

國立交通大學

電機與控制工程學系

碩 士 論 文

智慧型機器人之彈性化即時控制系統

**A Flexible Real-Time Control System for
Autonomous Mobile Robots**



研 究 生：林 嘉 豪

指 導 教 授：宋 開 泰 博 士

中華民國九十三年七月

智慧型機器人之彈性化即時控制系統

A Flexible Real-Time Control System for Autonomous Mobile Robots

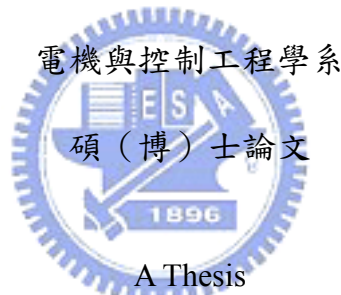
研究生：林嘉豪

Student: Chia-How Lin

指導教授：宋開泰 博士

Advisor: Dr. Kai-Tai Song

國立交通大學



Submitted to Department of Electrical and Control Engineering
College of Electrical Engineering and Computer Science
National Chiao-Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master
in
Electrical and Control Engineering
July 2004
Hsinchu, Taiwan, Republic of China

中華民國2004年7月

智慧型機器人之彈性化即時控制系統

研究生：林嘉豪

指導教授：宋開泰 博士

國立交通大學電機與控制工程學系

摘要

本論文以多代理人導向的方式設計一套具有高發展彈性的機器人即時控制系統。本系統的主要目的在於整合異質性機器人控制程式，同時確保系統能有即時的反應效率，並在彈性與通用性之間取得平衡。本架構可幫助開發團隊成員將一複雜之控制系統依照其功能性切割成多個代理人，如此系統的發展得以平行進行，各程式也能共享有限的硬體資源。本系統並提供即時的訊息流通以及決策機制，開發人員只需專注在演算法的開發而不是系統整合的細節部分。在軟體方面本論文使用 RTAI 這套即時 Linux 套件來實現，整個架構並提供開放式代理人樣版，使得開發者可以在維持擴充性以及相容性的前提下移植並整合各種異質性的程式。本論文以實驗室研製之家用機器人平台為主，實際製作環境探索、影像追蹤系統及雙機器人合作任務，以驗證本系統的控制效果。

A Flexible Real-time Control System for Autonomous Mobile Robots

Student: Chia-How Lin Advisor: Dr. Kai-Tai Song

Department of Electrical and Control Engineering
National Chiao-Tung University

Abstract

This thesis studies an agent-based approach to developing a flexible real-time robot control system. The main purpose of the system is to integrate heterogeneous algorithms and control methods while still guarantee real-time responding of robotic behaviors. The balance between generality and flexibility is also considered. The proposed architecture facilitates a robot research team to divide a complex control system into agents according to various functions. Therefore, the development can be carried out simultaneously in parallel and integrated in a latter phase. Moreover, the programs can share limited hardware resources onboard the robot. The system also provides a real-time message delivery and decision making mechanism, which help the designers to concentrate on the development of individual algorithm: The software framework of the system is realized using a Linux platform equipped with Real Time Application Interface (RTAI). An agent template is provided with scalability and compatibility to port and integrate heterogeneous programs. Finally, the control system has been tested using a laboratory mobile robot to realize environmental exploration, visual tracking, and a two-robot-cooperation. Experiment results verify the effectiveness of the proposed architecture.

ACKNOWLEDGMENT

First of all, I would like to express my deepest sense of gratitude to my advisor Dr. Kai-Tai Song for his patient guidance, advice encouragement and excellent advice throughout this study.

I am thankful to Professor Gary Anderson for his participation and advice for the experiment of my thesis.

I would like to thank Dr. Yu-Lun Huang and Dr. Jwu-Sheng Hu, for their comments and suggestions for the editing of my thesis.

I also thank my colleagues in isci lab, Hennry, Chih-Chieh, Chih-How, Yao-Qing, for sharing experiences and knowledge during the time of study.

Finally, I take this opportunity to express my profound gratitude to my beloved parent, and my friends for their moral support and patience during my study in NCTU.



CONTENTS

ABSTRACT(CHINESE)	i
ABSTRACT(ENGLISH)	ii
ACKNOWLEDGMENT	iii
CONTENTS	iv
LIST OF TABLES	vi
LIST OF FIGURES	vii
1 . INTRODUCTION	1
1.1. Motivation	1
1.2. Background and Related Work	1
1.2.1. Hybrid System	1
1.2.2. Software Frameworks for Robot Development.....	3
1.2.2.1. Classification of deliberation and reactive behaviors.....	4
1.2.2.2. Real-time responding vs. computing time.....	5
1.2.2.3. Team work in robot development.....	6
1.3. Problem Statements	7
1.4. Organization of the Thesis.....	7
2 . SYSTEM DESIGN	8
2.1. System Overview.....	8
2.2. Properties of Robot Control System.....	9
2.2.1. Flexibility	9
2.2.2. Real-time Response and Adjustable Execution Period.....	9
2.2.3. Connectivity	10
2.2.4. Loosely Classified	10
2.2.5. Generality	11
2.3. Proposed System Architecture.....	11
2.3.1. System Components	12
2.3.2. Inter Agent Communication	15
2.3.3. Examples	16
2.3.3.1. Multi-agent cooperation	16
2.3.3.2. Multi-robot control	18
2.4. Summary.....	20
3 . SOFTWARE REALIZATION	21
3.1. RTAI	21
3.1.1. Overview	21
3.1.2. Compare with Other RTOS	23
3.1.2.1. Microsoft Windows®	23
3.1.2.2. Commercial hard real-time operating systems	23
3.1.3. RTAI Features.....	24
3.1.3.1. Scheduling.....	24

3.1.3.2.	Inter Process Communication (IPC).....	24
3.1.3.3.	Memory management.....	25
3.1.3.4.	Soft-hard real time in user space: LXRT.....	25
3.1.3.5.	Real-time API for hardware.....	26
3.2.	Implementation.....	26
3.2.1.	Clock.....	26
3.2.2.	Short-term Memory.....	27
3.2.3.	Message Delivery and Router.....	28
3.2.4.	Agent.....	30
3.2.5.	Decision Maker and Knowledge base.....	32
3.2.6.	Resource Manager.....	32
3.3.	Summary.....	32
4 .	EXPERIMENTAL RESULTS	33
4.1.	The experimental mobile robots.....	33
4.2.	Experiment of single robot exploration.....	34
4.2.1.	Goal.....	34
4.2.2.	Implementation.....	34
4.2.2.1.	Sensor agents.....	35
4.2.2.2.	Actuator agents.....	36
4.2.2.3.	Functional agents.....	36
4.2.2.4.	Decision making.....	37
4.2.3.	Experimental Result.....	38
4.3.	Experiment of two robot cooperation.....	41
4.3.1.	Goal.....	41
4.3.2.	Implementation.....	42
4.3.2.1.	Localization.....	42
4.3.2.2.	Robot 1(Exploration).....	44
4.3.2.3.	Robot 2(Task Execution).....	45
4.3.3.	Experimental Result.....	47
5 .	CONCLUSIONS AND FUTURE WORK.....	52
5.1.	Conclusions.....	52
5.2.	Future work.....	52
	REFERENCES	54
	APPENDIX A- DISTANCE ESTIMATION OF CHECKPOINT.....	56

LIST OF TABLES

Table 1-1 A comparison of three different styles..... 5



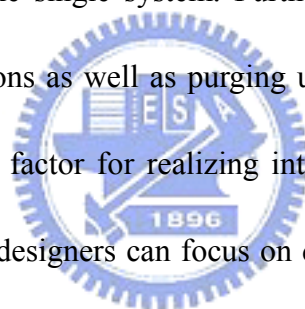
LIST OF FIGURES

Figure 1-1 Hybrid architecture for robot control.....	2
Figure 2-1 A simplified example for period adjusting	10
Figure 2-2 Global view of the control system architecture	12
Figure 2-3 Sensor agent pattern.....	13
Figure 2-4 Actuator agent pattern.....	14
Figure 2-5 An example of the control system	17
Figure 2-6 An example of a 2-robot cooperation system	19
Figure 3-1 RTAI architecture.....	22
Figure 3-2 The concept of the circular buffer.....	27
Figure 3-3 The connection list of the router	29
Figure 3-4 The basic design pattern of agents.....	30
Figure 4-1 (a) H1 robot platform (b) H2 robot platform.....	34
Figure 4-2 The system organization e robot exploration experiment.....	35
Figure 4-3 The sonar sensors module.....	36
Figure 4-4 Recorded trajectory of the robot.....	40
Figure 4-5 The environment of the experiment.....	40
Figure 4-6 The changes of the (a) velocity, (b) angular speed, and (c) angle versus time of this experiment.....	41
Figure 4-7 The artificial landmark: checkpoint.....	43
Figure 4-8 The programming organization of robot 1.....	44
Figure 4-9 The programming organization of robot 2.....	46
Figure 4-10 The recorded trajectories of two robots	48
Figure 4-11 (a) Robots at the starting point (b)Robot 2 at via point 2 (c) Robot 2 at via point 2 (d)Robot 2 start the right turn (e)Robot 2 approaching target point (e)Robot 2 arrived	49

1. Introduction

1.1. Motivation

The main purpose of this study is to develop a flexible control system for autonomous mobile robots. An autonomous robot is a complex system, which integrates various control functions for practical applications, such as autonomous navigation, human face tracking and recognition, etc. It is desirable to integrate these functions, developed by different people or on different platforms, into one single system. Furthermore, the system needs to keep the capacity of adding new functions as well as purging unnecessary functions. Flexible control architecture is therefore a key factor for realizing intelligent autonomous control. Via such control architecture, the robot designers can focus on developing their own sub-systems, and the architecture will do the rest: integrating these sub-systems, managing the resources, exchanging the messages, etc. Moreover, the architecture archives improved performance by integrating different control methods, rather than using a single complex controller.



1.2. Background and Related Work

1.2.1. Hybrid System

In the last two decades, many researchers have made their efforts in developing control architectures for autonomous and intelligent robotic systems. The trend converges to a similar architectural solution – a hybrid architecture that integrates deliberative and reactive strategies,

as shown in Fig. 1-1. The reactive approaches provide features such as fast response without complex representational symbolic knowledge. The deliberative systems, on the other hand provide capabilities such as planning, problem solving and goals achieving with a global view. The hybrid, multi-tiered architecture has the advantages of both and satisfies task-oriented specific requirements [1]. Arkin proposed the “Autonomous Robot Architecture (AuRA)” [2], known as the first example of hybrid architecture, which integrates behavioral, perceptual, and environment information. AuRA has been successfully implemented and affected other control architectures further on.

Recently, many researchers in the traditional AI field have become involved in robotics, and led to several interesting new design concept. The concept of agents and multi-agent system, for example, has been widely used in the field of intelligent robotics. Agents are computer systems situated in some environment that are capable of flexible, autonomous action in order to meet their design objectives. Agents can exist in a structure as complex as a global Internet or one as simple as a module of a common program. Agents can be autonomous entities, deciding their next step without the interference of a user, or they can be controllable, serving as a middleware between the user and other agents, thus they can exhibit deliberation, rationality, and intelligence in achieving their goals.

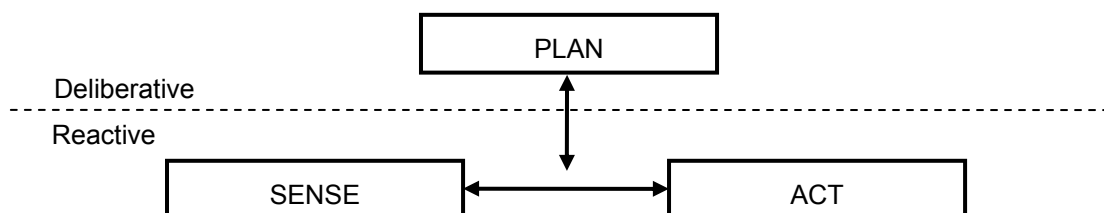


Figure 1-1 Hybrid architecture for robot control

Multi-agent systems focus on the development of computational principles and models for constructing, describing, implementing, and analyzing the patterns of interaction and coordination in agent societies [3].

The agent-oriented approaches help engineers to decompose and analysis the complete system. Saphira divided the deliberation activities of the robot among software agents [4]. Oliveira treats the intelligent robot as autonomous agents capable of flexible action in real world environments [5]. He proposed Autonomous Mobile Robot Control System (ARCoS), which uses a multi-agent system framework to give a flexible strategy for single agents' cooperation and enables a set of behaviors to have a certain degree of autonomy.

1.2.2. Software Frameworks for Robot Development

In parallel with the efforts of architecture and control methodologies, the software framework of intelligent autonomous system is another important issue. Since the rapid growth of quantity and complexity of robot components, the flexibility and reusability during the software development have become more and more important, in order to eliminate the duplicated efforts on programming. Some agent-based design exploits the traditional agent-development tools, such as April [6, 7]. BERRA [8] adopts several platform-portable software packages. CLARAty develops a framework for generic and reusable robotic components that can be adapted to a number of heterogeneous robot platforms [9]. OROCOS project (Open Robot Control Software) [10, 11] is a component-based, distributed, and configurable software framework. Its hard real-time core provides a generic control structure,

with plug-in facilities for customization, and gives optimal flexibility: distributed control over a network, portable over several (real-time) operating systems, configurable for high performance on special purpose hardware, scalable from limited to full functionality, etc.

Although much architecture have been proposed and examined successfully, still, there are some open questions needed to be discussed and some interesting areas needed to be studied. For instance, how the engineers distinguish different behaviors when analyzing the control system? How does the system reach a real-time response? How could team members work together efficiently? The detailed will be illustrated in the next section.



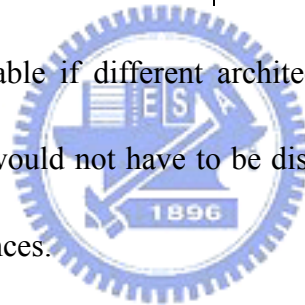
1.2.2.1. Classification of deliberation and reactive behaviors

Building a hybrid system requires cooperation between both deliberative and reactive layers. However, since boundary between deliberative and reactive behavior is still not well understood, it is difficult to classify all the behaviors into two layers without doubts. Various approaches have been used in different architectures. They can be loosely divided into three styles: managerial, state hierarchies, and model-oriented [1]. Managerial styles, such as AuRA [2], and SFX [1], are recognizable by their decomposition of responsibilities similar to business management. State hierarchies, such as 3T [1], use the knowledge about the robot's state to distinguish. In model-oriented styles, such as Saphira [4] and TCA [1], reactive model serves as virtual sensors. The features of these styles are listed in Table 1. It can be seen that each one has its own advantages and disadvantages, which can hardly be evaluated.

Table 1-1 A comparison of three different styles

	Deliberation	Reaction
Managerial styles	Global knowledge or world models	Behaviors which have some past / persistence of perception and external state
State Hierarchies	Requires past or future knowledge	Behaviors are purely reflexive and have only local, behavior specific; require only PRESENT
Model-oriented styles	Anything relating a behavior to a goal or objective	Behaviors are “small control units” operating in present, but may use global knowledge as if it were a sensor (virtual sensor)

Therefore, it would be admirable if different architectures can be integrated into a single system. Therefore, behaviors would not have to be distinguished in a regular way, and could vary under different circumstances.



1.2.2.2. Real-time responding vs. computing time

Real-time responding is an important factor for a robot control system, especially in dynamic environments. The hybrid architecture archives this requirement by distinguishing the deliberative and reactive behaviors and keeping the reactive one working at the higher priority and execution rate. However, it is not possible to execute every task in its highest rate since the CPU time is limited. Traditionally, the priority of the tasks decides which task should be launched first. However, it is still a style of art to solve the dead lock problem.

1.2.2.3. Team work in robot development

In most cases, a complex system such as a home robot system is a cooperative work of a group of researchers rather than an individual research results. Therefore, it is desired to have a good way to merge these works from different people. One favorite method is to use object oriented programming (OOP) technique. Researchers define the objects used in the project and then develop them simultaneously. This approach provides good reusability and portability, since these objects can be separately developed and modified. However, if the specification of the object is changed, there are some new objects added, or some unused objects are waived, the main program of the control system has to be altered to fit these modifications. Researchers therefore have to take extra effort maintaining different versions of main programs for different purposes, for instance, testing different modules, debugging, monitoring, etc.

Another good approach is to use agent oriented programming (AOP) technique. Researchers develop their own agents and execute them separately, without a launcher program. These agents can therefore forms a system by communicate with each other, and the system is therefore easier to maintain. However, in AOP agents need some specific languages to design, which may cause a strict learning curve for the researchers, and takes more extra time when porting the old programs.

1.3. Problem Statements

This study aims to construct a flexible system for robot control that solves the previous problems and remains good features of other systems. The main purpose is to develop a software architecture to integrate multiple research results into one program to control a home robot. Therefore, the objective of this study will include: (i) a good control strategy to negotiate between programs; (ii) a software framework including useful APIs and modules to simplify the process of porting/combining the programs and allocates/shares the resource properly. Moreover, through the implementation of this system, several advanced control functions of an experimental home robot will be demonstrated.

1.4. Organization of the Thesis

Chapter 2 describes the proposed system. It presents an overview of the design and its details. Chapter 3 describes the software implementation of the system. The OS, the API, and realization issues are presented. Chapter 4 illustrates experimental studies of the proposed control system. Experimental results are presented for performance evaluation. Finally, Chapter 5 is the conclusion and future works of this study

2. System Design

2.1. System Overview

The concept of the proposed system is from the neural system of human beings and the managerial architecture of an organization. Human sensors work continuously, whether the brain process its data or not. In other words, the information flows into the brains, muscles, and organs automatically, rather than requested.

On the other hand, managerial architecture can be considered as a group of people manipulating a robot. They work individually or in a small team, and are in charge of different jobs: sensing, interpreting, decision-making, and reacting, etc. They work autonomously and asynchronously. Moreover, they can exchange information to cooperate on a complex task and increase their performance.

In a designer's point of view, the programming process of a complex system can be treated as a circuit board assembled from ICs and other electronic components. In a similar way, the robot designer can build and modify the robotic system by connecting reusable parts. This flexible architecture helps the robot designer to decompose a complex system into many specific, independent parts, and develop them separately. The details will be discussed in the following sections.

2.2. Properties of Robot Control System

2.2.1. Flexibility

The proposed system is composed with loosely coupled agents, which means that agents can be added or purged without breaking or blocking the whole system. Therefore, the control method and performance can be customized easily by adding or purging agents, or changing the connections among agents. Furthermore, agents can be tested individually, or in small groups first, and then the whole system with those proved agents.

2.2.2. Real-time Response and Adjustable Execution

Period



The system is designed to keep a quick and predictable responding time even if numerous programs are running simultaneously. In such a system, it is possible to change the execution period of each task dynamically, as shown in Fig. 2-1. In this case the system can conserve the computing power for the most urgent or suitable tasks. The robot can therefore concentrate on the most important issue at the moment.

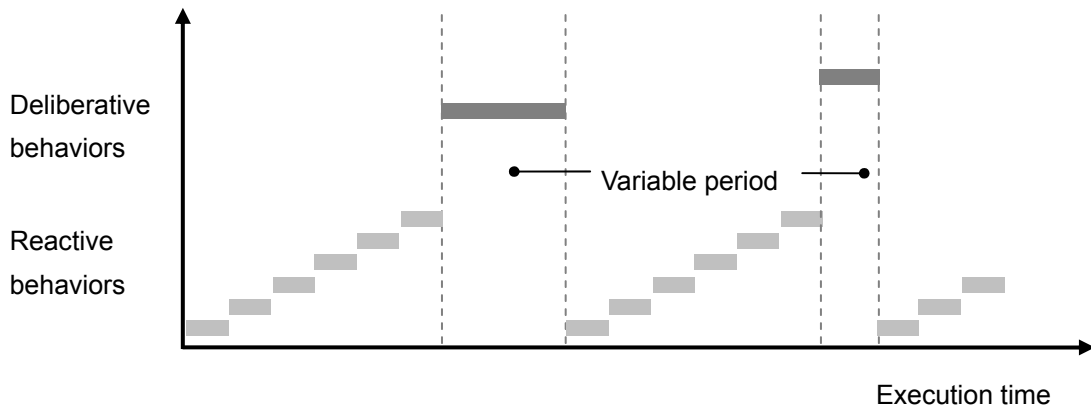


Figure 2-1 A simplified example for period adjusting

2.2.3. Connectivity

Good connectivity is an important feature for any architecture since it guarantees that all the entities in the architecture can communicate with each other. In this control system, the communication network or dataflow can be modified both off line and on line, in order to support the flexibility of the system. This feature is superior to a hierarchical layered architecture. Moreover, one can build the connection with agents in other system through internet, which provides the ability of building a distributed system.

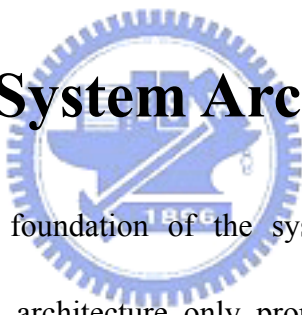
2.2.4. Loosely Classified

Each agent can be a simple function, assembled sub system with other agents, or a small hybrid system of its own. Each system can use different classifications of deliberation or reactive behaviors. In other words, user can port various kinds of algorithms or systems to our system, without doing lots of decoupling works.

2.2.5. Generality

Since the agents connected with message flows, the coding style of agents is much unlimited. Users can design their own agent with pre-defined templates, thus they can complete an agent easily by *filling in* the desired algorithm and do not have to take care of any low-level program features. One can also re-implement their own agent with new templates or styles, as long as they keep the same protocol of the message flow and execution time management. Therefore, porting a program to this system is simple.

2.3. Proposed System Architecture



The architecture is the foundation of the system since it usually determines its properties. In this system, the architecture only proposes an organization method for its construction. It defines the properties and characteristics of the necessary components, and the relationship among them. Fig. 2-2 presents the proposed architecture of the robot control system.

The design of architecture is based on the basic definition of agents, including the abilities of sensing, thinking, and acting. The core components in the architecture provide these abilities, and maintain the performance of the system. They will be introduced in depth in the next section.

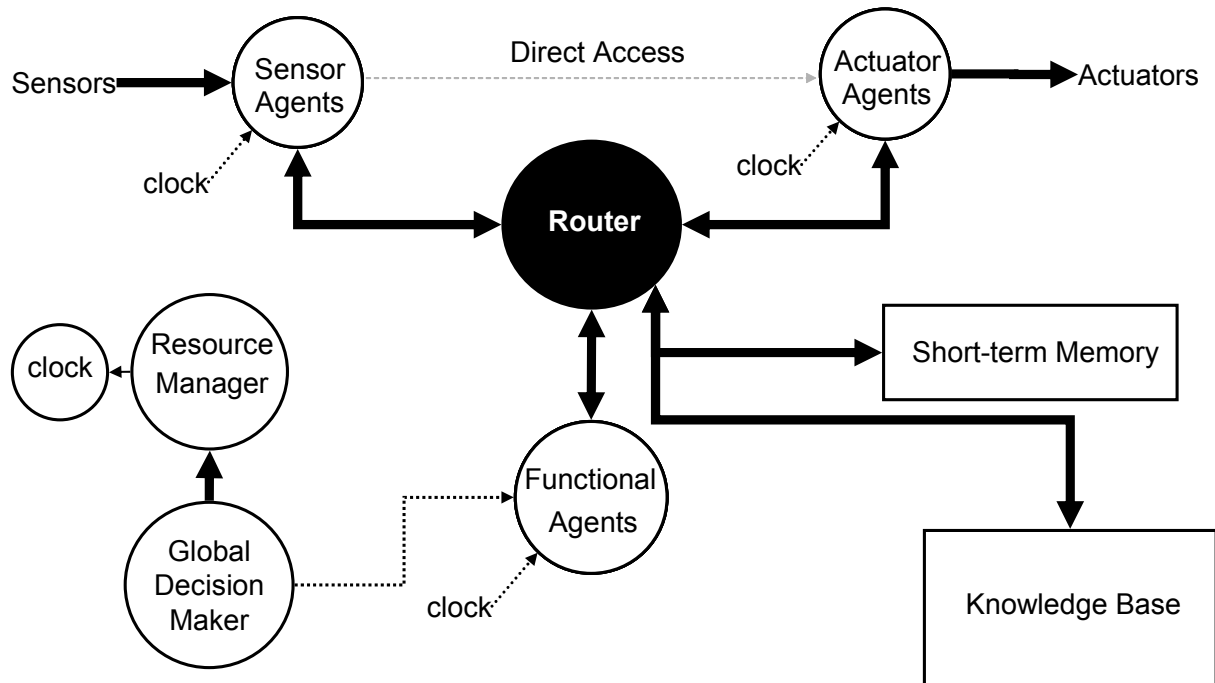


Figure 2-2 Global view of the control system architecture

2.3.1. System Components

- *Sensor Agents* are responsible for processing or interpreting the sensor input data into useful information. For example, image processing, noise filtering, etc. Furthermore, it is able to access the actuator agent directly for some special cases. In our design, sensor agents always get the highest priority.

Therefore, the system is capable of monitoring any changes and phenomena happened in the real world. The concept of the sensor agent is depicted in Fig. 2-3. The receiver reads the data from the hardware interface, synchronized with the hardware or asynchronous using an external clock. The data then flows into the filters or other processors in output layers, processed and output to other components. Notice that in the output layer, one can choose the

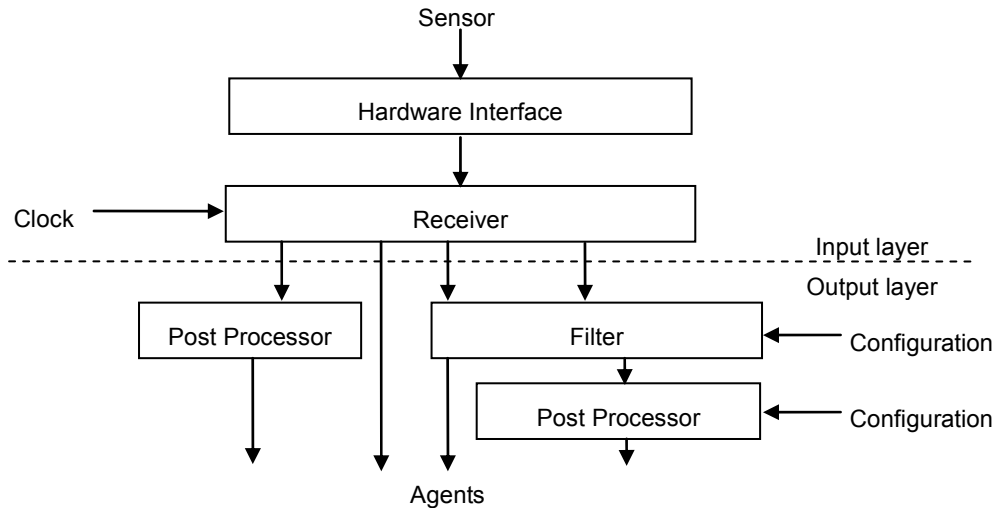


Figure 2-3 Sensor agent pattern

most proper output of the processors for different proposes. The parameters of the processors can be tuned by sending configuration commands. *Actuator Agents* are responsible for fusing input commands and controlling the hardware, for example, a wheel motor. Both linguistic and numeric commands are acceptable. The agents also filter out impractical inputs to fit the hardware limitation or for safety. The concept of the actuator agent is shown in Fig. 2-4. The filter receives and blocks out the improper command from other agents. If the command were in numerical form, the command would send to the controller; if the command were linguistic commands, the fuzzy behavior system will translate and fuse the commands for the controller.

The sensor and actuator agents solve the hardware-sharing problem when integrating various functions. Not only the raw sensory data, but also the processed data can now be

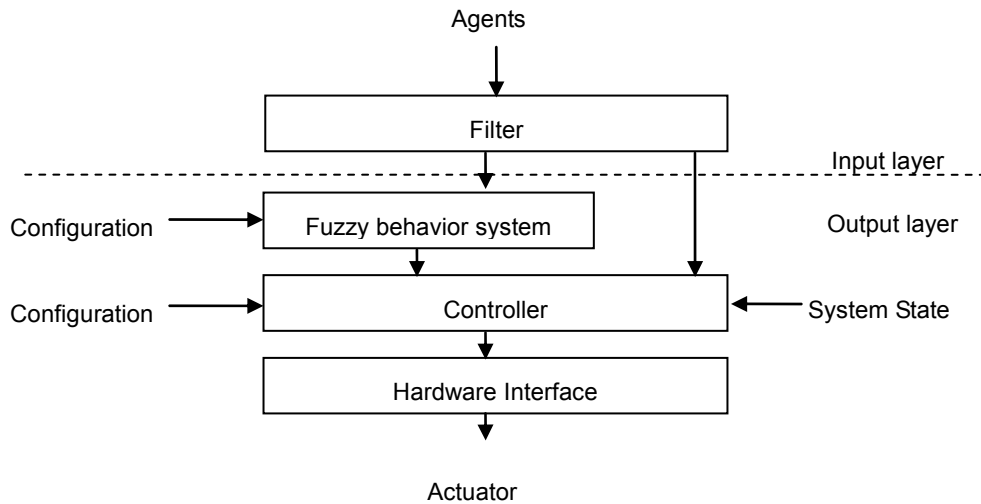


Figure 2-4 Actuator agent pattern

freely accessed. For example, the image sensor agent may provide raw image, edge image, motion vector of the image, etc, and thus the other agents do not need to waste CPU time for processing

- *Functional Agents* are in charge of all the functions and skills equipped on the robot. For example, face tracking, path planning, etc. To make the system work better, users should implement their function with anytime algorithm or multiple methods. Therefore, the system can choose a preferred result between high updating rate and accurate answer.
- *Router* records all the input and output paths of the agents connected to the system. It manages the message network of agents. The user therefore does not have to take care of the communication issues. Furthermore, the router makes the agents possible to recognize the composition of the system. Therefore, a newly added agent would be able to know what agent it could work with in the system.

- *Short-term Memory* records recent information used by the agents. Agents can share data and state through it.
- *Knowledge Base* stores the prepared information for deliberation, such as patterns, schedules, symbolic data, etc.
- *Resource Manager* manages the memory and computing power used, and adjusts the configuration of the clock and the agents.
- *Global Decision Maker* determines the current state and goal of the whole robot.

2.3.2. Inter Agent Communication

One of the main factor of this system is to keep the message flows smoothly and in real-time. Therefore, the procedure that is responsible for processing message / data flows of the system always works at the highest priority (as high as sensor agent). Under this design, every message will be delivered to the target agent at the very first time.

In order to fulfill the needs of various kinds of functions of the agent, the system provides different method for communication:

- **Direct Access:** For small messages such as commands or control signal, the router directly sends the messages to the agents. This can be used to send commands or linguistic input for algorithms using fuzzy set.
- **Shared Memory:** If the size of the data is huge (such as images) or is used by too many agents, we can only send a link and a signal to agents, and they can gather the data from

the shared memory, thus save a lot of time and memory. For each data in the shared memory, we use a circular buffer, logging the data with the storing time. Thus, it can be used for prediction, data mining, etc.

The system starts from the clock and the router, which works like the heartbeat and nerve system of a human, provide the execution pulse and message exchanges. The agents then joined in the architecture, provides the practical abilities of the robot. They can communicate and work together via the router. The memory and knowledge base will be read or updated by these agents. The functional agents may lead to different results, and will be determined by the global decision maker. When the computing power is running out, the resource manager will try to adjust the speed of the agents depending on its necessity for current work. The manager refers the data from the knowledge base.



2.3.3. Examples

2.3.3.1. Multi-agent cooperation

Fig. 2-5 shows an example of how the whole system works. This is a conceptual system of a home robot, which can navigate through the house, avoiding obstacles, and tracking an object or a moving person.

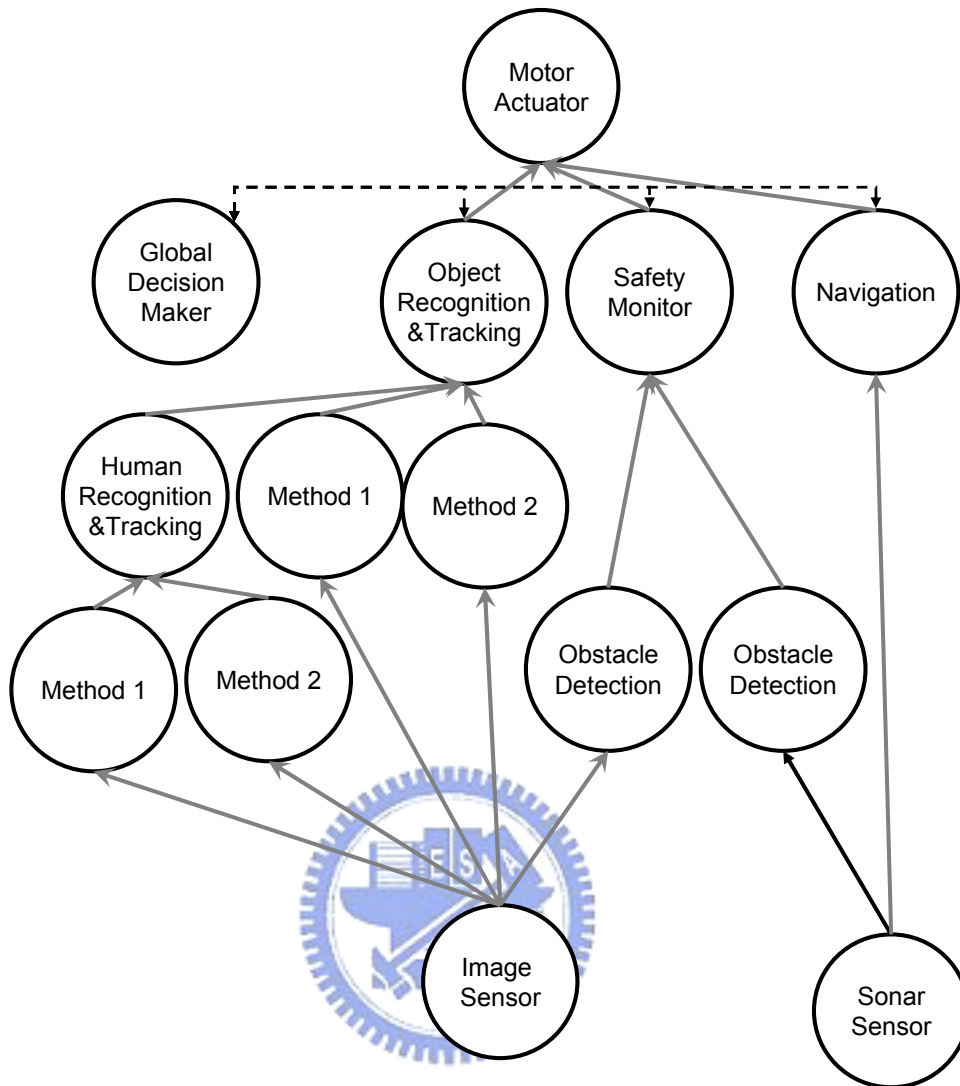


Figure 2-5 An example of the control system

The first feature is the global decision maker. Consider, for instance, the three different agents that Motor Actuator is connected to: Object Recognition and Tracking, Safety Monitor, and Navigation. Each of them is responsible for a specific function. It is obvious that their desired outputs may conflict in many cases. The global decision maker will decide the importance of the agent, and alter the weight of the output from each agent.

The second feature is the robustness brought by multi-agent approach. In the Safety Monitor case, two agents using different sensor sources (image and sonar respectively) are

then utilized concurrently. This is termed sensor fusing, which brings results that are more reliable. Another example is the object recognition. There are two different methods for general object recognition, and an extra one for human face recognition. Two different methods are employed for human face recognition. The successful rate is therefore increased. However, the drawback is the CPU time increases to a high degree. In our system, the resource manager will observe the CPU usage, and terminate some extra process if the real time concern cannot be fulfilled. On the other hand, the global decision maker will slow down or stop unused agent to preserve more CPU time, for instance, when the robot is focusing on tracking.

Finally, we observe that the actual behaviors, whether reactive or deliberative, are separated in different agents. Thus, the responding time is actually determined by the performance of the agent connected to the actuator. Therefore, these agents are suggested to be implemented with fast algorithm, or a hybrid behavior model.

2.3.3.2. Multi-robot control

The architecture can also be extended for a multi-robot control system. The communication among individual systems is then a key factor. For mobile robots, the communication is often realized by wireless communication such as wireless LAN or Bluetooth. Since the hardware varies and the transmission rate is non-deterministic, it is not a good idea to integrate the remote message delivering function into the real-time router. Instead, the transmitter and receiver (or the server and client in internet connection) are

treated as sensors and actuators. Different agents can be implemented to deal with different protocol. Figure 2-6 shows an example of a 2-robot cooperation system. Each robot contains pairs of senders and receivers, used to synchronize a global map and issue commands to each other. Therefore, no server is needed in this system.

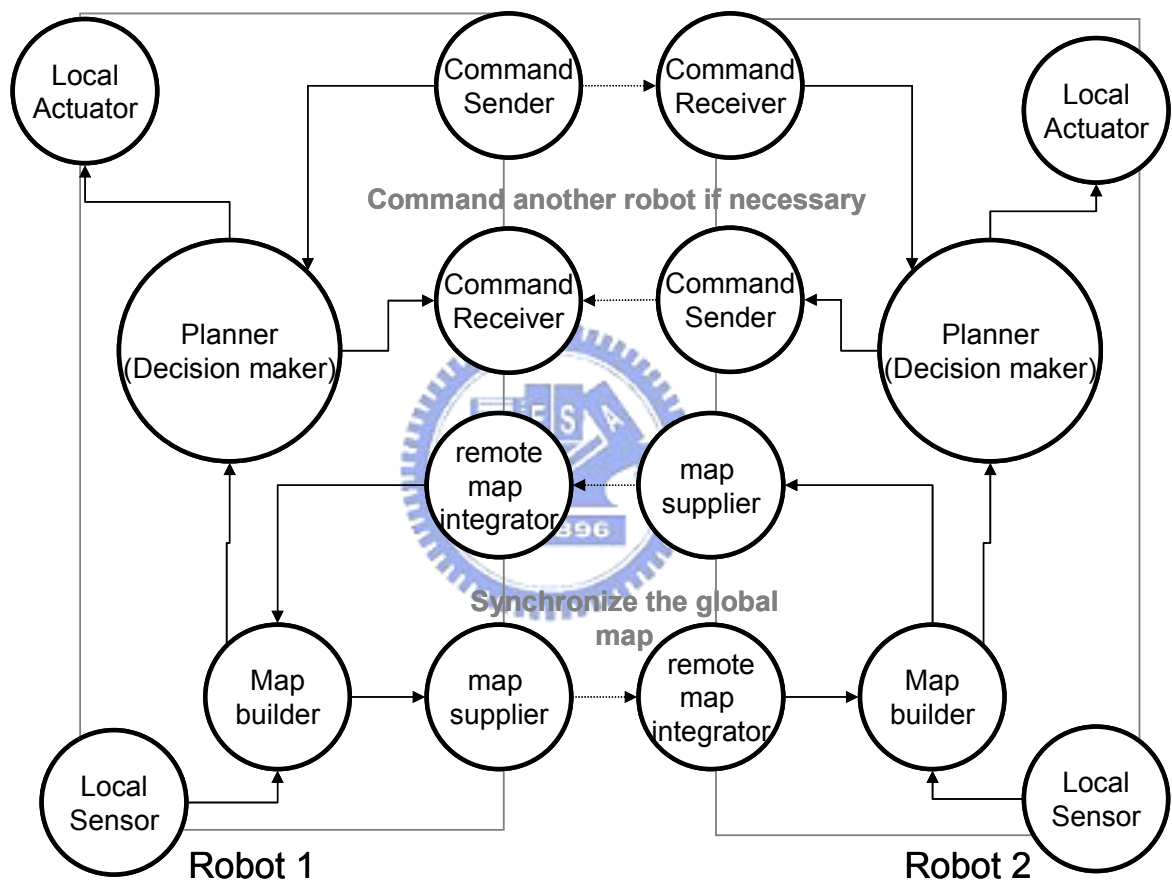


Figure 2-6 An example of a 2-robot cooperation system

2.4. Summary

In this chapter, a robot control system is proposed. This system features a comprehensive way to divide a complex control system into individual pieces, and reunion them efficiently by using multi-agent approach under a flexible architecture. The development effort is reduced since the agent can be reused and the system is easy to modify. The resource-sharing problem is solved by introducing the sensor and actuator agents, which manage the hardware. Functional agents can cooperate with each other in order to obtain a better result. The router simplifies the data exchanging issues, and provides the flexible composition ability of the system. The message-first policy grants the data transmitted in real-time, and results in the fast and real-time reaction of the system.



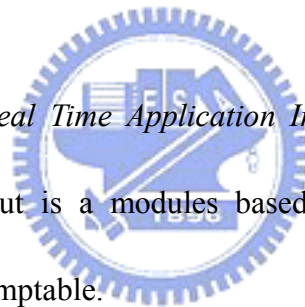
3. Software Realization

C and C++ are chosen for programming the agents, rather than inventing a new language, in order to provide the generality. The first version of system realization is written under Linux and RTAI, since RTAI provides many good features to reduce the effort of realization.

3.1. RTAI

3.1.1. Overview

RTAI [12] stands for *Real Time Application Interface*. Strictly speaking, it is not a real-time operating system, but is a modules based on the Linux kernel, providing the capability to make it fully preemptable.



Linux is a standard time-sharing operating system that provides good average performance and highly sophisticated services. Like other OS, it offers applications for at least the following services:

- Hardware management layer dealing with event polling or processor/peripheral interrupts
- Scheduler classes dealing with process activation, priorities, and time slice
- Communication among applications

Linux suffers from a lack of real-time support, however. To obtain correct timing behavior, it is necessary to make some changes in the kernel sources, i.e. in the interrupt handling and scheduling policies. In this way, a real time platform with low latency and high predictability requirements can be established within full non-real-time Linux environment (access to TCP/IP, graphical display and windowing systems, file and data base systems, etc.).

RTAI offers the same services as Linux kernel core, adding the features of an industrial real time operating system, as shown in Fig. 3-1. It consists of an interrupt dispatcher: RTAI mainly traps the peripherals interrupts and if necessary re-routes them to Linux. It is not an intrusive modification of the kernel; it uses the concept of hardware abstraction layer (HAL) to get information from Linux and to trap some fundamental functions. This HAL provides few dependencies to Linux Kernel. This leads to a simple adaptation in the Linux kernel, an easy RTAI port from version to version of Linux and an easier use of other operating systems instead of RTAI. RTAI considers Linux as a background task running when no real time activity occurs.

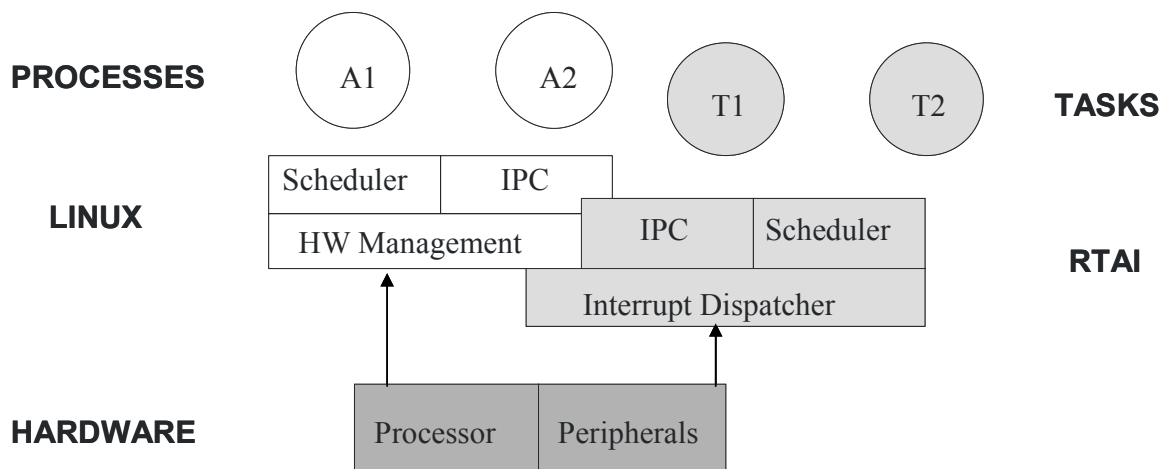


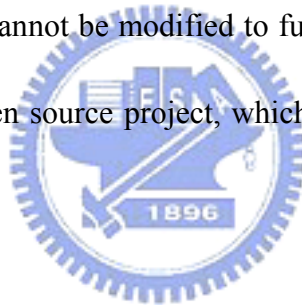
Figure 3-1 RTAI architecture

3.1.2. Compare with Other RTOS

There are other real time operating systems (RTOS), which can be used to implement the system, but RTAI seems to be the best choice for the first version of the realization. The comparative result is discussed in the following section:

3.1.2.1. Microsoft Windows®

While there is RTlinux /RTAI on Linux, Windows can also reach hard real-time by installing RTX (VenturCom ©) for Windows. Although the approaches are similar, RTX is a commercial product and thus cannot be modified to fulfill the requirement of the system. On the other hand, RTAI is an open source project, which is still maintained and updated by its original team.



3.1.2.2. Commercial hard real-time operating systems

Commercial hard real-time operating systems such as QNX are small and convenient to be used in embedded systems, but they usually support only limited numbers of devices and are lack of resources compare with Linux.

Therefore, RTAI is most efficient choice, which is free, resourceful, and up to date.

3.1.3. RTAI Features

3.1.3.1. Scheduling

The scheduling units of RTAI are called tasks. There is always at least one task, namely the Linux kernel, running as a low-priority task. When real time tasks are added, the scheduler gives those priorities higher than the Linux kernel. The scheduler provides operators such as *suspend*, *resume*, *yield*, *make_periodic*, *wait_until*, which are used in various real-time operating systems.

3.1.3.2. Inter Process Communication (IPC)

RTAI provides a variety of mechanisms for inter-process communication. Although the UNIX systems provide similar IPC mechanisms to the user-space processes, RTAI needs to provide its own implementation for them in order to make it possible for the real-time tasks to use the IPC mechanisms, as they cannot use the standard Linux system calls. Different IPC mechanisms are included in RTAI in the form of kernel modules, which can be loaded in addition to the basic RTAI and scheduler modules only when they are needed by the tasks. An additional advantage of using modules is that the IPC services can be easily customized and expanded. The mechanism supported by RTAI includes FIFO, shared memory, semaphores, and mailboxes.

3.1.3.3. Memory management

The present versions of RTAI include a memory management module, which allows dynamic allocation of memory in the real-time tasks by using an interface similar to the standard C library. RTAI pre-allocates a large piece of memory (with configurable size and number) before real-time execution. When a real-time task calls *rt_malloc ()*, the requested memory is given from a pre-allocated chunk. When the amount of free memory in the chunk is less than a threshold value, a new piece of memory is reserved for future allocations. Similarly, when *rt_free ()* is called, the released memory is left on the allocated chunk to wait for future reservations. When the amount of free memory is more than a high water mark value, the memory chunk is released.

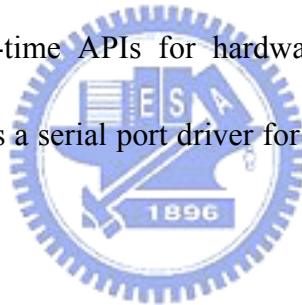
3.1.3.4. Soft-hard real time in user space: LXRT

LXRT is a group of APIs for RTAI, which makes it possible to develop real-time applications in user-space, without creating kernel modules. This feature is useful, since in the past, real-time execution is only allowed in kernel space, which has some drawbacks. First, it is inconvenient to debug a kernel module. Although there are certain debugging tools existing, they are more difficult to use and less matured than the one used in user space. Furthermore, many useful system calls, such as file manipulation and socket communication are not supported in kernel space. Therefore, users have to take a lot more effort implement these system calls in order to convert a user space program into a real-time kernel modules.

With LXRT, converting an application from user-space process to a real-time task becomes much simple, because LXRT provides a symmetrical API for inter-process communication and other RTAI services. This means that both kernel-space tasks and user-space processes can use the same API. The same LXRT API can also be used when two user-space processes or two real-time tasks communicate with each other. This implies that the various timers and messaging systems that LXRT provides could be used by a user-space application even if it would not actually have any real-time requirements.

3.1.3.5. Real-time API for hardware

RTAI also provide real-time APIs for hardware in order to handle data from the interface. For example, *spdrv* is a serial port driver for setting up, sending, and receiving data via RS-232 serial port.



3.2. Implementation

Referred to Fig 2-2, the system components determine the performance of the system.

With the aid of RTAI APIs, the components can be realized easily.

3.2.1. Clock

The system clock is hidden in the system scheduler of RTAI, as mentioned in the last section. The function *start_rt_timer ()* and *stop_rt_timer ()* can switch the system timer (clock) on and off, with desired period. Furthermore, one can use *rt_get_time ()* to get current system

time.

3.2.2. Short-term Memory

The short-term memory is a large piece of memory stores the recent variables, states of the agents, and can be accessed by every agent. Circular buffer is therefore a good data structure under this requirement. First, it can be allocated once and remain useful, which reduce the overhead of memory allocating time; Secondly, it allows the agents to read and write at the same time and therefore prevent the synchronization problem, as shown in Fig. 3-2.

In practice, the shared memory is allocated using the RTAI API *rt_malloc ()*. Every chunk of circular buffer can be identified with an unsigned long integer, or a six characters name.

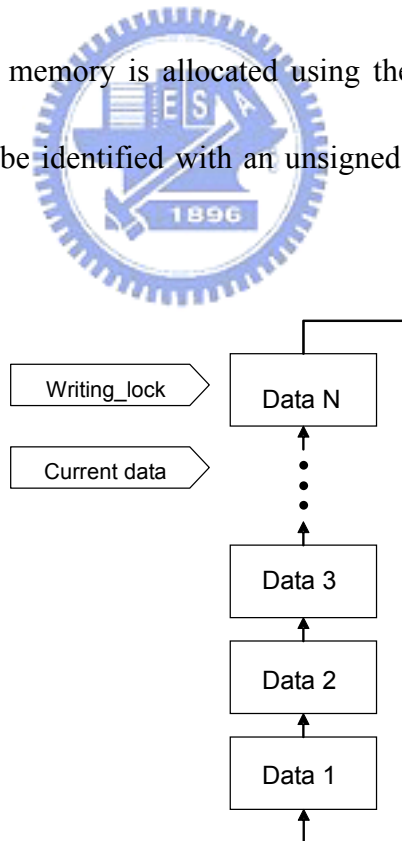
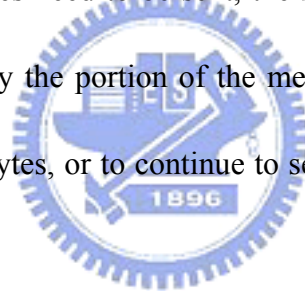


Figure 3-2 The concept of the circular buffer

3.2.3. Message Delivery and Router

The mailbox service is chosen in this system for message delivery. The mailbox service allows messages between processes to be automatically stored and retrieved as needed in a priority queue. In RTAI, mailbox service is very flexible.

- It can be explicitly setup to accept messages of customized sizes.
- Multiple receivers and senders can be connected to the same mailbox where the order in which messages are taken depends on the priorities of the receivers.
- When large messages need to be sent, the service provides functions to allow the process to send only the portion of the message that can be stored, returning the number of unsent bytes, or to continue to send the message until all of it has been accepted.



Mailboxes can be slightly less efficient than FIFO since it has twice more numbers of *memcpy ()* operations. However, the advantage is relatively high that it is worth such a minor penalty. Furthermore, the effect is unlikely to be noticeable for relatively short messages.

The router is a real-time task with two mailboxes, one for registry, and the other for message dispatching. If an agent wants to join the system, it has to register its own mailboxes (which means that an agent can own more than one mailbox), and subscribes the desired mailboxes of the agents from the router. It can also assign target mailboxes initiatively. The router therefore records the connection lists of the mailboxes. An agent sends its output

message to the router, and the router will look through the list and broadcast the message to every subscriber. This process produces some minor overhead, but introduces a dynamic linking between agents, instead of changing the code and recompiling it.

The connection list is maintained with a two-layered linking list, as shown in Fig 3-3, which takes advantages on adding and checking through mailboxes fast. However, removing a MBX from the list causes a chain effect: Every sub-list has to be checked and removed, which is $O(N^2)$. Therefore, the deletion of the MBX only removes it from the top level. When a sending task is processed, every MBX will be checked and the deleted one will be removed.

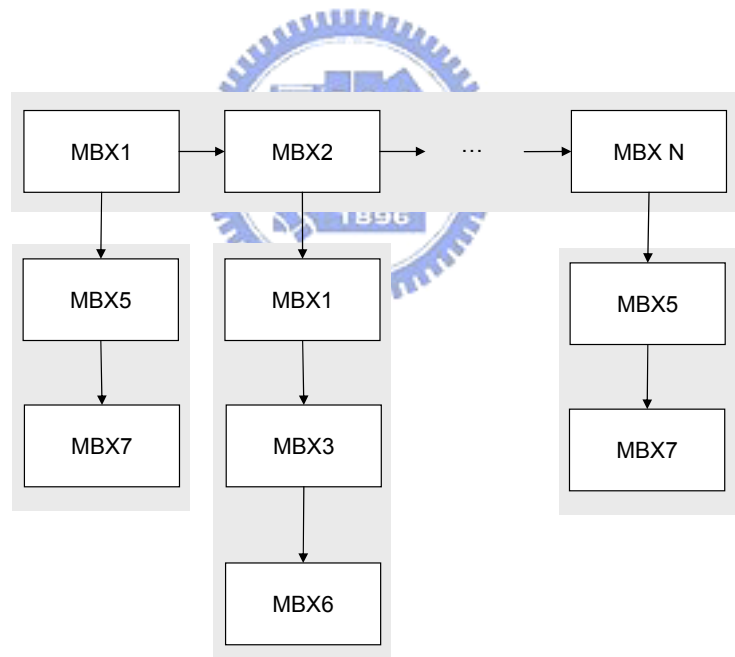


Figure 3-3 The connection list of the router

3.2.4. Agent

The agent defined in our system can be just part of algorithm or functions. Since this is a multi-process/ multi-threaded system, it suffers from the synchronization and deadlock problems, etc, and have to be carefully handled. In this implementation, the messages and main tasks used by the agent are separated. This approach guarantees that the functions will never be blocked when waiting for messages. Fig. 3-4 shows how this approach works.

The message checker is a high priority task that listens to the mailbox and responses immediately on receiving messages. The message checker will interpret the message, modify the parameters or states of the agent, and keep them in the memory. The interpretation of the messages is left for the user in order to be compatible with their design. The agent task can be classified into two different types:

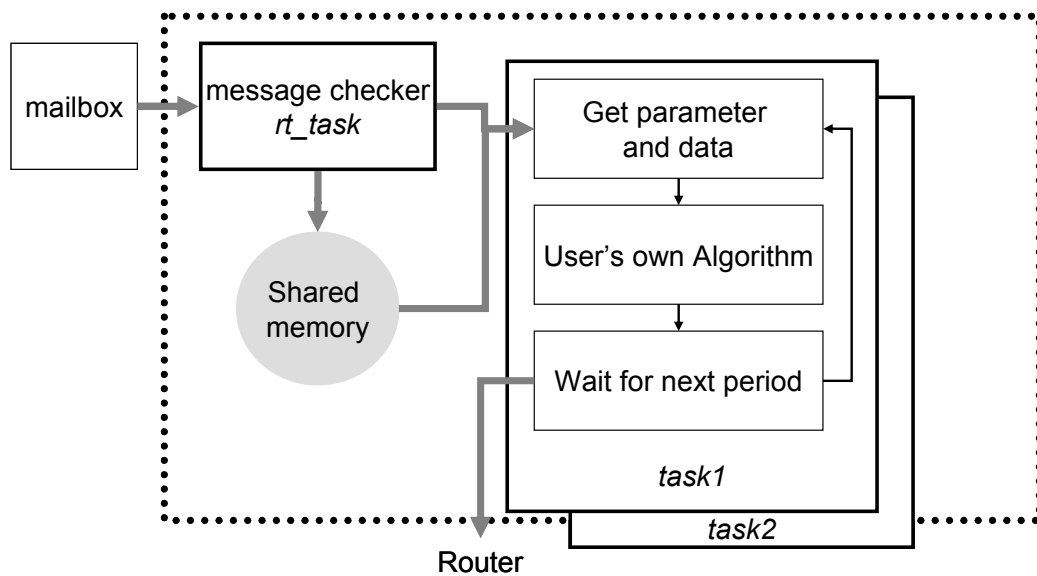


Figure 3-4 The basic design pattern of agents

- The event-driven (aperiodic) type is triggered right after the messages arrive. Therefore, its execution rate is determined by its message source. Once it is started, the function will lock itself to prevent reentrant
- The periodic type is triggered periodically with different priorities and rates. The system can adjust its rate to achieve better performance.

In the case of sensor agent, it does the same trick. The only difference is that there is one more task, which listens to the data transferred from the sensor devices.

As mentioned before, the RTAI task can be executed in kernel space or user space. Therefore, an agent may be implemented in two different forms. In the kernel space, the agent has to be written in the form of modules. The real-time tasks are initialed in *init_module ()*, which is the starting point when a module is loaded. The code of the real-time task is implemented as function and then linked to the task when it is initialized using *rt_task_init ()*.

In the user space, it takes more efforts to do the same job. First, the code of the real-time tasks is not linked to a function. The user has to use *pthread* to execute the functions with multi- threads, and initial the real-time task inside each thread. Another big difference is that the user space program always blocks the console and stops the batch loading process of agents. Therefore, the user space program has to be daemonized in order to execute as a background process as if the module does. In this implementation, the agent template is done for both kernel and user space, so the end user can choose a preferred one.

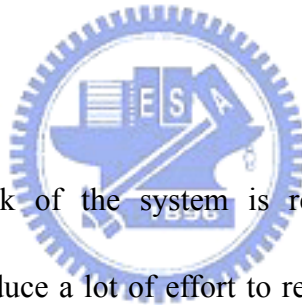
3.2.5. Decision Maker and Knowledge base

The decision maker works as a table look-up agent, which assigns different weight and execution rate under different circumstances reported by the agents.

3.2.6. Resource Manager

The resource manger works as a virtual sensor agent, gathering the CPU usage info and memory usage status, and reporting it to the decision maker. The system information can be gathered from the files under /proc directory.

3.3. Summary



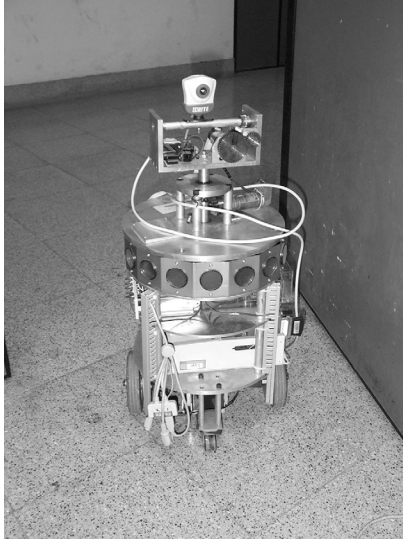
The software framework of the system is realized on RTAI Linux. Its favorite performance and properties reduce a lot of effort to reach the desired functions and enhance the system performance. The robot designer can easily port their work to the system by attaching their own C/C++ program into the agent template. User can also build their own templates using the system APIs.

4. Experimental Results

4.1. The experimental mobile robots

The proposed system is tested on our laboratory robot platforms, H1 and H2, which are self-constructed mobile robots in our lab. Fig. 6 shows recent pictures of H1 and H2. These two robots are similar and composed by the following components:

- An USB web Cam on a two DOF pan-tilt head for image capturing
- A 3DOF arm for grasping. (Only on H2)
- Twelve ultra-sonic sensors with RS-232 interface for environment sensing. (Only on H1)
- An IR receiver receives the digital code from the IR transmitter.
- A Celeron 300MHz PC for system control, using Linux (kernel 2.4.18) and RTAI (24.1.13).
- Two independent drive wheels and two casters for mobility
- DSP boards for motors controls. PC can access the boards to control the motors via serial ports.
- 12V to 5V DC-DC Converter
- Two 12V batteries and two 6V batteries, for power supply



(a)



(b)

Figure 4-1 (a) H1 robot platform (b) H2 robot platform

4.2. Experiment of single robot exploration

4.2.1. Goal

The goal of this experiment is to program a robot that explores autonomously in an unknown environment, searches for, and tracks a target object. All the functions are designed separately and integrated with the proposed architecture. The efficiency of programming the robot and the performance of the system will be presented in this experiment.

4.2.2. Implementation

The implementation for this experiment is carried out by going through the processes described in Chapter 3. Fig. 4-2 shows the organization of the agents used in this experiment.

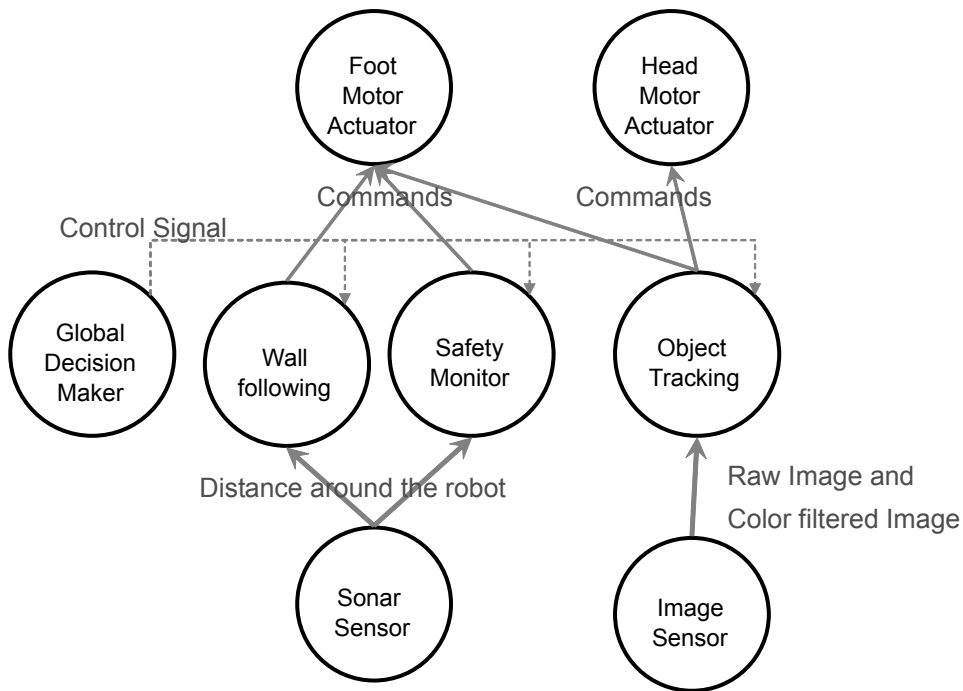


Figure 4-2 The system organization of a robot exploration experiment

4.2.2.1. Sensor agents

- *The sonar sensor agent* gathers the data from the ultrasonic sensor via serial port. The sensor modules offer distance from 12 different directions, as shown in Fig 4-3, with a 4 Hz sampling rate [13]. The agent measures the probability of risks at each direction by estimating the difference of the distance between two sample times.
- *The image sensor agent* captures a 320x240, RGB image from the Web Cam at the speed of 15 frames per second. The input image will be converted to HSI color space for further usages.

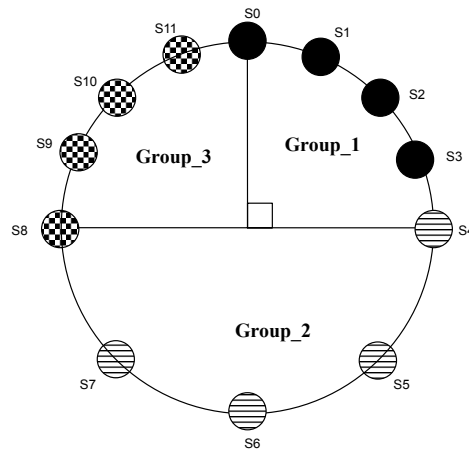


Figure 4-3 The sonar sensors module

4.2.2.2. Actuator agents

- *The head and foot motor agents* interpret the linguistic commands into the controller desired, numerical format, and send it to DSP controller via the serial port. For example, *turn right a little bit* or *increase speed*. The state of the motors such as speed and position is stored inside the agent and will be updated from the encoder data provided by DSP.

4.2.2.3. Functional agents

- *The safety monitor agent* will decrease the robot speed if it is approaching an obstacle detected by the sonar sensor, and try to make a left turn or a right turn to avoid the obstacle. The robot will go forward to the direction, which is detected as a safer, wider space. Furthermore, if something comes too close to an object, the agent will force the robot to stop moving (but not turning) immediately.
- *The wall following agent* keeps increasing the speed of the robot and try to

maintain the distance between the robot and the wall.

- *The image-tracking agent* tries to track the red object using its head and moves close to it. This is done in the following steps:
 1. Split the red part of the HSI image (H=350 to 10) to a binary image.
 2. Find the maximum area in the binary image, which is indicated as the target object.
 3. Find the centre of the object

The control policy is quite straightforward: move forward to the target object. If the centre of the object is far from the center of the image, the robot moves its head and the foot to locate the object to the center; if the area is small, the robot will try to come closer.



4.2.2.4. Decision making

For the head motor, there is only one agent connected, thus there is no need to make any decision. For the foot motor, the behavior is determined by two parameters: tendencies for danger and handling target tracking. The decision-making agent receives the environmental information from the agents, estimate the parameters, and activate relating agents. If the webcam observes a target to track, the tracking agent is activated. If the state is dangerous, such as something approaching nearby, or the environment becoming narrow, the object avoidance agent will be activated. Since the desire and danger compete with each other, the

tracking process will never collide with the target. The wall following agent works when the other two are suspend.

4.2.3. Experimental Result

Fig. 4-4 shows the recorded trajectory of how the robot managed to navigate safely in a crowded room. It passed a door to the corridor and tracked an object. The gray area in the figure only roughly represents obstacles, since the actual environment is quite complicated. See the pictures in the right side of Fig. 4-4. The agent-based control system successfully guided the robot through the door, without bumping into anything in a narrow space. After the robot went out the room, it was attracted by a red object hold by a person. The robot moved forward to it. However, the image-tracking agent stops the robot when it came too close to the person with the red object.



Fig. 4-5 illustrates the reactions of the robot at each moment during the experiment. The states of the robot change every 0.25 sec, which is the same as the update rate of the sonar sensor. Fig. 4-5 (a) shows that the robot suffered from sudden stops occasionally. It was caused by the strict safety control and erroneous sonar readings. At 160~180 sec, the robot increased its speed and moved straight forward since it faced a wide corridor and there was no danger. At 180 sec a person with the red object appeared, which caused the robot to stop. The person then stepped backward to lead the robot make a U-turn. As shown in Fig. 4-5 (b) and (c), the robot followed his steps.

This experiment shows the proposed control system successfully combines

heterogeneous agents in to a single system. The router works as expected, broadcasts messages between agents, makes the agents possible to receive the sonar information simultaneously. Agents can stop execution and restart during the experiment, without hanging up the whole system. The decision maker indicates the agent to execute at the proper time and thus merges three single-function agents into the multi-purpose robot. Therefore, all the components in the architecture works properly, except the resource manger since the CPU time is still sufficient (66%) in this experiment.

In every experiment made, the trajectories may be quite different. Sometimes the robot could not find the door and turned back to the starting point. This is because of the error distance measurement from the ultrasonic sensors.



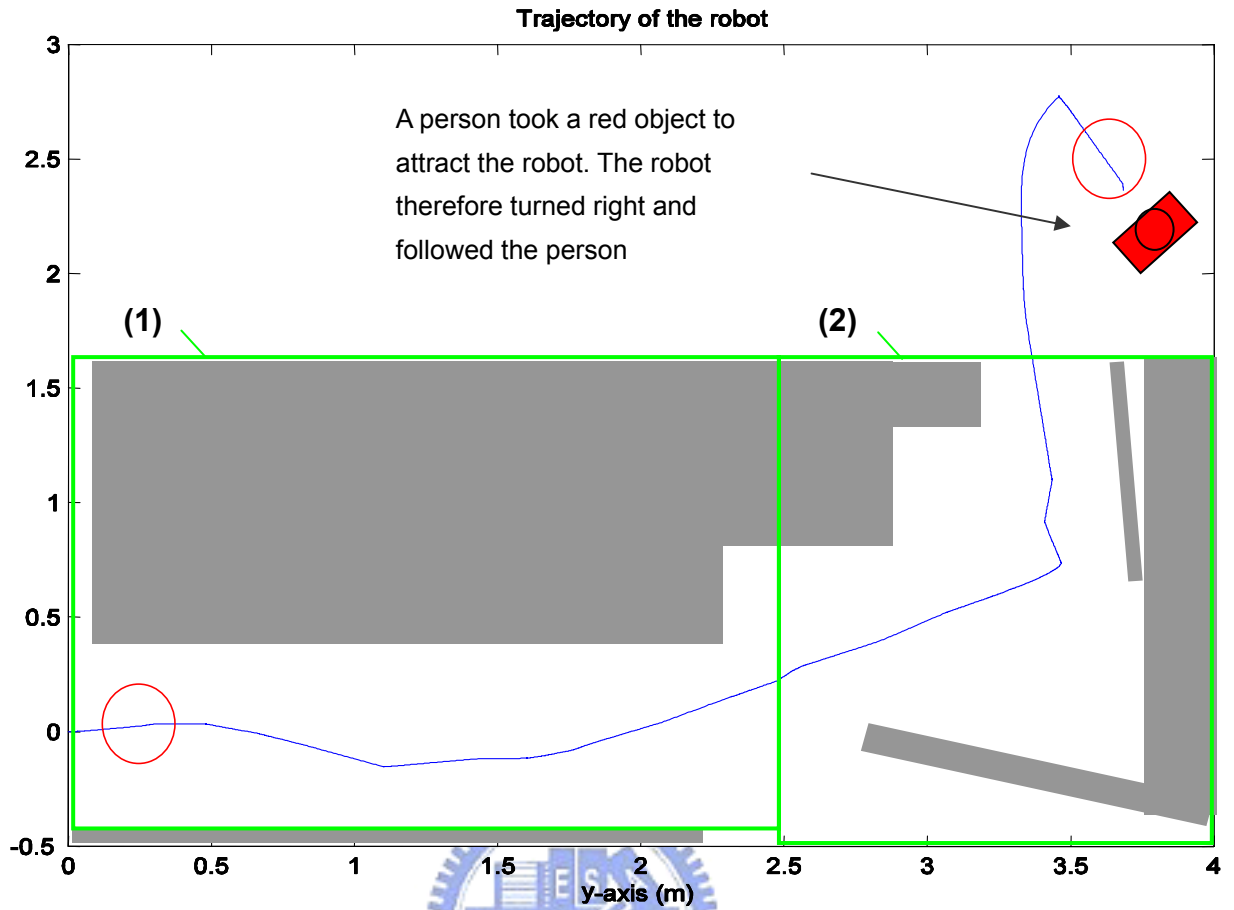


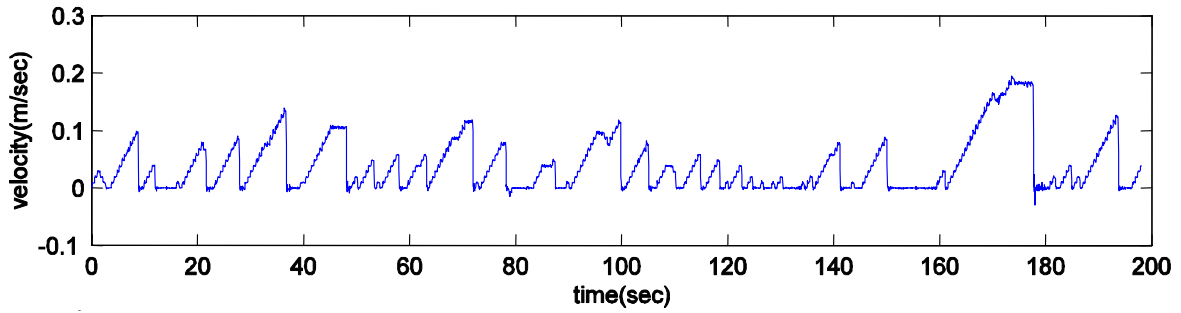
Figure 4-4 Recorded trajectory of the robot



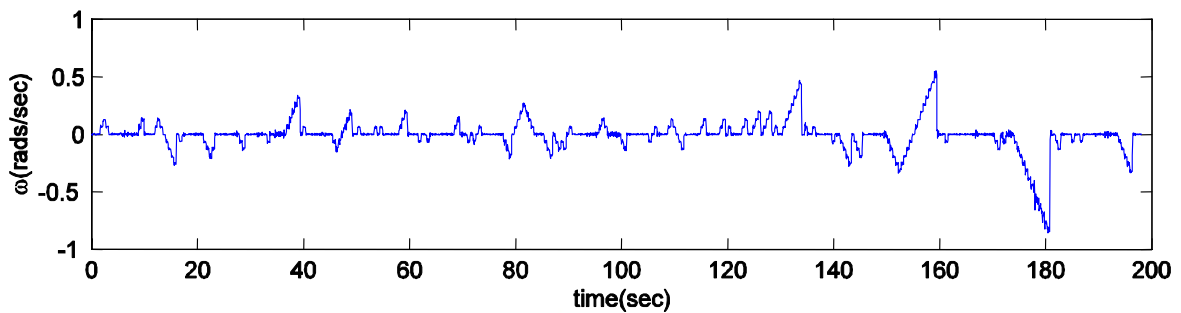
(a) Worktable and walls at (1) in Figure 4-4

(b) Doorway at (2) in Figure 4-4

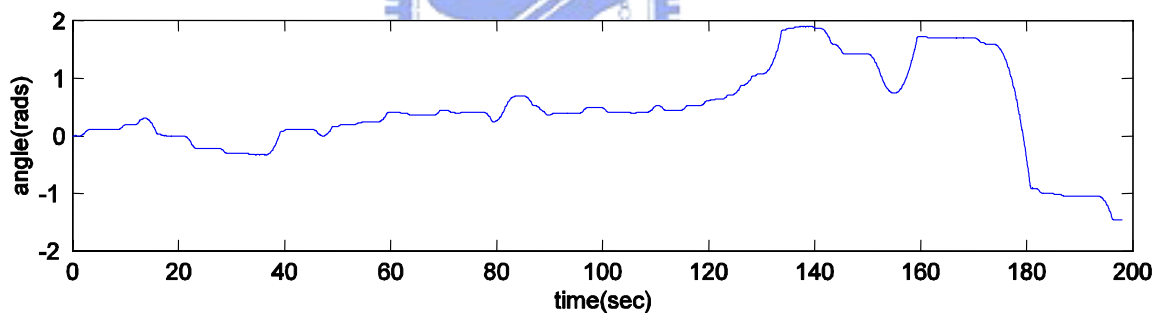
Figure 4-5 The environment of the experiment.



(a)The Magnitude of velocity versus time



(b)The Magnitude of angular speed versus time



(c)The angle versus time

Figure 4-6 The changes of the (a) velocity, (b) angular speed, and (c) angle versus time of this experiment

4.3. Experiment of two robot cooperation

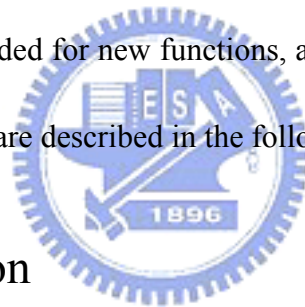
4.3.1. Goal

The goal of this experiment is to show how robots with different ability can cooperate

and accomplish an assigned task. The robots share their information obtained individually and finish the task according to their specific skills. Two heterogeneous robots are used in this experiment: the one equipped the sonar sensor is responsible for exploring the area and finding the path to the target; the robot with the grasping capacity otherwise follow the given path, achieving the target and perform its specialties. The cooperation model of multiple-robot cooperation (referred to Section 2.3.2.2) will be used and tested in this experiment.

4.3.2. Implementation

The control system of both robots are modified from the one introduced in the first experiment. New agents are added for new functions, and unnecessary agents are purged. The details for these modifications are described in the following sections:



4.3.2.1. Localization

In order to an obstacle-free path and latter on to follow it, the robots need a method to localize its coordinate. The dead reckoning can be use for a short distance, but is suffered from the accumulated error. Therefore, artificial landmarks, termed checkpoints, are utilized in this experiment for the localization of the robot. The checkpoint is a landmark with a two-color tube and four IR transmitters, shown in Figure 4-7.

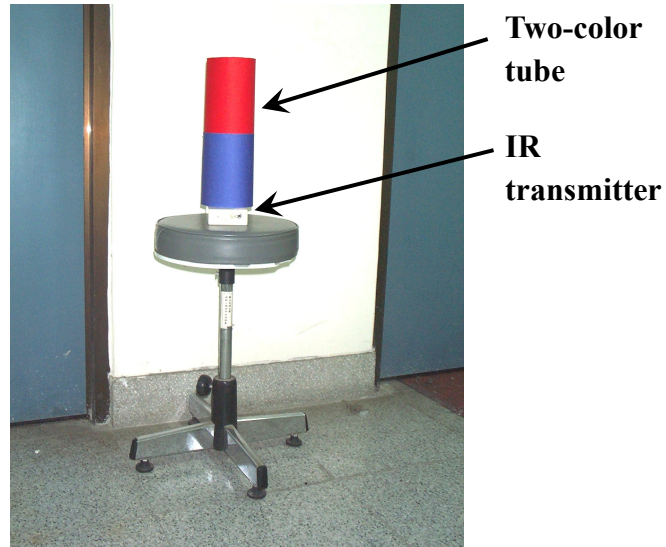


Figure 4-7 The artificial landmark: checkpoint

The two-color tube offers the distance between the robot and the checkpoint. The robot on-board webcam acquires the image, estimates the distance between the centers of two color areas, and estimates the distance using simple triangulation approaches. See Appendix A for detailed description. The range of distance measurement is from 0.8 m to 3 m

The IR transmitter offers an identification (ID) code of the checkpoint and the direction the robot orientation. The range IR sensor is from 0 m to 2 m (may vary with the available power of the battery). Each transmitter sends a unique digital code, represented by different frequencies. The receiver recognizes the code, and the robot will translate the code into the ID and the directions. Since the distance and orientation between the robot and checkpoint can be obtained, and the coordinates of checkpoints are already known in advance, the coordinate of the robot can therefore be estimated. This mechanism is useful for correction of accumulated error of odometer.

The *Checkpoint Sensor Agent* is responsible for this procedure. It obtains the information from the image sensor agent and IR sensor agent, and updates the recorded coordinate on board the robot. Furthermore, this agent is able to find the target, which is a special checkpoint with an inversed color tube.

In hits experiment, robot 1 first explores the environment, and locates the path to the target. The observed information will be transferred to robot 2, which can reach the target point directly.

4.3.2.2. Robot 1(Exploration)

The programming organization of robot 1 is shown in Figure 4-8. The exploration part is similar to the one described in the first experiment, which makes the robot be able to explore the area along the wall.

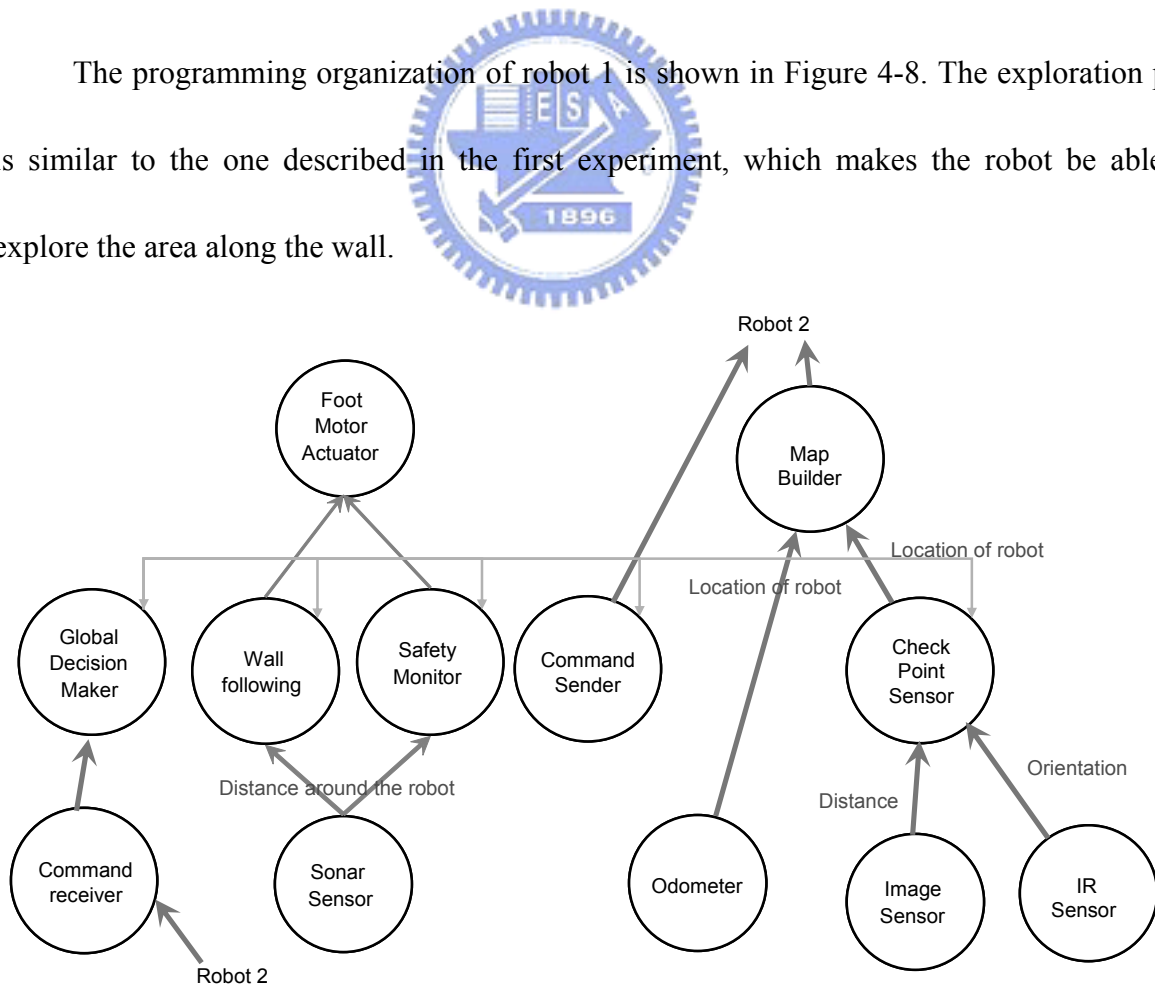


Figure 4-8 The programming organization of robot 1

During the exploration, the map builder obtains the coordinate of the robot, which will be updated by the odometer and Checkpoint Sensor Agent. When the target is found, the map builder will produce a path for robot 2, and send it to robot 2 via WLAN. For convenience, a two-colored tube is used to represent the target. It is up side down in order to be distinguished with others. The following steps produce the path:

Step 1: Detect the checkpoints, and record their coordinate.

Step 2: Mark via points between the robot and checkpoints (or the final target). The distance from via points is fixed to 1.6 meter far from checkpoints (or the final target).

Step 3: Connect via points and target point according to the sequence their relative location.



Meanwhile, the decision maker will launch the command sender and ask the robot 2 to come for the target. It will then wait for the arrival of robot2. If robot 2 finds the target, the command receiver will get the response from robot 2 and resume exploring the area.

4.3.2.3. Robot 2(Task Execution)

The programming organization of robot 2 is shown in Figure 4-9. Agents in robot 1, such as *Checkpoint Sensor*, *Command Sender*, etc, are reused. The *Target Achieving Agent* provides the ability to go from one target to another.

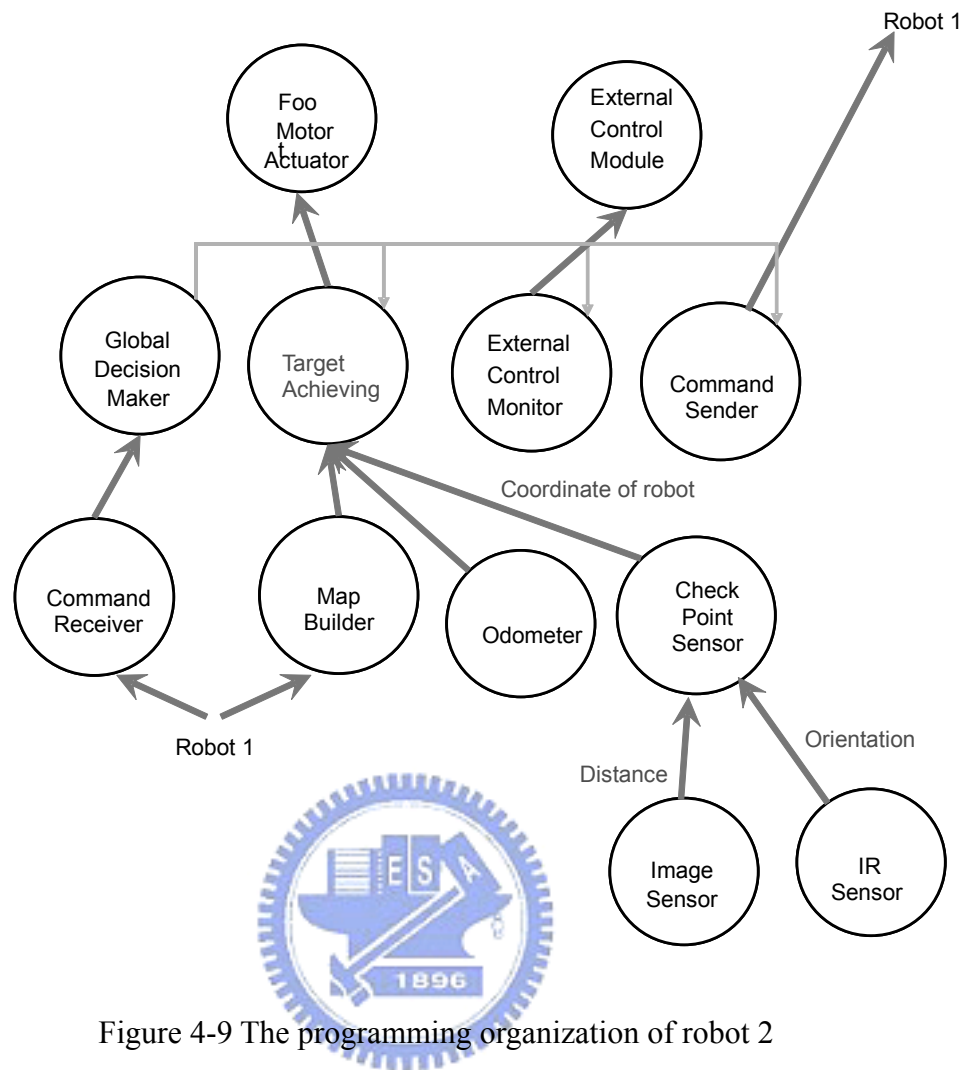


Figure 4-9 The programming organization of robot 2

The execution sequence is quite simple:

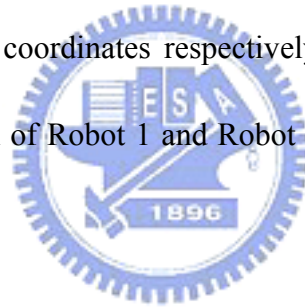
1. Update the coordinate of the robot from the *Checkpoint* Sensor Agent and the odometer.
2. Calculate the differential angle and distance between the robot and the target coordinate.
3. If the angle is more than ± 10 degrees, the robot will stop and rotate in order to face the target. Otherwise, the robot will approach to the target.

Once the map data and the command from robot 1 are received, the target-achieving

agent will be executed in order to go from one via point to the next. When robot 2 arrives the final target, it will give its control to the external control module to execute a special actuation function, for example, grasping. Meanwhile, robot 2 will inform robot 1 to start exploration again.

4.3.3. Experimental Result

Fig. 4-10 illustrates the environment of this experiment and the recorded trajectories of two robots. The experiment was carried out in the corridor of our laboratory. The target is situated at an open space in a corner of corridor, at the coordinate in meters (12.8,-3). Four checkpoints are placed at the coordinates respectively, (1.2, 1.2), (6, 1.2), (10.8, 1.2), and (15.6, 1.2). The initial location of Robot 1 and Robot 2 are (0, 0) and (-4, 0) respectively, as shown in Fig 4-11(a).



The process execution of this experiment is divided into two stages:

- **STAGE 1: Exploration of robot 1:**

Since the environment is unknown in the beginning of the experiment, robot 1 explores the environment using the sonar sensor. It first moves forward, since there were no obstacles in the corridor. Fig 4-10 shows that the recorded odometer data of robot 1 from its traveling along the corridor. However, the actual path of robot 1 is slanting to the right. This error was corrected every time robot 1 came by the checkpoint. These corrections cause a sudden change of the recorded trajectory, which can be easily observed in Fig 4-10.

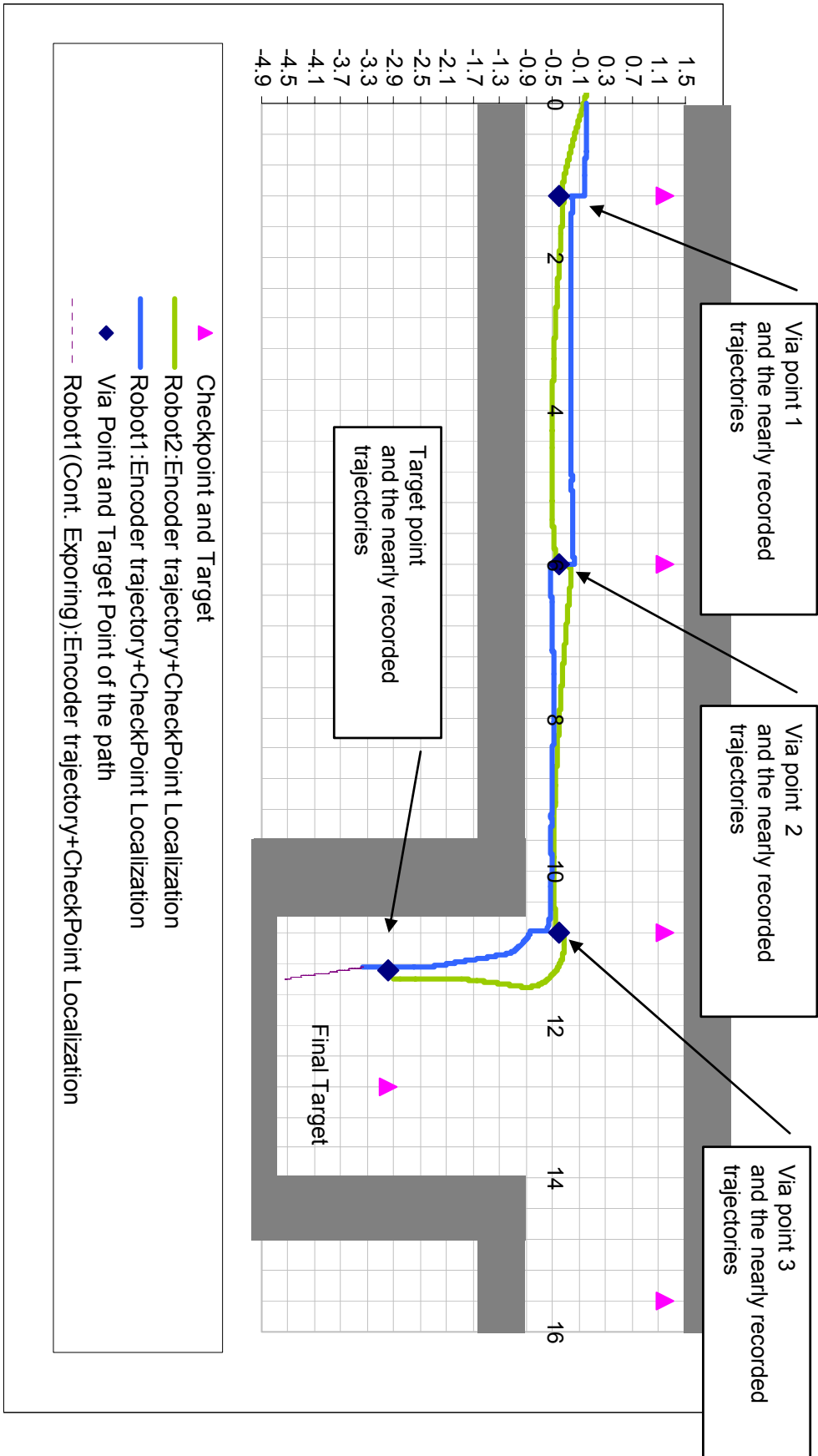


Figure 4-10 The recorded trajectories of two robots

(a)



(b)



(c)



(d)



(e)



(f)

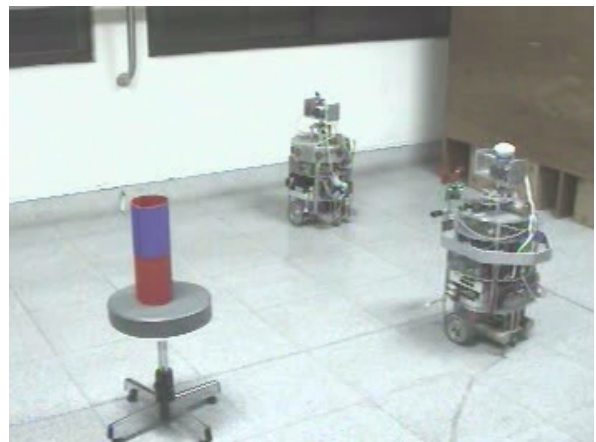


Figure 4-11 (a) Robots at the starting point (b)Robot 2 at via point 2 (c) Robot 2 at via point 2 (d)Robot 2 start the right turn (e)Robot 2 approaching target point (f)Robot 2 arrived

When robot 1 passed by the third checkpoint, the sonar sensor agent detected a free space at its right side. Robot 1 therefore made a right turn until it found a wall to follow again. This phenomenon can also be observed in Fig 4-10, at the coordinate (-0.5, 11). Later, robot 1 found the target with the webcam and stopped. During its journey, robot 1 marked via points 1.6m in front of every checkpoints and the target. The coordination of these via points is shown in Fig 4-10. Robot 1 then sent this information to robot2, and asks it to launch...

- **STAGE 2: Path tracking :**

After robot 1 reaches the target point, robot 2 started to move along the desired path, which is the exploration result of robot 1. Robot 2 first turned and reached via point 1. It adjusted its coordinate using the checkpoint, and continue to move toward the via point 2. At the beginning, robot 2 did not seem to move ahead to via point 2. It is because robot 2 would not adjust its direction if the angular error were within ± 1 degree, in order to avoid motion fluctuation. When the robot came closer to via point 2, the angular error increased, and the heading direction was adjusted. Robot 2 therefore approached via point 2 as expected, as shown in Fig 4-11(b). The trajectory in Fig 4-10 from via point 1 to via point 2 shows the u-shaped curve formed by the previous behaviors. Robot 2 then moved toward to via point 3, as shown in Fig 4-11(c). After it arrived, it turned right and moved forward to the target point, as shown in Fig 4-11(d). When robot 2 was arriving at the target point, as shown in Fig 4-11(e), it would inform robot 1 to resume exploring. At the end, robot 2 arrived the target point, as shown in Fig 4-11(f).

In Fig 4-10, it can be observed that the Checkpoint and path information collected by

robot 1 is correctly transferred to robot 2, and robot 2 uses the information to travel in the corridor and reach the target in a direct way. However, the accumulated odometer errors during the movement were corrected by using the Checkpoints. Fig 4-10 shows the adjustment of the trajectory of robot 2 at via point 1, 2 and 3. Without these Checkpoints, Robot 2 would suffer from the accumulated error, and cannot reach the given via points and target point as desired. The experimental result in Fig 4-10 show that the robot indeed use checkpoint to eliminate the accumulated error, and find via points and the target point accurately.

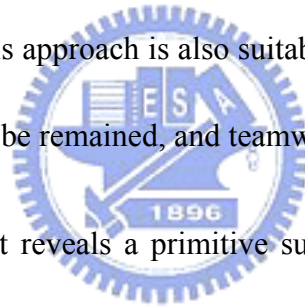
This experiment may fail under certain conditions. One of the main reasons is the error measurement of the ultrasonic sensor, as mentioned in experiment1. The robot may turn back before it finds the open space where the target is. The other one is the misunderstanding of the checkpoint data. The webcam suffers from the variance of the environmental illumination, which might result in an error on distance measurement and therefore a failed localization.



5. Conclusions and Future Work

5.1. Conclusions

This thesis presents a flexible real-time robot control system based on a multi-agent approach. The flexible architecture decomposes a complex robotic control system into agents, which makes teamwork more efficient. The software framework solves the resource sharing, message flows, and coding problems. It helps to reunite the separated works into a whole system with very few efforts. The core components guarantee a real-time responding and high performance of the system. This approach is also suitable for a long-term project in a research lab, since the former effort can be retained, and teamwork becomes much easier.



The practical experiment reveals a primitive success of the system. Agents fruitfully operate together, communicate with each other, give full scope to their specialist, and control the robot to explore the room autonomously and for a multi-robot cooperation task.

5.2. Future work

The proposed system only provides a skeleton for robot control. In the future, the improvement should be focused on two major parts. One is to provide a better development environment. For example, an integrated development environment (IDE) that can automatically transfer a piece of program into an agent suitable for the system will eliminate many unnecessary efforts doing copy-and-paste. A Graphics User Interface (GUI) that is able to

manage connection between agents will bring an effective way to design a robot controller in an intuitive way.

The other one is to increase the intelligence of the decision maker. Rather than using the pre-defined knowledge, the artificial intelligence and machine learning technique may provide an interactive and adaptive way to determine what should be done autonomously.



References

- [1] Robin R. Murphy, Introduction to AI robotics, The MIT press, MA 2000
- [2] R. C. Arkin, E. M. Riseman, and A. Hansen, “Aura: An architecture for vision-based robot navigation,” in Proc. of the DARPA Image Understanding Workshop, Los Angeles, CA, Feb 1987, pp. 417-431.
- [3] Nourredine Bensaïd and Philippe Mathieu, “A hybrid architecture for hierarchical agents,” Griffith University, Gold-Coast, Australia, Feb 1997, pp. 91-95.
- [4] K. Konolige, K. L. Myers, E. H. Ruspini, and A. Saffiotti, “The Saphira Architecture: A Design for Autonomy”, J. of Experimental and Theoretical Artificial Intelligence, 1997, pp. 215-235
- [5] Maria C. Neves and Eugénio Oliveira, "Fuzzy and Connectionist Paradigms in a Multi-Agent Control Architecture for a Mobile Robot," in Proceeding of the 3rd World Multi-Conference on Systems, Cybernetics and Informatics and The 5th International Conference on Information Systems Analysis and Synthesis (SCI/ISAS'99) ,Orlando - Florida - USA, July 31-Aug 4,1999
- [6] F. G. McCabe, and K. L. Clark, “April—Agent Process Interaction Language,” in Intelligent Agents, Lecture Notes in Artificial Intelligence, M. J. and Jennings, N. R., Eds, Springer-Verlag, 1995, pp.324-340,
- [7] Y. Takada, F. G. McCabe, and Y. Wada, “Multi-Agent Oriented Programming

- Language—April,” in Proc. of the 51st International IPSJ Conference, 1995.
- [8] Mattias Lindstrom, Anders Oreback, and Henrik I. Christensen, “BERRA : A Research Architecture for Service Robots,” in Proceeding of the IEEE Conference on Robotics and Automation, San Francisco, CA, USA, 2000, pp. 3278-3283
- [9] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. “The CLARAty architecture for robotic autonomy,” in Proc. of the 2001 IEEE Aerospace Conference, Big Sky, Montana, March 2001, pp. 121-132.
- [10] Herman Bruyninck, Peter Soetens, and Bob Koninck, “The Real-Time Motion Control Core of the Orocos Project,” in Proc. of the 2003 IEEE International Conference on Robotics & Automation, Taipei, Taiwan, September 14-19, 2003, pp. 2766-2771
- [11] Herman Bruyninckx, "Orocos (Open RObot Control Software)" [online], Nov. 2003, [cited Nov. 16, 2003], available from World Wide Web: <<http://www.orocos.org>>
- [12] Cloutier, et. al. DIAPM-RTAI Position paper. In Workshop on Real Time Operating Systems and Applications and 2nd Real Time Linux Workshop, November 2000. available from World Wide Web: < http://www.aero.polimi.it/projects/rtai/position_paper.pdf>
- [13] Wen-pin Chen, “Development of an Environment Exploration system for a Home Robot”, Master Thesis, National ChiaoTung University, July, 2002

Appendix A- Distance Estimation of Checkpoint

Assume that the distance between the two centers of the two-color-tube is known and noted as D . (See Fig A-1). f denotes the focal length of the camera. The distance between the camera and *Checkpoint* can be obtained by

$$Z = Df(d)^{-1} + \alpha_1(d-h)^{-2} + \alpha_2(d-h)^{-3} + \alpha_3(d-h)^{-4} + \alpha_4(d-h)^{-5},$$

where h and $\alpha_i, i=1 \sim 4$ are the parameters for correcting distance estimation. We can find these parameters using least-square estimation method.

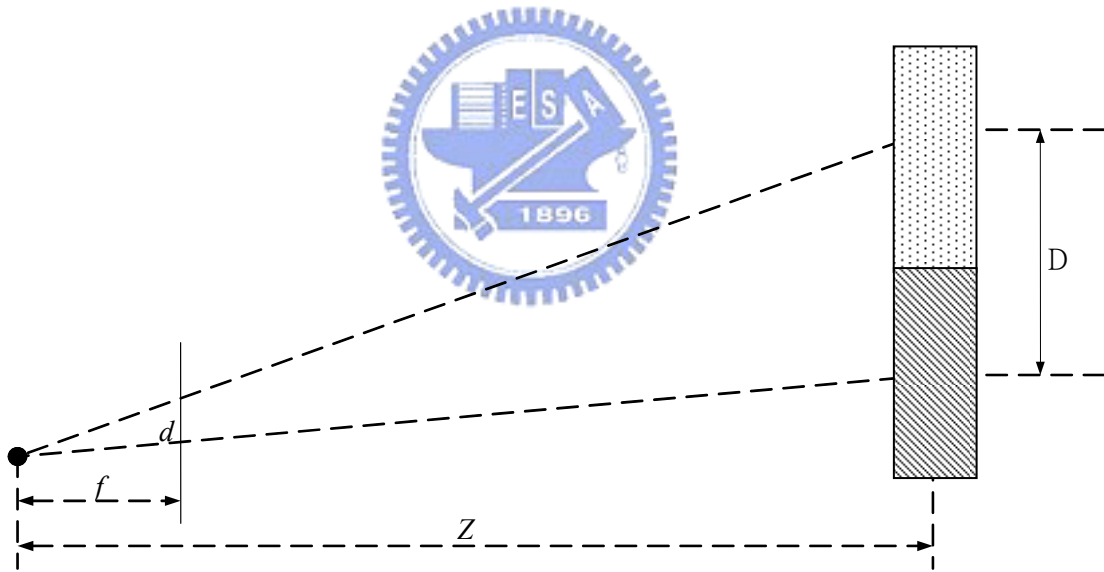


Figure A-1. Model of distance estimation.