topological maps to reduce the learning space in reinforcement learning.

Camacho et al. (2006) introduced behavior based coordination for robot soccer agents. In their work, they allocated each robot a duty for execution of the particular defined task. A rule-based RECCA control algorithm was also proposed.

1.4.1.2 Planning-based approach for coordination

Miyata et al. (2002) presented coordination of multiple mobile robots for cooperative transportation in unknown static environments. They introduced a centralized approach for task planning and assignment. Priority-based assignment algorithms were used for multi-robot coordination and motion planning.

1.4.1.3 Distributed control approach for coordination

Wang and de Silva (2005), Zaerpoora et al. (2005), Pereira et al. (2004), and Chaimowicz et al. (2004) presented the caging formation control problem in multi-robot coordination for cooperative transportation. They proposed an "Object Closure" strategy to coordinately move the object with multiple mobile robots while maintaining the formation. The major challenge was to establish a bounded movable area for the object during its transportation. 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 used 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.

Marshall et al. (2006) introduced a distributed control strategy called "cyclic pursuit" for multi robot coordination. The particular approach was found to be robust in the presence of un-modeled dynamics and delays due to sensing and information processing.

1.4.1.4 Market-based approach for coordination

Multi-robot task allocation is a vital and complex area of multi-robot coordination and the conventional approaches have seen low success in the presence of a large number of robots in the system. However, market-based approaches seem better for solving the complex multi-robot task allocation problem, and they are gaining popularity. For example, Mataric et al. (2002) presented an auction-based approach for multi-robot coordination. They utilized the publish/subscribe first price auction method and validated it in a multi-robot box pushing task. During the transportation process a "watcher" robot monitors the transportation task, and "publishes" the required auctions for use by the "pusher" robots in the same environment. The "pusher" robots determine the available task at a given instance though "subscribing" to the information from the "watcher" robot. By matching its own abilities to the required tasks, each "pusher" robot bids for a task. When the "watcher" robot receives all bids from the "pusher" robots, it selects the most suitable "pusher" for the particular tasks and then the "watcher" robot broadcasts the decisions to the "pusher" robots in the system. Particular auction-based approach was validated for a multi-robot box pushing task in a static environment without obstacles.

1.4.1.5 Learning-based approach for coordination

Typically multi-robot systems work in a dynamic and unknown environment where a traditional behavior-based approach can easily fail because the designer cannot foresee all possible world states the robot would encounter and design the corresponding behavior. A behavior-based multi-robot system will work well in a known and structured environment. Learning capabilities are important for multi-robot systems to work properly in this type of environments. Most existing work in multi-robot learning utilizes reinforcement learning due to its simplicity and good real-time performance. For example, Ito and Gofuku (2004) proposed a two-layer multi robot architecture for multi-robot coordination in cooperative object transportation. They introduced a centralized machine learning method in the top level of this architecture for decision making. A distributed approach of rule-based control was developed for the lower level of motion control of the robot, to reach a specified position and take a specific action according to the commands from the upper level. Their approach reduced the learning space by integrating reinforcement learning with genetic algorithms.

1.4.2 Machine Learning in Multi-Robot Systems

Learning capabilities are desirable for multi-robot systems when the working environment is unknown, unstructured and dynamic. Technologies of machine learning have been employed commonly for this purpose. Among the machine learning approaches, reinforcement learning and particularly Q-Learning has been quite popular due to its simplicity and good real-time performance.

A well-known work in this area, Parker et al. (2002) studied two significant aspects in multi-robot learning called learning new behaviors autonomously and learning parameter adjustments. They proposed two approaches for autonomously learning new behaviors in multi-robot systems. The first approach is called distributed pessimistic lazy Q-learning and it combines lazy learning with Q-learning. It also uses a pessimistic algorithm for the credit assignment problem. The second approach, called Q-Learning with VQQL (vector quantization with Q-learning) and the generalized Lloyd algorithm, addresses the generalization issue in reinforcement learning. Both approaches have been validated in a COMMMT (Cooperative Multi robot Observation of Multiple Moving Targets) project. Let us consider learning for parameter adjustment. Their previous paper (as mentioned under behavior-based approach) introduced the behavior-based architecture called L-ALLIANCE, which 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.

Kapetanakis and Kudenko (2002) established two of multi-agent learning techniques called multiple single-agent learning and social multi-agent learning. In the multiple single-agent learning technique, each learning agent observes other agents as a part of the environment and an individual agent does not have any explicit knowledge of about other agents. However, in social multi-agent learning, agents can have a high level of knowledge of other agents and integrate this knowledge in the learning process, potentially using local modeling techniques, communication and coordination to support the learning task.

Martinson and Arkin (2003) presented a multi-robot foraging task with Q learning. They used Q-learning to select roles of the robots dynamically and it was integrated with the behavior-based approach. The learning space was reduced by the behavior representation. They did simulation to verify their approach.

Dahl et al. (2004) presented a Q learning-based multi-robot cooperative transportation task in a simple environment. They proposed a two-layer multi-robot architecture. The Q learning-based behavior-selection subsystem was located in the upper level and the behavior-based reactive subsystem was located in the lower level. Further, they introduced two communication strategies to speed up the convergence of Q learning.

Kovac et al., (2004) addressed a "pusher-watcher" multi-robot box pushing problem with reinforcement learning. According to their approach one robot acted as a "watcher" which observed the current world state and broadcasted it to other robots. The other robots in the project acted as "pushers" who selected their respective pushing actions using Q learning. A major weakness of this approach is the need for a robust communication network.

Taylor and Stone (2005) studied a methodology to transfer learned knowledge from one task to another with a different state-action space. This is an important issue in reinforcement learning. This transfer is beneficial for the reduction of the training time in reinforcement learning. Particular methodology used the specific domain knowledge to construct a knowledge mapping function between two considered 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 recognized four factors that directly affect the distributed multi robot learning: the reward type, the learning algorithm, diversity, and the number of robots. They concluded that the reward type and the learning algorithm considerably affected the final results, and the scalability of the system would also affect the learning speed but had only a slight effect on the final outcomes.

It is clear that reinforcement learning is rather popular in multi-robot coordination. The environment in a multi-robot task is typically dynamic because the robots themselves also change in their shared environment. Therefore, the single agent reinforcement learning algorithm when extended to a multi-robot system violates its assumption of a stationary environment.

A number of multi-agent reinforcement algorithms have been introduced in the multi-agent area, namely, mini-max Q learning algorithm, Nash Q learning algorithm, friend or foe Q learning algorithms, team Q learning algorithm, and so on. The team Q learning algorithm 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 its own decisions.

The large learning space is the main hurdle in multi-agent reinforcement learning algorithms. Each robot needs to monitor its own actions and its peer actions as well. Therefore, when the scalability of the system increases, the resulting learning space will be excessive. However, when the number of robots increases, the single agent Q learning algorithm still can provides reasonable performance due to its fixed learning space.

Multi-robot community has tried to employ numerous approaches such as neuro-fuzzy, genetic algorithms (GAs), and so on to reduce the large learning space in multi-robot coordination. However, research in this area is still in its infancy, and there are many open questions yet to be explored.

1.4.3 Pose Estimation Technologies for Multi-robot Systems

Estimation of pose (i.e., position and orientation) is essential for object transportation in a multi-robot system. Technologies of pose estimation are utilized to estimate the real-time pose of objects, including obstacles and the other robots, in the work environment. It will help identify the world state and make appropriate decisions accordingly. Researchers have utilized various sensory methods such as digital CCD (charge coupled device) cameras, global positioning systems (GPS), laser range finders and sonar devices to identify the pose of an object in a robotic work environment. Also, sensing algorithms have been developed to recognize, identify objects and estimate features or poses of the objects in the environment.

Lang et al. (2008) presented a multiple sensor-based method for robot localization and box pose estimation. A CCD camera mounted on the robot was used to find the box in its environment. A laser range finder mounted on the robot was activated to measure the distance between the laser source and the object. Finally homogenous matrix transformation was applied to represent the global pose of the robot and the box. The approach was validated using computer simulation.

Park et al. (2006) introduced a method for global localization of an indoor environment, which employed object recognition using a stereo camera. Their method of global localization first made a coarse estimation of the pose and then refined it. The coarse pose was estimated by means of object recognition and least squares fitting through singular value decomposition, while the refined pose was estimated by using a particle filtering algorithm with omni-directional metric information. Even though a vision system has been used for pose estimation, the approach has a number of drawbacks. In particular, the presented schemes are computationally expensive, time consuming, have time delays, and are susceptible to changes in the ambient lighting conditions.

Ekvall *et al.* (2005) studied a computer vision approach for object recognition and pose estimation based on color co-occurrence histogram and a geometric model. Even though the approach was somewhat robust, it failed under large changes in lighting conditions.

Kato et al. (2003) presented a method for identifying a robot and obtaining its relative position in a multi-robot system using an omni-directional vision sensor. Their approach was validated using the simulation and experimentation. Similar work has been done by Hajjdiab et al. (2004). A vision based approach for multi-robot simultaneous localization and mapping (SLAM) was presented. They presented a method to calculate the locations of the robots using a collection of sparse views of the planar surfaces on which the robots were moving. The camera movements were estimated using inter-image homographs computed by matching the overhead transformed views.

Tracking of multiple moving objects using vision sensors is an important issue in road traffic control systems. Chen et al. (2005) introduced a framework for spatiotemporal vehicle tracking based on unsupervised learning segmentation and object tracking. 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.

Laser range finders and CCD cameras are commonly-used sensors in mobile robots. Most laser range finders have the ability to scan the surrounding environment and accurately detect objects within a radius of 50 meters, and return a distance measurement for each degree in a 180 degree range. On the other hand, CCD cameras can be used to detect the shape, color and surface features of the objects of interest and return the corresponding vision information. Researchers have combined these two sensors for object recognition and localization.

Murarka et al. (2006) proposed a hybrid method incorporating laser range finders and vision for building local 2D maps to help an intelligent robotic wheelchair distinguish between safe and hazardous regions in its immediate environment. Here, 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 obstacles was used to construct a local 2D safety map.

Tomono (2005) used a mobile robot equipped with a laser range finder and a monocular camera to build a 3-D environment 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.

Takahashgi and Ghosh (2005) proposed a method to identify the parameters of a moving object by integrating a laser range finder and a vision sensor. The approach has been reduced the dimension of the parameter ambiguity.

Gopalakrishnan et al. (2005) developed a mobile robot navigation project which integrated a laser range finder and a vision sensor. The laser range finder was used for localization and the vision sensor was used object detection. Also a route based navigation technique was used for mobile robot navigation.

1.5 Contributions and Organization of the Thesis

This thesis performs research and develops advance techniques which will help multi-robot systems operate in a robust and effective manner in a dynamic and unknown environment. The main contributions in the thesis are as follows:

- A multi-robot coordination mechanism and a fully distributed three-layer multi-robot hierarchical architecture are developed to carry out cooperative tasks, which integrates machine learning and behavior-based approach of robot coordination. This coordination technique enables efficient state identification and coordination among robots. The developed system architecture has the ability to dynamically allocate a task in an unknown environment.

- An improved reinforcement learning algorithm termed the "modified Q learning algorithm" is developed, which is able to provide efficient state identification and optimize the action selection conflicts in multi-robot learning. It also helps deal with some key issues in multi-robot learning; for example, task assignment, reduction of the learning space, and circumventing the Markov assumption.

- A method is developed to estimate the pose of an object, which can be used to track potential obstacles and the other robots in the work environment simultaneously during object transportation. This approach is more realistic than other available approaches. It is validated using two types of physical experimentation.

- A physical multi-robot transportation system, which integrates the developed techniques, is developed in our laboratory (Industrial Automation Laboratory). This multi-robot system works in a dynamic and unknown real-world environment and shows effectiveness, flexibility and overall good performance.

The organization of the thesis is as follows. Chapter 1 has discussed multi-robot systems and the research objectives of the thesis. It defined the object transportation problem as studied in the thesis, and discussed related past work. Chapter 2 addresses the multi-robot coordination mechanism and the three-layer multi-robot architecture. Two techniques of multi-robot coordination included in multi robot architecture - the behavior-based approach and the machine learning approach - are discussed and their relationship is presented. Several pertinent issues of distributed controls for multi-robot coordination are discussed. Chapter 3 proposes a modified Q learning algorithm in the multi-robot domain to facilitate robotic decision making in a dynamic and unknown environment. This algorithm is assessed using a multi-robot transportation task in simulation. The results are analyzed, and the advantages and disadvantages of the algorithm are indicated. In Chapter 4, an innovative method for determining the pose of an object is developed, which will facilitate the robots to identify useful objects in a work environment. The technology is validated using a task of multi-robot object transportation, in simulation. Chapter 5 presents a physical multi-robot cooperative transportation system operating in a dynamic and unknown environment. Microsoft Visual Studio 2005 – C^{++} distributed computing model is introduced to allow robots to effectively communicate and cooperate. Experimental results are presented to study the performance of the developed system, particularly concerning adaptively and robustness. Chapter 6 summarizes the primary contributions of the thesis and indicates some relevant issues for future research.

Chapter 2

Multi-Robot Coordination for Object Transportation

2.1 Overview

Multi-robot coordination can be defined as the ability to manage the interdependencies of activities between mobile robots. Coordination among a group of robots should in principle improve the overall performance of the team of robots, as robots may share their world views and may negotiate to identify the present world state and cooperatively select the actions. However, in practice, effectively handling in real-time, multi-robot information and coordination is a challenging task.

Many open questions remain in the area of multi-robot coordination. How should a group of robots divide tasks among its members? Once roles have been assigned to the robots, how should they position themselves to fulfill their roles without interfering with their team-mates? What happens if a robot fails or if the environment changes so that a different robot is more suitable for the task?

The coordination of multi-robot teams in dynamic environments is one of the fundamental problems in multi-robot transportation. The underlying question is, given a group of robots and a task to be performed in a dynamic environment, how the robots should be coordinated in order to successfully complete the task? Coordination normally implies synchronization of robot actions and exchanging of information among the robots. The level of synchronization and communication depends heavily on the task requirements, characteristics of the robots, and the environment. Thus, different levels of coordination are required in different situations.

Coordination of a group of mobile robots is performed according to a multi robot architecture developed in the present work. The architecture represents the organization structure of the key components and their functional descriptions in a multi-robot system. Promoting cooperation or coordination among robots and enabling a member robot to make rational decisions are the main objectives of the architecture.

In this chapter, a framework for a group of robots to coordinate their activities and a hierarchical multi-robot architecture are proposed. This hierarchical multi-robot architecture integrates machine learning and behavior-based approaches. In section 2.2 a fully distributed hierarchical multi-robot architecture for robotic decision making and detailed implementation is described. Section 2.3 presents a coordination framework for a group of robots. This framework for multi-robot coordination facilitates global sharing of information during the stage of object environment state identification. It also describes how obstacle identification in the object neighborhood may be combined with multi-robot coordination. Finally, in section 2.4, relevant distributed control issues in multi-robot systems are discussed. The multi-robot architecture and coordination framework developed in this chapter will be validated using computer simulations and physical experiments. Details of the validation process will be presented in the subsequent chapters.

2.2 General Hierarchical Architecture for Multi-Robot Coordination

There are five main approaches for multi-robot coordination as mentioned in section 1.4.1: the behavior-based approach, the planning-based approach, the distributed control-based approach, the market-based approach and the learning-based approach. Among these approaches, the behavior-based approach is the most popular one because of its simplicity and robustness. However, this approach is particularly suitable for a known and structured environment. The major drawback of this approach is that the designer must have an ability to predict all the available environment states, because he needs to design all possible behavior rules. However, most realistic systems need to make decisions in a dynamic and unknown environment, and this faces numerous challenges as indicated below.

First, a human must design all the behavior rules and individual robots must establish their preferences in advance. It is virtually impossible for a human designer to predict all possible environmental states that the robots will encounter and design the relevant behavior rules for them. On the other hand, within a dynamic and unknown environment, behavior-based robotic decision making can easily fail when the number of environmental states is large. Second, typically there will be a large number of environmental states in a dynamic and unstructured environment. The designer must design a general rule base to deal with all these environment states. However, it will be virtually impossible for the designer to arrange the preference for each behavior rule in an extensive rule base due to the problem of "combination explosion." Therefore a purely behavior-based approach for multi-robot coordination is not feasible in a unknown and dynamic environment.

Planning-based approach utilizes planning techniques of traditional artificial intelligence (AI) to establish optimal decisions for the robots (Miyata et al., 2002) in a multi-robot system. The centralized approach and the distributed approach are the two types of planning-based approaches for multi-robot coordination.

In the centralized planning approach, the actions are planned by a single robot in the system and this robot assigns subtasks to all other robots in the team. Task allocation among multiple robots is a NP (non-deterministic polynomial) time problem in the theory of computational complexity. In fact computational complexity is the key disadvantage of this approach. For a practical multi-robot systems consisting of large number of robots, finding an analytical solution is very difficult and even impossible, through this approach. A compromise would be to approximate it by employing an AI based optimization technique. Furthermore, the lack of communication will make the centralized approach weak. On the other hand, the distributed planning approach plans robot actions independently and achieves a globally optimal decision by integration the individual decisions of all the robots. In view of its difficulty, few successful examples have been reported in the literature. In addition, both centralized and distributed planning approaches can easily fail in a dynamic and unknown environment since these approaches are unable to alter their policies (action sequences) for the robots quickly enough to adapt to a rapidly changing environment.

The distributed control approach (Pereira et al., 2004; Wang and de Silva, 2005) utilizes a distributed control strategy such as "Form Closure," "Object Closure," or "Leader-Follower Control" to assign sub-tasks to the robots for controlling the team formation in the system. Distributed control is not a general approach. In the class of multi-robot cooperative transportation, it is only suitable for special task situations, and this is a major disadvantage of the approach. Furthermore, the approach of distributed control also suffers from the problem of computational complexity.

The market based approach (Mataric et al., 2002) is a rather promising approach for multi-robot coordination in a dynamic environment. However, there are many open questions related to this approach. Its main challenges are computational complexity and real time performance.

The machine learning-based approach attempts to circumvent the computational complexity problem which affects other multi-robot coordination approaches by approximating the optimal polices of the robots. In particular, in view of its good real-time performance and simplicity, reinforcement learning, particularly the Q learning algorithm, has been employed commonly (Parker et al., 2002; Martinson and Arkin, 2003; Martinson and Arkin, 2003; Dahl et al., 2004; Taylor and Stone, 2005; Wang and de Silva, 2007). In a survey paper, Arai et al. (2002) have recommended that the area of learning based multi-robot coordination.

The learning capability is essential for multi-robot coordination and task execution in a dynamic and unknown environment. As suggested in Chapter 1, the learning capability will enhance the adaptability to the dynamic environment and assist the robots to identify new environmental states. The robots can continuously adjust their action policies and improve the performance based on the experiences of past success and failure in a dynamic environment, thus completing the required tasks in a faster and more robust manner. Furthermore, Wang and de Silva (2006) have shown that the single agent learning scheme when extended to a multi-robot system, will violate the original assumption of a stationery environment. Consequently, the learning algorithm may not converge to its optimal policy.

These observations clearly show that the problem of decision making for multi-robot coordination is a challenging undertaking. A single approach may not be able to solve all the problems encountered in a cooperative multi-robot transportation process. Therefore, the combination of several existing techniques will be employed to develop a multi-robot architecture that can effectively support multi-robot coordination and cooperation in a dynamic and unknown environment.

This fully distributed multi-robot hierarchical architecture integrates two popular artificial intelligence (AI) techniques; namely, machine learning and behavior-based approaches for decision making in a dynamic, unknown and unstructured environment. Figure 2.1 shows the proposed three level multi-robot architecture that offers improved robustness and real-time performance for multi-robot object transportation.

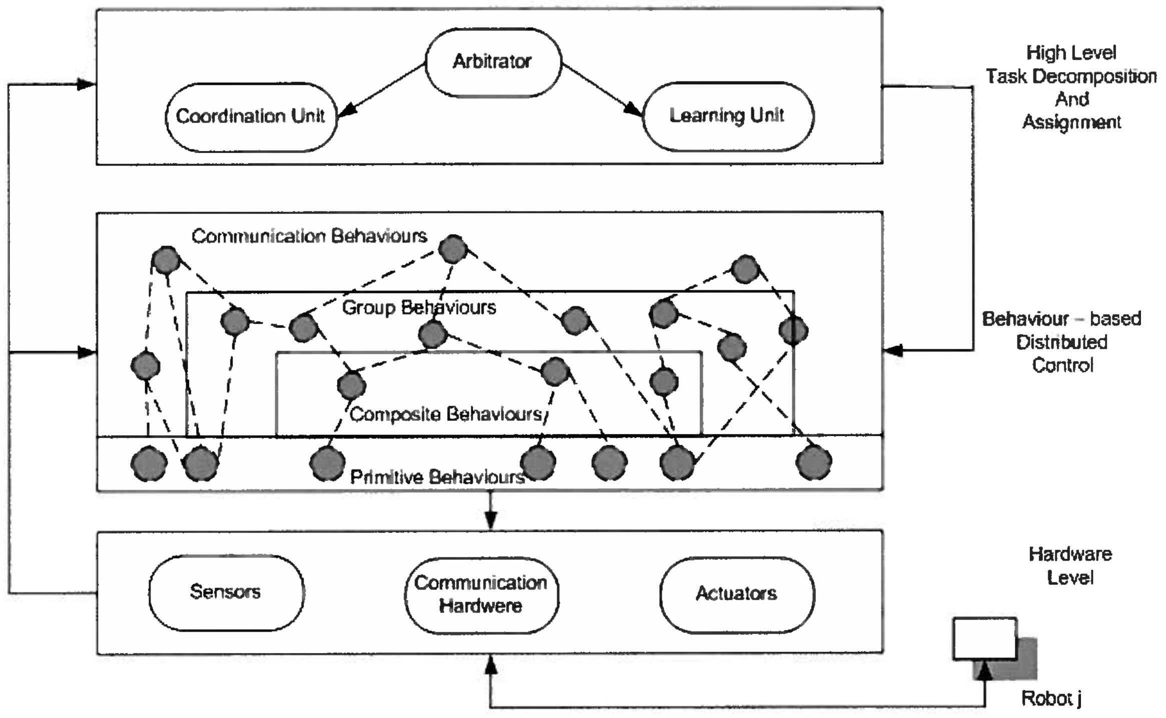


Figure 2.1: Hierarchical Architecture for Object Transportation.

The hierarchical architecture for object transportation integrates different techniques in a hierarchical manner to provide flexible and robust capabilities to the robots for decision making while familiarizing with the external environment dynamically. Basically, it consists of three layers: Task Decomposition and Assignment layer, Behavior-based Control layer, and Hardware level layer, subject to the constraints of the available robots.

Task Decomposition and Assignment represents a high level layer. It has a learning unit and a coordination unit which are used to implement inter-robot coordination. The distribution of available information among robots plays a vital role in multi-robot cooperative transportation. Particularly, one robot in the system will collect information from its own sensors, and will transmit the information to the system. All robots in the system will gain an understanding about the sensor information of the peer robots in addition its own, though the communication network.

Such communication will determine the current world state cooperatively, and decide the robotic actions sequentially using its learning or coordination units to reach the goal. The high level layer will perform inter-robot task decompositions and assignment.

During the transportation process each robot may use a learning unit or a coordination unit for making decisions. Particularly, in the stage of environmental state identification an arbitrator will be assigned dynamically who will monitor the environment state identification. A coordination unit is used to coordinate the activities of the particular stage. The arbitrator will decide which unit is used to make decisions for the stage of environment state identification. For example, when the robot navigates in a dynamic environment, usually the learning-based unit is activated. The arbitrator typically switches on the learning unit for decision making of the robot. However, once the arbitrator realizes that a robot within the system has identified the object and estimated its pose, it will turn off the learning unit temporarily and switch on the coordination unit. This will coordinate the state identification stage for the robots in the system within the bounded local region called the detection area (Details are fund in Chapter 3). When the robots complete the state identification stage, the arbitrator will return the decision making capability to the learning unit. Through this type of switch mechanism, the multi-robot architecture benefits from multi-robot coordination and the flexibly adapt to various environments.

A popular behavior-based approach is employed in the middle level of the architecture, to implement robust control of robotic actions. As stated above, the behavior based approach is known to be efficient in multi-robot cooperative control. The hierarchical architecture utilizes the behavior-based approach to execute sub-tasks as determined by the top level. When the task decomposition and assignment 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. The behavior-based control level consists of four types of behavior. The communication behaviors are responsible for communication among robots. The group behaviors execute coordination among robots; for example, "Assume Formation" and they are typically fired by the learning unit or the coordination unit in the top level of the architecture. Composite behaviors are responsible for the control of composite actions of individual robots such as "Find Object," "Move to Location" and "Avoid obstacle." These composite behaviors can be decomposed further into primitive behaviors such as "Turn left," "Go straight" and

"Move Forward." Through the design of hierarchical behaviors and by carefully organizing their preferences, the robots can complete the required tasks as defined by the Task Decomposition and Assignment level, in a effective and robust manner.

The low level of the architecture is the hardware level. It consists of sensors, actuators, and communication hardware of the robots. They compose the hardware platform of the hierarchical multi-robot architecture for object transportation.

What is proposed is a general architecture for distributed multi-robot coordination and decision making. This architecture is common for all robots in the system. A composite representation of the proposed architecture is given in Figure 2.2.

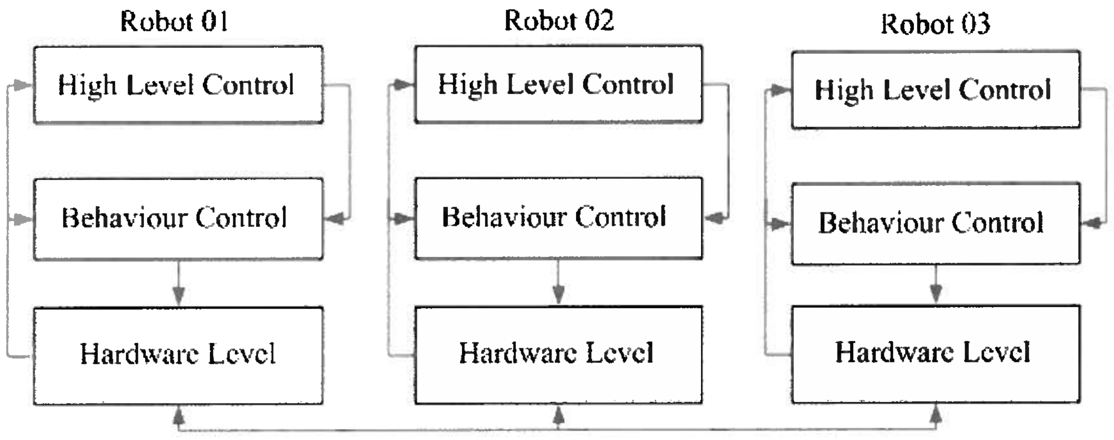


Figure 2.2: Composite representation of the proposed architecture

The top level of the architecture implements "soft" control of a robot while the middle level implements "hard" control behavior. In particular, the top level decomposes and assigns tasks among robots; and the behavior based control layer decomposes the actions of a single robot further into more detailed primitive behaviors and controls motions of the robots. By combining established mainstream approaches in multi-robot coordination, such as learning and behavior based methods, this multi-robot architecture provides more flexibility and cooperating power in a dynamic, unknown and unstructured environment than those presented in the literature.

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

2.2.1 Machine Learning Unit

The machine learning unit is located in the top level of the proposed multi-robot architecture. The main purpose of this unit is to learn optimal decision making in a given dynamic environment though continuously monitoring actions of robots and the corresponding environmental rewards. The principle of the machine learning unit is presented in Figure 2.3.

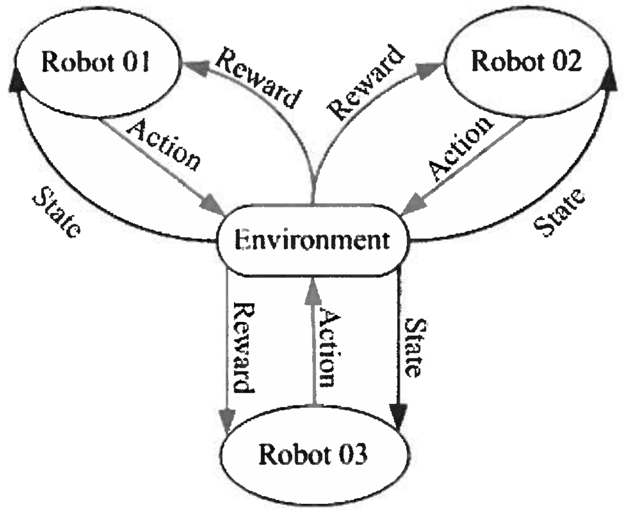


Figure 2.3: Principle of Machine Learning.

All robots in the system gain knowledge by continuously monitoring the current environmental state, selecting an action sequentially and executing it, and receiving rewards from the environment. At the sequential action selection stage, each robot needs to observe concurrently the actions of the peers in the system and review their effects on the environment. Each robot employs a machine learning technique to learn from the past experience with the work environment. The machine learning technique developed in the thesis continuously improves its performance and approximates its optimal policy for decision making.

The machine learning unit dynamically adjusts the mapping relation among the environment state and its actions. Consequently, the machine learning unit will lead to better performance in the system robots and will complement the behavior layer in the middle level.

In this thesis, Q learning is utilized and enhanced to establish the mapping relationship between the environment state and the robot actions. The major challenge arises in the modification of the single agent Q learning to accommodate a multi-robot environment. Next chapter will discuss this issue in detail.

2.2.2 Coordination Unit

In the prototype system there are three autonomous heterogeneous robots which attempt to push a rectangular object cooperatively from a starting position to a goal location. Many challenges exist in this transportation task. For example, there are obstacles distributed randomly in the environment and each robot possesses local sensing capabilities only. This means a robot will not know about an obstacle before it moves sufficiently close to the obstacle. Another challenge is that the object is rather long and heavy, so a single robot will not be able to handle it alone. The robots have to find optimal cooperative pushing strategies to move the object to the goal location while avoiding collisions with any obstacles in the environment.

In order to reach this goal, an arbitrator is designed in the top layer of the proposed architecture, which will select either the learning unit or the coordination unit for multi-robot decision making. When a multi-robot system begins its task, the arbitrator selects the machine learning unit as its decision making unit. Robots start exploring the environment for color coded object to be transported. Once an object is detected by a robot in the system, it rearranges its pose to center the color blob in the camera frame. Then the appropriate robot estimates the pose (position and orientation) of the object and the information will be transmitted to the other robots. After the object is detected and the pose is estimated, the particular robot will temporarily close the machine learning unit and transfer its decision rights to the coordination unit. At the same time the robot that detected the object is assigned as the leader of the team and all other robots in the system will stop their learning units and transfer their decision rights to the coordination unit of the leader robot. After all the robots have transferred their decision making rights to the leader robot, the coordination unit is activated temporarily. Hence, the leader robot should assign tasks to peers in the system who will assist in identifying the obstacle distribution around the object. The leader robot transmits all the processed sensor information to the system, and the robots will be able to establish the current local environment state individually after switching on their learning unit. After the state identification stage, the coordination unit of the leader robot will stop working and will transfer the decision rights to all the learning units of the robots.

Figure 2.4 shows a representation of the local environment state in the multi-robot object pushing task which employs the modified Q learning algorithm. This will be described in detailed in the next chapter. In the present subsection it is demonstrated why the coordination unit is necessary for multi-robot tasks.

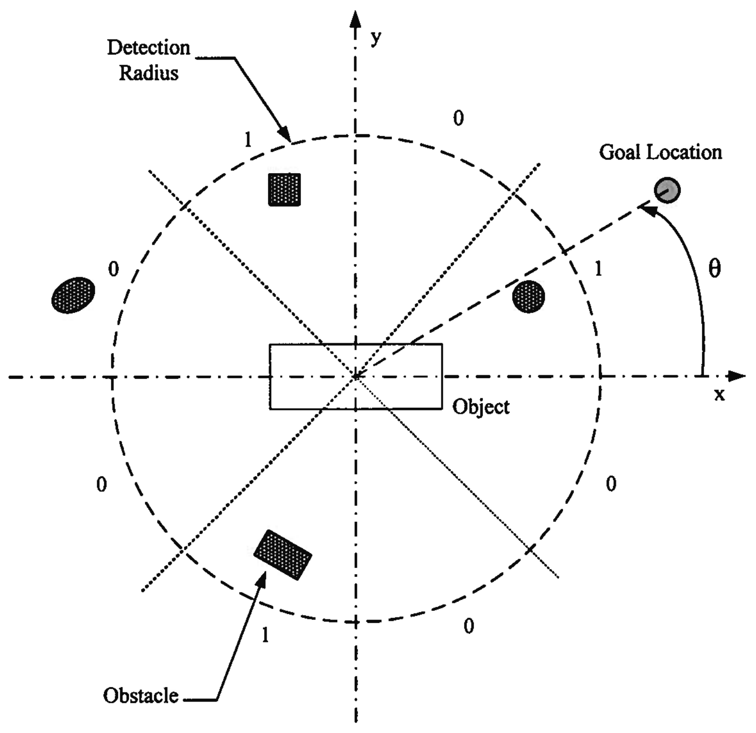


Figure 2.4: The environment state representation for coordination.

Both learning and coordination units 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.

2.2.3 Behavior-Based Distributed Control

The proposed hierarchical multi-robot architecture is represented in Figure 2.1. The behavior based distributed control layer is designed to control the local actions of the robots in the system. It is located between the high-level task decomposition and assignment layer and the low-level hardware layer. Normally, the high-level task decomposition and assignment layer selects quite abstract actions, which cannot be directly executed using actuators in the hardware layer. Hence, these high-level abstract actions need to be decomposed into more detailed actions in the layer of behavior based distributed control.

The hierarchical behavior control architecture CAMPOUT has been developed at NASA Jet Propulsion Laboratory - JPL (Huntsberger et at., 2003) to control the robot behavior in a robust manner. Their hierarchical control architecture consists of two types of behaviors called group behaviors and primitive behaviors. The group behavior layer further decomposes them into composite and communication behaviors.

The mid-level behavior control layer based on CAMPOUT clearly indicates the hierarchy of communication behaviors, group behaviors and composite behaviors. Communication behaviors are the high level behaviors relative to the group behaviors and group behaviors are accomplished though the communication behaviors. Composite behaviors are the low level behaviors relative to group behaviors. The general idea of the behavior control layer is presented in Figure 2.5.

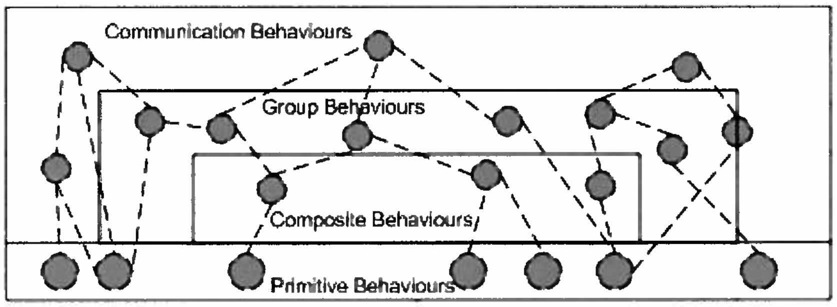


Figure 2.5: The hierarchical behavior-based control layer.

Behaviors serve as basic building blocks for robotic actions in the "Behavior-Based Systems." Simply put, behavior is a reaction to a stimulus. Thus, in some literature, behavior systems are also called as reaction systems. In reactive robotic systems, actions of robots are tightly coupled with their perceptions without a knowledge reasoning process or time history. The foundation for behavior-based systems has originated from animal models. Normally, biology has provided an existence proof that many of the tasks we want the behavior-based robotic systems to undertake are indeed doable. Additionally, the biological sciences, such as neuroscience, ecology, and psychology have elucidated various mechanism and models that may be useful in implementing behavior-based robotic systems.

Behavior-based systems have several advantages over planning-based systems. First, a behavior-based system is simple and easy to design. For a fast changing environment, a planning-based system is difficult or even impossible to implement because it cannot adapt to a fast changing environment. A behavior-based system is much easier to design than a traditional system with planning and reasoning by tightly coupling perceptions with actions and avoiding complicated time consuming reasoning and planning. Also, a behavior-based system can be successful even in a dynamic and unknown environment and it has shown better performance and robustness in physical robotic systems, if the designer can predict all possible environmental states and arrange the corresponding behavior rules to deal with them.

2.2.3.1 Behavior Representation

Several methods are available for expressing robotic behavior. These methods are stimulus-response diagram (SR), functional notation, and finite state acceptor (FSA). SR diagrams are used for graphical representation of specific behavior configurations. Here, behaviors are expressed using Stimulus Response (SR) diagrams. Figure 2.6 shows an SR diagram.

Stimulus
Behaviour
Response

Figure 2.6: SR diagram of a simple behavior.

According to Figure 2.6 a behavior can be expressed using the stimulus, response, and behavior. Therefore, the triple (S, R, β) represents the behavior, 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.

Binary coded duple (p, λ) represents an individual stimulus $s(s \in S)$, where p indicates the type of stimulus and λ denotes the strength of the stimulus. A threshold value τ is assigned for each stimulus, and the response will be generated only if the value of stimulus is bigger than τ.

In addition, a response r can be expressed with a six dimensional vector consisting of six parameters for mobile robots. For each of the six degrees of freedom of motion, each parameter encodes the magnitude of the translation and orientation responses:

$$r = [x, y, z, \theta, \phi, \varphi]\text{, where } r \in R \tag{2.1}$$

The gain value g is introduced for a particular behavior β in order to further modify the magnitude of its response. A behavior is expressed as

$$r = g * r = g * \beta(s) \tag{2.2}$$

2.2.3.2 Behavior Composition and Cooperation

Behavior coordination has two predominant classes of coordination. These classes are competitive (composition) class and cooperative class. Each class has several strategies for realization. First consider the composition of behaviors which is described in Figure 2.5, which clearly shows how composite behaviors are decomposed further into primitive behaviors. Therefore, simple primitive behaviors can be joined to construct more abstract composite behaviors. Composite behaviors are behavior packages on which a behavior-based robot system is built. A behavior coordination operator and a number of behavioral components are included with each composite behavior. The behavioral components may be either primitive behaviors or composite behaviors.

High level behaviors can be built on some simple behaviors. These high level behaviors can be referred without needing to know their behavioral components. Composite behavior is given as follows:

$$\rho = C(G * R) = C(G * B(S)) \tag{2.3}$$

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

Here, B stands for the vector with n number of current composite behaviors $(\beta_1, \beta_2, \cdots, \beta_n)$. S represents a vector of all detectable stimuli relevant for each behavior β_i at time t; R represents a vector of response of n number of behaviors $(\beta_1, \beta_2, \cdots, \beta_n)$ at time t; G represents a gain matrix which modifies the response matrix R; and C represents the behavior coordination operator for composite behavior (There are two types of behavior coordination operators, called competitive operator and cooperative operator, for two classes of behavior coordination). Figure 2.7 shows the SR diagram of a composite behavior with a competitive operator.

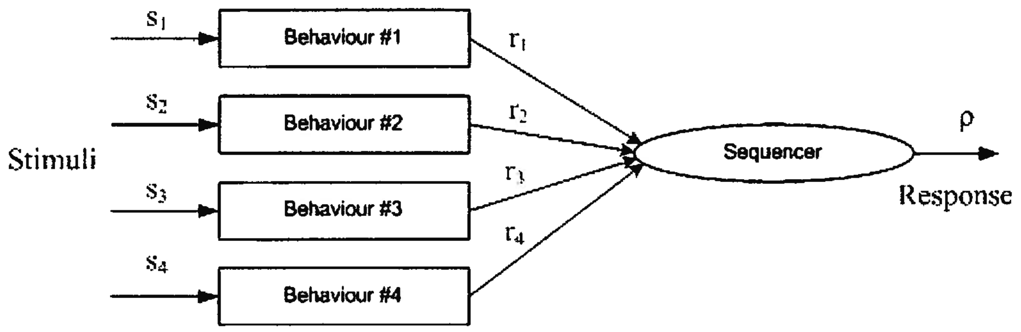


Figure 2.7: The SR diagram for a composite behavior with a competitive operator.

According to this figure, conflict can result when two or more behaviors are active simultaneously, each with its own independent response. Competitive method provides a means of coordinating behavior response for conflict resolution. This competitive operator is designed to select the winner behavior from among the simultaneously activated behaviors. Only the winner's response is directed to the robot for execution. There are numerous competitive operators; for example, suppression-based arbitration, arbitration via action selection, voting based arbitration, and so on.

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

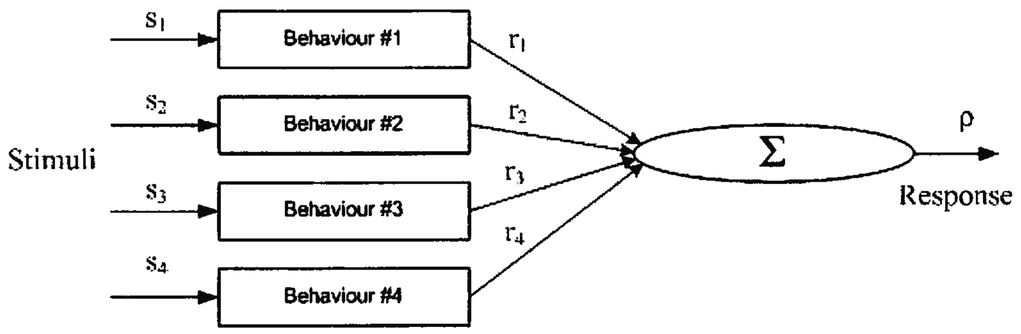


Figure 2.8: The SR diagram of a composite behavior with a cooperative operator.

The cooperative method provides an alternative to a competitive method such as arbitration. It uses behavior fusion and provides the ability to concurrently use the output of more than one behavior at a time. The outputs of multiple behaviors are fused and directed to the robot for execution. It has been pointed out that multiple behaviors are executed by the robot one at a time if they do not conflict with each other. The simplest cooperative operator is performed through vector addition or super-positioning. The key to a cooperative operator is to design the behaviors so that their responses can be fused. The potential field-based method is the most popular approach, which simply adds the outputs of multiple behaviors.

2.2.3.3 Communication Behaviors for Coordination

For coordinating the activities of the robots in a multi-robot system, communication behaviors are designed to receive information from the high level task decomposition layer, exchange sensing information, objectives, and internal states of the robots. The communication behavior coordination is implemented through the high level task decomposition layer which is located in the top level of the behavior-based control layer. The behavior communication layer sends its received information to the group behavior layer for low level control. Explicit communication and implicit communication are the two types of communication. Explicit communication concerns sending or receiving information via computer network data links. However, robots also have the ability to exchange information indirectly with its teammates by interacting with the environment or using sensing. For example, a robot can estimate its relative distance from another robot or from closest obstacle by using its CCD camera or laser distance finder.

2.2.3.4 Group Behaviors for Coordination

The group behavior layer is responsible for decomposing the abstract behaviors, which are received from the high level task decomposition layer though the communication behavior layer. The group behavior layer will carefully coordinate the activities of the robots with the purpose of managing and solving conflicts among robots and making them contribute to the common tasks. The learning or coordination unit continuously decomposes subtasks 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.

2.2.3.5 Behavior Coordination for Object Transportation Subtask

A fully distributed multi-robot coordinated object transportation system is developed in this thesis. The behavior-based distributed control layer facilitates the implementation of the sub-tasks which are assigning by the learning/coordination units in the top level of the proposed architecture.

Figure 2.9 shows the behavior hierarchy in the behavior-based distributed control layer for the coordinated object transportation subtask. In the object transportation subtask, first the robots wander in the environment, looking to find an object of interest. The group behavior of the "Assume Formation" makes the robot move to an intended site around the transported object so as to establish a formation with its peer robots. The learning/coordination unit in the top level of the proposed architecture establishes the transportation formation of the robots; it decomposes into the specific position and orientation of the robots, and sends the determined subtasks to the behavior control layer for execution. When the "Assume Formation," parent behavior is activated, it will first turn on its child behavior of "Find Object." It attempts to search for the object to be transported. If this object found from the environment, it estimates the object pose (location and orientation) in the environment. A CCD camera and a laser range finder are used to accomplish this estimation. Simultaneously, the composite behavior "Find Object" will call the "Search Color Blob" composite behavior to make the robot to rearrange its pose continuously and try to locate the predefined color blob attached to the object and move it to the center of the camera frame, by using its CCD camera. The predefined color blob can be used to distinguish the object of interest from other objects or obstacles in the environment. Then the "Laser measurement" primitive behavior is triggered to estimate the pose (position and orientation) of the object using the laser range finder. After that the composite behavior "Find Obstacle" will call the "Laser measurement" primitive behavior to measure the distances to the obstacles from the object of interest, angles and sizes. Then, the behavior of "Move to a Location" will attempt to move the robot to the designated action position as decided by its higher level learning/coordination unit. The composite behaviors "Move" and "Obstacle Avoidance" are activated simultaneously, until the robot moves to its designated action position.

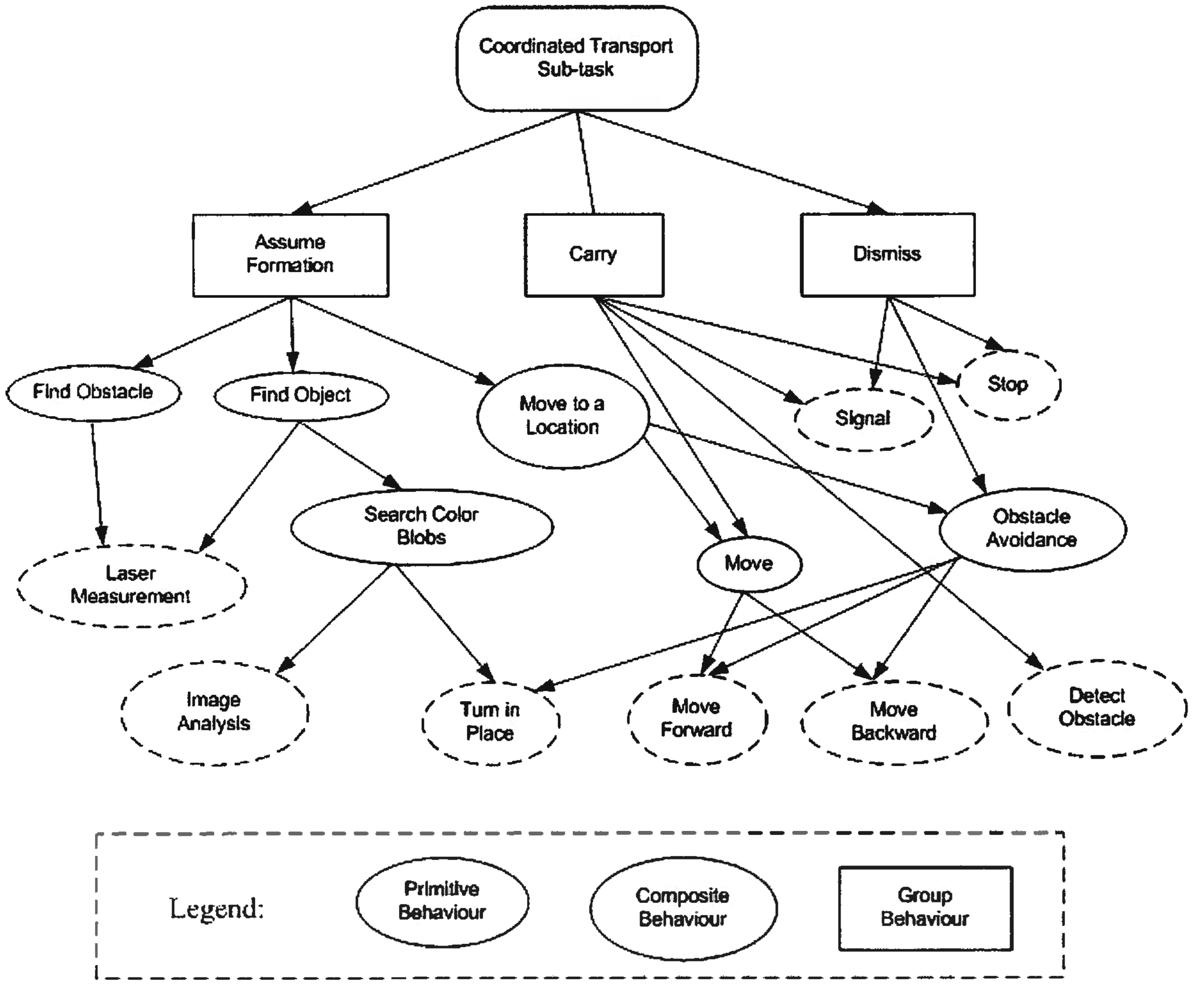


Figure 2.9: Behavioral hierarchy for the coordinated transport sub-task.

The group behavior "Carry" is activated after all robots reach their designated action positions, and creates a formation around the object of interest. When the "Carry" parent behavior is activated, it will first turn on its child behavior of "Signal" which attempts to start a synchronization process to transport the object. During the transportation process, the robots need to monitor whether the object collides with any obstacles in the environment. If the object collides with an obstacle, the robots will abandon the "Carry" behavior and inform the incident to their learning/coordination units in the top level task decomposition and assignment layer. Then learning/coordination units plan their formation again.

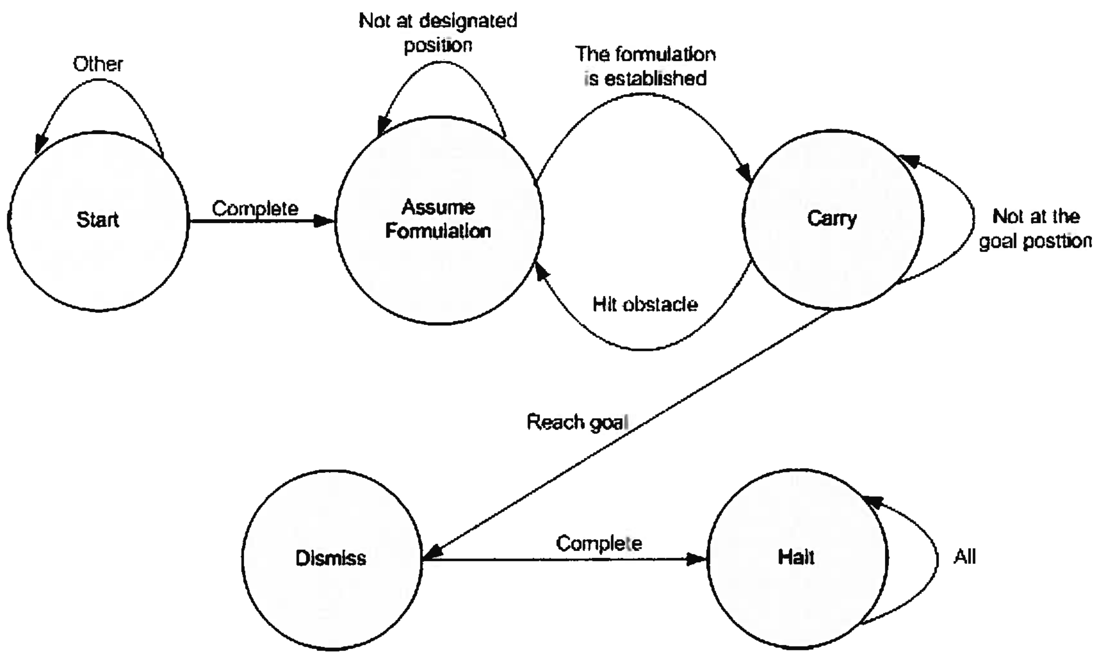


Figure 2.10: FSA describing the transitions of the group behaviors in the task of coordinated object transportation (Wang, 2007).

Finite State Diagrams (FSA) is a method for expressing robotic behaviors. Here, Finite State Diagrams (FSA) are 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 behavior 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 Figure 2.10.

2.3 Mechanism for Multi-Robot Coordination

A multi-robot coordination mechanism must cope with different types of requirements, such as robot positioning and synchronization. It should provide flexibility and adaptability, allowing multi-robot teams to execute tasks efficiently and robustly. To accomplish this, a dynamic role assignment mechanism is proposed in which robots can dynamically change their behavior, enabling them to successfully execute different types of cooperative tasks. This approach is validated using environmental state identification. A role is defined as a function like environment state identification, which one or more robots perform during the execution of a cooperative transportation task. The particular state identification stage consists of three roles (modes): s_1 - Wandering; s_2 - Object Identification, and s_3 - Obstacle Identification.

2.3.1 Dynamic Role Assignment Mechanism for Coordination Unit

In general, to execute a cooperative role a team of robots must be coordinated; specifically, they have to synchronize their activities and exchange information. In this approach, each robot performs a role that determines its activity during the cooperative task. According to its internal state and information about the other robots and the role received through communication, a robot can dynamically change its role. The mechanism for coordination is centralized, but it is activated only for a short time. As mentioned in subsection 2.2 the leader robot has the power to make decisions based on its local information and the information received from peers. Each team member has a specification of the possible activities that should be performed during each phase of cooperation in order to complete the state identification task. These activities must be specified and synchronized considering several aspects, such as robot properties, task requirements, and characteristics of the environment. The dynamic role assignment will be responsible for allocating the correct activities to each robot and synchronizing the cooperative execution though the leader robot in the system.

Before describing the details of the role assignment mechanism, it is necessary to define what a role is in a cooperative transportation task. A role is defined as a function that one or more robots perform during the execution of the state identification stage, which is a cooperative robotic task according to the modified Q learning algorithm. Each robot will perform a role while certain internal and external conditions are satisfied, and will assume another role otherwise. The role will define the present activity of the robot in that instance. Here three distinct conditions are considered for role transition. CCD camera and laser range finder is used as sensors for this particular task, generating high level stimulus to the system.

The role assignment mechanism allows the robots to change their roles dynamically during the execution of a task, adapting their activities according to the information they have about the system and the task. Basically, according to the state identification stage; there are two ways of changing roles during the execution of a cooperative task. The simplest way is the **Allocation,** in which a robot assumes a new role after finishing the execution of another role. In the **Reallocation** process, a robot terminates the performance of one role and starts or continues the performance of another role. Figure 2.11 shows a diagram with the two types of role assignment. The vertical bars inside the robots indicate the percentage of the role execution that has been completed.

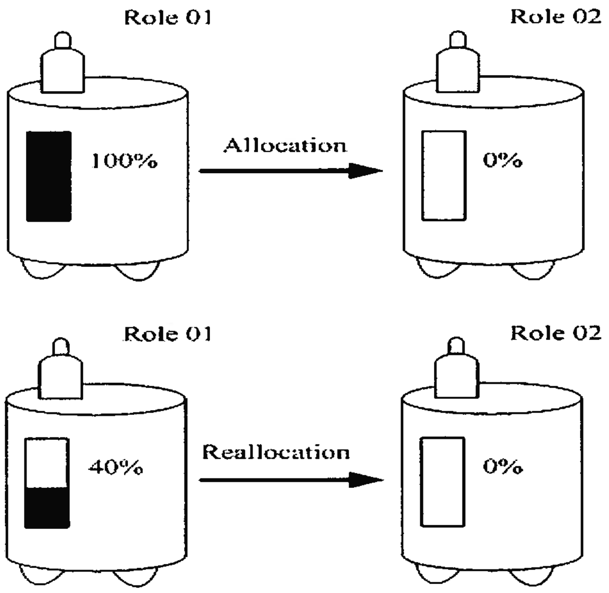


Figure 2.11: Types of role assignment.

The control software for a robot includes programs for each role it can assume. The local information consists of the robot's internal state and its perception about the environment. Global information contains data about the other robots and their view of the system and is normally received through explicit communication (message passing). A key issue is to determine the amount of global and local information necessary for the role assignment. This depends on the type of the activity that is being performed.

The present approach allows for two types of explicit communication: synchronous and asynchronous. In synchronous communication, the messages are sent and received continuously in a constant rate, while in asynchronous communication an interruption is generated when a message is received. Synchronous messages are important in situations where the robots must receive constant updates about the activities of the others. On the other hand, asynchronous communication is used for coordination when, for example, the leader robot needs to inform the others about the object pose or the leader robot need to interrupt the wandering role of the other robots, and so on.

An important requirement is to define when a robot should change its role. In the role allocation process, the robot detects that it has finished its role and assumes another available role. For example, consider the robots which wander in the work environment to find a color coded object. If a robot detects the color coded object then it needs to identify the object (particularly, estimate the object pose). In the reallocation process, the robots should know when to abandon the current role and assume another role. A possible way to do that is to use a function that measures the utility of performing a given role. A robot which performs role r has a utility given by μ_r. When a new role r' is available, the robot computes the utility of executing the new role $\mu_{r'}$. If the difference between the utilities is greater than a threshold: $\tau((\mu_{r'} - \mu_r) > \tau)$ the robot will change its role. The function μ can be computed based on local and global information and may be different for different robots, tasks and roles. Also, the value τ must be chosen such that the possible overhead of changing roles will be compensated for by a substantial gain on the utility and consequently producing a better overall performance.

2.3.2 Dynamic Role Assignment Model for Coordination Unit

The dynamic role assignment can be described and modeled in a more formal framework. In general, a multi-robot system can be described by the robotic state (X), which is a concatenation of the states of the individual robots:

$$X = [x_1, x_2, ..., x_n]^T \tag{2.4}$$

Considering a simple control system, the state of each robot varies as a function of its continuous robotic state (x_i) and the input vector (u_i). Also, each robot may receive information about the other robots in the system $(\hat{z}_i)$. This information consists of estimates of the robotic states of the other robots that are received mainly through communication. We use the hat $(\hat{})$ notation to emphasize that this information is an estimate because the communication can suffer delays, failures, and so on. Using the role assignment mechanism, in each moment a robot will be controlled using a different continuous equation according to its current role in the state identification stage. Therefore, we use the subscript q to indicate the current role of a robot. By following this description, the state equation of robot i during the execution of the task is given by:

$$\dot{x}_i = f_{i,q}(x_i, u_i, \hat{z}_i) \tag{2.5}$$

Since each robot is associated with a control policy, we have

$$u_i = k_{i,q}(x_i, \hat{z}_i) \tag{2.6}$$

and since $\hat{z}_i$ is a function of the state X, we can rewrite the state equation:

$$\dot{x}_i = f_{i,q}(X) \tag{2.7}$$

or, for the whole team,

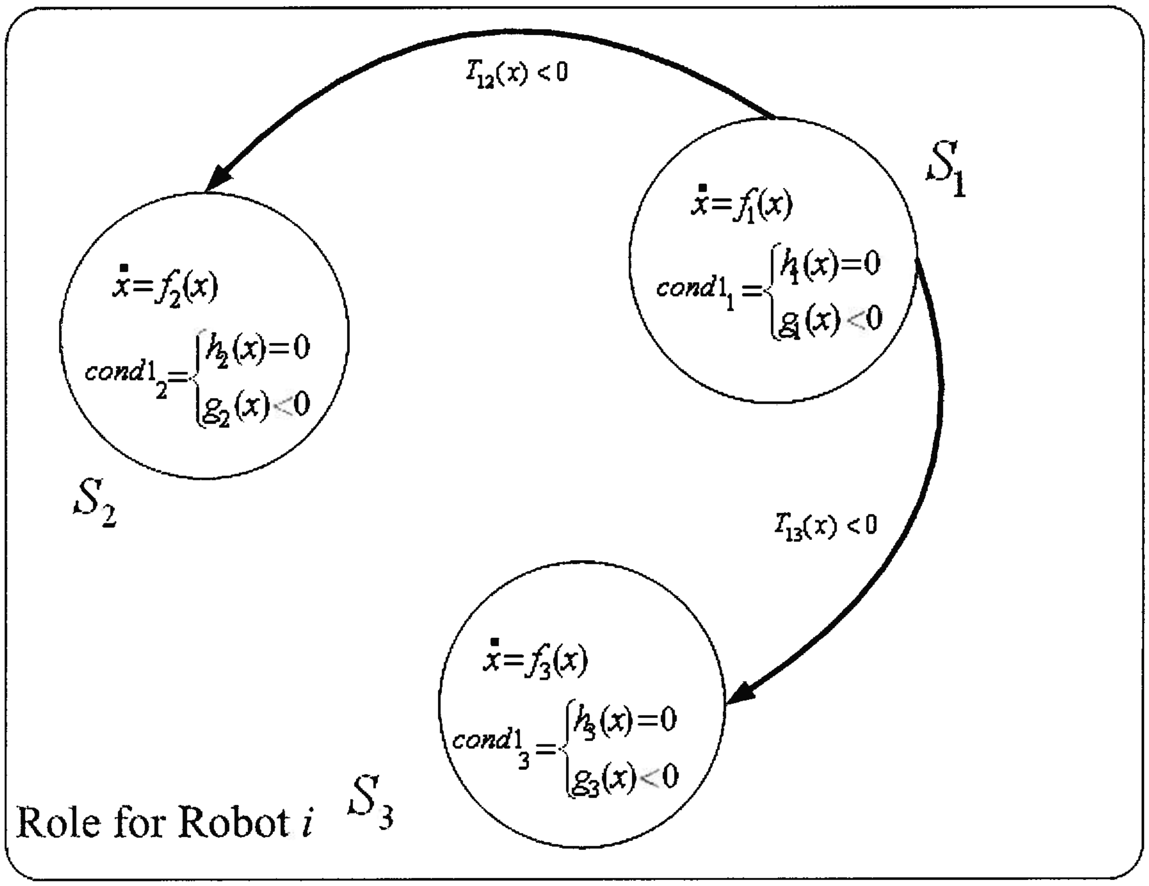
$$\dot{X} = F_q(X), where F_q = [f_{1,q}, ..., f_{n,q}]^T \tag{2.8}$$

The continuous behavior of each robot (it represent the robotic state at any instance) is modeled by these equations. Based on these equations, the continuous actions of the whole team of robots are implemented during the execution of a cooperative state identification stage. These equations, together with the roles, role assignments, continuous and discrete variables, communication, and synchronization can be better understood and formally modeled using a function Δ, as described next. This function consist of a finite number of real-valued variables that change continuously, as specified by differential equations and inequalities, or discretely, according to specific assignments. The state identification stage consists of both discrete states (roles) and continuous states (robotic states). Therefore, the nature of the particular multi-robot system in state identification stage is a hybrid, given by

$$\Delta = \{S, V, T, f, cond1, cond2, init, R\} \tag{2.9}$$

Here $S = \{1, 2, ..., k\}$ is the set of roles and this is a discrete set of states of the robot. According to Figure 2.12, a role is composed by other roles, which are called hierarchical/sequential compositions. Therefore, variables of the system V $(V = V_d \cup V_c)$ can be composed by discrete information (V_d) and continuous information (V_c) . V_c represents the robotic states and V_d represents the sensory information within each role. The dynamics of the continuous information are determined by the flows f . Generally continuous information is updated according to the differential equations inside each role (f_q). Discrete transitions between pairs of roles (p, q) are specified by transition T, which represents the role assignment. $cond1$ and $cond2$ are predicates related to the set of roles (S) and transition T , respectively and both $cond1$ and $cond2$ are defined when each robot will assume a new role. The system can stay in a certain control mode while its $cond1$ is satisfied, and can take transition when its $cond2$ (jump condition) is satisfied. The initial states of the system are given by $init$, and each transition T can also have an associated reset statement R, to change the value of some variable during a discrete transition using discrete information. Furthermore, variables are also updated according to the reset statement of each discrete transition. Finally, we can model the cooperative state identification stage execution using a parallel composition.

Figure 2.12 shows a role composed by the three roles: s_1 - Wandering; s_2 - Object Identification; s_3 - Obstacle Distribution. Each role is characterized by differential equations and algebraic constraints. The condition under which transitions between roles occur and the conditions under which robots stay in a particular role, are specified.



$T_{12}(x) < 0$
S_1
$\dot{x}=f_1(x)$
$cond1_1=\begin{cases} h_1(x)=0 \\ g_1(x)<0 \end{cases}$
S_2
$\dot{x}=f_2(x)$
$cond1_2=\begin{cases} h_2(x)=0 \\ g_2(x)<0 \end{cases}$
$T_{13}(x) < 0$
S_3
$\dot{x}=f_3(x)$
$cond1_3=\begin{cases} h_3(x)=0 \\ g_3(x)<0 \end{cases}$
Role for Robot i

Figure 2.12: Schematic representation of a role composed by three roles.

Communication among robots can also be modeled in this framework. We use a message passing mechanism to exchange information among the agents. To model message passing in a hybrid automaton, we consider that there are communication channels between agents, and use the basic operations send and receive to manipulate the messages. In the hybrid automaton, messages are sent and received in discrete transitions. These actions are modeled in the same way as assignments of values to variables (reset statements). It is rather common to use a self transition, i.e., a transition that does not change the discrete state, to receive and send messages.

2.4 Distributed Control Issues for Multi-robot Coordination

Normally, a multi-robot system with a distributed control architecture does not employ a leader robot to make decisions for all other robots in the system. The proposed hierarchical multi-robot architecture is also distributed control architecture. In a distributed control system, 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 coordination unit in the proposed architecture which requires centralized decision making when the robots need to identify the environmental state. However, the centralized coordination unit works only for the state identification phase which exists for a very short period of time compared to the transportation task. Once the multi-robot system leaves the state identification stage, the learning unit will takeover the control of the robot again. A distributed architecture has many advantages over a centralized control architecture, especially for multi-robot systems.

First, a distributed multi-robot system has an ability to increase the number of robots in the robot team without significantly increasing its computational cost. This represents the scalability. For example, the architecture proposed in this chapter can be expanded from to 3 to 20 robots without serious problems. If a member robot quits or a new robot enrolls into the system during the transportation process, the system is capable of adapting to changes of the team size dynamically. For a centralized multi-robot system, scalability is a difficult issue because the computational complexity of a centralized decision making algorithm increases rapidly with the number of robots.

Second, distributed systems have a low communication volume. A decentralized distributed system does not assign a leading robot. Therefore, bottlenecks in communication can appear. However, a centralized system has a central leader and when the number of robots in the system increases the communication bottleneck can be so serious that the algorithm becomes worthless. Furthermore, distributed systems are capable of accomplishing some of the cooperative tasks even without communication. Particularly, most realistic cases of mobile robot systems are expected 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, the bottom of a deep ocean or on a planetary surface. If the system does not establish reliable communication among robots, centralized decision making becomes weak because the robot members cannot receive commands from their leader in a timely and accurate manner. However, distributed multi-robot system may work well even when communication among robots is not reliable.

Third, the distributed systems are robust. Each robot in the system can make their decisions individually to accomplish a common task. When one robot in the system malfunctions or quits, other robots still can make decisions by themselves to continue their tasks. However, in a centralized system one leader robot makes decisions for all other robots in the team. Hence, once the leader robot fails, the whole system will fail entirely. Therefore, such systems are not robust compared to distributed control systems.

Low efficiency is a major disadvantage of a distributed system, where multiple robots reach their agreements through some cooperation technique such as voting, action, and so on. These cooperation approaches are usually time consuming and are not efficient. Normally, distributed learning algorithms are much more difficult and complex than centralized equivalents. Therefore, a distributed system is usually not optimal, because the final policy reached by the members is not optimal. It is still an open question as to how to implement efficient and optimal distributed decision making.

2.5 Summary

In this chapter, a three layer distributed multi robot architecture, which accommodates distributed machine learning, behavior-based distributed control and low level "hard" control, was developed. The objective of the architecture is to support multi-robot systems to properly coordinate the robotic tasks in a dynamic and unknown environment. The proposed architecture is a general platform that merges several existing techniques of multi-robot decision making, and accommodates homogenous or heterogeneous robots in the same framework.

Before proposing the architecture, several popular multi-robot coordination approaches given in the literature were discussed, and their advantages and disadvantages were compared. The discussed multi-robot coordination approaches are the behavior-based approach, the planning-based approach, the distributed control-based approach, the market-based approach, and the learning-based approach.

Next, the machine learning unit and the coordination unit were introduced. In the proposed architecture, an arbitrator dynamically selects either the learning unit or the coordination unit to make decisions for the robots. Then the behavior-based control layer and their sub-layer were presented. In this thesis, it is combined with the learning and coordination units in the top level of the architecture to control the actions of a robot. After that, the mechanism of multi-robot coordination was presented. This framework, which facilitates a group of robots to coordination their activities, allows global sharing of information during the stage of object environment state identification. Finally, some distributed control issues of multi-robot systems were discussed. The advantages and disadvantages of both centralized and distributed architectures were pointed out and some open questions in this field were identified.

Chapter 3

Machine Learning for Multi-Robot Decision Making

3.1 Overview

Multi-robot decision making is a challenging task in many real world problems, which involve real-time operation of robotic systems in uncertain and dynamic environments. In these problems, actions and interactions of multiple mobile robot have to be considered concurrently so as to reach a global optimal policy. Among the available approaches, planning-based approach is a contender for dealing with this problem. However, planned optimal policy usually is feasible in static environments, and it is shown that the planning-based approach is inapplicable to fast changing environments.

Current approaches for decision making in uncertain and dynamic multi-robot environments such as planet surfaces and urban environments have two main difficulties. First, many approaches do not take into account the uncertainty inherent in these problem domains. For example, in the multi-robot environment, terrain features can change continuously. Second, for systems with many robotic agents, state of the art methods can grow to be too cumbersome to be and suboptimal. For example, even though the physical environment is stationary, from the perspective of a single robot, the particular static environment is still a dynamic environment because the other robots of a multi-robot system take actions, continuously making changes to the environment.

In view of these considerations, the behavior-based approach and the machine learning approach are the primary approaches that are capable of overcoming the challenges of multi-robot systems operating in dynamic, unstructured, and unknown environments. As mentioned in the previous chapter, the behavior-based approach is more robust for a dynamic environment than the planning approach. It is so because it can quickly adapt to a fast changing environment, by identifying the current environment state and couple it with a specific action. In addition, the behavior-based approach is known to become unsuccessful in a complex environment with a large number of environment states.

However, it is highly effective in simple environments. Particularly, the behavior-based approach needs a human designer to design all behavior rules and the designer needs to foresee all possible world states that the robots will encounter. Therefore it is almost impossible for this approach to be effective in a dynamic and unknown environment with a large number of states.

Considering how humans accomplish physical tasks in a dynamic environment, it is not difficult to come to the conclusion that a human employs not only planning or reactive (behavior-based) techniques but also learning techniques. Humans identify the new environment states through their learning capabilities, come across corresponding optimal actions from the past experience, and improve their planning and reactive techniques. In other words, humans explore the unexpected environment states with less uncertainty and make their decisions more robust and effective in a dynamic environment.

Therefore, machine learning has become an important subject in the in the multi-robot field. Machine learning is considered a vital development due to the highly dynamic nature of typical multi-robot environments. Among the existing machine learning approaches, reinforcement learning (RL), especially Q learning, is the most commonly used one in multi-robot systems due to its simplicity and good real time performance.

The single agent Q learning algorithm is the most common Q learning algorithm. If we directly extend the single agent Q learning algorithm to a multi-robot system, the assumption of static environment is violated, and as a result the values of the Q-table may not converge. Although the robots still can find some good policies for cooperation, the performance of the whole team of robots can be degraded when that approach is extended in this manner. (Wang and de Silva, 2007)

Therefore, an improved version of the single agent Q learning algorithm is proposed here for robot decision making, which can come up with an optimal solution to the action selection conflicts of the robots in a multi-robot system. Hence the proposed Q learning algorithm is suitable for real-time operation of cooperating multi-robot systems.

In this chapter machine learning is used to make decisions for a multi-robot object transportation task. Markov decisions processes (MDP) and the single agent Q Learning algorithm are introduced in section 3.2 and section 3.3, respectively. The single agent Q learning algorithm will converge to an optimal policy in an MDP problem without a transition model. Modified Q learning algorithm is presented in section 3.4 and it is validated though computer simulation. Section 3.5 presents the distributed decision making unit of Q learning-based multi-robot object transportation, which is validated though computer simulation.

3.2 Markov Decision Process (MDP)

The Markov decision process framework for decision making, planning, and control is surprisingly rich in capturing the essence of purposeful activities in various practical situations. MDP models and associated problem arise in many areas, including medical decision making, maintenance planning, and mobile robot navigation. Therefore MDP is a popular topic in artificial intelligence and it facilitates an intelligent agent to make decisions in real-world problems. The basic concept of MDP is introduced now.

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

- $S = \{s_1, s_2, ..., s_n\}$, is a set of states of the environment
- $A = \{a_1, a_2, ..., a_n\}$,is a set of actions by the robot
- $T : SxA \rightarrow \Pi(s)$, 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 s.
- R: S x A $\rightarrow \Re$, is a reward function. When the robot takes an action a_i under the current state s, it determines the immediate reward.

MDP presents a sequence of decisions made by an agent or a robot in an environment with many states. At time *t*, the agent or a robot requires 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 denotes the action needed to be taken by the robot or agent to reach the desired state. Further, there is a reward function which denotes the immediate reward when the agent or robot takes an action a under the current state s.

Two major attributes of MDP make it interesting and challenging. First, that particular subsequent state s' is not deterministic when the agent or robot obtains 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 $T(s, a, s')$. Second, MDP's transitions among states are Markovian; that is, the probability of reaching s' from s depends only on s and not on the history of previous states.

The performance of an agent or robot policy π is measured by the rewards it received when it made a sequence of decisions according to the policy and the sequence of states it visited. 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 probabilistic nature of the transition function $T(s,a,s')$, given a policy, the sequence of states visited by the agent or robot each time is not fixed and it 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, the way to find all optimal policies has to be carefully considered, if more than one optimal policy exists. The value iteration algorithm has been developed to find the solution to this problem (Russel and Norvig, 2003). In this 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, action selection (agent's or robot's decision) will become easy; specifically, to select the action that maximizes the expected utility of the subsequent state:

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

However, determination of the utilities of all states is not an easy task. Bellman found that the utility of state could be divided into two parts: the immediate reward for that state and the expected discounted utility of the next state, as given by

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

Equation (3.5) is the Bellman equation. There is a corresponding Bellman equation for each state. For example, if there are n states totally, then there are n number of Bellman equations for the states. 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 iterative techniques. The value iteration algorithm is an approach to find the utilities of all states, and is given below.

```
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
            $U'(s) \leftarrow R(s) + \beta \max_a \sum T(s,a,s')U(s')$
            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 (Russel and Norvig, 2003).

3.3 Reinforcement Learning

The value iteration algorithm is introduced in section 3.2 to solve MDP problems. However, the environment model $T(s,a,s')$ and the reward function $R(s)$ are usually unknown in a real robot decision making situation. Therefore, the value iteration algorithm is not practically viable for real robot decision making. Usually, robots are unable to obtain perfect information of the environment model for employing the value iteration algorithm to solve the Markov Decision Process (MDP) problem. Reinforcement Learning is a viable solution to overcome this challenge.

In reinforcement learning, an agent or a robot observes a sequence of state transitions and receives rewards, through trials of taking actions under different states in the environment. This can be used to estimate the environment model and approximate the utilities of the states. This implies that reinforcement learning is a model-free learning approach, which is an important feature. There are numerous variants of reinforcement learning among which Q-learning is the most common algorithm. A typical Q-learning algorithm is introduced below.

```
•   For each state $s_i \in (s_1, s_2, ..., s_n)$ and action $a_i \in (a_1, a_2, ..., a_m)$, initialize the table
    entry $Q(s_i, a_j)$ to zero. Initialize $\tau$ to 0.9. Initialize the discount factor $0 < \beta \le 1$
    and the learning rate $0 < \eta \le 1$
•   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 $Q(s, a_k)$
        • $Q(s, a_k) = (1-\eta)Q(s, a_k) + \eta(r + \beta \max_{a'} Q[s', a'])$
        • $s \leftarrow s', \tau \leftarrow \tau * 0.999$
```

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

An empty Q-table is set up and all its entries are initialized to zero at the initiation of the Q-learning algorithm. The Q-table is a 2D table. Its rows represent the environment states and its columns represent the actions available to the agent or the robot. $Q(s_i, a_j)$ represents the value of a Q-table entry and describes 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 it is for the agent or the robot 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 the operation, the agent or the robot observes the environment, determines the current state s, selects an action a_k probabilistically with probability given in equation (3.6) and executes it.

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

Once the agent or the robot executes the action a_k, it will receive a reward r from the environment and will observe the new environment state s'. Based on the information of the immediate reward (r) and the new environment state s', the agent will update its Q-table by utilizing the following equation:

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

After updating the Q-table, 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 or robot has an ability to select a unique action uniformly and randomly among all possible actions, and with probability 1-ε, the agent or the robot selects the action with a high Q value under the current state. In addition, the probability ε is decreased gradually. The benefit of the ε - 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 improves its Q table. In equation (3.6), the probability ε is controlled though decreasing the "temperature" parameter τ.

3.4 Modified Q Learning Algorithm

The mentioned single agent Q learning algorithm is the most common Q learning algorithm. It is employed to find optimal action policies for the members of a multi-robot system and it can help robots find good cooperation policies. However, this algorithm has some serious disadvantages in a multi-robot environment. In particular, directly extending the single agent Q learning algorithm to a multi-robot system violates its assumption of static environment, and as a result the values of the Q-table may not converge. Although the robots still can find some good policies for cooperation, the performance of the whole team of robot can be degraded when that approach is used. (Wang and de Silva, 2007)

Accordingly, a novel Q learning algorithm is developed, which is suitable for real-time operation of cooperative multi-robot systems. It determines the optimal solution to the action selection conflicts and provides a more realistic state identification methodology for a multi-robot system. The developed algorithm is suitable for real-time operation of cooperating multi-robot systems.

The single agent Q learning algorithm is improved and is named the "Modified Q learning algorithm." The basic idea of the modified 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. The group members typically do not observe the environment state individually and decide the environment state independently. Instead, one of them may handle one side, and another member may handle a complementary side. Then they share their reactive information and come to a conclusion about the present environment state. Likewise other group members typically do not select their pushing locations and forces concurrently. One person among 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 would determine their actions, and 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 modified Q-learning algorithm for multi-robot transportation tasks. A modified algorithm is presented in Figure 3.3.

- Assume that there are n robots, $R_1, R_2, ..., R_n$ which are arranged in a special sequence. The subscripts represent their positions in this sequence.
- $S_1, S_2, ..., S_m$ is the corresponding role set available for the robots. In particular, the robot R_i has S_j role available for the particular instant.
- Robots $R_1, R_2, ..., R_n$ exploring the environment for color coded object
- If the robot R_i detects the color coded object
    - R_i estimates the pose (position and orientation) of the object.
    - Pose information will be transmitted to $R_1, R_2, ..., R_n$
    - R_i assigned as the leader of the team.
    - $R_1, R_2, ..., R_{i-1} and R_{i+1}, R_{i+2}, ..., R_n$ transfer the decision rights to the coordination unit of the R_i
    - R_i should assign tasks to $R_1, R_2, ..., R_{i-1} and R_{i+1}, R_{i+2}, ..., R_n$ in the system who will assist in identifying the obstacle distribution around the object
    - $R_1, R_2, ..., R_n$ are transmitting their processed sensor information to the system.
    - $R_1, R_2, ..., R_n$ are will be able to establish the local environment state individually
- else If Robots $R_1, R_2, ..., R_n$ exploring the environment for color coded object

Figure 3.3(a): Environment state identification algorithm.

- Assume that there are n robots, $R_1, R_2, ..., R_n$ which are arranged in a special sequence. The subscripts represent their positions in this sequence.
- $A_1, A_2, ..., A_n$ are the corresponding actions sets available for the robots. In particular, the robot R_i has m_i actions available for execution as given by $R_i : A_i = (a_1^i, a_2^i, ..., a_{m_i}^i)$
- $Q_1(s_1, a_1), Q_2(s_2, a_2), ...Q(s_i, a_j)$ are the corresponding Q tables for the robots. All entries in the Q tables are initialized to zero.
- Initialize τ to 0.9.
- Ω is a set including all actions selected by the robots thus far, and ϕ represents the empty set.
- Do repeatedly the following:
  - o Initialize $\Omega = \phi$
  - o Observe the current environment state s
  - o For $(i = 1$ to $n)$
    1. Generate the current available action set $\Upsilon_i = (A_i - (A_i \cap \Omega))$
    2. The robot R_i selects the action $a_j^i \in \Upsilon_i$ with probability

       $$P(a_j^i) = \frac{e^{Q(s,a_j^i)/\tau}}{\sum_{l=1}^{m} e^{Q_l(s,a_l^i)/\tau}} \tag{3.8}$$

       Where $a_j^i \in \Upsilon_i (r = 1, 2, ..., k)$ and k is the size of the set Υ_i
    3. Add action a_j^i to the set Ω
  - a. End For
  - b. For each robot $R_i = (i = 1, 2, ..., n)$, execute the corresponding selected action a_j^i
  - c. Receive an immediate reward r
  - d. Observe the new state s'
  - e. For each robot $R_i = (i = 1, 2, ..., n)$, update its table entry for $Q_i(s, a_j^1, a_j^2, ..., a_j^i)$ as follows:

    $$Q_i(s, a_j^1, a_j^2, ..., a_j^i) = (1-\eta) Q_i(s, a_j^1, a_j^2, ..., a_j^i) + \eta[r + \beta \max_{a^1, a^2, ..., a^i} Q_i(s', a_j^1, a_j^2, ..., a_j^i)] \tag{3.9}$$

    where $0 < \eta < 1$ is the learning rate and $0 < \beta < 1$ is the discount rate.
  - o $s \leftarrow s', \tau \leftarrow \tau * 0.999$

Figure 3.3(b): Modified Q learning algorithm

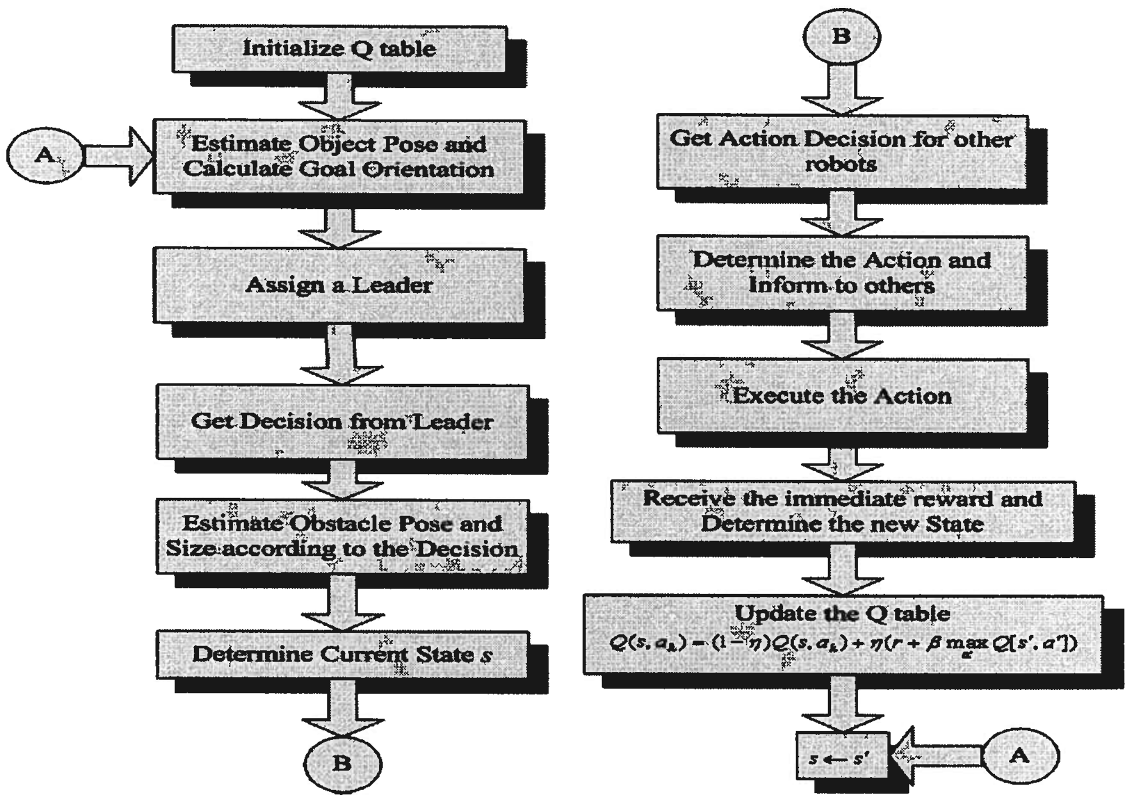

Initialize Q table
A
Estimate Object Pose and Calculate Goal Orientation
Assign a Leader
Get Decision from Leader
Estimate Obstacle Pose and Size according to the Decision
Determine Current State s
B
B
Get Action Decision for other robots
Determine the Action and Inform to others
Execute the Action
Receive the immediate reward and Determine the new State
Update the Q table
$Q(s, a_k) \leftarrow (1-\eta)Q(s, a_k) + \eta(r + \beta \max_{a} Q[s', a'])$
$s \leftarrow s'$
A

Figure 3.3(c): The flow diagram of the modified Q learning algorithm.

During operation, robots in the system start wander in the environment for color coded object to be transported. Once an object is detected by a robot in the system, it rearranges its pose to center the color blob in the camera frame. Then the appropriate robot estimates the pose (position and orientation) of the object and the information will be transmitted to the other robots. . The robot that identified the object will temporarily close the machine learning unit and transfer its decision rights to the coordination unit. Particular robot is assigned as the leader of the team for the state identification process and all other robots in the system will stop their learning units and transfer their decision rights to the coordination unit of the leader robot. Thereafter, the leader robot will assign tasks to peers in the system, enabling to identify the obstacle distribution around the object. All members in the system will transmit their information while processing the information continuously. All this information would determine the current local environment state s of the object individually by all robots. According to the present state, robots will individually select their respective actions.

In the action selection stage, a robot probabilistically and sequentially selects an action a_j^i with probability; and executes it. The proposed modified Q learning algorithm uses a sequential action selection method to overcome the action selection conflicts among robots. When two robots select the same action to transport the object to the same location, it violates the cooperation strategy. According to the proposed sequential action selection method, if one robot selects an action, it should transmit the information to the system. This would help to overcome the conflict.

After the agent takes the action a_k according to the sequential action selection method, 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

$$Q_i(s,a_j^1,a_j^2,...,a_j^i) = (1-\eta)Q_i(s,a_j^1,a_j^2,...,a_j^i) + \eta[r + \beta \max_{a^1,a^2,...,a^i} Q_i(s,a_j^1,a_j^2,...,a_j^i)] \quad (3.10)$$

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. Learning rate η determines to what extent the newly acquired information will override the old information. Discount factor β determines the importance of future rewards. The "temperature" parameter is τ. The developed approach is validated through experimentation.

3.5 Multi-Robot Transportation Using Modified Q Learning

Multi-robot object transportation is an important subject in multi-agent robotics. In an autonomous transportation system, several autonomous mobile robots move cooperatively to transport an object to a goal location and orientation in a static or dynamic environment while avoiding fixed or removable obstacles. It is a rather challenging task. For example, in the transportation process, each robot needs to sense the positions of the obstacles and the other robots in the neighborhood, and any change in the environment. Then it needs to communicate with the peers, exchange the sensing information, discuss the coordination strategy, suggest the obstacle avoidance strategy, assign or receive the subtasks and roles, and coordinate the actions so as to move the object quickly and successfully to the goal location. Such operations have many practical applications in such fields as space exploration, intelligent manufacturing, deep sea salvage, dealing with accidents in nuclear power plants, and robotic warfare.

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 and make the entire system increasingly flexible and autonomous. Although most of the existing commercial multi-robot systems are controlled remotely by a human, 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.

A schematic representation of the developed system is shown in Figure 1.2 and it is repeated in Figure 3.4.

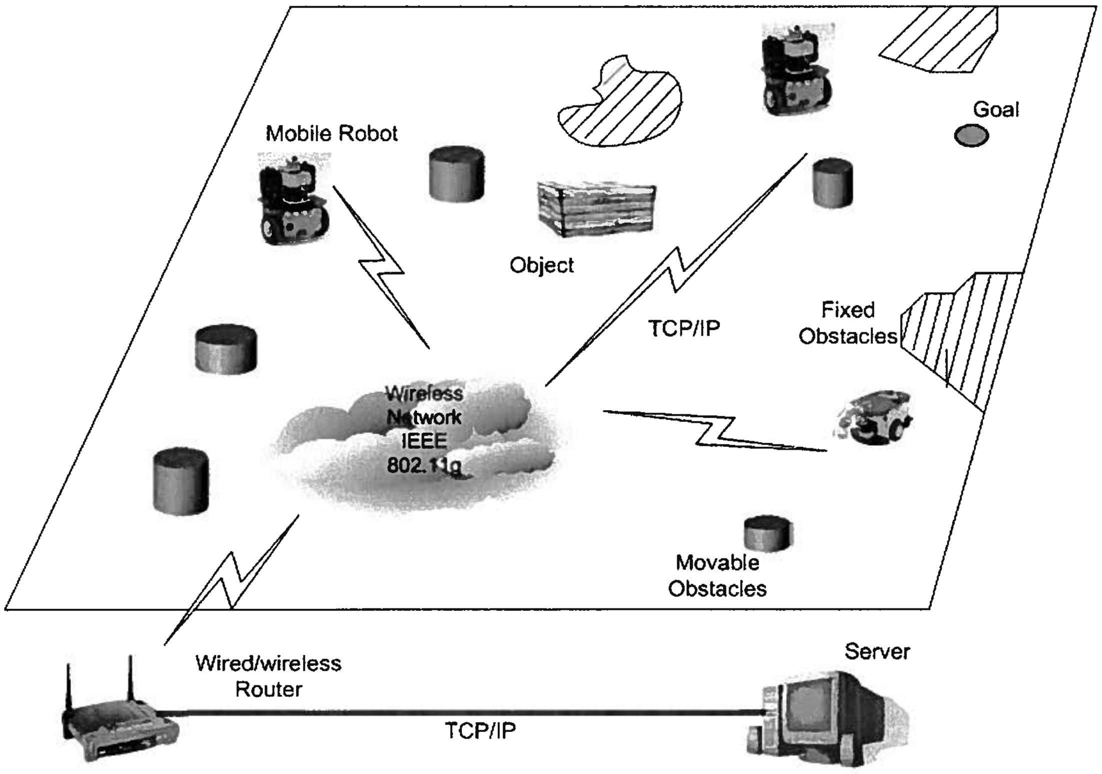


Figure 3.4: The developed Multi Robot System.

In this section, a multi-robot transportation system outfitted with distributed Q-learning is studied, which works in a dynamic and unknown environment with a complex obstacle distribution. Modified Q-learning algorithm is used to accomplish this task.

3.5.1 Distributed Multi-robot Q-leaning

There are two ways to extending single-agent reinforcement learning into the multi-robot field. One way is to treat multi-robot learning as a combination of multiple single-agent learning processes across the team members. The other way employs SG-based (stochastic game-based) reinforcement learning As mentioned above, Modified Q learning is also an improved version of typical single agent Q learning. Therefore the multiple single agent nature of the typical single agent Q learning algorithm plays a significant role in multi-robot systems. In this section we consider the combination of multiple single-agent learning type only. Each robot is equipped with the reinforcement learning technique of the multiple single agent learning type , 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.

In this method, 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. Consequence of this is a small learning space and a low computational cost. 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.

It follows that, "distributed learning" may find some utility in the multi-robot learning domain because of its simplicity. Distributed learning may work in a purely cooperative multi-robot task (Yang and Gu, 2004). Theoretical proof of its convergence under some conditions is available (Littman, 2001). On the other hand, "distributed Q-learning" is applied to a multi-robot object transportation task in a dynamic and unknown environment, which is a challenging and important research topic in multi-robot coordination systems. Next, some important issues of "distributed learning" are discussed and effectiveness, robustness and adaptivity are studied through computer simulation of a learning-based multi-robot system.

3.5.2 Object Transportation Task

Even though considerable work has been done in multi-robot object transportation, it is still difficult problem. In a task of object transportation, multiple autonomous robots have to cooperatively push an object to a goal location and orientation in a dynamic and unknown environment. Pushing is the transportation method used to accomplish the particular task. The particular object has a rectangular cross section. It is large and heavy, and a single robot is unable to accomplish the task alone. Furthermore, each robot only possesses local sensing capability and the obstacles may appear and disappear randomly in the transportation path. Therefore robots cannot plan the trajectory in advance. Figure 3.5 presents a dynamic and unknown environment in a continuous space. Both dynamic and static planning technologies will fail if the environment changes quickly and the computational speed of the robots is not sufficiently fast. The learning based approach may be a better solution to this challenging problem than a static/dynamic planning approach. The modified Q learning approach described in Figure 3.3 is utilized to execute this multi-robot object pushing problem.

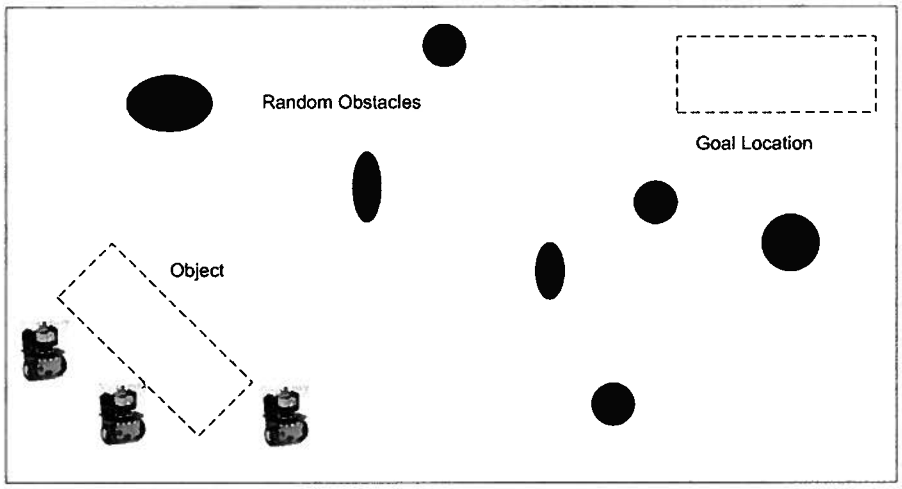


Figure 3.5: Description of a multi-robot object pushing task.

3.5.3 Multi-robot Object Pushing with Modified Q-learning

Modified Q-learning is used as the learning technique for the multi-robot object pushing problem. The operational method is as follows. Each robot is equipped with modified Q-learning. When the robots begin to transport the object, each robot identifies the current environmental state cooperatively utilizing the state identification technique mentioned earlier, which consists of the object pose estimation technology and obstacle identification technology, and then individually develops the current environment state as a binary word (Details are given in subsection 3.5.3.1). Next, the robots independently employ the Modified Q-learning algorithm given in Figure 3.3 to select their actions sequentially to push the object. After the actions are taken, each robot cooperatively utilizes the state identification technique and observes the new environmental state individually and receives its immediate reward$r$. The robots then update their own Q-tables with the reward and the new environment state information, and continue with the next step of object-pushing until the object reaches to goal location. Each robot individually maintains their Q-tables and makes their decisions. Therefore, it is a fully distributed learning-based decision making system. In other words, the multi-robot object pushing problem is solved with a purely learning-based approach.

However, the proposed "Distributed Reinforcement Learning" system also violates the assumption of stationary environment in the MDP problem. While a robot is pushing the object of interest, its peer robots are also pushing it. Therefore, the environment is not stationary anymore from the viewpoint of this robot. Furthermore, the random obstacles make the environment more complicated. However, the simulation results in the following section will show that the object pushing task can be completed successfully albeit with less efficiency.

3.5.3.1 The Environmental States for Modified Q Learning

The Modified Q-learning algorithm is employed in the task of multi-robot object pushing. As mentioned before, environmental state identification plays a vital role in accomplishing the overall task. Normally, a multi-robot working environment is continuous and complex. Therefore, a methodology is needed to express the continuous environment where the infinite number of environmental states have to be represented by a finite number of states. The environmental state is coded into 16-bit binary code as shown in Figure 3.6.

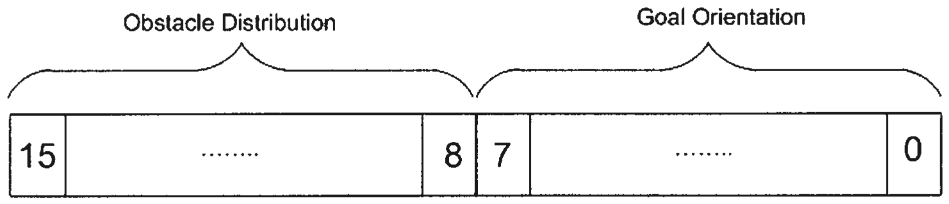


Figure 3.6: Environmental State Representation using Binary Code.

The current obstacle distribution around the object within the detection radius is represented by the higher 8 bits (from bit 8 to bit 15) in the binary code. The orientation of the goal location relative to the current object position and orientation (pose) is represented by the lower 8 bits (from bit 0 bit 7) in the binary code. The coding protocol is presented in Figure 3.7.

The coding protocol assumes that the robots only possess local sensing abilities. Consequently, the system needs to identify the local environment and detect the obstacles that appear within the detection radius. In order to describe the obstacle distribution in Figure 3.7, 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 3.7, the higher 8-bits of the environmental state is "10100100" which indicates the current obstacle distribution around the object within the detection circle.

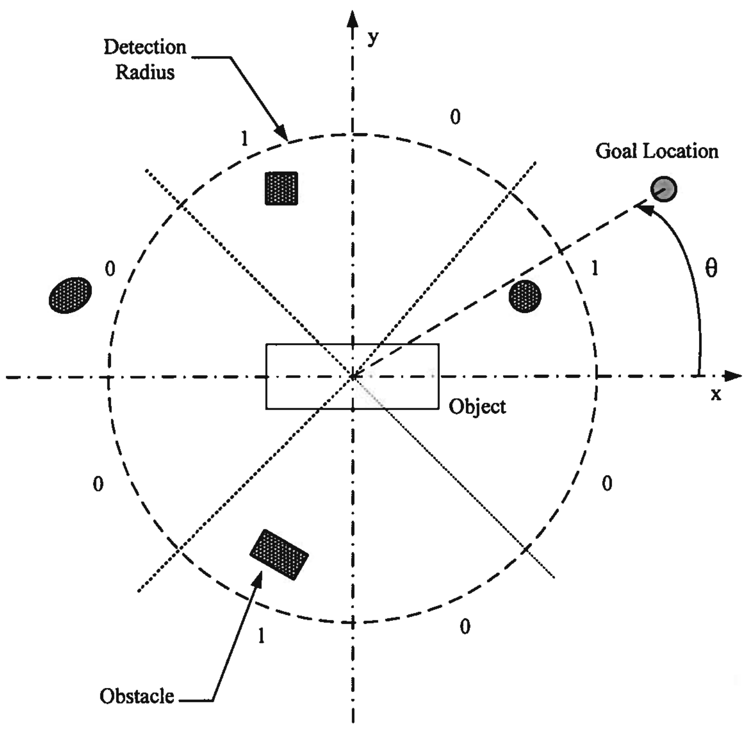


Figure 3.7: Coding protocol of the environment states.

The orientation of the goal relative to the current object pose is represented by the lower 8 bits of the environmental state code. The orientation angle θ of the goal in Figure 3.7 is used to calculate the lower 8 bits of the state using the following equation:

$$State_{bit(0-7)} = \text{int}(\theta / (360.0/256)) \tag{3.11}$$

int() function is used to convert a float value into an integer number.

3.5.3.2 Robot actions

Normally, a robot has the ability to push an object of rectangular cross section at any contact point on the surfaces of the object. Therefore, the object provides an infinite number of action points for the robots to push. However, a finite number of action points needs to be defined to implement the modified Q learning algorithm. Accordingly, only six action points (a1~a6) will be available to each robot, as shown in Figure 3.8 (Wang, 2007).

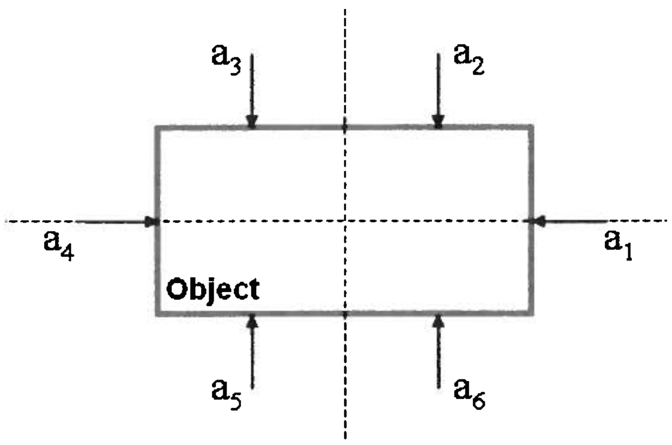
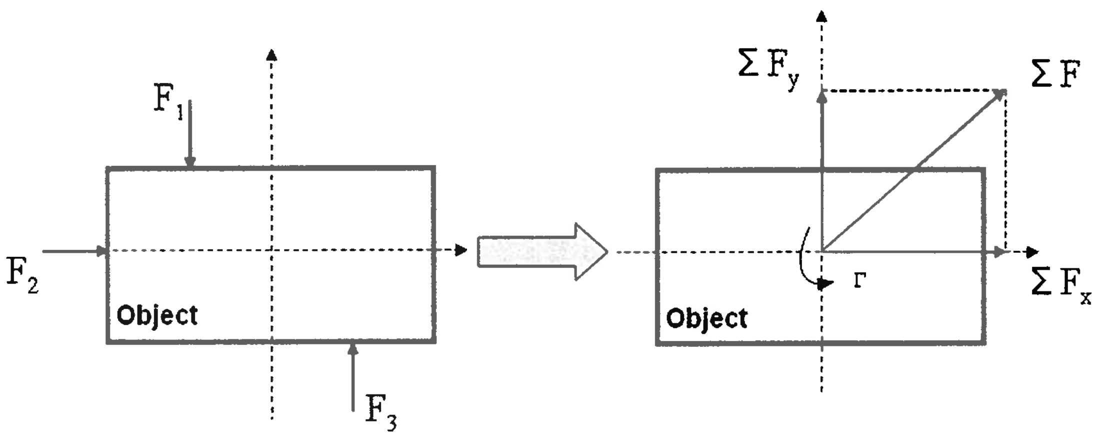


Figure 3.8: Pushing action points available to each robot.

3.5.3.3 Object dynamics

This section discusses the application of the modified Q learning algorithm to the multi-robot object pushing problem. The object dynamics is simplified in the simulation system. It is assumed that the object displacement is proportional to the net force exerted by the robots, and the rotational angle of the object is also proportional to the net torque exerted by the robots. In Figure 3.9, the dynamics of the object is illustrated, where the object is being pushed by three mobile robots (Wang, 2007).



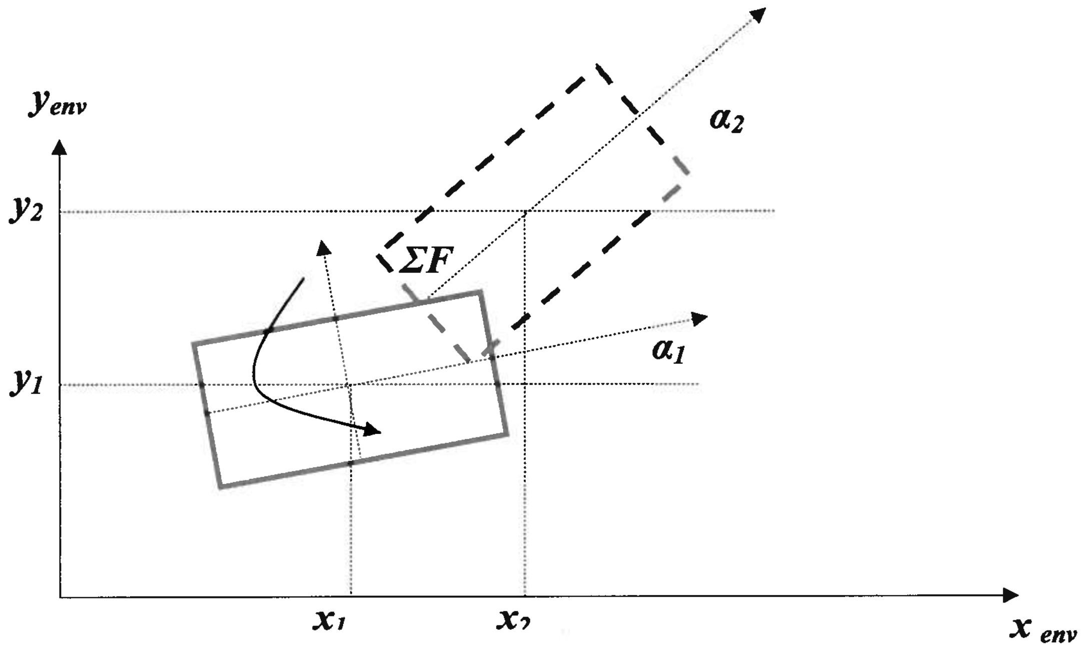


Figure 3.9: Dynamics of the object with rectangular cross section.

According to Figure 3.9, three robots push the object with rectangular cross section and F_1 to F_3 are their pushing forces. The object dynamics is described by the following equations: (Wang, 2007)

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

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

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

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

3.5.3.4 Reward function

A common immediate reward policy is presented in this section. Specifically, each robot receives the same reward after it pushes the object for a short period of time Δt. The reward consists of the following three parts: (Wang, 2007)

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

$$R_{distance} = (D_{old} - D_{new})c_d \tag{3.15}$$

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

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

$$R_{rotation} = \cos(|\alpha_2 - \alpha_1|) \text{ with } 0 \leq |\alpha_2 - \alpha_1| \leq \pi \tag{3.16}$$

where, $(\alpha_2 - \alpha_1)$ is the rotational angle of the object, 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 of the object difficult for the robots.

(3) The reward for obstacle avoidance

$$R_{obstacle} = \begin{cases} 1.0; if\_no\_obstacle \\ 1.5(\sin(\psi / 2)) \end{cases} \tag{3.17}$$

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 to w_3 are the weights and $w_1 + w_2 + w_3 = 1.0$.

3.5.4 Simulation Results

The simulation system is developed to validate the proposed modified Q learning technique. Microsoft Visual C++ language is used to program a learning-based multi-robot object pushing system, which has been described in the previous sections. The dimensions of the simulated environment are 1000×700 units and the dimensions of the object with rectangular cross section are 120×80 units. Gray color circles with a radius of 2 units are used to represent the three robots, which are used for the simulation system in the subsequent figures to push the object to the goal location. The object with a rectangular cross section is large and heavy, and an individual robot cannot handle it without the help of its peers.

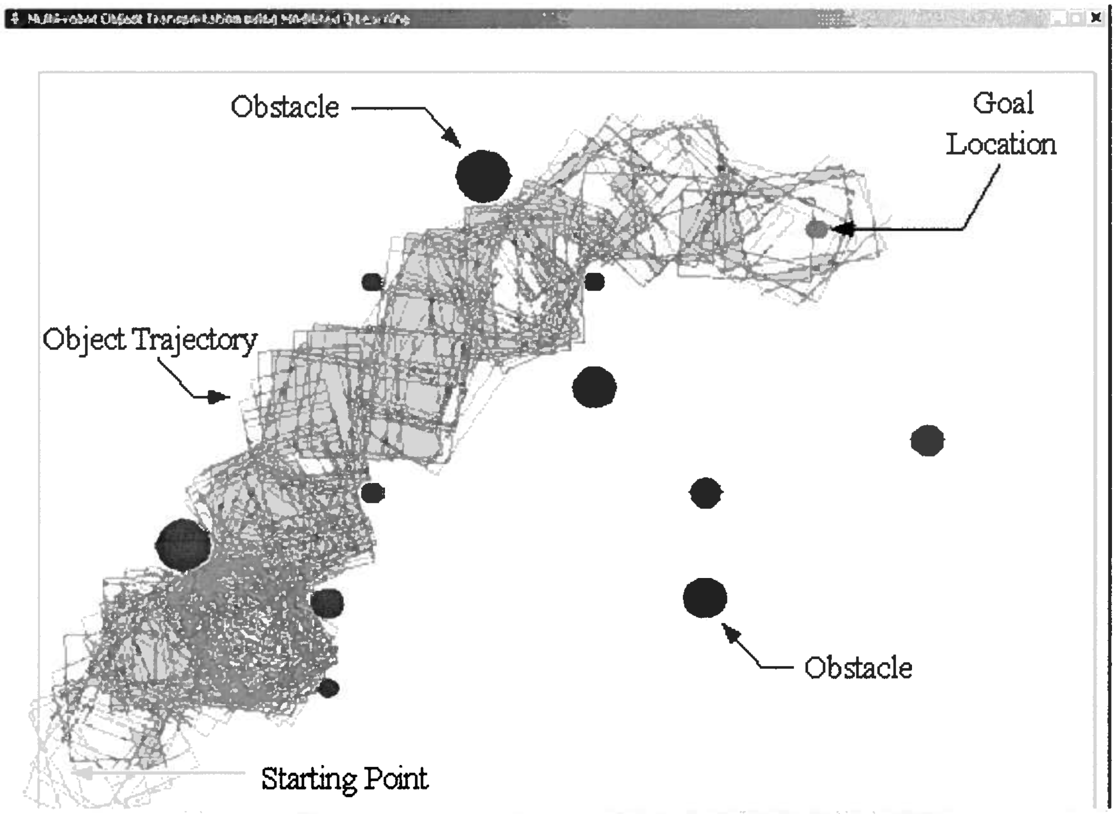
During the simulation, based on the available environmental state, the robots sequentially select their actions using the modified Q-learning algorithm. After selecting their actions sequentially, they begin to push the object after synchronization. The object is pushed for a short period of time Δt, and the same reward is given all the robots. Next, observing the new environmental state, based on the reward and the new environmental state, the robots independently update their Q-tables. Next step of pushing starts, and so on, until the goal location is reached or the total number of steps exceeds the maximum limit.

The object pushing is completed ones the object reaches the goal location or the total number of steps for one round exceeds the limit. When a round of object pushing is completed, a new round of object pushing is started. In addition, the starting location of the object, the goal location, and the obstacles are selected randomly, every time the next round of object pushing is started. However, the Q-table of each robot is inherited form the last round. The simulation system is stopped after the total number of rounds reaches the desired number.

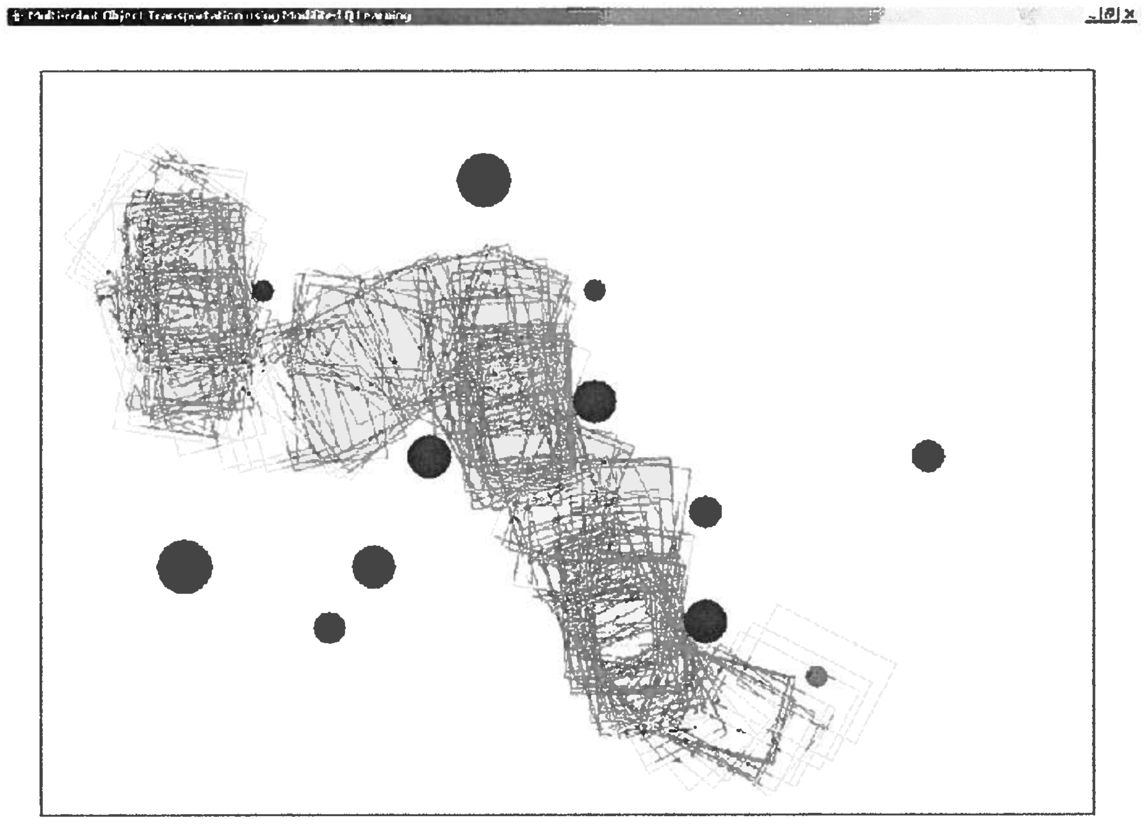
The parameter values of the simulation system are: $\beta = 0.9$, $\tau = 0.9$, $\eta = 0.8$, $c_f = 0.275$, $c_t = 0.0175$, $c_d = 0.5$, $w_1 = 0.6$, $w_2 = 0.04$, $w_3 = 0.36$, and $l = 10$. Eleven obstacles appear randomly in the environment during the simulation, and the maximum number of steps per round is 6000.

3.5.4.1 Simulation Results for Object Pushing

Two different object pushing snapshots under different obstacle distributions and goal location are presented in Figure 3.10. Three robots represented by very small circles are located on the sides of the object. The eleven large blue color circles represent the obstacles and the red color circle represents the goal location in the environment.



(a). Snapshot of a successful object pushing task



(b). Another successful task with different obstacle distribution and goal location

Figure 3.10: Two object-pushing snapshots under different conditions.

The three robots first cooperatively push the object upwards to the goal location in the top right corner, as presented in Figure 3.10 (a). During the object pushing process, the object reaches an obstacle in the transportation path. Successfully avoiding the obstacle after several failed trials, the robots select to slightly push the object downwards to avoid the obstacle. The object is pushed by the robots upward again to reach the direction of the goal location. However, before the object reaches the goal location, it reaches another couple of obstacles. The robots again adjust their actions according to the new environment states and avoid the obstacles during transportation. Finally, the object is successfully pushed to the goal location.

Figure 3.10(b) presents another object pushing task with a different obstacle distribution. Both these results demonstrate that the modified Q learning algorithm which is also a single agent reinforcement learning algorithm, can be extended to multi-robot object pushing in an effective and successful manner. In the process of pushing and obstacle avoidance, the robots demonstrate a flexible and adaptive sequential action selection strategy in a complex and unknown environment. In addition, the system employs a fully distributed multi-robot decision-making system with local sensing capability. 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 respect. It can suffer from the lack of local sensing. However, the particular problem can be solved by increasing the detection radius. That will lead to rapid expansion of the learning space, which can make the algorithm ineffective.

3.5.4.2 Effectiveness of Modified Q-learning in a Multi-Robot Cooperative Task

As mentioned before, immediate reward is a same common reward and it is distributed among the robot. Because a robot's reward is affected by the action of its peers, the assumption of a stationary environment is violated in the Modified Q-learning algorithm as well. This means the particular algorithm has less sensitivity in a multi-robot environment. However, it was possible for single agent reinforcement learning to learn the coordination among robots in a purely cooperative multi-robot task (Yang and Gu, 2004). This observation is verified with the multi-robot object pushing simulations presented in this section. Additionally, some probability analysis results are used to evaluate the Modified Q-learning in a multi-robot cooperative task as presented next.

For the present simulations, the environment state is selected randomly for analysis, whose state value represents the binary number "1000000000" or decimal number "512." Its corresponding Q-values for the six action candidates are also examined. According to the coding protocol of the environmental states in section 3.5.3.1, the binary value of "1000000000" represents an environmental state, which is shown in Figure 3.11. The robots are allowed to push the object continuously for 100,000 rounds to gain experience and update their respective Q-tables.

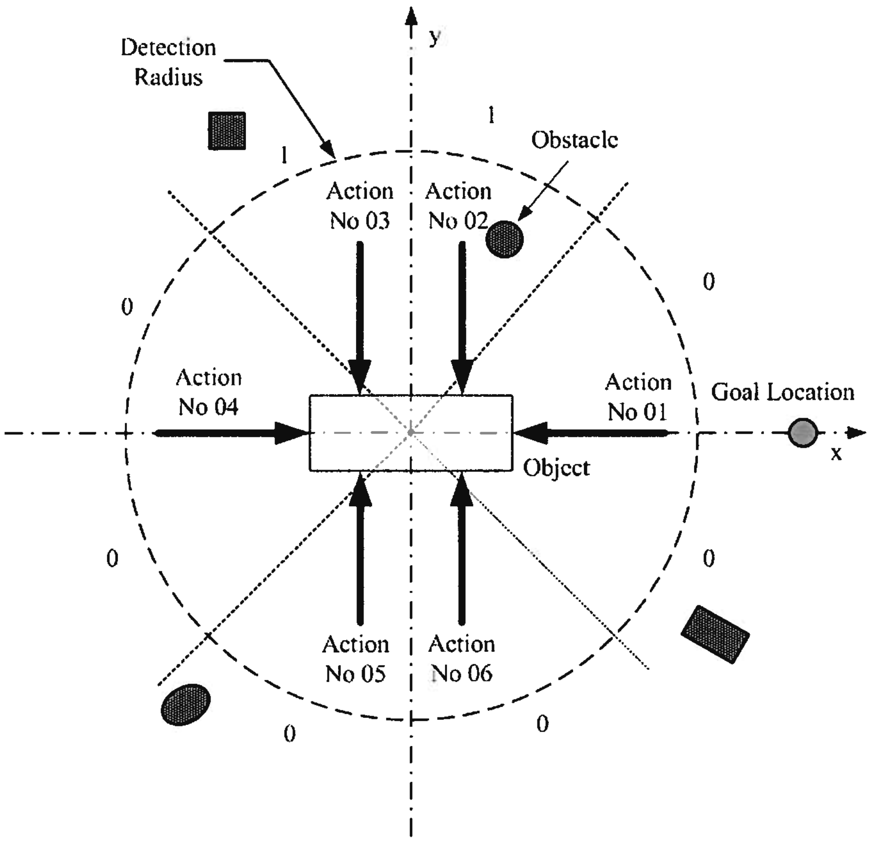


Figure 3.11: The environmental state with decimal value "512" and six pushing actions available to the robots.

The Q-values under the state of "512"in 100,000 rounds of object pushing are recorded and shown in Figure3.12, and the probability density estimate of these Q-values is given in Figure3.13.

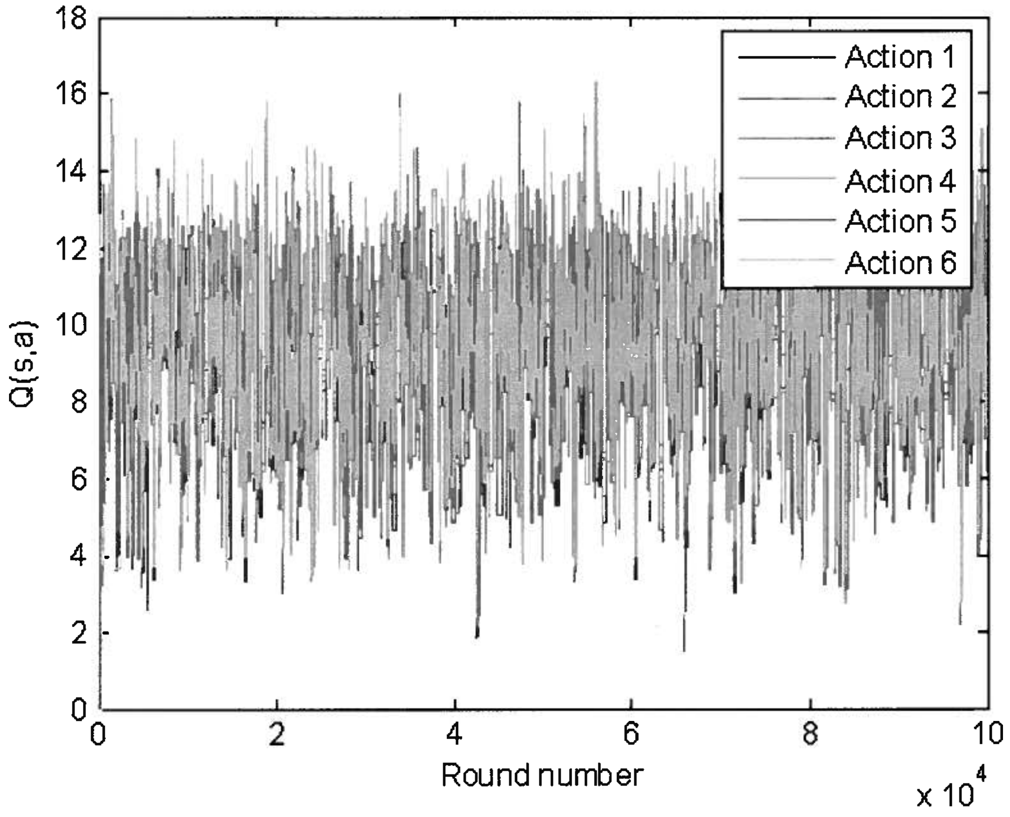


Figure 3.12: The Q-values under state of "512" in 100,000 rounds of object pushing.

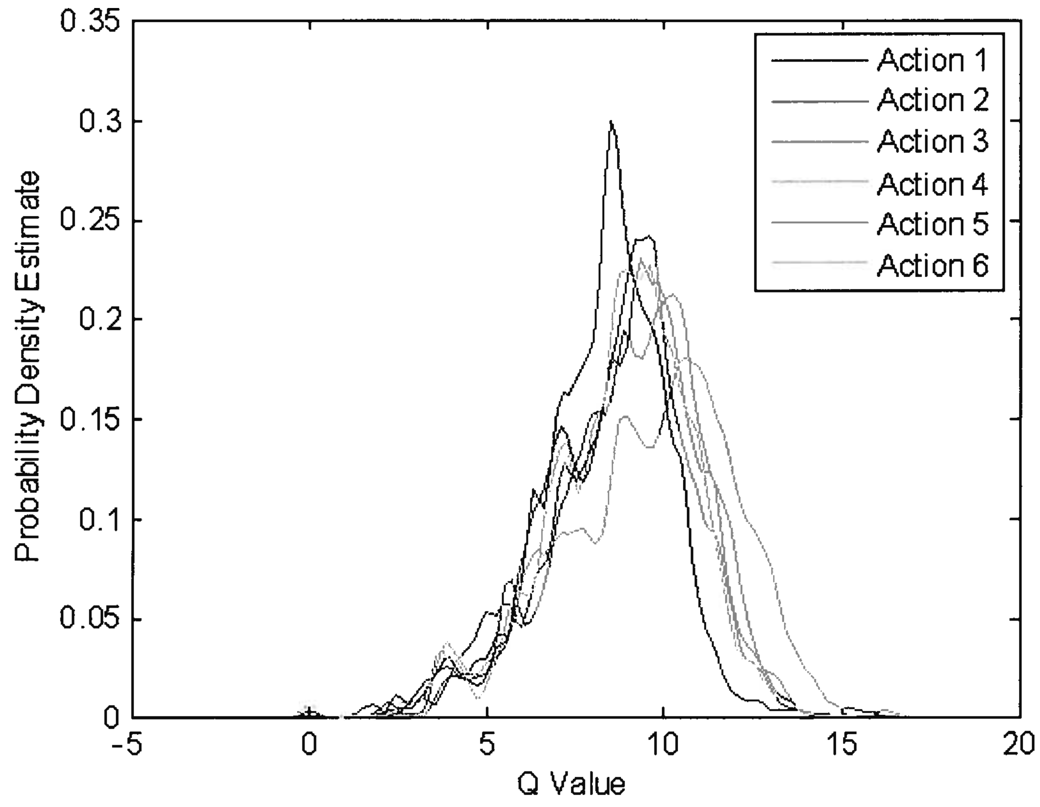


Figure 3.13: Probability density estimate of Q values under the state of "512" in 100,000 rounds of object pushing.

When 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). Figure 3.14 shows that the Q-values increase rapidly from the initial values of zero to some positive values in a very early stage. Then the Q-values start to oscillate in a bounded area with an upper bound of 16 and a lower bound of 3. From Figure 3.14 and Figure 3.15 it is rather difficult to find which action is selected at higher probability under the state of "512." The Q-values in Figure 3.14 appear to oscillate randomly and the curves overlap with each other. 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 3.12.

According to the sequential action selection strategy learned by the robots, the action selection probability in each round is calculated using the equation $P(a_j^i)=\frac{e^{Q(s,a_j^i)/\tau}}{\sum_{l=1}^{m} e^{Q_l(s,a_l^i)/\tau}}$.

Figure 3.16 presents the probability density estimate of the action selection probability with the sample of 100,000 rounds of object pushing.

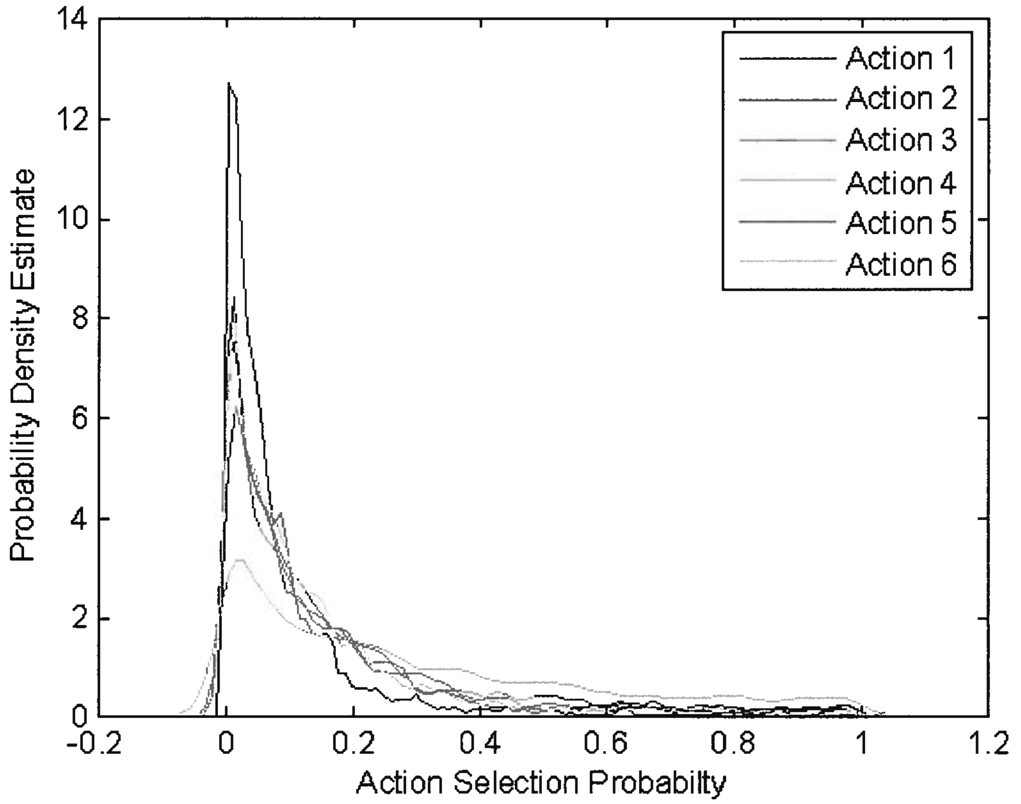


Figure 3.14: Probability density estimate of an action selection probability under the state of "512" in 100,000 rounds of object pushing.

Figure 3.14 gives the probability density estimate of sequential action selection probability under a special state in 100,000 rounds of object pushing. It is shown that there is a high probability that the robots select the action number 01 in a low probability, and there is a lower probability that the robots select action number 04 in a low probability. In other words, action number 01 is selected by the robots in low probability under the environmental state of "512"and action number 04 is selected by the robots in higher probability under that particular state. According to Figure 3.11, it is observed that action number 01 pushes the object away from the goal location, and action number 04 pushes the object towards the goal without colliding with the obstacle. By comparing Figure 3.14 with Figure 3.11, it is easy to conclude that the robots learn the correct sequential action selection strategy when they cooperatively push the object to the goal location. This explains why the robots in Figure 3.10 can cooperate to push the object to the goal successfully in a complex and dynamic environment with random obstacle distributions.

3.6 Summary

This chapter addressed the application of machine learning to multi-robot decision making. 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, where the traditional task planning becomes ineffective because a planned optimal policy can become inappropriate a few minutes later due to changes in the environment. Therefore, multi-robot systems face some special challenges; for example, dynamic and unknown environments, which the traditional AI technologies cannot effectively deal with.

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 introduced for multi-robot decision making. The basic idea of the single agent Q learning and its framework of Markov Decision Processes were introduced in section3.2 and section 3.3. In section 3.4, a more challenging multi-robot environment was assumed, which is dynamic and unknown and has a more complex obstacle distribution. 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. For such an environment, a distributed multi-robot object pushing system based on the modified Q learning algorithm was developed. The simulation results showed that the robots still could complete the task well at considerable speed.

A modified Q-learning algorithm, which is suitable for multi-robot decision making was developed. The simulation results showed that it has disadvantages when used to make decisions in a multi-robot system, even though it can enable the robots to make good decisions and properly complete tasks in most of cases. Because it violates the assumption of a static environment, directly applying the learning algorithm to the multi-robot environment will result in low convergence speed, and there is no guarantee that the algorithm will converge to the optimal policies. However, the developed learning algorithm showed good performance with regard to more realistic of state identification and action selection in a simulated multi-robot object-pushing exercise.

Chapter 4

Pose Estimation for Multi-Robot Object Transportation

4.1 Overview

A multi-robot system consists of several autonomous robots for carrying out a common task cooperatively. This represents an important research area in robotics. Such systems should be able to monitor and learn their cooperative strategy. Because of their flexible configuration, high efficiency, robustness, low cost and similarity to human groups, there are numerous research issues and applications associated with them.

Several autonomous robots share the environment in a multi-robot system. Therefore, the team members need to know their exact position and orientation in the shared environment. Not only that an individual robot needs to know its own latest pose (position/orientation), but also the pose of other robots and potential obstacles in the shared environment in order to make rational decisions. Therefore, pose estimation plays a vital role in a multi-robot system.

There are many means to measure the pose of a robot or an obstacle; for example, using digital cameras, sonar, or laser distance finders. However, most multi-robot systems employ digital cameras for this purpose, which offer three key advantages. First, a digital image provides a rich source of information on multiple moving objects simultaneously in the operating environment. Second, there is a possibility to build fast and accurate vision subsystems at low cost. Third, robot cameras observe and understand their operating environment in a "natural" manner as how humans use eyes to observe the world (Wang and de Silva, 2007). However, numerous obstacles exist in employing a computer vision system. First, since the object detection has to be done through feature extraction from the image, if the lighting conditions of the environment change, the result can become inaccurate. Second, in the image capturing process, hidden features of the image will not be in the image. Therefore, it will be difficult to detect an object occluded by another object . Third, different light sources may have different light intensity levels from different directions in the background. In a practical environment, with different light sources and lighting directions, identifying an object can become difficult. Fourth, object detection using digital cameras becomes challenging when the actual orientation or the extracted features are different from those employed in the training process. Fifth, during image processing with large amounts of training data, computational processing power may not be adequate, and this will affect the overall performance of the system.

Use of computer vision as a stand alone sensory approach in a dynamic environment to detect objects has been attempted by others. In particular, Kay and Lee (1991) used images from a stereo camera mounted on the end-effecter of a robot manipulator to estimate the pose of an object with respect to the base frame of the robot. The orientation of the object was estimated by using the least-squares method. A Kalman filter was used to smooth the estimation error. 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. Simon *et al.* (1994) developed a system which was able to perform full 3-D pose estimation of a single arbitrarily shaped rigid object. They used a high speed VLSI range sensor to acquire data on the object, and then performed pose estimation by fitting the data to a triangular mesh model using an enhanced implementation of the iterative closest point algorithm. 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. Ekvall *et al.* (2005) presented a computer vision approach for object recognition and pose estimation based on color co-occurrence histogram and a geometric model. Even though the approach was somewhat robust, it failed under large changes in lighting conditions. Park *et al.* (2006) presented a method for global localization of an indoor environment, which employed object recognition using a stereo camera. Their method of global localization first estimated a coarse pose and then a refined pose. The coarse pose was estimated by means of object recognition and least squares fitting through singular value decomposition, whereas the refined pose was estimated by using a particle filtering algorithm with omni-directional metric information. Even though a vision system has been used for pose estimation, their work has a number of drawbacks. In particular, the presented schemes are computationally expensive, time consuming, have time delays, and are susceptible to changes in lighting conditions. Lang *et al.* (2008) presented a multiple sensor based method for robot localization and box pose estimation. A CCD camera mounted on the robot was used to find the box in its environment. A laser range finder mounted on the robot was activated to measure the distance between the laser source and the object. Finally, homogenous matrix transformation was applied to represent the global pose of the robot and the box. The approach was validated using Microsoft Robotics Studio simulation environment.

In a multi-robot object transportation task, 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. Therefore, the conventional time-consuming vision algorithms are not feasible here. These algorithms are too complex and computationally demanding for meeting real-time constraints in a multi-robot object transportation system.

On the other hand, in an object transportation task, the multiple mobile robots move in a large area that has different levels of illumination. Multi-robot systems usually work in large, unstructured, and unknown 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 the existing algorithms do not consider this problem.

Object and obstacle recognition and pose estimation are essential for an object transportation system with mobile robots. Therefore, computer vision alone is not an adequate solution for the detection of objects and potential obstacles. Sensor fusion can be proposed as an alternative to overcome the problems mentioned above. This chapter investigates an approach for pose (position and orientation) estimation of an object or potential obstacle using a laser range finder and a CCD camera mounted on a mobile robot. In the next section, the multi-robot transportation system in the laboratory is introduced. In Section 4.3, the proposed pose estimation approach is presented. Section 4.3.1 presents the robot global localization. Section 4.3.2 outlines color blob tracking for object recognition. Section 4.3.3 presents object pose estimation. An experimental evaluation of object pose estimation is given in Section 4.4. The conclusions of the chapter are given in Section 4.5.

4.2 Multi-robot Transportation System

The primary objective of the multi-robot object transportation project in the Industrial Automation Laboratory (IAL) is to develop a physical multi-robot system where a group of intelligent autonomous robots has the ability to work cooperatively to transport an object to a goal location and orientation in an unknown and dynamic environment. Obstacles may be present or even appear randomly in the work environment during the transportation process. Three or more autonomous mobile robots are present in the system, which enable cooperative transportation of an object to a goal location. During the course of the transportation, they have to interact with each other to develop the coordination and cooperative strategy and decide on the feasible transportation locations and suitable action positions selected by each robot so that the object is transported quickly and effectively while avoiding obstacles that may be present during the transportation process. Furthermore, the system may have to consider avoiding damage to the transported object. Charge coupled device (CCD) camera systems, laser range finders and sonar are used to monitor and measure the current location and orientation of the object. A schematic representation of the initial version of the developed system is shown in Figure 1.2 and is repeated in Figure 4.1.

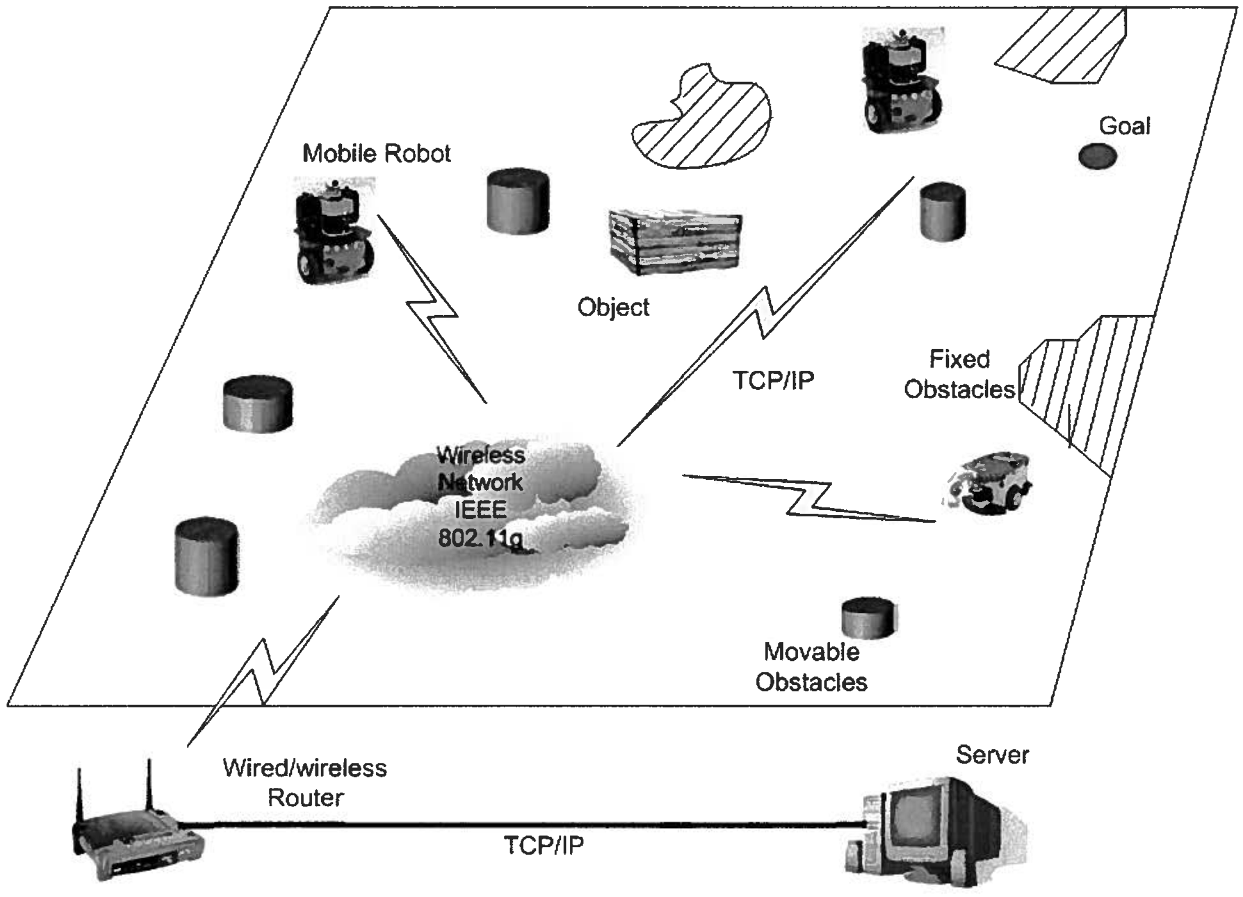


Figure 4.1: The developed physical multi-robot system.

The dynamic and unpredictable work environment contains fixed obstacles and movable obstacles, which may appear randomly. The robots and the sensory system are separately linked to their host computers, which are connected though a local network to implement complicated controls and machine intelligence. 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.

It is important to consider three major challenges in the development of the system, which are also major challenges in multi-robot systems. The first one is the dynamic and unpredictable nature of the environment. Especially, dynamic random obstacle are scattered within the working environment of robots. The obstacles may appear and disappear randomly during the transportation process. In addition, there are multiple robots working in parallel. While one robot makes decisions, other robots may have changed the environment according to their behaviors. This also makes the environment dynamic from the viewpoint of the first robot. As a result, the decision making process becomes very complex for the robots within the dynamic environment. A major challenge is to make decisions in real-time while identifying hidden or invisible features of objects and the surrounding environment conditions from the local viewpoint of the robots.

The second challenge results from the local sensing capabilities of the robots. In the present project, it is assumed that each robot is capable of detecting an object or an obstacle within a limited detection radius. It means that the robots possess local sensing capabilities only and therefore the global environment is unknown to them. Robots do not have an understanding about the location or the shape of an obstacle until a robot moves close to the obstacle. The unknown environment is a major challenge in the multi-robot domain because the robots do not have a complete understanding about the environment in advance, partly due to the unfamiliarity and partly due to the dynamic nature. Thus the system cannot utilize the traditional planning technologies for decision making.

The third challenge is the lack of a reliable communication capability among robots, where the disturbances are represented by white noise. Due to unreliable communication, the robots may end up selecting a wrong coordination strategy. Improvement of the robustness of multi-robot decision-making in an environment with unreliable communication is a challenging issue in the multi-robot domain.

4.3 Pose Estimation Technique

Pose estimation in real time is a fundamental requirement in multi-robot cooperation for object transportation. Though there has been substantial growth of research activity in pose estimation of a robot, the pose estimation of objects and potential obstacles has not received the much needed attention. Estimation of the Cartesian coordinate position (x, y) and orientation (θ) of an object can be used in diverse applications like object grasping and robotic manipulation, motion prediction, and object transportation. This chapter presents an approach to acquire meaningful data on distance and orientation of an object using the laser range finder of a mobile robot for calculating the pose (center and orientation) of the object with respect to the robot in the global coordinate system. The developed methodology as presented here is an integral part of the multi-robot cooperative object transportation system. A group of mobile robots explore the environment to detect a color coded object. Once the object is detected, that particular robot estimates the object pose. This includes the center and the orientation of the object. However, the object of interest is large and heavy, and cannot be transported by a single robot. Therefore, a team of robots in the multi-robot system is called upon to cooperatively transport the object. Figure 4.2 presents the general scheme of pose estimation of an object for cooperative object transportation.

According to the proposed method, successful transportation constitutes six steps, as outlined next.

Initialization of all robots: Robot global localization. Local localization of all robots are transformed into global localization

Search for an object: Robots start searching for an object of interest to be transported to the goal location and orientation.

Rearrange of robot pose: Once the robot in the multiple robot system finds a color coded object, it rearranges its pose to center the color blob in the camera frame.

Object pose estimation: The particular robot, which found the color blob, estimates the pose of the object and takes over the leadership in the rest of the state identification process. The particular robot assigns tasks to the member robots in the system during the state identification stage.

Obstacle pose estimation: The other robots in the system identify the local obstacle distribution around the object and build the state binary code of themselves. Then the system determines suitable points of contact (action points) with the object for transporting it.

Transportation of the object: Multiple robots transport the object by pushing the object of interest. The process is repeated until the object reaches the goal location and orientation.

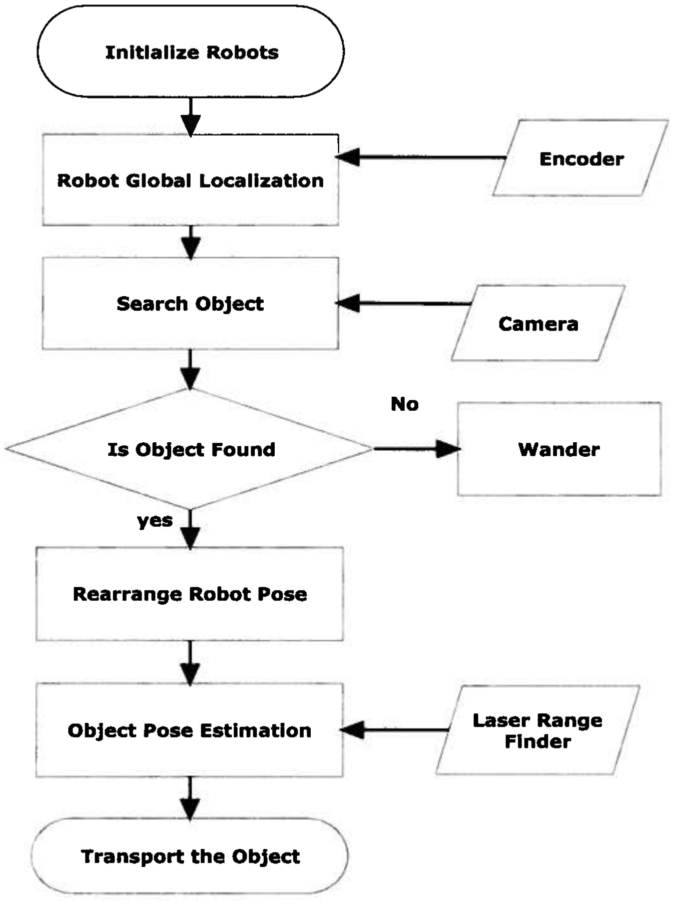


Figure 4.2: General scheme of pose estimation for cooperative object transportation.

4.3.1 Global Localization of Robot

Global pose estimation is required since there is more than one robot in the environment of cooperative object transportation. An important means of estimating pose of a robot is odometry. This method uses data from the shaft encoder mounted on a wheel of the robot. The encoder measures the speed and the displacement of the drive wheel after a specific time step and is added to the pose of the robot in the previous time step. With reference to Figure 4.3, the kinemati model of the mobile robot is expressed by the relations presented next. The state vector representing the pose of the mobile robot in the global frame is given by:

$\underline{q}(k) = [x(k) \;\; y(k) \;\; \theta(k)]^T$, where $x(k)$ and $y(k)$ are the coordinates of position P in mm, and $\theta(k)$ is the orientation in degrees. Also, $D(k)$ is the distance traveled between the time steps k and $k+1$; $\upsilon_t(k)$ is the robot translational speed in mm/s; T is the sampling time in seconds; $\theta(k)$ is the angle between the robot and the global x-axis; $\Delta\theta(k)$ is the rotation angle between time steps k and $k+1$; $\omega_L(k)$ and $\omega_R(k)$ are the angular velocities of the left and right wheels, respectively; r is the radius of the two drive wheels; and d is the distance between the two wheels. The kinematic model of the mobile robot is given by:

$$x(k+1) = x(k) + D(k).\cos\theta(k+1) \tag{4.1}$$

$$y(k+1) = y(k) + D(k).\sin\theta(k+1) \tag{4.2}$$

$$\theta(k+1) = \theta(k) + \Delta\theta(k) \tag{4.3}$$

$$D(k) = \upsilon_t(k).T \tag{4.4}$$

$$\Delta\theta(k) = \omega(k).T \tag{4.5}$$

$$\upsilon_t(k) = \frac{\omega_L(k).r + \omega_R(k).r}{2} \tag{4.6}$$

$$\omega(k) = \frac{\omega_R(k).r - \omega_L(k).r}{d} \tag{4.7}$$

It follows that the updated pose state vector is $\underline{q}(k+1) = [x(k+1) \;\; y(k+1) \;\; \theta(k+1)]^T$

By using equations 4.1 through 4.7, the robot global pose $\underline{q}(k+1)$ for a wheeled robot is computed as:

$$\underline{q}(k+1) = \begin{bmatrix} x(k) \\ y(k) \\ \theta(k) \end{bmatrix} + \begin{bmatrix} \left(\dfrac{\omega_L(k)+\omega_R(k)}{2}\right)\cos\left(\theta(k)+\dfrac{T.r}{d}\left(\omega_R(k)-\omega_L(k)\right)\right) \\ \left(\dfrac{\omega_L(k)+\omega_R(k)}{2}\right)\sin\left(\theta(k)+\dfrac{T.r}{d}\left(\omega_R(k)-\omega_L(k)\right)\right) \\ \dfrac{\omega_R(k)-\omega_L(k)}{2} \end{bmatrix} \qquad (4.8)$$

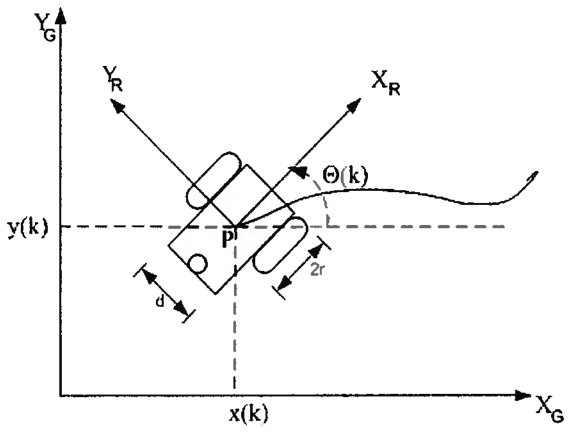


Figure 4.3: Global Pose Estimation of Wheeled Robot.

4.3.2 Color Blob Tracking

The phenomenon of color blob template matching is utilized for color blob tracking. The approach presented in this section entails detection of the object to be transported with the help of two different color blobs attached to its vertical surfaces. For this purpose the Advanced Color Tracking System (ACTS) capability is employed. ACTS is a software tool which in combination with a color camera, allows the application to track up to 320 colored objects at a speed of 30 frames per second.

The robots explore the surrounding environment within the sensory range and a robot receives a stimulus if it locates an object with the color blob. As it locates the object, the robot rearranges its pose so as to move the color blob attached to the object to the center of the camera frame. As illustrated in Figure 4.5, the camera frame is divided into four quadrants. Here, the size of the camera frame is (640 × 480) pixels, while the center of the camera frame is represented by a square of (40×40) pixels. If the color blob falls within the square, it is believed to be in the center. Otherwise the robot rotates its base so that the detected color blob is approximately located at the centerline of the camera frame. By centering the color blob in the camera frame, a suitable position is provided for the Laser range finder to scan the object for pose estimation. A flow diagram of the above mentioned process is shown in Figure 4.4.

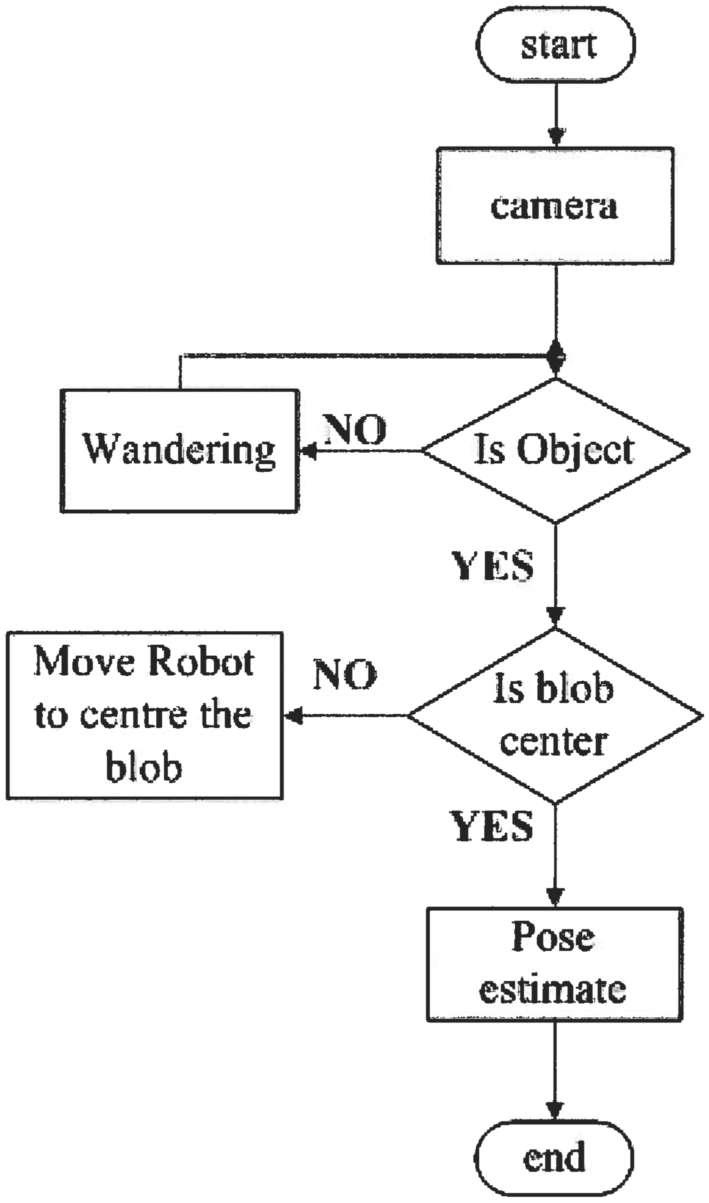


Figure 4.4: flow diagram of color blob tracking

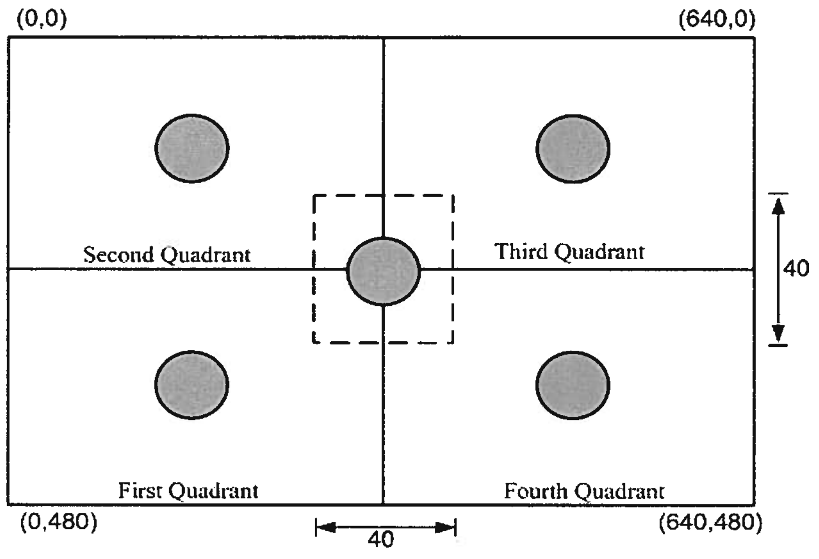


Figure 4.5: Division of Camera Frame into Four Quadrants.

4.3.3 Object Pose Estimation

As noted in section 4.3.1, once the robot rearranges its pose to center the color blob within camera frame, the laser range finder that is mounted on the robot is activated to estimate the pose of the object. Figure 4.6 indicates how the laser range finder is used to measure the range. Specifically, the distance is determined by calculating the phase difference between the incident and reflected signals. If the transmission time from A to B is *T*, phase difference is ϕ, and the modulation frequency is f_m, then the distance between A and B is given by: $L = c\frac{\phi}{2\pi f_m}$. The pose of the object is estimated relative to the robot, which is then transformed into global pose estimation (Gao and Xiong, 2006). This is needed because the robot may have to communicate the object pose to other robots in order to transport the object cooperatively.

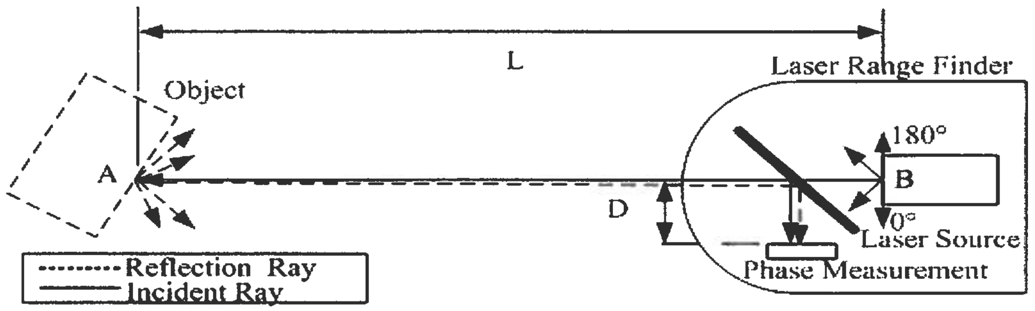


Figure 4.6 (a): Schematic drawing of laser range sensor.

Figure 4.6 (b): A 180-degree laser range sensor.

4.3.3.1 Relative Pose Estimation

With reference to Figure 4.7, data on distance and angle are acquired through the laser range finder to calculate the center and the orientation of the object relative to the robot coordinate system.

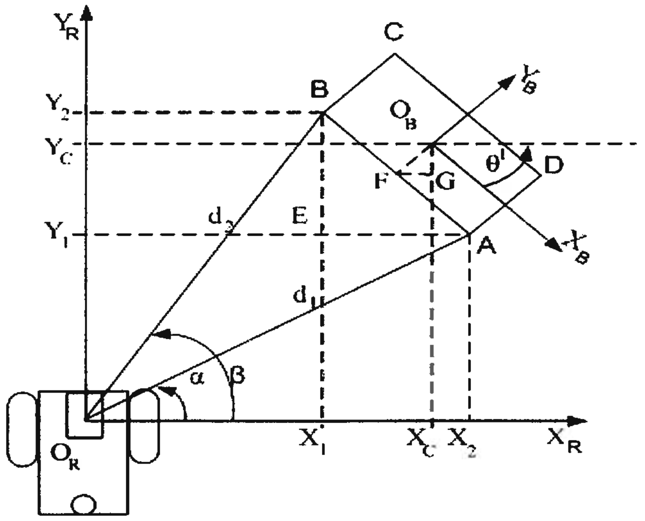


Figure 4.7: Relative Object Pose Estimation.

Note that α and β are the angles corresponding to the distances d_1 and d_2, respectively. Also, $d_1 = AO_R$, $d_2 = BO_R$; Y_R, O_R, and X_R are robot coordinates; and d_1 and d_2 are the distances from laser range finder to two distinct edges of the object. We have:

$$d_1 \cos\alpha = O_R X_3 \tag{4.9}$$

$$d_2 \cos\beta = O_R X_1 \tag{4.10}$$

$$d_1 \sin\alpha = AX_3 \tag{4.11}$$

$$d_2 \sin\beta = BX_1 \tag{4.12}$$

By using equations 4.9 through 4.12, the angle which represents the orientation of the object is given by:

$$\theta' = \tan^{-1} \frac{d_2 \sin\beta - d_1 \sin\alpha}{d_1 \cos\alpha - d_2 \cos\beta} \tag{4.13}$$

Object orientation can be found by using equation 4.13.

The center point of the object O_B is given by:

$$O_B \equiv \left(\begin{array}{c} \frac{1}{2}\left[d_1 \cos\alpha + d_2 \cos\beta + \frac{AD}{AB}(d_2 \sin\beta - d_1 \sin\alpha) \right], \\ \frac{1}{2}\left[d_1 \sin\alpha + d_2 \sin\beta + \frac{AD}{AB}(d_1 \cos\alpha - d_2 \cos\beta) \right] \end{array} \right)$$

$$X_C = \frac{1}{2}\left[d_1 \cos\alpha + d_2 \cos\beta + \frac{AD}{AB}(d_2 \sin\beta - d_1 \sin\alpha) \right] \tag{4.14}$$

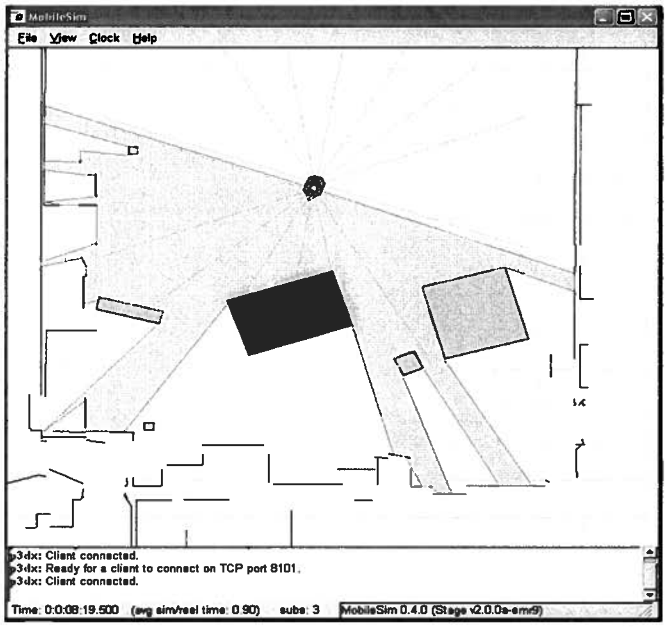
$$Y_C = \frac{1}{2}\left[d_1 \sin\alpha + d_2 \sin\beta + \frac{AD}{AB}(d_1 \cos\alpha - d_2 \cos\beta) \right] \tag{4.15}$$

Equations 4.13 through 4.15 represent the object pose relative to robot pose, and can be given by the vector:

$$O_P = \begin{bmatrix} X_C & Y_C & \theta' \end{bmatrix} \tag{4.16}$$

Equation 4.16 describes the object with respect to the mobile robot, and is called the relative pose.

Figure 4.8 shows the results of the laser range finder using MobleSim simulator. There are six objects within the laser scanner according to the results given in Figure 4.9. Because the object with a color blob label should be in the center of the laser range scan after applying the color blob tracking in Section 4.3.2, it is very easy to establish that the object must be in the range between - 14° and 32°.



```
p3dx: Client connected.
p3dx: Ready for a client to connect on TCP port 8101.
p3dx: Client connected.
Time: 0:0:08:19.500  (avg sim/real time: 0.90)  subs: 3  MobileSim 0.4.0 (Stage v2.0.0a-emr9)
```

Fig. 4.8 (a) MobileSim simulator visualized results from laser range finder.

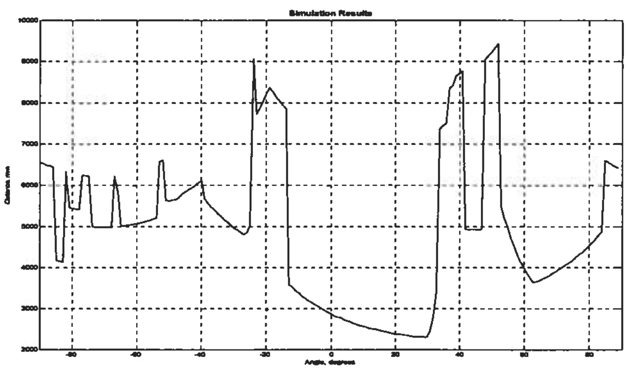


Fig. 4.8 (b) Laser range finder results using MobileSim simulator.

4.3.3.2 Object Global Estimation

A homogenous transformation matrix (Spong *et al*, 2006), which represents the translational and rotational motion of one frame with respect to another frame, is applied to represent the relation between the three coordinate systems: the robot coordinate system, the object coordinate system, and the global coordinate system.

The homogeneous transformation matrix between the object coordinate system and the robot coordinate system can be expressed by using the result obtained in section 4.3.2.1:

$$T' = \begin{bmatrix} \cos\theta' & -\sin\theta' & X_C \\ \sin\theta' & \cos\theta' & Y_C \\ 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

The robot global pose is known from equation 4.8 in section 4. The homogeneous transformation matrix between the robot coordinate system and the global coordinate system can be established as:

$$T'' = \begin{bmatrix} \cos\theta & -\sin\theta & X \\ \sin\theta & \cos\theta & Y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.18}$$

By using equations 4.17 and 4.18, the homogeneous transformation matrix T between the object coordinate system and the global coordinate system can be computed as:

$$T = T''.T' = \begin{bmatrix} \cos\theta & -\sin\theta & X \\ \sin\theta & \cos\theta & Y \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} \cos\theta' & -\sin\theta' & X_C \\ \sin\theta' & \cos\theta' & Y_C \\ 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} \cos(\theta+\theta') & -\sin(\theta+\theta') & X_C\cos\theta - Y_C\sin\theta + x \\ \sin(\theta+\theta') & \cos(\theta+\theta') & X_C\sin\theta + Y_C\cos\theta + y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

From equation 4.19, the pose of the object in the global coordinate system is determined as:

$$O'' = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} X_C \cos\theta - Y_Y \sin\theta + x \\ X_C \sin\theta + Y_C \cos\theta + y \\ A\tan(\sin(\theta+\theta')/\cos(\theta+\theta')) \end{bmatrix}$$

4.4 Experimental Results

In this section the experimental results are presented using the test bed described in section 4.4.1. Experiments are conducted under two different situations. In the first set of experiments, the robot is kept stationary and the objects are placed at different positions and orientations. In the second set of experiments, the robots move, and once an object is detected by a robot it calculates the pose of the detected object. In both sets of experiments the objects are placed in different poses, as indicated in Figure 4.9.

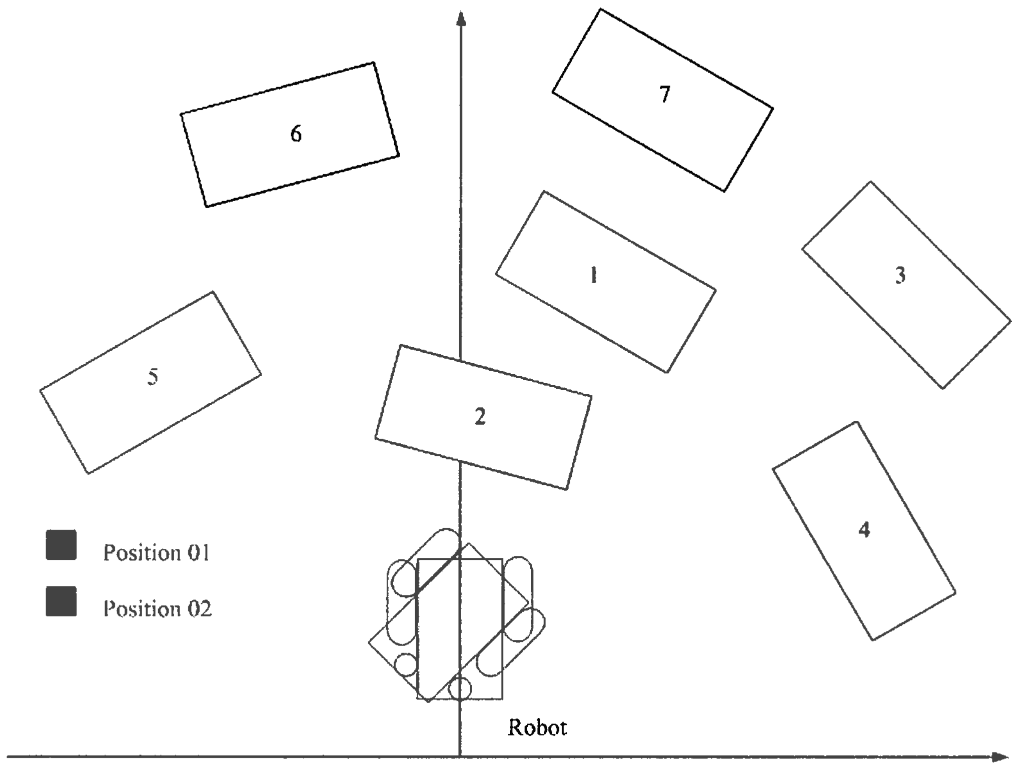


Figure 4.9: Arbitrary layout of objects.

4.4.1 The Test-bed

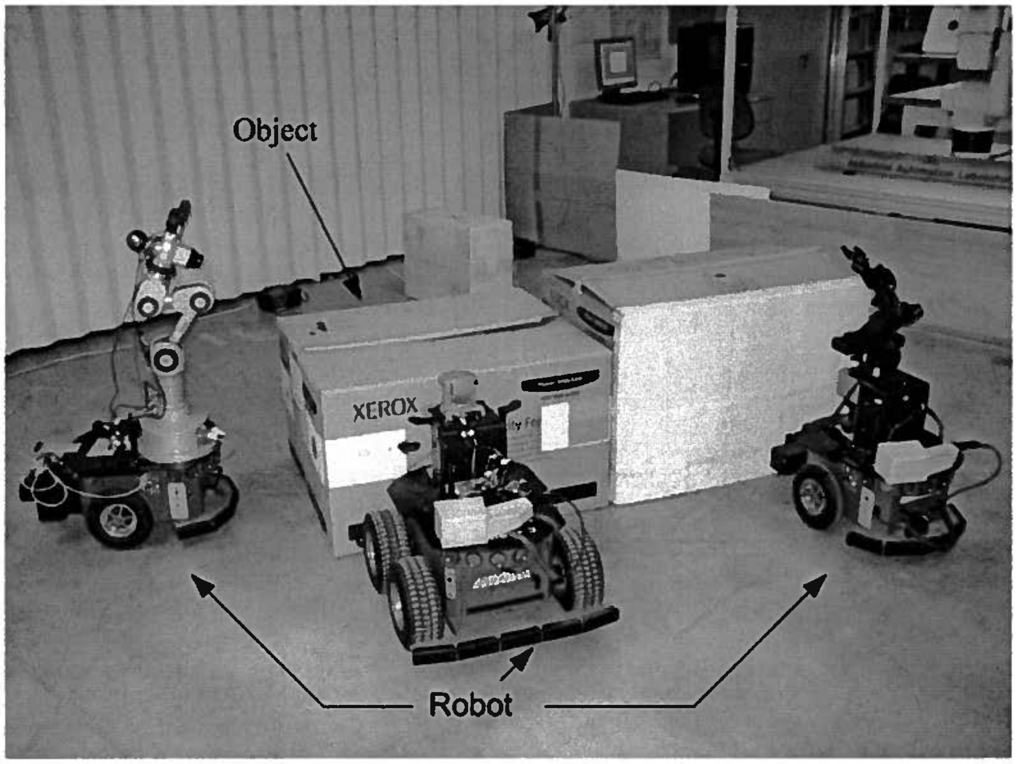


Figure 4.10: The experimental system in IAL at UBC.

A test bed has been developed in the Industrial Automation Laboratory (IAL) for multi-robot transportation, as shown in Figure 4.10. It includes one four-wheel-driven ActiveMedia PioneerTM 3-AT robot and two three-wheel-driven ActiveMedia PioneerTM 3-DX robots. All three robots include wireless Ethernet, eight sonar sensors and a gyro. Two robots include a CCD camera and a laser range finder. Each robot houses a Pentium based computer running windows 2000. The laser range finder in the present system uses a SICK LMS 200 2D scanner, which has a horizontal range of 180° with a maximum resolution of0.5°, which is used to monitor the positions and orientations of the object, potential obstacles and mobile robots. The device produces range estimation using the time required for the light to reach the target and return (Nourbaksh *et al*, 2004).

4.4.2 Experiments with Stationary Robot

Two sets of experiments are conducted in the category of stationary robot. In the first set of experiments, the robot is placed at the global position [0, 100, 0], which gives the *X*-coordinate, *Y*-coordinate, and orientation θ, respectively. The seven test points in Table 1 represent an object having dimensions 525mm × 257mm × 446mm, placed in the workspace at seven different positions and orientations. First the actual center of the test point is measured, which is compared with the center estimated through the laser range finder that is mounted on the robot. Figures 4.11, 4.12 and 4.13 correspond to the information in Table 4.1. Specifically, Figure 4.11 gives the difference between the actual and the estimated values of the x-coordinate of the center of the object. Figure 4.12 represents the difference between the actual and the estimated values of the y-coordinate of the center of the object. Figure 4.13 indicates the difference between the actual orientation and the estimated orientation of the object.

Table 4.1: The actual and estimated object pose results in the first set of experiments with a stationary robot.

Test Points	*X - actual*	*X - estimated*	*error*	*%error*	*Y - actual*	*Y - estimated*	*error*	*%error*	*θ - actual*	*θ - estimated*	*error*	*%error*
1	248	286.36	38.36	15.47	2095	1985.69	109.31	5.22	16	14.17	1.83	11.44
2	-219	-241.24	22.24	10.16	1184	1150.34	33.66	2.84	-12	-13.08	1.08	9
3	943	929.25	13.75	1.46	1460	1324.37	135.63	9.29	38	36.96	1.04	2.74
4	1471	1387.21	83.79	5.7	463	350.17	112.83	24.37	70	68.66	1.34	1.91
5	-1364	-1370.01	6.01	0.44	538	536.67	1.33	0.25	-66	-66.84	0.84	1.27
6	-1368	-1388.36	20.36	1.49	1354	1317.24	36.76	2.71	-32	-33.97	1.97	6.16
7	-710	-727.02	17.02	2.4	2298	2274.81	23.19	1.01	-7	-9.31	2.31	33

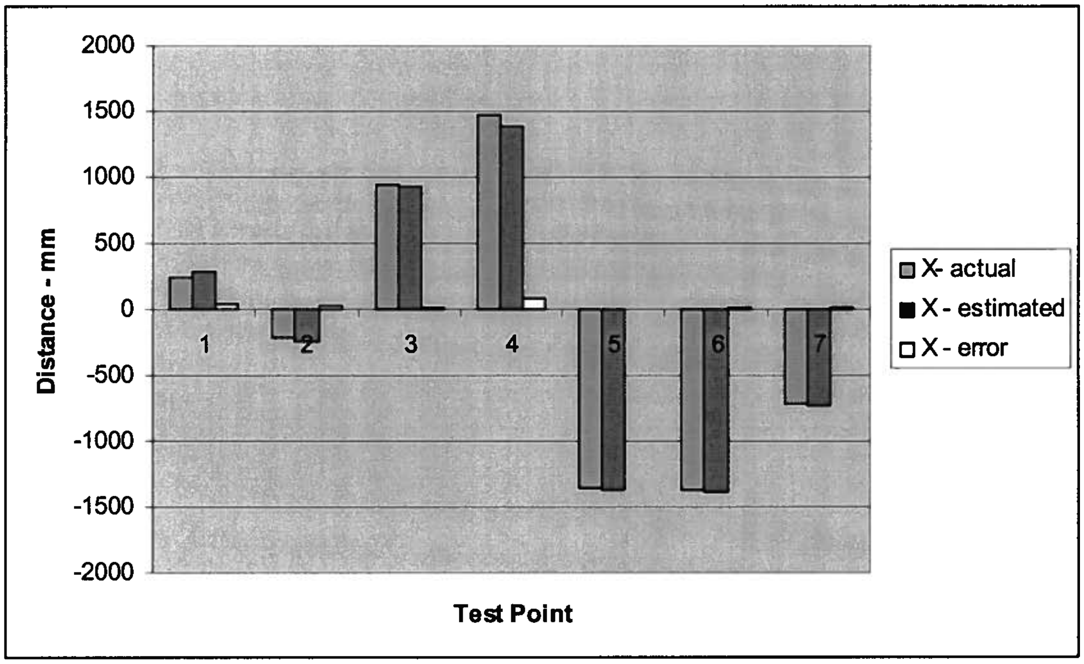


Figure 4.11: The x-axis error from Table 4.1.

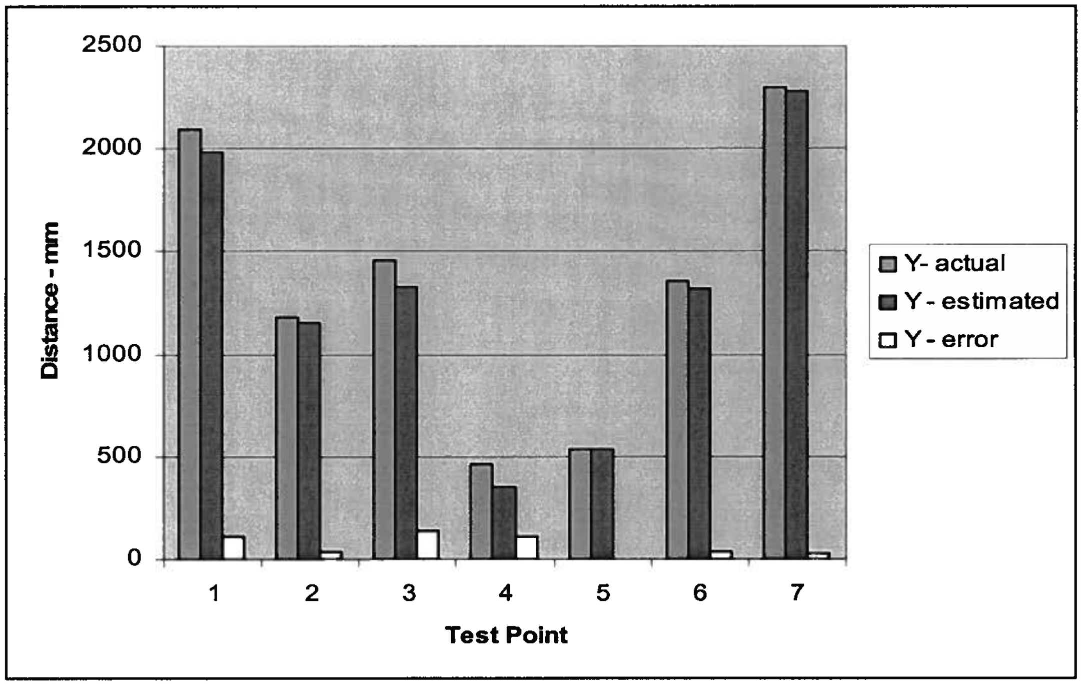


Figure 4.12: The y-axis error from Table 4.1.

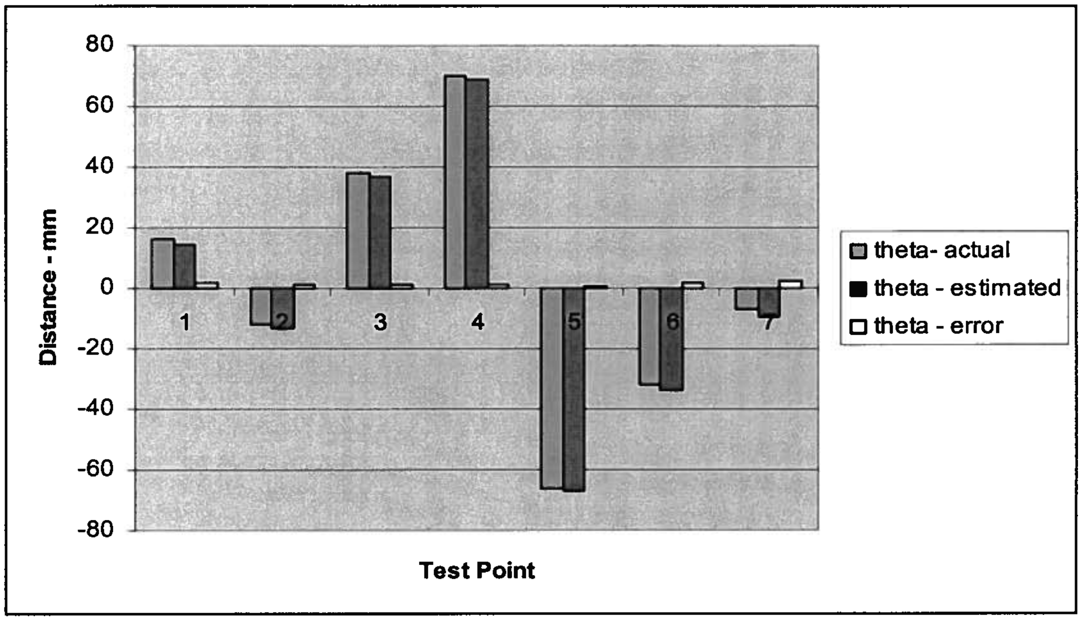


Figure 4.13: The orientation error from Table 4.1.

In the second set of experiments the robot global pose is changed to [40, 105, 60]. The dimensions of the object in this experiment are 525mm × 257mm × 446 mm. Table 4.2 presents the result for the second experiment, and Figures 4.14, 4.15, and 4.16 show the error results corresponding to the information given in this table. The same parameters as in the previous experiment are used. In both sets of experiments, with a few exceptions, the % error is well within an acceptable range, which is 100 mm in the considered application.

Table 4.2: The actual and estimated object pose results for the second set of experiments with a stationary robot.

Test Points	*X - actual*	*X - estimated*	*error*	*%error*	*Y - actual*	*Y - estimated*	*error*	*%error*	*θ - actual*	*θ - estimated*	*error*	*%error*
1	248	266.56	18.56	7.48	2095	1964.69	130.31	6.22	16	13.06	2.94	18.38
2	-219	-169.26	49.74	22.71	1184	1145.34	38.66	3.27	-12	-13.54	1.54	12.83
3	943	931.65	11.35	1.2	1460	1303.52	156.48	10.72	38	36.56	1.44	3.79
4	1471	1364.56	106.4	7.24	463	327.65	135.35	29.23	70	67.66	2.34	3.34
5	-1364	-1293.14	70.86	5.2	538	534.66	3.34	0.62	-66	-67.67	1.67	2.53
6	-1368	-1339.67	28.33	2.07	1354	1299.93	54.07	3.99	-32	-34.86	2.86	8.94
7	-710	-692.93	17.07	2.4	2298	2258.81	39.19	1.71	-7	-10.21	3.21	45.86

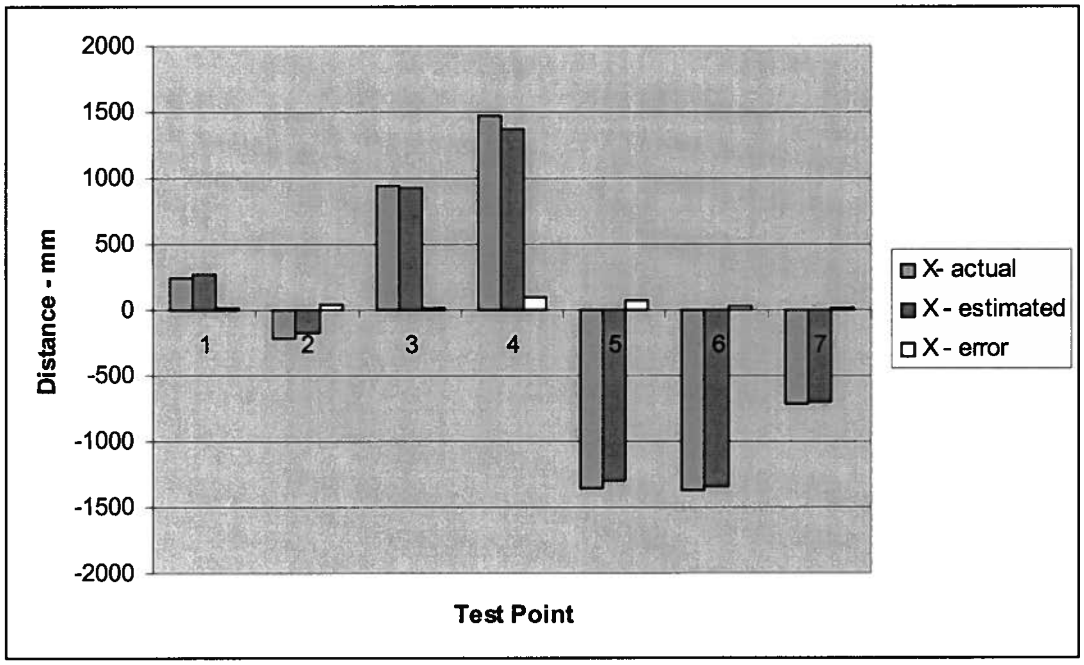


Figure 4.14: The x-error from Table 4.2.

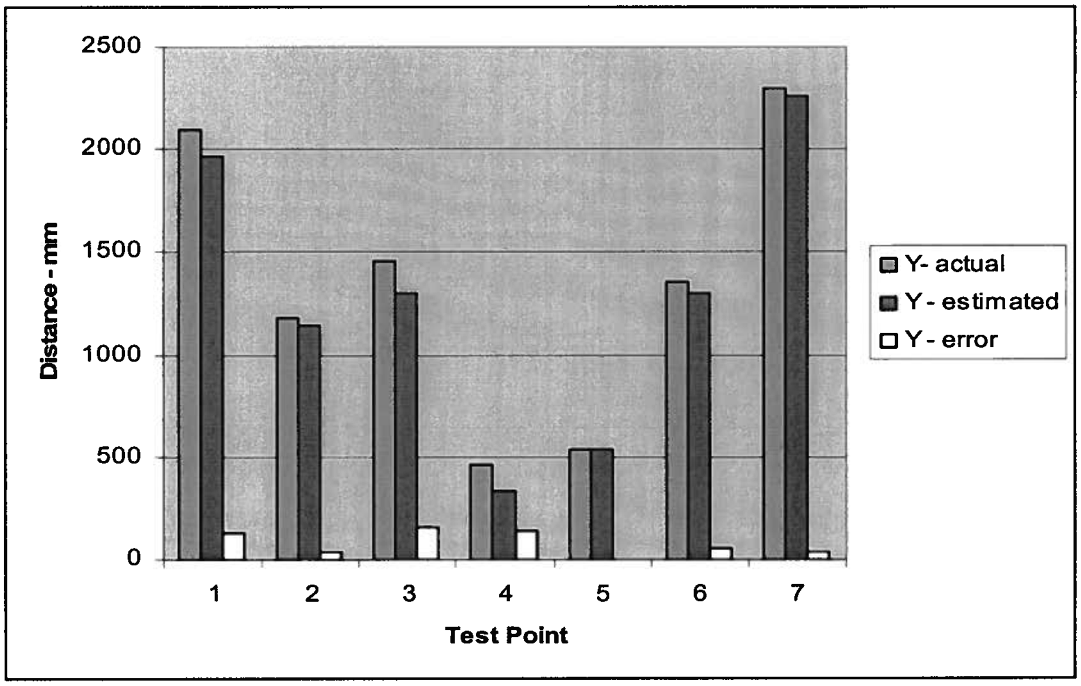


Figure 4.15: The y-error from Table 4.2.

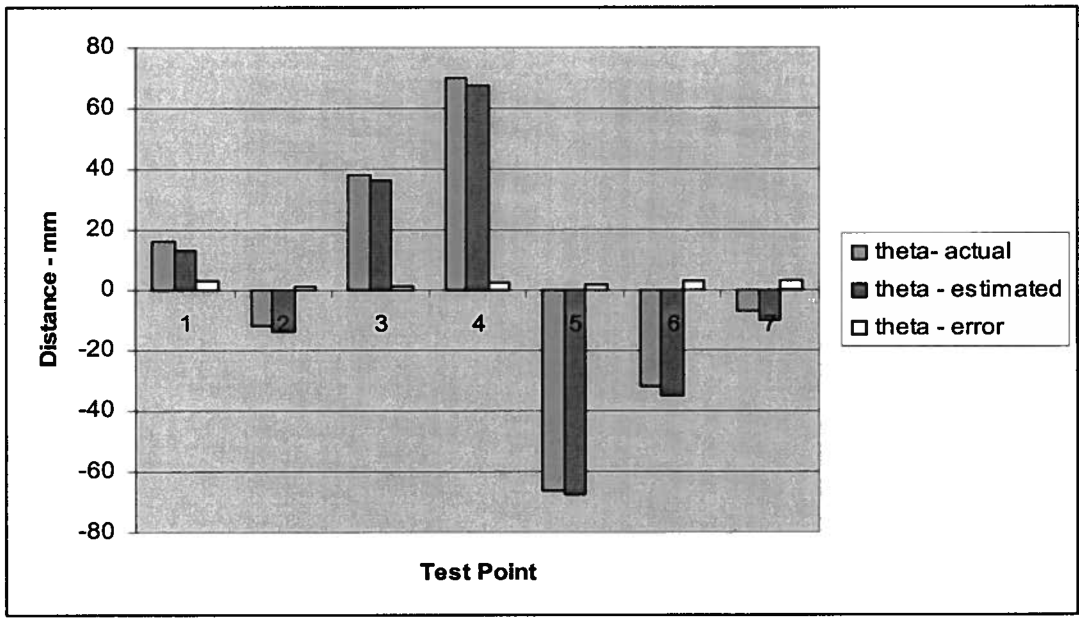


Figure 4.16: The orientation error from Table 4.2.

4.4.3 Experiments with a Moving Robot

Two sets of experiments are conducted in this category as well. Here, robots explore the environment in searching for an object to be transported. Once a robot finds the color coded object, it rearranges its pose to center the color blob within the camera frame. Then the laser range finder is activated to estimate the orientation and the center of the object.

In the first set of experiments, an object having dimensions 430 × 537 × 594 is used. Seven different experiments are carried out. Table 4.3 presents the results of the experiments. Figures 4.17, 4.18, and 4.19 correspond to this table, which show the difference between the actual and the estimated *x*-coordinate, *y*-coordinate, and orientation, respectively, of the center of the object.

Table 4.3: The actual and estimated object pose results for the first set of experiments with a moving robot.

Test Points	*X - actual*	*X - estimated*	*error*	*%error*	*Y - actual*	*Y - estimated*	*error*	*%error*	*θ - actual*	*θ - estimated*	*error*	*%error*
1	1318	1384.52	66.52	5.05	3455	3514.78	59.78	1.73	30	28.57	1.43	4.77
2	2467	2543.12	76.12	3.09	4047	4149.39	102.39	2.53	15	12.95	2.05	13.67
3	1205	1237.39	32.39	2.69	6147	6219.28	72.28	1.18	45	44.21	0.79	1.76
4	2745	2790.32	45.32	1.65	6123	6183.43	60.43	0.99	-15	-16.43	1.43	9.53
5	1465	1487.45	22.45	1.53	7442	7543.27	101.27	1.36	-60	-61.25	1.25	2.08
6	2063	2092.14	29.14	1.41	8522	8632.02	110.02	1.29	-30	-31.37	1.37	4.57
7	2842	2914.42	72.42	2.55	9712	9828.21	116.21	1.2	-75	-76.65	1.65	2.2

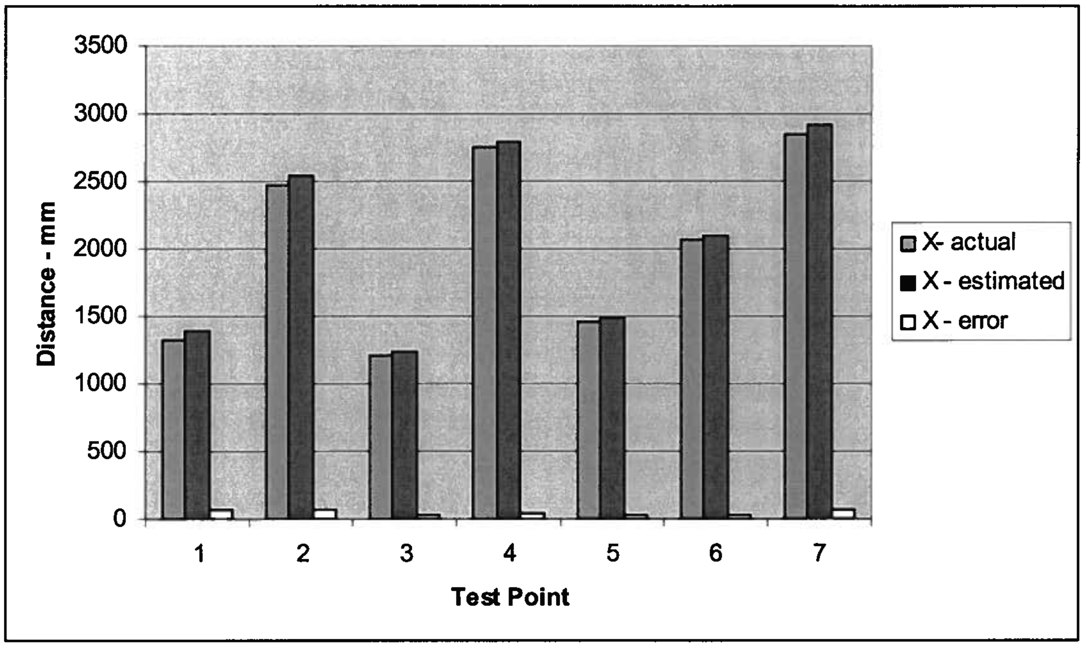


Figure 4.17: The x - error from Table 4.3.

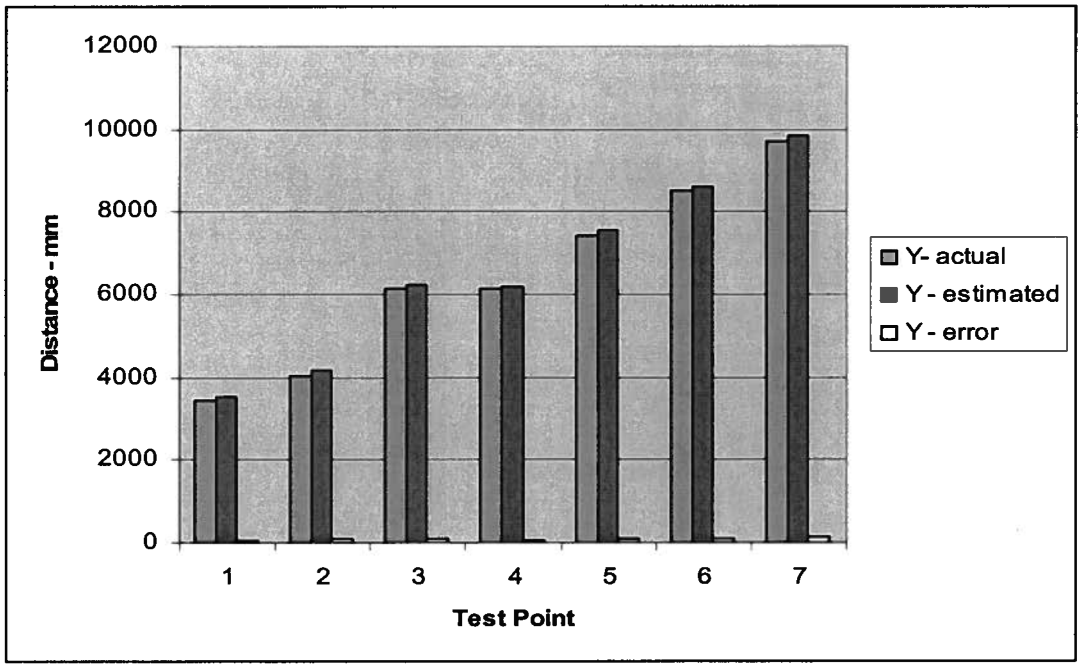


Figure 4.18: The y - error from Table 4.3.

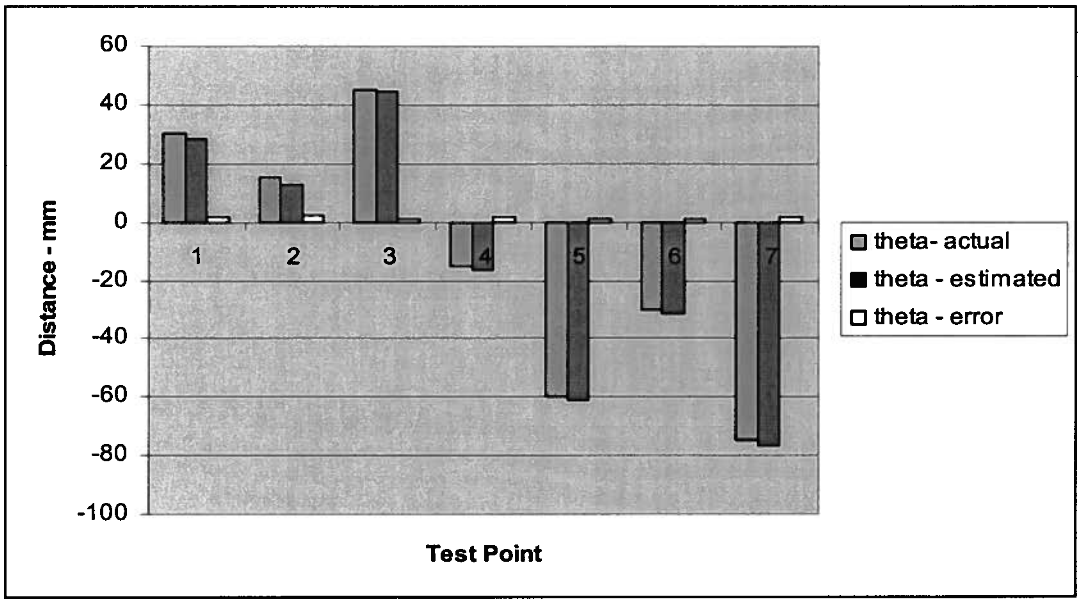


Figure 4.19: The orientation error from Table 4.3.

In the second set of experiments, an object having dimensions 590mm × 726mm × 1715mm is used. Table 4.4 presents the results from these experiments. Figures 4.20, 4.21, and 4.22 correspond to this table, which show the difference between the actual and the estimated *x*-coordinate, *y*-coordinate, and orientation, respectively, of the center of the object. Here too the error readings are well within acceptable range.

Table 4.4: The actual and estimated object pose results for the second set of experiments with a moving robot.

Test Points	*X - actual*	*X - estimated*	*error*	*%error*	*Y - actual*	*Y - estimated*	*error*	*%error*	*θ - actual*	*θ - estimated*	*error*	*%error*
1	1321	1356.43	35.43	2.68	3486	3509.52	23.52	0.67	30	28.27	1.73	5.77
2	2488	2502.32	14.32	0.58	4023	4112.23	89.23	2.22	15	13.95	1.05	7
3	1223	1235.43	12.43	1.02	6123	6179.72	56.72	0.93	45	43.77	1.23	2.73
4	2724	2787.21	63.21	2.32	6147	6184.17	37.17	0.6	-15	-16.43	1.43	9.53
5	1437	1462.76	25.76	1.79	7412	7513.23	101.23	1.37	-60	-62.24	2.24	3.73
6	2040	2072.34	32.34	1.59	8541	8612.98	71.98	0.84	-30	-32.06	2.06	6.87
7	2825	2903.59	78.59	2.78	9703	9812.73	109.73	1.13	-75	-77.06	2.06	2.75

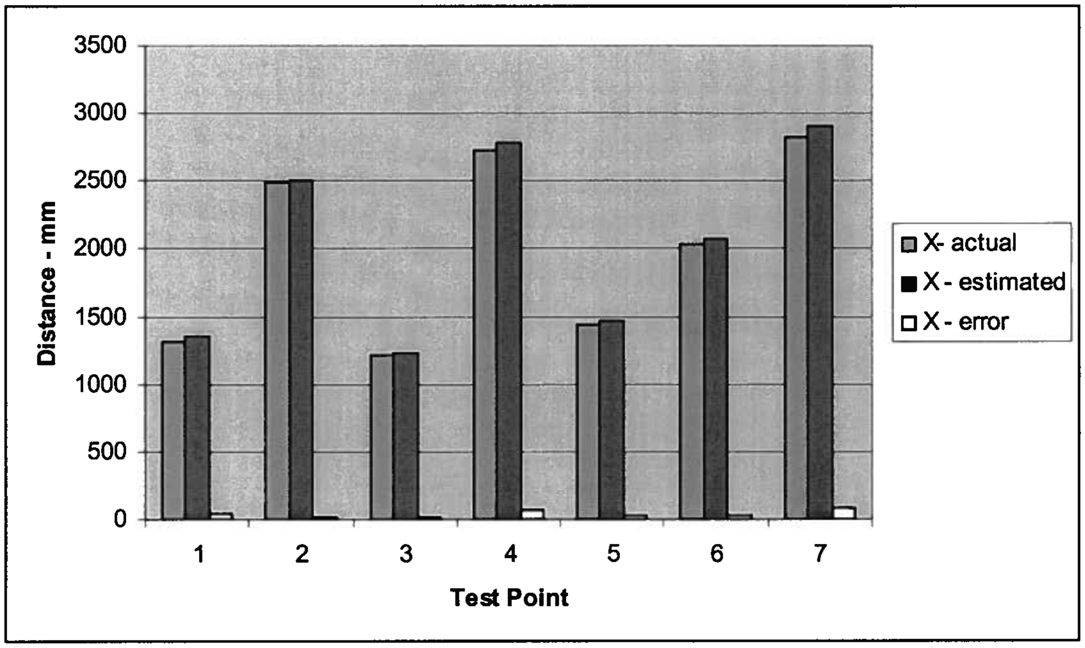


Figure 4.20: The x-axis error from Table 4.4.

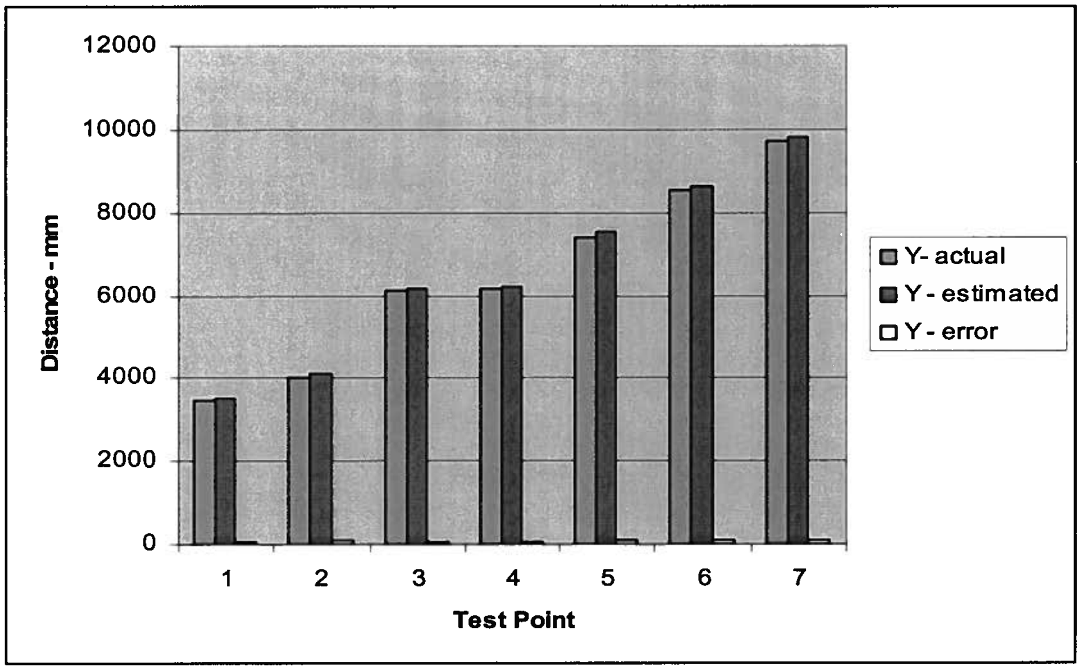


Figure 4.21: The y-axis error from Table 4.4.

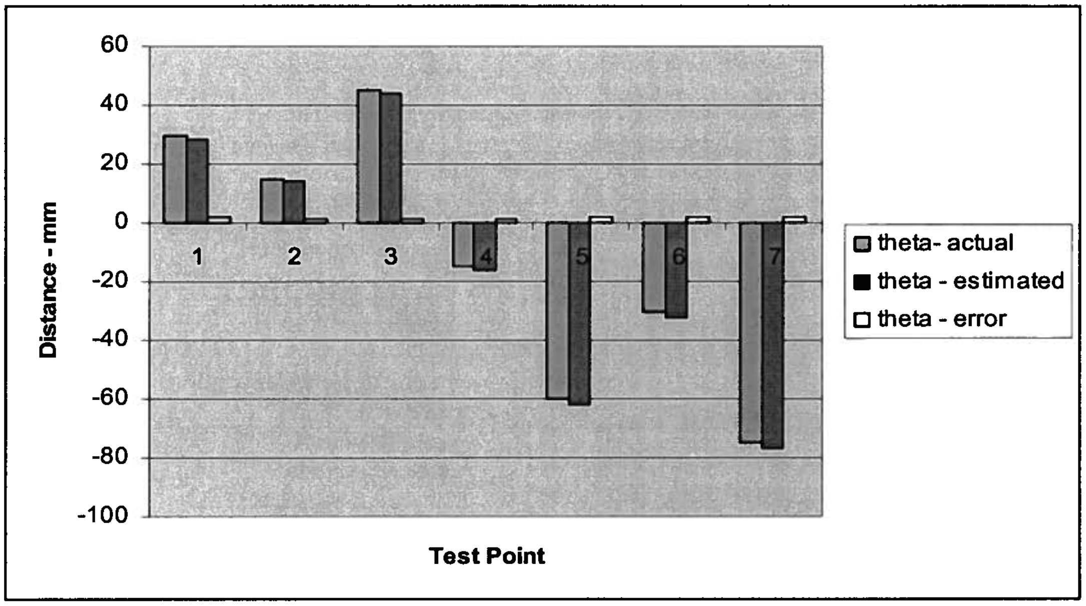


Figure 4.22: The orientation error from Table 4.4.

By comparing the % error of the results obtained when the robots were stationary with those when the robots were moving, it is seen that the latter results are better. When a robot autonomously explores the surroundings and finds an object, it stops at a suitable distance from the object, rearranges its pose to centre the color blob within the camera frame and then estimates the pose of the object. On the other hand, in the case of a stationary robot, the object is placed in front of the robot, and the associated distance may not be a suitable. Consequently, human error may have contributed to the enhanced % error in pose estimation with a stationary robot.

4.5 Summary

A method for pose estimation of an object for application in cooperative object transportation by mobile robots was developed in this chapter. A CCD camera, optical encoders and laser range finders were the sensors utilized by the robots. In order to transport the object by robots, first robot global localization was established. Next, the CCD camera was used to find the object in the work environment. Once the object was identified by using a color blob tracking approach, the robot rotated its base to move the color blob into the center of the camera frame. Finally a laser range finder was used to scan the object and to determine the distance and the orientation angle. The developed approach was thoroughly tested using a series of physical experiments.

Chapter 5

Experimentation in Distributed Multi-Robot Cooperative Object Transportation

5.1 Overview

Multi-robot object transportation is an active research area in the robotics community. The particular multi-robot system as developed in our laboratory (Industrial Automation Laboratory - IAL at the University of British Columbia - UBC) consists of three heterogonous multiple robots. It is intended to transport an object of interest to a goal location cooperatively in both static and dynamic environmental conditions. There are practical and theoretical issues, which need to be addressed for realizing the cooperative transportation task. First, techniques such as the modified Q learning algorithm and the multi-robot coordination framework in a multi-agent environment are used in the developed cooperative transportation system. These techniques deal with important issues that arise in a multi-agent environment. The related issues include task assignment, dynamic environment, and distributed control. Second, though multi-robot transportation itself applies in industrial projects and scientific explorations, it faces some challenges, which are addressed in the present thesis. As a rationale for a possible practical application, 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, multi-robot coordination mechanism, the modified Q learning algorithm, and the pose estimation technology, are integrated to implement a physical multi-robot transportation system operating in a dynamic and unknown environment. These techniques are tested and validated with the available mobile robots in IAL at UBC.

5.2 Test Bed

An experimental system has been developed in the IAL at UBC to implement a physical multi-robot object transportation system. An overview of this system is presented in Figure 5.1, which is a repetition of Figure 1.2.

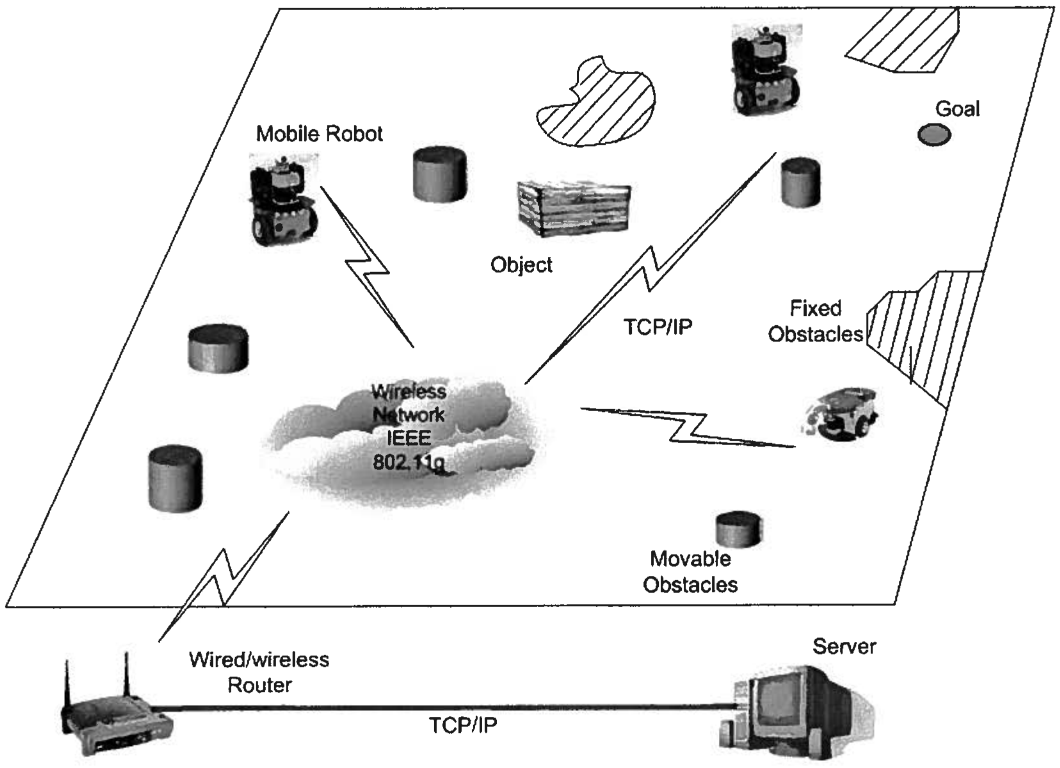
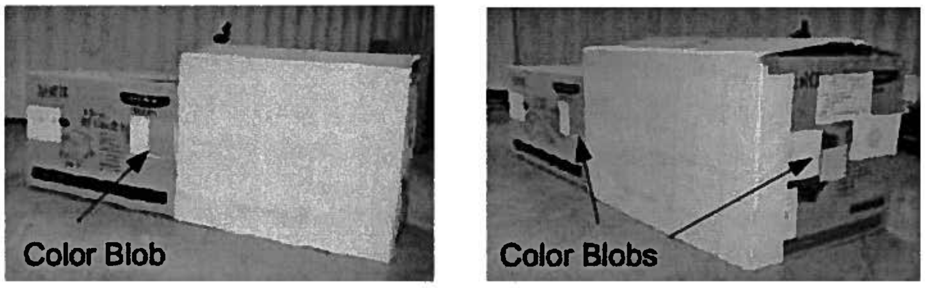


Figure 5.1: The multi-robot object transportation system.

Three autonomous mobile robots are used in the developed system to transport a heavy and long rectangular shaped object to a goal location in the testing area of our laboratory. According to the environment in the testing area, a simple global coordinate system is employed to validate the techniques developed in the previous chapters. In the beginning of the operation, each robot is informed of their initial position and orientation with respect to the global coordinate system. It estimates the latest pose (position and orientation) of a robot by recording and analyzing the data of the wheel encoders and the data of its compass sensor while it moves in the environment. Each robot in the system transmits continuously the current pose via the wireless network. In addition, the robots are assumed to know the global coordinates of the goal location in advance.

The object and the obstacles are randomly placed within the environment. When the multi-robot system begins to operate, robots have to search the environment in order to detect the object. Once a robot in the system detects the object, it will estimate the pose of the object. Then the particular robot will transmit the object pose to the system and this robot will act as a leader of the group for that particular instance. The leader assign task to the other robots for detecting obstacles in the environment. The task of detection and feature estimation of objects and obstacles is accomplish by fusing the sensory data from the sonar, laser range finder and CCD camera of a robot. The sensors are assumed to only detect obstacles within the detection radius of 2 meters, and a robot is able to exchange its sensory information with its peers via wireless communication to establish a binary word for state identification. The robots possess local sensing capabilities only and therefore, they are able to know only a local area bounded by the detection radius of the environment.

Two color blobs are present on the object of interest. One color blob is placed on one lateral side and on its opposite side. The other color blob is on the adjoining side and its opposite side. After a robot detects the color blobs using its CCD camera, the robot rearranges its pose to estimate the pose of the object by fusing information from the sonar and the laser range finder of the robot. If an object without any color blobs is detected within the environment, it will be regarded as an obstacle. Large and heavy objects with their color blobs are shown in Figure 5.2.



(a) (b)

Figure 5.2: The color blobs used to identify the orientation of an object.

Each robot makes decisions independently to build the state binary code utilizing the pose information of the objects and obstacles. Robots need to exchange this information with their peers in order to form a cooperation strategy and for making decisions. This subject has been addressed in Chapter 3.

The developed system consists of a computer server, 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 ActivMedia Robotics, which is an established company in the mobile robot industry. One four-wheel-driven Pioneer^{TM} 3-AT robot and two three-wheel-driven Pioneer^{TM} 3-DX robots are used in the cooperative object transportation system developed at our laboratory (IAL). 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 communication, laser sensing, and other autonomous functions. The P3 robots store up to 252 watt-hours of hot-swappable batteries. They have a ring of eight forward sonars and ring of eight rear sonars. Their powerful motors and 19 cm wheels reach speeds of 1.6 m/s 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 and accessories include laser based navigation, bumpers, gripper, vision, compass and a suite of other options. A P3-DX robot and a P3-AT robot are shown in Figure 5.3 and Figure 5.4, respectively.

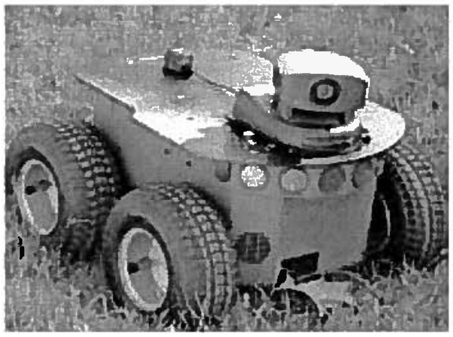


Figure 5.3: Pioneer^{TM} 3-AT mobile robot.

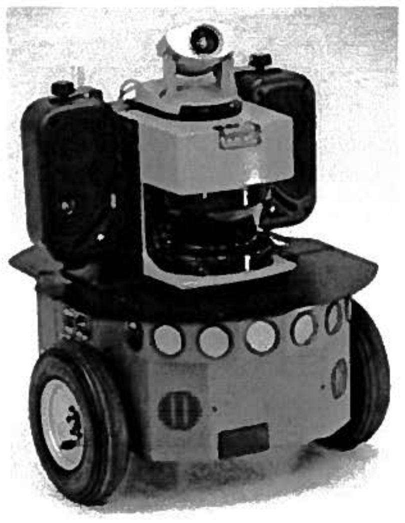


Figure 5.4: Pioneer^{TM} 3-DX mobile robot.

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

- Wander randomly
- Plan paths with gradient navigation
- Provide C/C++ development platform
- Localize using sonar and laser distance finder
- Display a map of its sonar and/or laser readings
- Drive under control of software, keys or joystick
- Communicate sensor and control information related to sonar, motor encoder, motor controls, user I/O, and battery charge data
- Simulate behaviors offline with the MobileSim^{TM} simulator, which accompanies each development environment.

The robot can building maps and constantly updates its position using its Laser Mapping and Navigation System and MobileEyes, within a few centimeters while traveling within mapped areas. The user is able to observe the view of the robot remotely, speak, and hear audio, with appropriate accessories and send the robot on patrol (wandering).

In summary, the Pioneer-3 DX or AT robots are all-purpose robots, used for research and applications involving mapping, tele-operation, localization, monitoring, reconnaissance, vision, manipulation, cooperation and other behaviors.

5.3 Physical Multi-Robot Transportation System

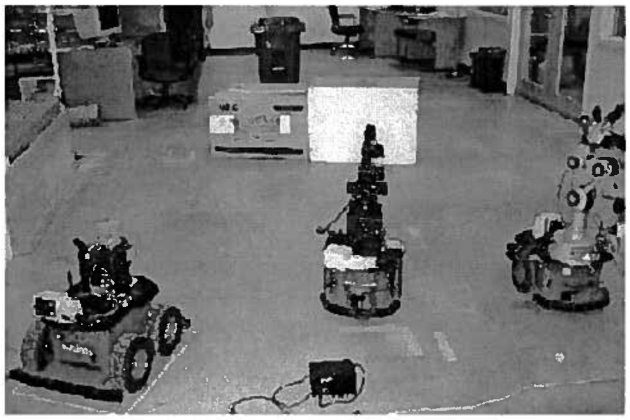
A physical multi robot transportation system has been developed to integrate various approaches developed in chapters 2 though 4. The system has local sensing capabilities only. An experiment is carried out to validate these approaches in a real environment in the presence of sensor noise.

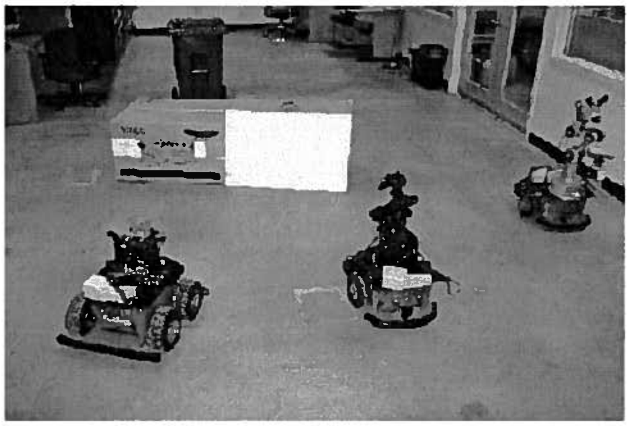
In this experiment, three mobile robots equipped with the modified Q learning algorithm are employed to test the cooperative transportation strategy of a large and heavy object. The object is to be transported from one end of the environment to a predetermined goal location in the presence of an obstacle.

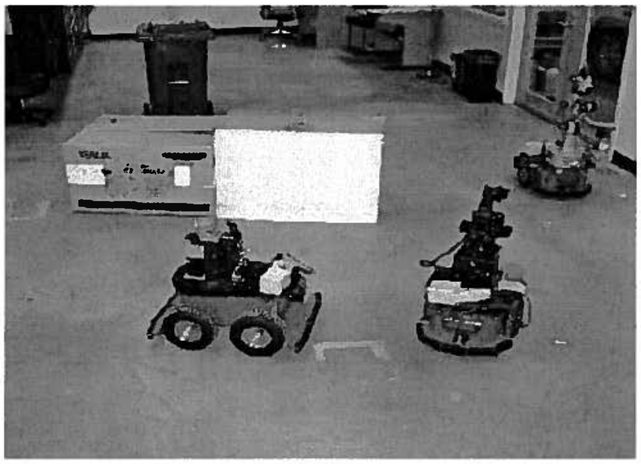
The simulation system described in Chapter 3 is employed to train the robots, before the experiment is carried out. In this manner, the Q-tables of the robots are improved according to the environment model. After 10,000 rounds of simulated object-pushing, the Q-tables of the three simulated robots are exported to the three physical robots to complete a physical cooperative object-pushing task.

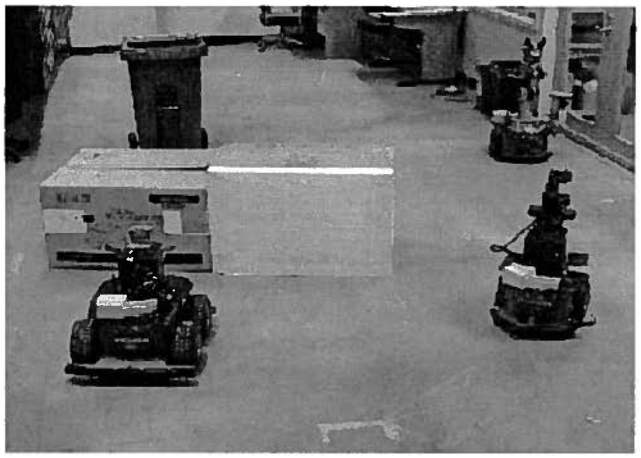
In the simulation system described in Chapter 3, after each step of object-pushing, the robots will identify the new environment state and select the corresponding actions according to the MQLA. The same policy is employed in the experimental system. However, a larger distance is selected as the detection radius for the physical multi-robot object-pushing system in comparison to the simulation system.

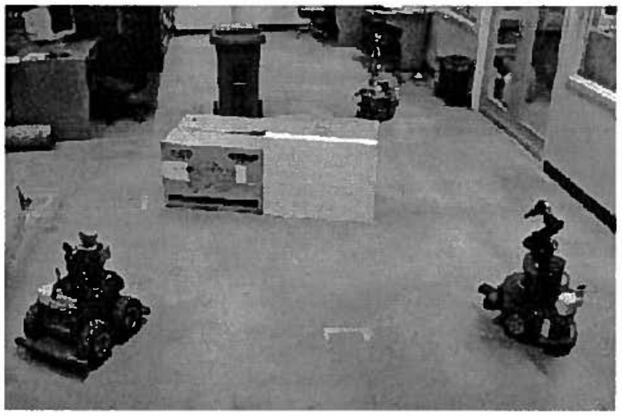
The experimental results of cooperatively transporting an object in a real environment are presented in Figure 5.5.


(a)

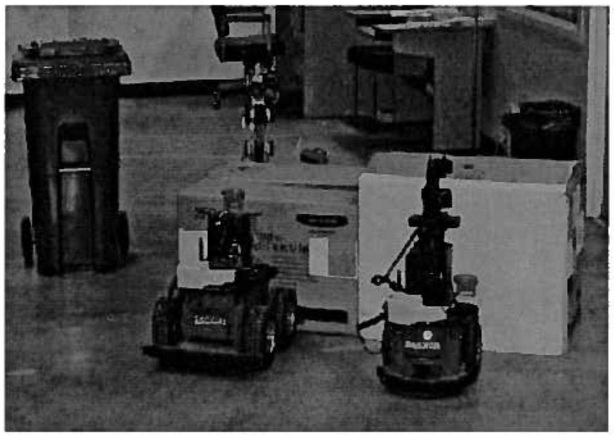

(b)


(c)


(d)

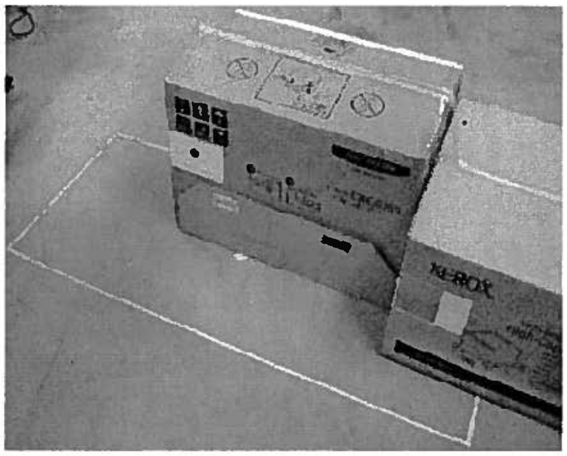

(e)

(f)


(g)

(h)

(i)



(j)

Figure 5.5: Multi-robot cooperative transportation.

The three mobile robots, the large object and an obstacle have been employed in our experimental setup. The three mobile robots are informed about their initial pose (positions and orientations) with respect to the global coordinate system before they begin their work. As shown in Figure 5.5 (a), first a large object is placed in the testing area of our laboratory. When the system operation starts, each robot uses its CCD camera to wander within the testing area and to detect the color blob on the object surface. Figure 5. 5 (b) shows one of the robot in the system detecting the object. If a robot among the three robots detects the color blob on the object surface, then that particular robot rearranges its current pose so as to center the color blob in its camera frame. Figure 5. 5 (c) shows the robot, which detects the object, has rearranged its current pose to center the color blob in its camera frame. After that, the pose of the object is estimated relative to the current pose of the particular robot using the sensory data from its laser range finder. Figure 5.5 (d) shows the robot, which detects the object, goes to estimate the pose of the object. By fusing relative pose of the object, the robot can estimate the pose of the object in the global coordinate system. This information is then transmitted to the other robots in the group. At the same time, the robot that detected the object will be assigned the leadership of the group and all other robots in the system will transfer their decision rights to the coordination unit of the leader robot through their arbitrators. After all the robots have transferred their decision making rights to the leader robot, it will assign tasks to peer robots in the system in order to assist them in identifying the obstacle distribution around the object. Figure 5.5 (e) shows how all the robots in the system try to identify the obstacle distribution around the object. The leader robot transmits all processed sensor information to the system. Then the robots will be able to establish the state of the current local environment individually by fusing the information of the object pose with the information of local obstacle distribution. Local environmental state is established by the robots, and the optimal action under this state is determined using its Q-table. Figure 5.5 (f) shows how the robots push the object with the selected actions.

If none of the robot in the system can find the object, those robots will wander in the environment until one of them detects the object.

While each robot tries to identify the existing obstacles within its detection area, simultaneously it scans its local environment utilizing the laser range finders. If an object is detected within this area, the robot needs to verify whether it is an obstacle or it is a peer robot.

Figure 5.5 (g) and (h) show the robots detecting an obstacle (the blue garbage bin) in the path and avoiding that obstacle. This obstacle had not been detected before by the sensors of the robots due to the limited detection radius.

Figure 5.5 (f) and (g) show that the robots have changed their formation to adapt to the new environment state. Here they attempt to change the orientation of the object in order to avoid the obstacle.

Figure 5.5 (g) and (h) show that the robots have avoided the obstacle successfully. Figure 5.5 (i) and (j) show that the robots effectively have completed a cooperative transportation task in an environment with one unknown obstacle. However, there is an error due to wheel slip, which corresponds to the difference between the actual location of the robot and its desired location.

From Figure 5.5 (a)-(j), 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 to select good cooperation strategies in a robust and effective manner.

5.4 Summary

A fully distributed multi-robot transportation system is physically implemented in the Industrial Automation Laboratory (IAL) at the University of British Columbia (UBC), which integrated the approaches developed in chapters 2 through 4. In the system, three mobile robots were employed to push a heavy object to a goal location in the laboratory environment. There were three challenges to achieve 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, the experimentation system presents sensor noise and uncertainty of actions, which did not exist in the simulation system. These constraints degrade the performance of the robots. Finally, the real environment is dynamic due to the presence of random obstacles. The distributed computing using C++ was introduced into the developed multi-robot system, which enabled the robots to cooperate in an easy and efficient manner.

An experiment was carried out to evaluate the performance of the developed distributed multi-robot system. In this experiment, cooperative robotic transportation of a long and heavy object in an environment with unknown obstacles was carried out. The experimental results showed that the developed multi-robot system has the ability to work well in a dynamic environment, with sufficient accuracy. It can be concluded that the approaches developed in chapters 2 through 4 are useful and effective in enabling a multi-robot system to operate in a dynamic and unknown environment.

Chapter 6

Conclusions

6.1 Primary Contributions

Multi-robot systems have to undergo many improvements before they can be used in real life applications. For example, the challenges that have to be faced are enormous in developing a system of multiple autonomous robots to cooperatively transport an object of interest in a dynamic and unknown environment like the surface of Mars. In this thesis, several techniques have been developed and integrated to support the operation of a multi-robot object transportation system in a complex and unstructured dynamic environment with an unknown terrain. 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 indicated below.

First, a general, distributed multi-robot architecture and a multi-robot coordination method were 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 coordination unit were combined to establish good cooperative strategies for the robots while they attempt to cooperatively transport an object. 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. In the state identification stage, the coordination unit would be temporarily activated by an arbitrator to search the environment according to the coordination mechanism for the object in order for the robots to estimate the pose and build the state binary word. In the middle level of the architecture, a 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 executed. This architecture also included a communication module so that any robot could easily exchange information with other robots in the team using standard network protocols. By combining the learning, coordination and behavior-based approaches, and carefully designing the robot coordination mechanism, 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 prototype system has the to cope with scalability, computational complexity and communication bottleneck when the number of robots is increased. To overcome the associated problems, a distributed Q-learning approach for multi-robot object transportation 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 adaptively and effectiveness in a complicated 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 fact, single-agent Q-Learning showed better performance (as measured by the average number of steps required in a round of object-pushing).

In order to overcome various disadvantages of the traditional single-agent Q-learning algorithm when extended to a multi-agent scenario, a modified Q-Learning algorithm termed MQLA was developed in this thesis. By enabling the robots to learn in a predefined sequence, the new MQLA algorithm was able to overcome several major shortcomings in the single-agent Q-Learning. Consequently it is more suitable for multi-robot tasks. The simulation results showed that the MQLA algorithm needed less time to complete a multi-robot transportation task and received better reward than the traditional Q-Learning algorithm.

Third, a pose estimation technique was developed to detect the object pose using two color-blobs simultaneously so that the orientations and positions of the robots could be determined in a multi-robot object transportation task. This method 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. With 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 constrains. In addition, in order to implement effective communication among multiple robots and facilitate proper execution of the new MQLA algorithm, the Microsoft visual studio (C++) technology was incorporated into the system. The experimental results showed that the developed physical system was able to operate effectively and robustly in a dynamic physical environment with obstacles.

6.2 Limitations and Suggested Future Research

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

6.2.1 Improvement of the Model and Algorithm of MQLA

The credit assignment is rather difficult in multi-robot learning. For example, when multiple robots cooperatively push an object to a goal location, how to split the observed global reward among the robots is an important issue. In MQLA developed in this thesis assumes each robot makes a positive contribution to the collective task, which may not be true in a real multi-robot system. In practice, each robot makes a different contribution to the collective task. If 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, MQLA simulation results in Chapter 3 have shown that the learning agents can converge slowly or even become confused in a multi-robot cooperative task, thereby making the MQLA algorithm ineffective. It follows that updating the Q-values with a global reward is not proper in a dynamic and unknown environment. Therefore, more work may be needed to estimate the real reward of each robot from the global reward signal for the MQLA algorithm.

6.2.2 Integration of Learning and Coordination

Learning and coordination are both important for multi-robot systems when operating in an unknown and dynamic environment. In this thesis, a type of switching strategy between learning and coordination was developed, where a coordination unit was temporarily activated when the stage of state identification of the machine learning approach was encountered. However, more advanced integration scheme would be useful so as to make use of coordination in a multi-robot task.

6.3 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 and coordination 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 important work has been carried out in this thesis towards this general objective, specifically with regard to stage of state identification, improved robustness, object pose estimation, obstacle pose estimation, and real-time operation.

Bibliography

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.

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, 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.

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.

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., "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.

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.

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, 2002.

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.

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.

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.

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

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.

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.

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.

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.

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, 2001.

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.

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.

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.

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.

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.

Parker, L.E., "Current state of the art in distributed autonomous mobile robotics," *Distributed Autonomous Robotic Systems 4*, Springer-Verlag, Tokyo, pp. 3-12, 2000.

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.

Russell S. and Norvig P., *Artificial Intelligence: A Modern Approach*, Second Edition, Pearson Education, Inc., Upper Saddle River, NJ, 2003.

Spong, M.W., Hutchinson, S., and Vidyasagar, M., *Robot Modeling and Control*, John Wiley & Sons, Inc., Hoboken, NJ, 2006.

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.

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.

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.

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, 2005.

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, 2006.

Wang, Y. and de Silva, C.W, "A machine learning approach to multi-robot coordination," *Engineering Applications of Artificial Intelligence (Elsevier)*, 2007.

Wang, Y., "Cooperative and Intelligent Control of Multi-robot Systems Using Machine Learning," *Thesis for The Degree of Doctor of Philosophy (University of British Columbia)*, 2007.

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.

Appendix 1– Program Functions

Summary

This appendix outlines some important classes and functions as used in the present work.

Classes

Development of the application in this thesis has used following important classes:
MyClient, MyACTS, MyPose MyDecodeEncode MyQLearningRobot

Functions

MyClient.h

```
class MyClient
{
public:
      MyClient(ArRobot* robot);
      void connectServer(int myNo);
      bool sendingString(char* string,bool broadcast,const char*
      WinnerIP);
};
```

MyACT.h

```
class MyGeneralACTS: public ArRobot
{
public:
      MyGeneralACTS(ArRobot* robot){};
      int ACTSColorBlob();
      void CentreTheBlob(ArRobot* robot,int x1,int x2,int y1,int
      y2);
};
```

MyPose.h

```
class MyPose: public ArRobot
{
public:
      MyPose(){};
      void FindObstacleGroup(ArRobot* robot, ArSick* sick,
      double* ObstacleArrayX, double* ObstacleArrayY);
      void poseOfTheBox( double x1, double y1, double x2, double
      y2, double theta1);
      ArPose poseEstimate(ArRobot* robot, ArSick* sick);
```

```
    };
```

MyQLearning.h

```
    class MyDecodeEncode: public ArRobot{

    public:
          MyDecodeEncode(ArRobot* robot){};
          void GetMailHeading(char* MailHeading);
          char* StringPackage(double x, double y, double theta);
          ArPose StringDecode(char* strToDecode);
    };

    class qLearningRobot: public ArRobot{

    public:
          qLearningRobot(ArRobot*robot):tao(0.9),action(0),actiomNum(
          6),successProcessCount(0),learningRate(0.8),discountRate(0.
          9),w1(0.7),w2(0.25),w3(0.05){};
          double detectionRadius(int width,int height);
          void selectAction(int StateCode);
          int probaSelect(double p[], int n);
          void calculateRobotActionPositions(int action, ArPose
          boxPose, ArPose p1);
          void calForcePara();
          void boxResponse(ArPose position);
          void calNextPosition(ArPose boxPose);
          bool qLearningRobot::obstacleDetect(ArPose boxPose,double*
          obstacleArrayNewX,double* obstacleArrayNewY);
          void calculateReward(ArPose oldPose);
          void updateQ(int state, int newState, double reward);
          void frameConvert(ArPose p1, ArPose p2, ArPose origin);
          void frame2To1(ArPose p1, ArPose p2, ArPose origin);
    };
```

General.cpp

```
    int BuildStateCode()
    void qTableInitialization()
    int main(int argc, char **argv)
```

Appendix 2– Program Codes

Summary

This appendix includes important Microsoft C++ files and their implementation codes.

C++ Files

MyClient.h, `MyACT.h`, MyPose.h, MyQLearning.h, General.cpp

MyClient.h

```
/*******************************************************************
This program will uses to to communicate between robots during the
transportation process. It acts as a client to all robots individually
in the group to receive the handshaking signal from other robots.
*******************************************************************/

#include "string.h"
#include <iostream>
#include <sstream>


class MyClient
{
public:

      MyClient(ArRobot* robot);
      ~MyClient(){};
      void connectServer(int myNo);
      bool sendingString(char* string,bool broadcast,const char*
WinnerIP);
      const char* ipAddress;
      int myOwnNo;
      void close();

private:

      ArRobot* myRobot;
      ArPose currentPose;
      double currentX;
      double currentY;
      double currentTh;
      char *strToSend;
      char strToBroadcast[128];
      //const char* ipAddress;
      char *tok;
      char strToGet[10];
      char *otherIP[7];
      char *strToEnd;
      char buffer[33];
      int roboNo;
      int maxRoboNo;
      ArMutex myCycleCallback;
      ArSocket globalOtherSoc[5];
      size_t strSize;
      time_t timeAuctionStart;
};


MyClient::MyClient(ArRobot* robot):
strToEnd("BYE!!!"),
myRobot(robot)
{
      myExitPlanCB.setName("myExitPlan");
      Aria::addExitCallback(&myExitPlanCB, 21);


      //Other robots' IP address
      otherIP[0] ="192.168.1.3;
      otherIP[1] ="192.168.1.4";
      otherIP[2] ="192.168.1.5";

}

void MyClient::connectServer(int myNo)
{
      //connect to other servers
      Sleep(10000);
      int counter;
      myOwnNo = myNo;
      for (roboNo = 0; roboNo <maxRoboNo+1; roboNo++)
      {

            if (roboNo == myNo)
                  roboNo++;
                  if (roboNo > maxRoboNo)
                  break;
            for (counter = 0; counter < 1; counter++)
            {
                  if (globalOtherSoc[roboNo].connect(otherIP[roboNo],
7777+roboNo, ArSocket::TCP))

                  {
                    break;
                  }

            }
            if (counter == 1)
                        }
}

bool MyClient::sendingString(char* string, bool broadcast)
{
      strToSend = string;
            if (broadcast == true)
```

```
        {
                        for (roboNo = 0; roboNo <maxRoboNo+1; roboNo++)
                {

                        if (roboNo == myOwnNo && roboNo < maxRoboNo)
                                roboNo++;
                        if (WinnerIP != NULL)
                                if (strcmp(WinnerIP,otherIP[roboNo]) == 0 )
                                {

                                 continue;
                                }
                        if (roboNo > maxRoboNo)
                                break;

        globalOtherSoc[roboNo].write(strToSend,strlen(strToSend));
                }
                return true;
        }
        else
        {
                if (WinnerIP != NULL)
                        ipAddress = WinnerIP;
                else



                for (roboNo = 0; roboNo < maxRoboNo+1;roboNo++)
                {
                        if (strcmp(ipAddress,otherIP[roboNo]) == 0)
                        {

        if(globalOtherSoc[roboNo].write(strToSend,strlen(strToSend)) !=
strlen(strToSend))
                                {
                                        return false;
                                }
                                return true;
                        }
                }
        }
        return false;
}

void MyClient::close()
{
        myCycleCallback.tryLock();
        for(roboNo =0; roboNo < maxRoboNo+1; roboNo++)
        {
        if(globalOtherSoc[roboNo].write(strToEnd, strlen(strToEnd)) ==
strlen(strToEnd))
                printf("See You again!!!");
        }
        myCycleCallback.unlock();
}
```

MyACT.h

```
#include "Aria.h"
#include "ACTSClient.h"

#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include <signal.h>

using namespace std;

class MyGeneralACTS: public ArRobot
{
public:
      MyGeneralACTS(ArRobot* robot){};
      ~MyGeneralACTS(){};
      int ACTSColorBlob();
      void CentreTheBlob(ArRobot* robot,int x1,int x2,int y1,int y2);
      int getXcg(){return xcg;}
      int getYcg(){return ycg;}
private:
      int xcg,ycg;
      int channel;
      blob generalblob;
      ACTSClient actsClient;
};

int MyGeneralACTS::ACTSColorBlob()
{

            if(!actsClient.openPort("5001"))
      {
            cout<<"Sorry can't open acts port\n"<<endl;
            return 0;
      }

      actsClient.requestPacket();

            if( actsClient.receiveBlobInfo())
      {


            if (actsClient.getBlob(0,0, &generalblob)&&
generalblob.area > 5000)

                  {
                        cout<<"Blob Area :" <<generalblob.area<<endl;
                        cout<<"Received the Blob\n"<<endl;
                        xcg = generalblob.xcg;
                        ycg = generalblob.ycg;

                        actsClient.closePort();
                        return 1;
                        }
            else
                  {
                        cout<<"Blob Area :" <<generalblob.area<<endl;
                        cout<<"Blob is not the correct one\n");
                        actsClient.closePort();
                        return 2;
                        }
      }
      else
      {
            cout<<"The blob is not received\n"<<endl;
            {
                  actsClient.closePort();
                  return 2;
            }
      }
}

void MyGeneralACTS::CentreTheBlob(ArRobot* robot,int x1,int x2,int
y1,int y2)
{
// 1st quadrant
            if(((x1-x2)> 40) && ((y1-y2)< -40))
                  {
                  // (1)move backward           (2)move left

      cout<<"The Blob is in the 1st Quadrant"<<endl;
      double robotAngle = robot->getPose().getTh();

      cout<<"robotAngle: "<<robotAngle<<endl;

// the angle needs to be determined by either getTh or boxPoseEstimate
      robot->setHeading(robotAngle+90);
      ArUtil::sleep(8000);

      robot->move(250);
      ArUtil::sleep(8000);

      robot->setHeading(robotAngle+180);
      ArUtil::sleep(8000);

      robot->move(700);
      ArUtil::sleep(8000);

      robot->setHeading(robotAngle);
      ArUtil::sleep(8000);
                  }

// 2nd quadrant
            if(((x1-x2)> 40) && ((y1-y2)> 40))
                  {
                  // (1)move forward            (2)move left

      cout<<"The Blob is in the 2nd Quadrant"<<endl;
      double robotAngle = robot->getPose().getTh();

      cout<<"robotAngle: "<<robotAngle<<endl;
// the angle needs to be determined by either getTh or boxPoseEstimate
      robot->setHeading(robotAngle+90);
      ArUtil::sleep(8000);

      robot->move(250);
      ArUtil::sleep(8000);

      robot->setHeading(robotAngle);
      ArUtil::sleep(8000);

      robot->move(100);
      ArUtil::sleep(8000);

                  }

// 3rd quadrant
            if(((x1-x2)< -40) && ((y1-y2)> 40))
                  {
                  // (1)move forward            (2)move right

      cout<<"The Blob is in the 3rd Quadrant"<<endl;
      double robotAngle = robot->getPose().getTh();
      // double robotAngle = ((22.0 * (90 +
(robot.getPose().getTh())))/ (180.0 * 7.0));
      cout<<"robotAngle: "<<robotAngle<<endl;

// the angle needs to be determined by either getTh or boxPoseEstimate

      robot->setHeading(robotAngle-90);
      ArUtil::sleep(8000);

      robot->move(250);
      ArUtil::sleep(8000);

      robot->setHeading(robotAngle);
      ArUtil::sleep(8000);

      robot->move(100);
      ArUtil::sleep(8000);
                  }

// 4th quadrant
            if(((x1-x2)< -40) && ((y1-y2)< -40))
                  {
                  // (1)move backward           (2)move right
      cout<<"The Blob is in the 4th Quadrant"<<endl;
      double robotAngle = robot->getPose().getTh();
      cout<<"robotAngle: "<<robotAngle<<endl;

// the angle needs to be determined by either getTh or boxPoseEstimate

      robot->setHeading(robotAngle-90);
      ArUtil::sleep(8000);


      robot->move(250);
      ArUtil::sleep(8000);

      robot->setHeading(robotAngle-180);
```

```
	ArUtil::sleep(8000);

	robot->move(700);
	ArUtil::sleep(8000);

	robot->setHeading(robotAngle);
	ArUtil::sleep(8000);

		}
}
```

MyPose.h

```
/*****************************************************************
This program uses to detect the pose of the box (position and
orientation),which uses data from a laser rangefinder to detect a
continues line of points.
******************************************************************/

#include <cmath>

using namespace std;



class MyPose: public ArRobot{

	public:
	double center_a, center_b;
	MyPose(){};
	~MyPose(){};

	void FindObstacleGroup(ArRobot* robot, ArSick* sick);
	void poseOfTheBox( double x1, double y1, double theta1);
	ArPose poseEstimate(ArRobot* robot, ArSick* sick);

	private:

	ArPose currentPose;
	ArPose boxPose;
	double boxCentreX;
	double boxCentreY;
	double boxCentreTh;

	double currentX;
	double currentY;
	double currentTh;

};
```

```
/***************** This method use to find obstacles ***************/

void MyPose::FindObstacleGroup(ArRobot* robot, ArSick* sick){


      double angle,dist,dist1,angle1, CenterDist, CenterAngle;
      std::list<ArPoseWithTime *> *readings;
      std::list<ArPoseWithTime *>::iterator it;

      sick->lockDevice();
      dist1 = sick->currentReadingPolar(-60,60,&angle);

      readings = sick->getCurrentBuffer();

      int a1=0, a2=0, a4=0;
      int ObstacleMinLineDiff = 300;

      a1 = readings->size();

      double* distArray = new double[a1];
      double* angleArray = new double[a1];
      double* ObstacleDistArray = new double[a1
      double* ObstacleAngleArray = new double[a1

     for (it = readings->begin(); it != readings->end(); it++){

        distArray[a2] = robot->getPose().findDistanceTo(ArPose((*it)->
getX(),(*it)->getY()));
        angleArray[a2] = robot->getPose().findAngleTo(ArPose((*it)->
getX(),(*it)->getY()));

        a2++; Sleep(100);
      }

    for (int a3=0; a3<a2-1; a3++){
      if (abs(distArray[a3] - distArray[a3+1])> ObstacleMinLineDiff){
                  if (distArray[a3] > distArray[a3+1]){
                        ObstacleDistArray[a4] = distArray[a3];
                        ObstacleAngleArray[a4] = angleArray[a3];

                  }
                  else
                  {     ObstacleDistArray[a4] = distArray[a3+1];
                        ObstacleAngleArray[a4] = angleArray[a3+1];
                  }
                  a4++;
            }

      }
      int indexLocal = (int)((a4/2)-1);
      double* LocalObstacleArrayX = new double[indexLocal];
      double* LocalObstacleArrayY = new double[indexLocal];

    for (int a3=1; a3 < ((a4/2)+1); a3++){

     CenterDist =(ObstacleAngleArray[a3] + ObstacleAngleArray[a3+1])/2;
     CenterAngle = (abs(ObstacleAngleArray[a3] -
ObstacleAngleArray[a3+1])/2) + ObstacleAngleArray[a3+1];
     LocalObstacleArrayX[a3-1] = CenterDist * sin(CenterAngle);
     LocalObstacleArrayY[a3-1] = CenterDist * cos(CenterAngle);

// above corrdinate related to robot, convert into Global coordinate

      }
      ObstacleArrayX = LocalObstacleArrayX;
      ObstacleArrayY = LocalObstacleArrayY;
}

void MyPose::poseOfTheBox( double x1, double y1, double theta1){
      double x_face, y_face;
      int height = 1700;
      int width = 720;
      int half_height = 850;
      int half_width = 360;

      x_face = (x1 + x2)/2;
      y_face = (y1 + y2)/2;
      center_a = x_face + (half_width * cos (theta1));
      center_b = y_face + (half_width * sin (theta1));
      cout<<" Box Center X and Y: \n"<< center_a <<"  "<<center_b
<<endl;
}


ArPose MyPose::poseEstimate(ArRobot* robot, ArSick* sick)
{

  double angle,dist,dist1,angle1,startangle,startdist,endangle,enddist;
  double temp_angle,temp = 0.0;

      std::list<ArPoseWithTime *> *readings;
      std::list<ArPoseWithTime *>::iterator it;

      ArUtil::sleep(1000);

      sick->lockDevice();
      dist1 = sick->currentReadingPolar(-60,60,&angle);

      readings = sick->getCurrentBuffer();

      int i=0, j=0, a=0, b=1, c=1;
      int a1=0, a2=0, a3, minAngle=0, minPosi;
      double distance=300.0, s=0;

      a1 = readings->size();
      double* distArray = new double[a1];
      double* angleArray = new double[a1];
      double dist11, dist2, angle11, angle2, frontDistance;
      double maxmumLineDiff=300;
      for (it = readings->begin(); it != readings->end(); it++){

      double dist001 = robot->getPose().findDistanceTo(ArPose((*it)->
getX(),(*it)->getY()));
      double angle001 = robot->getPose().findAngleTo(ArPose((*it)->
getX(),(*it)->getY()));

      distArray[a2] = robot->getPose().findDistanceTo(ArPose((*it)->
getX(),(*it)->getY()));
      angleArray[a2] = robot->getPose().findAngleTo(ArPose((*it)->
getX(),(*it)->getY()));

            a2++;
               }

         for (a3=0; a3<=a1; a3++) {

              if ((angleArray[a3])>= minAngle){
                  frontDistance = distArray[a3];
                  minPosi = a3;
                  break;
               }
         }

         for (a3=minPosi-1; a3>=0; a3--){

          if (abs(distArray[a3+1]- distArray[a3])>maxmumLineDiff){

            dist11 = distArray[a3+1];
            angle11 = angleArray[a3+1];
                  break;
               }

         }

         for (a3=minPosi+1; a3<=a1; a3++){

          if (abs(distArray[a3-1]- distArray[a3])>maxmumLineDiff){

            dist2 = distArray[a3-1];
            angle2 = angleArray[a3-1];
                  break;
               }

         }

           Startdist = dist11;
           startangle= angle11;
           enddist = dist2;
           endangle= angle2;


        // Start and End angle calculation based on the XY plan
           double alpha=0,beta=0,d1,d2;

           alpha = ((22.0 * (90 + startangle))/ (180.0 * 7.0));
           beta = ((22.0 * (90 + endangle))/ (180.0 * 7.0));
```

```
            d1= startdist;
            d2= enddist;

double x1_new, y1_new, x2_new, y2_new, theta1, num, dnum;
double newcenter_b ,newcenter_a;

            x1_new = d1 * cos(alpha);
            y1_new = d1 * sin(alpha);
            x2_new = d2 * cos(beta);
            y2_new = d2 * sin(beta);

            num = y2_new - y1_new;
            dnum = x1_new - x2_new;
            theta1 = (180 * (7 * atan(num / dnum)))/22;

            poseOfTheBox(x1_new, y1_new, x2_new, y2_new, theta1);

/******************* Error Handler-Start *******************/
            newcenter_a = center_a;
            newcenter_b = value + center_b;

            boxCentreX =newcenter_a;
            boxCentreY =newcenter_b;
            boxCentreTh =theta1;
            boxPose.setPose(boxCentreX,boxCentreY,boxCentreTh);

/******************* Error Handler-End *********************/


/*************** Globle Box Pose - Start *******************/
            double x_globle_new,y_globle_new,Th_globle_new;
            double angle_new,gama_globle_new;

        Th_globle_new = ((22 * (robot->getPose().getTh()))/ (180 * 7));
        angle_new = (robot->getPose().getTh()) - theta1;

        x_globle_new = (robot->getPose().getX()) + center_a *
cos(Th_globle_new) + center_b * sin (Th_globle_new);
        y_globle_new = (robot->getPose().getY()) + center_a * cos
(Th_globle_new) - center_b * sin(Th_globle_new);
        gama_globle_new = (180 * 7 * angle_new)/22;

        cout<< "****************************************** \n" <<endl;
        cout<< "Globle Box Position : ( " << x_globle_new << " , " <<
y_globle_new << " , " << angle_new << " )"<<endl;
        cout<< "****************************************** \n"<<endl;

/******************** Globle Box Pose - End ********************/

  sick->unlockDevice();

  robot->waitForRunExit();

  return boxPose;
}
```

MyQLearning.h

```
class MyDecodeEncode: public ArRobot{

public:

      MyDecodeEncode(ArRobot *robot, ArSick* sick, ArAnalogGyro*
gyro){};
      ~MyDecodeEncode(){};
      void GetMailHeading(char* MailHeading);
      char* StringPackage01(bool boxOrRobot,double x, double y, double
theta);
      char* StringPackage02(int state);
      ArPose StringDecode(char* strToDecode);
      int StringDecodeState(char* strToDecode);
      int capabilityCheck(bool arm,bool gripper,bool camera);
      char* dec2bin(int statecode,char *binaryequivalent);
      int totalStateCode01(int statecode01,int statecode02);
      int totalStateCode02(int stateCodeInt01,int stateCodeInt02);

private:
      double posex,posey,posetheta;
      int state;
      char* strToSend ;
      char strToGet[128];
      char strToBroadcast[128];
      char* strToDecode;
      char* actionToSend ;
      char actionToGet[128];
      char actionToBroadcast[128];
      char* actoinToDecode;
      char buffer[33];
      char *tok;
      char* localMailHeading;
      bool isFinished;
      double forceValue[6];
      size_t strSize;
      ArPose boxOrRobotpose;
      ArPose position;
};

char* MyDecodeEncode::StringPackage01(bool boxOrRobot,double x, double
y, double theta)
{
      if (boxOrRobot == true){

            strcpy(strToBroadcast,"boxPose!!!");
            strcat(strToBroadcast,itoa((int)x,buffer,10));
            strcat(strToBroadcast,"!!!");
            strcat(strToBroadcast,itoa((int)y,buffer,10));
            strcat(strToBroadcast,"!!!");
            strcat(strToBroadcast,itoa((int)theta,buffer,10));
            strToSend = strcat(strToBroadcast,"!!!");

      }
      else {

            strcpy(strToBroadcast,"robotPose!!!");
            strcat(strToBroadcast,itoa((int)x,buffer,10));
            strcat(strToBroadcast,"!!!");
            strcat(strToBroadcast,itoa((int)y,buffer,10));
            strcat(strToBroadcast,"!!!");
            strcat(strToBroadcast,itoa((int)theta,buffer,10));
            strToSend = strcat(strToBroadcast,"!!!");

      }
      return strToSend;
}

char* MyDecodeEncode::StringPackage02(int state)
{
            strcpy(actionToBroadcast,"An environment state!!!");
            strcat(actionToBroadcast,itoa((int)state,buffer,10));
            strToSend = strcat(actionToBroadcast,"!!!");

            return actionToSend;
}

ArPose MyDecodeEncode::StringDecode(char* strToDecode)
{
      tok = strtok(strToDecode,"!!!");
      localMailHeading = tok;

      if ( strcmp(tok,"boxPose") ==0)
      {
            tok = strtok(NULL,"!!!");
            posex =atoi(tok);
            tok = strtok(NULL,"!!!");
            posey = atoi(tok);
            tok = strtok(NULL,"!!!");
            posetheta = atoi(tok);
            boxOrRobotpose.setPose(posex,posey,posetheta);
      }


      if ( strcmp(tok,"robotPose") ==0)
      {
            tok = strtok(NULL,"!!!");
            posex =atoi(tok);
            tok = strtok(NULL,"!!!");
            posey = atoi(tok);
            tok = strtok(NULL,"!!!");
            posetheta = atoi(tok);
            boxOrRobotpose.setPose(posex,posey,posetheta);
      }
            return boxOrRobotpose;
      }
```

```
int MyDecodeEncode::StringDecodeState(char* strToDecode)
{
      tok = strtok(strToDecode,"!!!");
      localMailHeading = tok;

      if ( strcmp(tok,"An environment state") ==0)
      {
            tok = strtok(NULL,"!!!");
            state =atoi(tok);

      }
      return state;
      }

int MyDecodeEncode::capabilityCheck(bool arm,bool gripper,bool camera)
{
      int capabilityCheck =0;
// armonly=1; gripperonly=2; cameraonly=3; arm&&gripper=4;
// arm&&camera=5; gripper&&camera=7; arm&&gripper&&camera=8;
// nocomponent=0;

      if (arm == true)
      {
            capabilityCheck ++;
      }

      if (gripper == true)
      {
            capabilityCheck =capabilityCheck+2;
            capabilityCheck ++;
      }

      if (camera == true)
      {
            capabilityCheck =capabilityCheck+3;
            capabilityCheck ++;
      }

      return capabilityCheck;
}

char* MyDecodeEncode::dec2bin(int statecode,char *binaryequivalent)
{
      int k =0,n = 0;
      int  remain,oldStatecode;
      char temp[16];
      do
        {
          oldStatecode = statecode;
          remain     = statecode % 2;
          statecode   = statecode / 2;
          temp[k++] = remain + '0';
        } while (statecode > 0);

            while (k >= 0)
                  binaryequivalent[n++] = temp[--k];
                  binaryequivalent[n-1] = 0;
      return binaryequivalent;
}
int MyDecodeEncode::totalStateCode01(int statecode01,int statecode02)
{

      int  b, k, m, n;
      int  sum = 0;
      char binaryequivalent01[16];
      char binaryequivalent02[16];


      for (int i = 0;i < 16;i++)
      {

            binaryequivalent01[i]= '\0';
            binaryequivalent02[i]= '\0';
      }

      bitset<16> binary01 ( string (dec2bin(statecode01,
      binaryequivalent01)));
      bitset<16> binary02 ( string(dec2bin(statecode02,
      binaryequivalent02)));
      bitset<16> binaryequivalent = binary01 | binary02;

      for(k = 0; k <= 15; k++)
        {
            n = (int)(binaryequivalent[k] - '0');
                  if ((n > 1) || (n < 0))
                        {
                          return (0);
                  }
                  for(b = 1, m = 15; m > k; m--)
                        {
                          b *= 2;
                  }
                     sum = sum + n * b;
        }
            return(sum);
}

int MyDecodeEncode::totalStateCode02(int stateCodeInt01,int
stateCodeInt02)
{
      int totalStateCode;
      totalStateCode=(stateCodeInt01 | stateCodeInt02);
      return totalStateCode;
}

class MyqLearningRobot: public ArRobot{

public:

   MyqLearningRobot(ArRobot *robot, ArSick* sick, ArAnalogGyro* gyro)
      :tao(0.9),action(0), learningRate(0.8),discountRate(0.9)
      ,lArm(20),w1(0.7),w2(0.25),w3(0.05){};
   ~MyqLearningRobot(){};
   double detectionRadius(int width,int height);
   int selectAction(int StateCode);
   int probaSelect(double p[], int n);
   ArPose calculateRobotActionPositions(int action, ArPose boxPose);
   void calForcePara();
   void boxResponse(ArPose position,ArPose newPose);
   void calNextPosition(ArPose boxPose);
   bool obstacleDetect(ArPose boxPose);
   void calculateReward(ArPose oldPose);
   void updateQ(int state, int newState, double reward);
   void frameConvert(ArPose p1, ArPose p2, ArPose origin);
   ArPose frame2To1(ArPose p2, ArPose origin);
   bool arriveGoal(ArPose theGoal,ArPose boxPose);


private:
   double posex,posey,posetheta;
   size_t strSize;
   ArPose boxpose;
   int forcevalue;
   double forceValue[6];
   int actionNum;
   int state;
   int reward;
   int action;
   double angle;
   double netForceAngle;
   double netForceValue;
   double netTorque;
   double* obstacleArrayNewX;
   double* obstacleArrayNewY;
   double q[6];
   double sum;
   double minQ;
   double learningRate;
   double discountRate;
   double tao;
   int lArm;
   double Mf,Mt;
   double w1,w2,w3;

};

double MyqLearningRobot::detectionRadius(int width,int height){
   double detectRadius;
   detectRadius = (sqrt((double)(((width/2)*(width/2))+
                  ((height/2)*(height/2)))))+400;
   return detectRadius;
}

int MyqLearningRobot::selectAction(int state)
{
      sum=0;
      minQ=100000000;
      double ranSelProb=0.1;
      double v=rand()*100;
```

```
        if(v<=ranSelProb*100)
        {
              action=(int)(rand()*6);
              return action;
        }

        for(int i=0; i<=5; i++)
        {
              q[i]=qTable[state][i];

              if(q[i]<minQ)
                    minQ=q[i];
        }

        for(int i=0; i<=actionNum-1; i++)
        {

              q[i]=q[i]-minQ;
        }

        for(int i=0; i<=actionNum-1; i++)
        {

        sum=0;

        sum=sum + exp(q[i]/tao);
        }

        double p[6];

        for(int i=0; i<=actionNum-1; i++)
        {
              p[i] = exp(q[i]/tao)/sum;
        }

        tao=tao*0.999;
        action=probaSelect(p,actionNum);

        return action;
  }

int MyqLearningRobot::probaSelect(double p[], int n)
        {
        double accumSum[7];
        accumSum[0] = 0;
        for(int i=1; i<=n;i++)
        {
              accumSum[i]=accumSum[i-1]+p[i-1]*100;
        }

        double v=rand()*100;
        for(int i=0; i<=n; i++)
        {
              if(v>=accumSum[i])
                    continue;
              else
                    return i-1;
        }
        return 0;
}


ArPose MyqLearningRobot::calculateRobotActionPositions(int action,
ArPose boxPose)
{

        double w=1700; double h=720;
        ArPose p2;

        if (action==0)
        {
                p2.setX(w/2); p2.setY(0);
        }

        if (action==1)
        {
                p2.setX((w/2)-50); p2.setY(h/2);
        }

        if (action==2)
        {
                p2.setX(-((w/2)-50)); p2.setY(h/2);
        }

        if (action==3)
        {
                p2.setX(-w/2); p2.setY(0);
        }

        if (action==4)
        {
                p2.setX(-((w/2)-50)); p2.setY(-h/2);
        }

        if (action==5)
        {
                p2.setY((w/2)-50); p2.setY(-h/2);
        }

        ArPose p1 = frame2To1(p2,boxPose);
        return p1;

}

void MyqLearningRobot::calForcePara()
{
    double PI = 3.14159265358979323846264338327950;
    for(int i=0; i<6;i++)
        {
          if (forceValue[i]>30)
            forceValue[i]=0;
        }
    double fx=forceValue[3]-forceValue[0];
    double fy=forceValue[4]+forceValue[5]-forceValue[2]-forceValue[1];
```

```
   netForceAngle=(atan2(fy,fx))/PI*180;
   netForceValue=sqrt((fx*fx)+(fy*fy));
   netTorque=(forceValue[5]+forceValue[2]-forceValue[1]-forceValue[4])*
      lArm;
}

bool MyqLearningRobot::obstacleDetect(ArPose boxPose)
{
      double rr=0;
      ArPose obsta;
      ArPose obstacleLocalPos(0,0);

      for(int i=0;i<sizeof (obstacleArrayNewX);i++)
      {

            obsta.setX(obstacleArrayNewX[i]);
            obsta.setY(obstacleArrayNewY[i]);
            frameConvert(obsta,obstacleLocalPos, boxpose);

            double x=obstacleLocalPos.getX();
            double y=obstacleLocalPos.getY();
            double width=1700;
            double height=720;

       if((abs(x)<=width/2) && (abs(y)<=(225+height/2)))
                return true;
       if((abs(y)<=height/2) && (abs(x)<=(225+width/2)))
                return true;

       bool tmp=false;
       tmp=((x-width/2)*(x-width/2)+(y-height/2)*(y-height/2))<=225*225;
       tmp=tmp || (((x-width/2)*(x-width/2)+(y+height/2)*
               (y+height/2))<=225*225);
       tmp=tmp || (((x+width/2)*(x+width/2)+(y-height/2)*(y-height/2))
                <=225*225);
       tmp=tmp || (((x+width/2)*(x+width/2)+(y+height/2)*
               (y+height/2))<=225*225);
       tmp=tmp && (abs(x)>(width/2)) && (abs(y)>=height/2);

       if(tmp)
               return true;
      }
               return false;
}

void MyqLearningRobot::boxResponse(ArPose position,ArPose newPose)
{
      bool isFinished;
      isFinished = false;
      ArPose oldPose;

      oldPose.setX(position.getX());
      oldPose.setY(position.getY());
      oldPose.setTh(position.getTh());
      int tMax=1;
      for(int i=0; i<tMax;i++)
        {
            calNextPosition(newPose);

            if (obstacleDetect(newPose))
            {
                  break;
            }
            else
            {
                  position.setX(newPose.getX());
                  position.setY(newPose.getY());
                  position.setTh(newPose.getTh());

            }
   }

   calculateReward(oldPose);

   isFinished=true;

}


void MyqLearningRobot::calNextPosition(ArPose boxPose)
{
   ArPose position;
   double PI = 3.14159;

      boxPose.setX(position.getX()+Mf*netForceValue*cos((netForceAngle+
                                    position.getTh())/180*PI));
      boxPose.setY(position.getY()+Mf*netForceValue*sin((netForceAngle+
                                    position.getTh())/180*PI));
      boxPose.setTh(position.getTh()+netTorque*Mt);

}


void MyqLearningRobot::calculateReward(ArPose oldPose)
{

      //  calalculate reward Distance
      double dOld=pow((oldPose.getX()-theGoal.getX()),2)+
      pow((oldPose.getY()-theGoal.getY()),2);
      dOld=sqrt(dOld);

      double dNew=pow((position.getX()-theGoal.getX()),2)+
      pow((position.getY()-theGoal.getY()),2);
      dNew=sqrt(dNew);

      double ratioE=0.3;
      double rDist=(dOld-dNew)*ratioE;

      ArPose tmpObst;
      ArPose obstLocalPosition;
      double obstAngle;
      double min=1000;

      //  calalculate reward Obstacle
      for(int i=0;i<sizeof (obstacleArrayNewX);i++)
      {

       tmpObst.setX(obstacleArrayNewX[i]);
       tmpObst.setY(obstacleArrayNewY[i]);

       double tmpDist=pow((oldPose.getX()-tmpObst.getX()),2)+
             pow((oldPose.getY()-tmpObst.getY()),2);
       tmpDist=sqrt(tmpDist);
         if(tmpDist>(detectionRadius(1700,720)))
            continue;

            frameConvert(tmpObst, obstLocalPosition, oldPose);

      obstAngle=atan2(obLocPosition.getY(),obLocPosition.getX())/PI*180;

         double tmpDiff=abs(netForceAngle-obstAngle);
            if (tmpDiff>180)
                  tmpDiff=360-tmpDiff;

            if (tmpDiff<min)
                  min=tmpDiff;
      }

         double rObstacle=0;
            if(min==1000)
                  rObstacle=1;
            else
                  rObstacle=(sin(min/180*PI/2.0)-0.3)*1.5;

      //  calalculate reward Roation

      double rRotation=abs(position.getTh()-oldPose.getTh());
      rRotation=cos(rRotation/180*PI)-0.9;

      //  total reward
      reward=w1*rDist+w2*rObstacle+w3*rRotation;


}


void MyqLearningRobot::updateQ(int state, int newState, double reward)
{
      int actionNum = 6;
      double learningRate = 0.8;
      double discountRate = 0.9;
      double q[6];
      double vMax = -10000;
```

```
	for(int i=0; i<=actionNum-1; i++)
	{
		q[i]=qTable[newState][i];
		if(q[i]>=vMax)
			vMax=q[i];
	}

	qTable[state][action]=(1-learningRate)*qTable[state]
			[action]+learningRate*(reward+discountRate*vMax);

}



void MyqLearningRobot::frameConvert(ArPose p1, ArPose p2, ArPose origin)
{

	angle = (angle/(180*(PI)));

	double x1 = p1.getX();
	double y1 = p1.getY();
	double x2 = origin.getX();
	double y2 = origin.getY();

	p2.setX((x1*cos(angle)+(y1*sin(angle)));
	p2.setY((y1*cos(angle)-(x1*sin(angle)));
}

ArPose MyqLearningRobot::frame2To1(ArPose p2, ArPose origin)
{
	ArPose p1;

	double x1 = p2.getX();
	double y1 = p2.getY();
	double x2 = origin.getX();
	double y2 = origin.getY();
	double theta = origin.getTh();

	p1.setX((x1*cos(theta)) - (y1*sin(theta))+ x2);
	p1.setY((x1*sin(theta)) + (y1*cos(theta))+ y2);
	return p1;
}

bool MyqLearningRobot::arriveGoal(ArPose theGoal,ArPose boxPose)
{
	double minSuccessDist = 1000;
	double d=0;

	double x = boxPose.getX()- theGoal.getX();
	double y = boxPose.getY()- theGoal.getY();
	d=sqrt((x*x)+(y*y));
	if (d<=minSuccessDist)
		return true;
	else
		return false;
}
```

General.cpp

```
/*******************************************************************
This program will uses to push the Box to the given location in the
control environment at the IAL. During the transportation task need
to consider about the speed limiting actions to the collision avoidance
as well. Press escape to shut down Aria and exit.
******************************************************************/
#include <bitset>
#include<iostream>
#include<conio.h>
#include<stdlib.h>
#include "Aria.h"
#include "Server.h"
#include "Client.h"
#include "Pose.h"
#include "Qlearning.h"
#include "GeneralACTS.h"

using namespace std;


double objectPoseX,objectPoseY,objectPoseTh;
double* obstacleArrayX; double* obstacleArrayY;
double* obstacleArrayNewX;double* obstacleArrayNewY;

double detectionRobotArrayX[5];double detectionRobotArrayY[5];
double detectionRobotArrayTh[5]; double dR;


int BuildStateCode(ArPose objectCenter)
{
      double objectCenterX = objectCenter.getX();
      double objectCenterY = objectCenter.getY();
      double objectCenterTh = objectCenter.getTh();
      ArPose theGoal;
      theGoal.setPose(500,7000);
      double PI = 3.14159265358979323846264338327950;
      int stateCodeOfThisRobot;

      double goalAngle=(atan2((theGoal.getY()-objectCenterY),
                        (theGoal.getX()-objectCenterX)))/PI*180;

      if(goalAngle<0)
            goalAngle=goalAngle+360;
            stateCodeOfThisRobot = (int)(goalAngle/(360.0/256
      for(int i=0; i< sizeof(obstacleArrayNewX);i++)
        {
            ArPose localPos;

            objectCenterTh=objectCenterTh/180*(PI);
            double x1 = obstacleArrayNewX[i]; //ObstacleArrayX[i];
            double y1 = obstacleArrayNewY[i]; //ObstacleArrayY[i];
            double x2 = objectCenterX;
            double y2 = objectCenterX;
```

```
     localPos.setX((x1*cos(objectCenterTh))+(y1*sin(objectCenterTh)) -
                  (x2*cos(objectCenterTh))-(y2*sin(objectCenterTh)));
     localPos.setY((y1*cos(objectCenterTh))-(x1*sin(objectCenterTh)) +
                  (x2*sin(objectCenterTh))-(y2*cos(objectCenterTh)));
     double angle=(atan2(localPos.getY(), localPos.getX()))/PI*180;

            if(angle<0)
             angle=angle+360;
             int obstaStat=(int)(angle/45.0);
             obstaStat= (int)pow(2.0,obstaStat);
            stateCodeOfThisRobot=stateCodeOfThisRobot  |  (obstaStat<<8);
      }
      return stateCodeOfThisRobot;
}

void qTableInitialization()
{
      double qTable[8192][6];
      for (int i = 0;i < 8192; i++){
            for (int j = 0;j < 6; j++){
                  qTable[i][j]=0;
            }
      }
}

int main(int argc, char **argv)
{
  Aria::init();
  int robotNo = 2;

  ArPose initialPose(xvalue,yvalue);
  ArPose currentPose;
  ArPose theGoal;
  theGoal.setPose(500,7000);

  ArRobot robot;
  ArSonarDevice sonar;
  ArSick sick;
  ArKeyHandler keyHandler;
  ArArgumentParser parser(&argc, argv);
  ArSimpleConnector simpleConnector(&argc, argv);

  robot.addRangeDevice(&sonar);
  robot.addRangeDevice(&sick);

  // Create objects
  MyServer myServer;
  MyClient myClient(&robot);
  MyReading myReading(&robot);
  MyGeneralACTS myACTS(&robot);
  MyPose myPose;
  MyDecodeEncode myDecodeEncode(&robot, &sick, &gyro);
  MyqLearningRobot myqLearningRobot(&robot, &sick, &gyro);

  Aria::setKeyHandler(&keyHandler);
  robot.attachKeyHandler(&keyHandler);
```

```
// the actions we'll use for wander
  ArActionStallRecover recover;
  ArActionBumpers bumpers;
  ArActionAvoidFront avoidFrontNear("Avoid Front Near", 225, 0);
  ArActionAvoidFront avoidFrontFar;
  ArActionConstantVelocity constantVelocity("Constant Velocity", 150);
  ArActionGoto gotoPoseAction("goto",ArPose(0,0,0),100,100,80,2);

  // Parse all command line arguments
  if (!simpleConnector.parseArgs() || argc > 1)
  {
      simpleConnector.logOptions();
      keyHandler.restore();
      return 1;
  }
  // Connect to the robot
  if (!simpleConnector.connectRobot(&robot))
  {
    printf("Could not connect to robot... exiting\n");
    Aria::exit(1);
  }
  // Connect to the Laser Range Finder

  if (!simpleConnector.connectLaser(&sick))
  {
    printf("Could not connect to Laser Range Finder... exiting\n");
    Aria::exit(1);
  }
  if (!Aria::parseArgs() || !parser.checkHelpAndWarnUnparsed())
  {
    Aria::logOptions();
    exit(1);
  }

  cout<<"Robot Th: "<< robot.getPose().getTh()<<endl;

  // add the actions
  robot.addAction(&recover, 100);
  robot.addAction(&bumpers, 75);
  robot.addAction(&avoidFrontNear, 51);
  robot.addAction(&avoidFrontFar, 35);
  robot.addAction(&constantVelocity, 25);
  robot.addAction(&gotoPoseAction, 50);

//*********************************************************************
  myServer.open(7777);
  Sleep(1000);
  myServer.runAsync();
  myReading.runAsync();
  myClient.connectServer(7777);
  robot.setAbsoluteMaxTransAccel(40);

  robot.runAsync(true);
  robot.comInt(ArCommands::ENABLE,1);

  const char* myIP= "IP Adress";
  char* newMail = NULL;
```

```
int noOfRobot=0;
int count1 = 0;
int count2 = 0;
int totalNoOfRobot = 3;
double dR,dRX,dRY,thetaDr;
int width = 1700;
int height = 720;
ArPose aa,bb;
char *ss;
char *messageStr;
char *messageState;
double* x; double* y; double* th;
char* mailheading = NULL;
int stateWord = 0; int action = 0;
int stateCodeInteger = 0;
char *otherIP[7];
otherIP[0] ="192.168.1.3";
otherIP[1] ="192.168.1.4";
otherIP[2] ="192.168.1.5";
dR = myqLearningRobot.detectionRadius(1700,720);


while (Aria::getRunning())
{
   robot.clearDirectMotion();
   newMail = myReading.checkMail();

  if(newMail=="STOP")
     {
       robot.stop();
       cout<<"wait! "<<endl;
     }

  if (newMail=="Estimated Object Pose Transmitted")
     {
       aa = myDecodeEncode.StringDecode(ss);
       myDecodeEncode.GetMailHeading(mailheading);

  if(mailheading == "boxPose")
      {
        objectPoseX = aa.getX(); objectPoseY = aa.getY();
        objectPoseTh = aa.getTh();
        thetaDr = atan((objectPoseX-robot.getPose().getX())/
                      (objectPoseY-robot.getPose().getY()));
        dRX = aa.getX()+dR*cos(thetaDr);
        dRY = aa.getY()+dR*sin(thetaDr);

        gotoPoseAction.setGoal(ArPose(dRX, dRX, 0));
        // move to detection radious

          robot.addAction(&bumpers,75);
          robot.addAction(&avoidFrontNear, 52);
          robot.addAction(&avoidFrontFar, 20);

   while(!gotoPoseAction.haveAchievedGoal())
    Sleep(10000);
   myClient.sendingString("Leader I'm in detection radius",true,NULL);
```

```
    }
   }

   if (newMail=="Leader I'm in detection radius")
    {
      if (((robot.getPose().getX()- dRX) <= 50) &&
                 ((robot.getPose().getY()- dRY) <= 50)){
           for (int IP=0; IP<3; IP++)
            {
              if ((!strcmp(myIP,otherIP[IP])==0) &&
                 (!strcmp((char*)1010,otherIP[IP])==0))
               {
                 messageStr = myDecodeEncode.StringPackage01
                        (false,robot.getPose().getX(),robot.getPose().
                               getY(),robot.getPose().getTh());
 myClient.sendingString(messageStr,true, otherIP[IP]);
 myClient.sendingString("Close to detection radius",true, otherIP[IP]);
 robot.stop();
               }
            }
     }
 }

if (newMail=="I am in close to detection radius")
 {
   bb = myDecodeEncode.StringDecode(messageStr);
   myDecodeEncode.GetMailHeading(mailheading);
   if(mailheading == "robotPose")
     {
       if (!(robotNo == 1010))
         {
           detectionRobotArrayX[noOfRobot]= bb.getX();
           detectionRobotArrayY[noOfRobot]= bb.getY();
           detectionRobotArrayTh[noOfRobot]= bb.getTh();
           noOfRobot ++;
            if (noOfRobot == 2)
             {
              double twoX = detectionRobotArrayX[0]-
                                    detectionRobotArrayX[1];
              double twoY = detectionRobotArrayY[0]-
                                    detectionRobotArrayY[1];
              double V=(double)sqrt((twoX*twoX)+(twoY*twoY));

                 if (V >= ((2*dR)-500))
                   {
                    myPose.FindObstacleGroup(&robot, &sick);
                    if ((sizeof(obstacleArrayX)== 0) &&
                                   (sizeof(obstacleArrayY)== 0))
                    int index = 0;
                    double obstacleArrayNewX[sizeof
                                          (obstacleArrayX)];
                    double obstacleArrayNewY[sizeof
                                          (obstacleArrayX)];
                    for (int i=0; i <(sizeof (obstacleArrayX)-1); i++)
                                          {

                      double bbx = aa.getX();
```

```
                        double bby = aa.getY();
                        double oAx = obstacleArrayX[i] - bbx;
                        double oAy = obstacleArrayY[i] - bby;
                        double oAxy = sqrt((oAx*oAx)+(oAy*oAy));

                        if (oAxy <= dR)
                         {
                         obstacleArrayNewX[index] = obstacleArrayX[i];
                         obstacleArrayNewY[index] = obstacleArrayY[i];
                         index++;
                         }
                      }
                   }
                }
             }
          }
       }

stateCodeInteger = BuildStateCode(aa,obstacleArrayNewX,
                                             obstacleArrayNewY);

   for (int infoTransmit=0; infoTransmit<3; infoTransmit++)
    {
     if (!strcmp(myIP,otherIP[infoTransmit])==0)
      {
      messageState =  myDecodeEncode.StringPackage02
                                             (stateCodeInteger);
      myClient.sendingString(messageState,true,otherIP[infoTransmit]);
      myClient.sendingString("generate the satate code",true,
                                             otherIP[infoTransmit]);

      }
    }

     if (newMail=="generate the satate code")
      {
       stateWord = myDecodeEncode.StringDecodeState(messageState);
       myDecodeEncode.GetMailHeading(mailheading);

       if (mailheading == "An environment state")
         {
          if  (robotNo == 1010)
                  {

                  action = myqLearningRobot.selectAction(stateWord);
                  myClient.sendingString("I received an action i"
                                                      ,true,NULL);
                  }
         }

      if (newMail == " I received an action i ")// i = 1,2,3,4,5,6
       {
         count1 ++;
         if (count1 == totalNoOfRobot)
          {
          ArPose actionPosition = myqLearningRobot.calculateRobotAction
                                             Positions(action, aa);
```

```
           gotoPoseAction.setGoal(actionPosition);

           robot.addAction(&bumpers,75);
           robot.addAction(&avoidFrontNear, 52);
           robot.addAction(&avoidFrontFar, 20);

            while(!gotoPoseAction.haveAchievedGoal())
                  Sleep(100);
                  robot.stop();
                  myClient.sendingString("I am in the action
                                              position!!!",true,NULL);
                  robot.remAction(&bumpers);
                  robot.remAction(&avoidFrontNear);
                  robot.remAction(&avoidFrontFar);

      }
   }
           if (newMail == "I am in the action position")
             {
                count2 ++;
                  if (myqLearningRobot.arriveGoal(theGoal,aa) == true)
                               {
                                     cout<<"Now in the goal"<<endl;
                                     break; //end program
                               }
                        bb = myPose.poseEstimate(&robot, &sick, &gyro);

              }

  while(robot.checkRangeDevicesCurrentPolar(-10,10)>1200)
   {
      currentPose = robot.getPose();
      currentX = currentPose.getX();
      currentY = currentPose.getY();
      currentTh = currentPose.getTh();
            Sleep(500);
                        }

if(robot.checkRangeDevicesCurrentPolar(-30,30)<1200)
         {
            robot.stop();

            //    2     |     3
            //    ----|----
            //    1     |     4

            int blobx1,blobx2,bloby1,bloby2;
            int blobInfo = myACTS.ACTSColorBlob();

                  cout<<"blobInfo: "<<blobInfo<<endl;
                  ArUtil::sleep(8000);

          switch(blobInfo)
           {
             case 1:    // received the blob

              myClient.sendingString("STOP",true,NULL);
              myACTS.getXcg();
              myACTS.getYcg();
              cout<<"Blob readings -> XCG: "<<myACTS.getXcg()<<"YCG:
                                     "<<myACTS.getYcg()<<endl;
              blobx1 = 320; bloby1 = 240;
              blobx2 = myACTS.getXcg();
              bloby2 = myACTS.getYcg();
              myACTS.CentreTheBlob(&robot,blobx1,blobx2,bloby1,bloby2);
              ArUtil::sleep(8000);
              double x, y, theta;
              myClient.sendingString("Object Pose Estimating",
                                                           true,NULL);

              aa = myPose.poseEstimate(&robot, &sick, &gyro);
              x = aa.getX();
              y = aa.getY();
              theta = aa.getTh();

            // robot detects the color blob; change its number
             myClient.myOwnNo = 1010;
             myClient.sendingString("Object  Pose  Estimated",true,NULL);

             ss = myDecodeEncode.StringPackage01(true,x, y, theta);

             for (int robotId=0; robotId<3; robotId++)
              {
               if (!strcmp(myIP,otherIP[robotId])==0)
                 myClient.sendingString(ss,true,otherIP[robotId]);
              }
                ArUtil::sleep(8000);
                myClient.sendingString("Estimated Object Pose
                                           Transmitted",true,NULL);
                  break;


             case 2:    // not received the blob
             cout<<"Sorry you haven't luck this time"<<endl;
             cout<<"I'll try my best find bolb here if possible"<<endl;
                  int randomAngle = rand()%90 - 45;
                  cout<<"Random Angle: "<<randomAngle<<endl;
                  ArUtil::sleep(8000);
                  robot.setHeading(randomAngle);
                  ArUtil::sleep(8000);
                  break;
            }
      }

 // Robot disconnected, shut down
 Aria::shutdown();
 return 0;
 }
}
```