






This is to certify that the  
dissertation entitled  
CLIENT SERVER CONTROL ARCHITECTURE  
FOR ROBOT NAVIGATION

presented by  
HANSYE SUDIANA DULIMARTA

has been accepted towards fulfillment  
of the requirements for

PH.D. degree in COMPUTER SCIENCE

  
PROFESSOR ANIL K. JAIN  
Major professor

Date MARCH 29, 1994

**LIBRARY**  
**Michigan State**  
**University**

**PLACE IN RETURN BOX to remove this checkout from your record.**  
**TO AVOID FINES return on or before date due.**

DATE DUE	DATE DUE	DATE DUE
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

**MSU is An Affirmative Action/Equal Opportunity Institution**

c:\circ\datedue.pm3-p.1





# **CLIENT-SERVER CONTROL ARCHITECTURE FOR ROBOT NAVIGATION**

By

*Hansye Sudiana Dulimarta*

A DISSERTATION

Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science Department

1994

h  
i  
c  
i  
P  
P  
o  
M  
M  
g  
s  
n  
a

# ABSTRACT

## CLIENT-SERVER CONTROL ARCHITECTURE FOR ROBOT NAVIGATION

By

*Hansye Sudiana Dulimarta*

Autonomous navigation is one of the major issues in mobile robotics. This capability is the key to making mobile robots useful to humankind. When a mobile robot is navigating inside its domain, the state of the world, as observed by the robot, changes with time. In such a situation, the role of sensors is important for achieving a robust navigation system. Moreover, the amount of information that must be processed is immense. This vast amount of data and the dynamic behavior of the perceived domain make the navigation task difficult.

This thesis addresses the problem of task decomposition in a mobile robot navigation system. The underlying supposition in this approach is that in a typical robot navigation system, there are a number of modules running concurrently and each module is assigned a specific subtask. To accomplish the common goal of the navigation task, these modules share the resources and common data in the system. In such a system, resource access control and information sharing among the modules must be properly managed.

In this thesis, a robot navigation system is decomposed into a number of *client* and *server* modules. Resource access control, resource sharing, information sharing,

and process synchronization in the entire robot navigation system are delegated to the server modules. The clients send appropriate requests to avail of these facilities. There are two types of server modules defined in the system: *data server* and *hardware server*. The core of the system consists of one data server and several hardware servers. The data server acts as a common information exchange medium for all the clients, while the hardware servers provide access interface to the hardware or peripherals on the robot. By decoupling the hardware access routines from the hardware servers, the modules in the navigation system can be made independent of hardware platform being used.

An indoor navigation system developed using the Client-Server model described above is presented. Probabilistic finite state automata have been developed for representing the confidence level of successfully completing a navigation task given to the system.

*O LORD, you are my God;  
I will exalt you and praise your name,  
for in perfect faithfulness  
you have done marvelous things,  
things planned long ago.*

*– Psalm 25:1*

Th  
individ  
individ

My  
encour  
and a  
ways.  
wonder  
had v

A  
vidin  
Duri  
a re  
grat  
tory  
Lab

Dr.  
The  
tati

CP:

## ACKNOWLEDGMENTS

This dissertation would not have been completed without the help given by many individuals during my research and graduate study. I would like to acknowledge those individuals here.

My first sincere thank goes to my dear wife Liana for her patience, sacrifices, encouragement, support, and understanding. Being both a wife of a Ph.D. student and a mother of a toddler has made her go through several difficult times. In many ways, I share my academic accomplishment with her. Thanks to Louisa for being a wonderful daughter. I feel pity for her for losing some of the time she should have had with her dad.

Also, I would like to thank my academic adviser, Professor Anil K. Jain for providing me insight, guidance, encouragement and support during my research work. During the first several months I worked with him, Professor Jain let me search for a research topic before I came up with one to pursue. In this respect, I am very grateful to him for his patience and kindness. As the Director of the PRIP laboratory, Professor Jain provided me a Research Assistantship by assigning me as a PRIP Laboratory manager.

My gratitude also goes to my other committee members: Dr. Matt Mutka, Dr. Richard Phillips, Dr. George Stockman, Dr. Lal Tummala, and Dr. John Weng. The discussions I had with them have sharpened the ideas expressed in this dissertation. It was Dr. Stockman who introduced me to Robotics when I was taking his CPS806 class. Dr. Tummala allowed Jeff Schneider and me to use his LABMATE

mobile

provide

State U

Dir

system

source

acknow

drafts.

Jon. S

then j

modu

Re

equip

Unas

W

peopl

Jinlor

Qian

Nalin

Lee, S

them.

Fin

for the

Barb C

Engels



mobile robot for our project in the class. Dr. Richard Enbody, my former adviser, provided me with the initial directions when I started my graduate study at Michigan State University.

During my assignment as a PRIP manager, I learned many things about Unix system administration from John Lees, my office mate. He is also my ultimate resource of T<sub>E</sub>X and L<sup>A</sup>T<sub>E</sub>X help whenever I came to a dead end. I would also like to acknowledge Jon Courtney's help in carefully proofreading most of my dissertation drafts. He and Steve Walsh did the "hammer and nail" work on our mobile robot. Jon, Steve and I were the robotics "subgroup" in the PRIP laboratory. Rick Herrell then joined the "subgroup" and assisted me in implementing and testing some of the modules used in my work.

Roxanne Fuller provided invaluable assistance by letting me use the EE shop equipment when I had to fix the hardware on the robot. Brian Wright, Shanti Vedula, Umashankar Iyer helped me in designing the circuit for the digital compass interface.

While working in the PRIP Laboratory, I also met a number of knowledgeable people from different parts of the world. Sushma Agrawal, Yao Chen, Shaoyun Chen, Jinlong Chen, Yuntao Cui, Chitra Dorai, Marrie-Piere Dubuisson, Sally Howden, Qian Huang, Michiel Huizinga, Jan Linthorst, Jian-Chang Mao, Sharathcha Pankanti, Nalini Ratha, Dan Swets, Marilyn Wulfekuhler, Yu Zhong, and former PRIPpies Greg Lee, Sateesh Nadabar, Tim Newman, Narayan Raja, and Deborah Trytten, are among them.

Finally, I would like to acknowledge my bible study group and Christian friends for their spiritual support and prayers; thanks to Dr. Tom Manetsch, Kelly Bartlett, Barb Czerny, Ralph Dicosty, Utami Rahardja, Ndibu Muamba, Christian Trefftz, Jon Engelsma, Iskandar Winata, Richard Setyabudhy, and Sutarto Hartono.

LIST C

LIST C

1 Intr

1.1

1.2

1.3

1.4

1.5

1.6

1.7

1.8

1.9

2 Sem

2.1

2.2

2.3

2.4

2.5

2.6

3 Con

3.1

# TABLE OF CONTENTS

<b>LIST OF TABLES</b>	<b>x</b>
<b>LIST OF FIGURES</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 History of Robotics . . . . .	2
1.2 Robotics Research . . . . .	7
1.2.1 Kinematics and Dynamics . . . . .	8
1.2.2 Trajectory Planning, Motion Control, and Obstacle Avoidance	8
1.2.3 Mobility . . . . .	9
1.2.4 Map Building and Navigation . . . . .	13
1.2.5 Multi- and Micro-Robotics . . . . .	15
1.3 Formal Models for Robot Navigation . . . . .	16
1.3.1 Finite State Machines . . . . .	16
1.3.2 Robot Schema . . . . .	18
1.3.3 Petri Nets . . . . .	20
1.4 Problem Definition and Approach . . . . .	21
1.5 Overview of the Thesis . . . . .	26
1.6 Achievements in Mobile Robotics . . . . .	26
1.7 Difficult Problems in Mobile Robotics . . . . .	29
1.8 Contributions of the Thesis . . . . .	31
1.9 Summary . . . . .	32
<b>2 Sensors</b>	<b>34</b>
2.1 Redundant Sensing . . . . .	36
2.2 Sensor Model and Uncertainty . . . . .	40
2.3 Computer Vision . . . . .	42
2.3.1 Stereo Matching Algorithms . . . . .	44
2.3.2 Stereo Vision in Mobile Robot Applications . . . . .	45
2.4 Sonar . . . . .	46
2.5 Infrared Proximity Detectors . . . . .	48
2.6 Summary . . . . .	48
<b>3 Control Architectures</b>	<b>49</b>
3.1 Multilevel Architecture . . . . .	50
3.1.1 NASREM . . . . .	51

3.1.2	ARTICS . . . . .	53
3.1.3	Task Control Architecture . . . . .	54
3.1.4	HAREMS . . . . .	58
3.1.5	Layered Architecture . . . . .	58
3.2	Connectionist Architecture . . . . .	60
3.2.1	Subsumption Architecture . . . . .	62
3.2.2	Colony-Style Architecture . . . . .	64
3.2.3	Autonomous Robot Architecture . . . . .	67
3.2.4	ACBARR . . . . .	69
3.2.5	Blackboard-Based Architecture . . . . .	70
3.3	Hybrid Architecture . . . . .	72
3.3.1	HRL/Voting-Based Architecture . . . . .	72
3.3.2	Rational Behavior Model (RBM) . . . . .	73
3.3.3	Concurrent Behavior Control Architecture . . . . .	73
3.3.4	Independent Agents Architecture . . . . .	75
3.4	Toolkit-Based Approach to Control Architecture . . . . .	76
3.5	Summary . . . . .	77
<b>4</b>	<b>Client-Server Control Architecture . . . . .</b>	<b>81</b>
4.1	Client-Server Model . . . . .	81
4.2	Remote Procedure Call . . . . .	84
4.3	Client-Server Control Architecture . . . . .	87
4.4	Related Work . . . . .	91
4.5	Implementation . . . . .	92
4.5.1	Interprocess Communication . . . . .	92
4.5.2	Client Interface Functions . . . . .	93
4.5.3	Server Interface Functions . . . . .	95
4.6	Servers . . . . .	98
4.6.1	DServer . . . . .	100
4.6.2	PServer . . . . .	103
4.6.3	RServer . . . . .	104
4.6.4	CServer . . . . .	110
4.7	Controlling Multiple Robots . . . . .	110
4.8	Emulating Other Control Architectures . . . . .	112
4.8.1	Emulating Hierarchical Systems . . . . .	113
4.8.2	Emulating the Colony-Style Architecture . . . . .	113
4.9	Summary . . . . .	122
<b>5</b>	<b>Indoor Navigation . . . . .</b>	<b>124</b>
5.1	World Representation . . . . .	125
5.1.1	Map Construction . . . . .	128
5.2	A Model of Robot Navigation . . . . .	132
5.3	Path Planner . . . . .	135
5.4	Navigator . . . . .	138
5.4.1	Center-Hallway state . . . . .	141

5.4.2	Corner state . . . . .	144
5.4.3	L-Goal . . . . .	145
5.4.4	D-Goal . . . . .	146
5.4.5	E-Goal . . . . .	146
5.4.6	Entering and Exiting Elevators . . . . .	146
5.5	Local Mapper for Heading Correction . . . . .	150
5.6	Ceiling Light Tracking . . . . .	152
5.7	Door Number Plate Detection . . . . .	156
5.8	Information Sharing Among Clients . . . . .	160
5.9	Summary . . . . .	161
<b>6</b>	<b>Experimental Results</b>	<b>162</b>
6.1	Indoor Navigation Experiments . . . . .	162
6.2	Door Number Plate Detection . . . . .	168
6.3	Confidence Levels of Indoor Navigation . . . . .	169
6.4	Summary . . . . .	178
<b>7</b>	<b>Conclusions and Future Research Directions</b>	<b>180</b>
7.1	Conclusions . . . . .	180
7.2	Limitations and Suggestions for Improvements . . . . .	183
7.3	Contributions of the Thesis . . . . .	185
7.4	Future Research Directions . . . . .	186
<b>A</b>	<b>A Map Generated from StickRep</b>	<b>188</b>
	<b>BIBLIOGRAPHY</b>	<b>190</b>

## LIST OF TABLES

1.1	Navigation control program vs. finite state machine. . . . .	17
3.1	Comparison of various control architectures. . . . .	79
4.1	Summary of commands to RServer. . . . .	106
4.2	Replacement rules for binary tree constructions. . . . .	119
5.1	Encoding of ceiling positions in the StickRep. . . . .	132
5.2	Relationship between modes, states, and goal functions. . . . .	140
5.3	Camera setup for ceiling light tracking. . . . .	154
5.4	Information sharing via the Data Server. . . . .	160
6.1	Results of the indoor navigation in the old wing of the MSU Engineer- ing Building. . . . .	165
6.2	Results of the indoor navigation in regions of the new wing of the MSU Engineering Building. . . . .	167
6.3	Door number plate detection results. . . . .	170
6.4	Interpretation of states in Figure 6.6. . . . .	173
6.5	Confidence levels of reaching goal positions. . . . .	175
6.6	Transition probabilities in Figure 6.9. . . . .	177

## LIST OF FIGURES

1.1	Robotics research. . . . .	7
1.2	Driving mechanism of KR I . . . . .	12
1.3	State transition diagram used by Crowley . . . . .	18
1.4	Client-server interconnections. . . . .	24
1.5	A StickRep representation (b) of a building structure (a). . . . .	25
1.6	Front view of RoME. . . . .	27
2.1	Stereo imaging geometry. . . . .	44
3.1	The NASREM Architecture for telerobots. . . . .	52
3.2	An example of a system using the TCA architecture. . . . .	56
3.3	Example of TCA task tree with a temporal constraint from task C to D. . . . .	57
3.4	HAREMS control architecture. . . . .	59
3.5	Crowley's Layered Architecture. . . . .	61
3.6	An example of a module in the Subsumption Architecture. . . . .	63
3.7	An example of module interconnection in the Subsumption Architecture (reproduced from [Brooks, 1986]). . . . .	63
3.8	Structure of two modules in the Colony-style Architecture. (a) memoryless module; (b) module with memory. . . . .	66
3.9	An example of a network in the Colony-style Architecture with two suppressor nodes and one inhibitor node. . . . .	67
3.10	Navigation system in AuRA. . . . .	69
3.11	Structure of the distributed agent in blackboard-based architecture. . . . .	71
3.12	The structure of the Concurrent Behavior Control Architecture. . . . .	74
3.13	A model of Independent Agents Architecture. . . . .	76
4.1	Client-server relationship. . . . .	83
4.2	An example of a Remote Procedure Call. . . . .	86
4.3	Typical configuration of the Client-Server Architecture. . . . .	88
4.4	A segment of <b>SockStr</b> class declaration. . . . .	93
4.5	An example of a program communicating to <b>RServer</b> . . . . .	94
4.6	Public interfaces of <b>Server</b> class declaration. . . . .	95
4.7	An example of a request handler in a server. . . . .	97
4.8	Library of functions defined on top of <b>SockStr</b> class. . . . .	99
4.9	An example of a multiple Client-Server Control Architecture. . . . .	111
4.10	Intermodule communication in: (a) hierarchical systems, and (b) its equivalent construct in the Client-Server Architecture. . . . .	113

4.11	The internal structure of a module in Colony-Style architecture. . . .	114
4.12	Inhibition network in the Colony-style architecture. . . . .	115
4.13	Suppression network in the Colony-style architecture. . . . .	117
4.14	Two different suppression networks with the same priority assignment.	118
4.15	Binary trees constructed from suppression networks. . . . .	120
5.1	Client modules and their data flow. . . . .	126
5.2	A partial map of the third floor of the MSU Engineering Building. . .	128
5.3	A hypothetical building structure. . . . .	129
5.4	A StickRep representation of the structure shown in Figure 5.3. . . .	130
5.5	Ceiling light projection for three lights (1,2,3) in a hallway. . . . .	131
5.6	Navigation model. . . . .	133
5.7	Different operating modes of the robot. . . . .	137
5.8	Control loop in the Navigator. . . . .	139
5.9	Sonar configuration. . . . .	141
5.10	<i>Far</i> and <i>near</i> sonars. . . . .	142
5.11	Barcode-like Markers representing binary values: (a) 000 and (b) 100.	148
5.12	An elevator door pasted with a barcode-like marker. . . . .	149
5.13	An HIMM map created from a corner in the hallway. . . . .	151
5.14	RoME in the hallway of the MSU Engineering Building. . . . .	153
5.15	Camera setup for detecting ceiling lights (facing up) and door number plates (facing sideways). . . . .	154
5.16	Door number plate detection result. (a) input image, (b) door number plate detected. . . . .	156
5.17	Number Plate Detection Geometry. . . . .	158
5.18	Two overlapping snapshots. . . . .	159
6.1	Regions in the map. . . . .	163
6.2	Output produced by the digital compass while the robot is moving north.	163
6.3	A finite state automaton representing a landmark-based navigation task.	171
6.4	A simplified representation of Figure 6.3. . . . .	171
6.5	A finite state automaton for a path with $k$ corners. . . . .	172
6.6	A generalized representation of the FSA in Figure 6.5. . . . .	172
6.7	The FSA in Figure 6.6 after the estimation of the probability values.	174
6.8	A probabilistic automaton for the PIFinder. . . . .	175
6.9	The modified finite automaton of Figure 6.8. . . . .	176
A.1	A map of the third floor of the MSU Engineering Building. . . . .	189



# CHAPTER 1

## Introduction

**ro·bot** [Czech, fr. *robota*, work] **1:** a machine in the form of a human being that performs the mechanical functions of a human being but lacks sensitivity **2:** an automatic apparatus or device that performs functions ordinarily ascribed to human beings or operates with what appears to be almost human intelligence **3:** a mechanism guided by automatic controls. \*

For some people, when they hear the word “robot”, the image that comes to their mind is a human-like figure with one head, two arms, and two legs, that is able to move its arms and legs, or even “say” a few words. However, others have different perception about the word “robot”, and they even might not come up with a simple answer when they are asked to define what “robots” are.

In fact, a robot can be viewed as a complex system. How complex is it? It depends on the level one considers. At the highest level, one could view a robot as composed of *hardware* and *software* subsystems. At the next highest level, the hardware subsystems can be decomposed into several sub-subsystems such as: sensing, manipulation, locomotion, controller, and power subsystems. In general, these components can be classified into: effectors, sensors, computers, and auxiliary equipments. On the software side one can find, for instance, the navigation, perception, communication,

---

\*Electronic version of Webster’s 7th Collegiate Dictionary.

path-planning, motion control, and object recognition sub-subsystems. In some of these sub-subsystems, there is no clear distinction between software and hardware.

The complexity of a robot system has led us to many different research areas in robotics. Each one of these areas might be associated with one or more “nodes” in the subdivision described above.

## 1.1 History of Robotics

The history of today’s robots could be traced back to the eighteenth century when *automata*<sup>†</sup> were popular. Then, *automata* (or “*automatons*”) referred to mechanical objects that moved automatically. Besides clockworks, some of the well-known automata during that era were de Vaucanson’s mechanical duck created in 1738, and Jacquer-Droz’s automatic scribe created in 1774 [Asimov and Frenkel, 1985]. Later in 1745, de Vaucanson invented the first automatic weaving loom. His invention marks the advance of controllable machines in which a program could be designed to control their operation. Jacquard improved de Vaucanson’s idea and used punch cards for controlling the motion of the needles to create intricate patterns on the cloth.

In the meantime, inventions that led to digital calculating machines were also in progress. Pascal invented the adding machine; Leibnitz discovered how to do multiplication mechanically. In 1823 Babbage attempted to design the first digital calculating machine which was based on the new algebra developed by Boole. Babbage’s work on the “Difference Engine” and the “Analytical Engine” was not completed. These “engines” were derived from Jacquard’s loom, and punch cards were used to control the arithmetic operations. While working with the U.S. Census Bureau, Hollerith

---

<sup>†</sup>In this context, the word *automata* has a different meaning from what is used in theoretical computer science.

develop

tronech

in the t

first ful

designe

digital

ENIAC

Mo

"intelli

sidered

used t

play, r

of sen

and p

human

ciety's

as ma

In

Z

developed a method of recording statistics on punched cards and invented his electromechanical tabulating machine used in the U.S. censuses of 1890 and 1900. Later in the twentieth century, Bush and his associates at MIT made history with the first fully automatic calculator (also the first analog computer); Aiken at Harvard designed his electromechanical scientific calculating machine called Mark I (the first digital computer); Mauchly and Eckert at the University of Pennsylvania designed ENIAC, the first electronic computer.

Motivated by all these inventions, people then began to consider incorporating “intelligence” into computers to create “true” automatons. This idea had been considered not only by scientists but also by science-fiction writers. In 1920, Capek used the word *robot* in a play entitled R.U.R. (*Rossum’s Universal Robot*). In the play, robots were artificially manufactured persons, mechanically efficient, but devoid of sensibility. Henceforth, the word *robot* was widely used in place of *automaton* and popular robot science-fiction stories often portrayed robots as dangerous, wicked, human-like figures. In 1939, Isaac Asimov, a young science-fiction writer, changed society’s perception of robots. Contrary to contemporary ideas, he viewed robots merely as machines, and he proposed his Three Laws of Robotics [Asimov and Frenkel, 1985]:

**First Law:** A robot may not injure a human being, or, through inaction, allow a human being to come to harm.

**Second Law:** A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.

**Third Law:** A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

In 1985, Asimov revised his Three Laws of Robotics and proposed the following:

**Zeroth Law:** A robot may not injure humanity, or, through inaction, allow humanity to come to harm.

**First Law:** A robot may not injure a human being, or, through inaction, allow a human being to come to harm, unless this would violate the Zeroth Law of Robotics.

**Second Law:** A robot must obey the orders given it by human beings except where such orders would conflict with the Zeroth or First Law.

**Third Law:** A robot must protect its own existence as long as such protection does not conflict with the Zeroth, First, or Second Law.

As computer technology advanced, smaller computers were widely used in controlling machine operations. Robots were then viewed as *mobile computers* which did not have to look like humans, but instead were expected to duplicate human capabilities. The outcome of this idea was the advent of computerized manipulator arms which also made a debut in industry (and hence the name *industrial robots*). Devol held the first patent on industrial robots. Together with Engelberger, Devol launched industrial robot manufacturing by establishing their company, Universal Automation (or “Unimation” for short). With this debut, Engelberger and Devol were also known as the “Father and Grandfather of Industrial Robots”.

It is worth noting that automated manufacturing does not necessarily imply robotization, because robotization is more than just automatization. The main criteria used to justify automatization vs. robotization are flexibility and adaptability [Brady, 1985]. In general, robotization provides flexibility in operation but automatization does not. Therefore, automatization is sometimes also called *hard automation*. In hard automation, preprogrammed machines are used in place of manually operated mechanical tools. Preprogrammed machines are dedicated to doing a single task consisting of a *known* sequence of operations. On the other hand, programmable machines offer more flexibility because they can be programmed for a number of different tasks. Robot manipulator arms can be categorized as this type of system. The Robotics Institute of America (RIA) has provided the following definition of robots:

A robot is a reprogrammable, multifunctional manipulator designed

One

tions.

moving

robot sh

that is E

Some ta

inspecti

building

Mol

that wh

manip

the wo

robots.

robots

as a re

independe

As

also a

This r

the or

robots

sensor

robots

tasks

roboti

to move materials, parts, tools, or specialized devices through variable programmed motions to perform a variety of tasks.

One of the objectives for building robots is to make them duplicate human functions. In accomplishing a task, humans do not just sit at one place, but might be moving actively from place to place. Careful scrutiny of the RIA's definition of a robot shows that robots are designed to *move* objects. To move objects to a place that is beyond the reach of the robot manipulator, the robot must possess mobility. Some tasks that require mobility of the robots are: room cleaning, drainage pipe inspection, exploration (outerspace as well as underwater), and cleaning of high-rise buildings.

Mobile robots are often considered different from industrial robots, in the sense that when people talk about industrial robots, they usually refer to programmable manipulator arms and not to wheeled robots (or other types of locomotion). Likewise, the word *mobile robot* often refers to wheeled, legged, submersible, and free-flying robots. One of the factors which contributes to this distinction is that industrial robots were invented by industrial demands whereas mobile robots were invented as a result of academic research. Furthermore, research in both fields has evolved independently.

As industrial robots were being developed, researchers in computer science were also aiming to incorporate perceptual and problem-solving capabilities into robots. This required attaching sensors to the robots that would bring the information from the outside world to the computer—the “brain”—of the robot. The aim was to make robots that could “see” (e.g., through TV cameras) and “feel” (e.g., through touch sensors on robot grippers). In a more general setting, the overall goal is to make robots “think”, “walk”, “grab”, “hear”, etc., enabling them to perform intelligent tasks as humans do. Here, the role of sensory input is very important. Brady views robotics as the intelligent connection between perception and action [Brady, 1985].

This

other

sense.

But

Kashya

levels of

• I

I

c

•

•

•



This view extends the meaning of *sensing* to *intelligent sensing*, i.e., *perception*. In other words, intelligent machines are required to perceive or interpret, not merely sense, the information obtained from their sensors.

Building intelligent machines has proved to be a difficult task. Iyengar and Kashyap characterized the evolution of intelligent robots and cited the following four levels of autonomy of robots [Iyengar and Kashyap, 1989]:

- Teleoperation — where machines are remotely operated under human control. This setting merely follows a master-slave configuration, where the slave machine at a remote distance follows the instructions given by a master controller.
- Telesensing — where machines have additional sensing capability about the teleoperator and environment. The perceived information is communicated back to the human operator who controls the machine continuously. Here, the human operator can sense what is happening at the remote site.
- Telerobotics — where human operators do not have to control the machines continuously, instead the operator intermittently communicates via a computer about goals, plans, constraints, etc., and receives feedback from the telerobot. The telerobot executes the task or goal given to it by its own sensors and knowledge base.
- Autonomous Intelligent Machines — where a human operator supplies a single high-level command and the robot does all the necessary task sequencing and planning to execute the command. In doing so, the robot needs a priori knowledge for reasoning.

## 1.2

The  
the h  
able  
requ  
pose  
Solv  
such  
gras  
bolt  
enou  
such

I

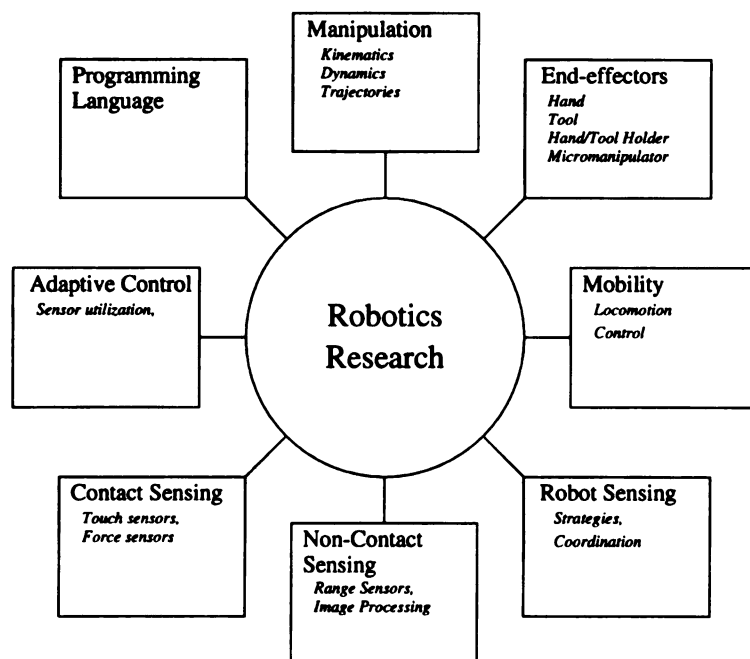


Figure 1.1. Robotics research.

## 1.2 Robotics Research

The ultimate goal of robotics research is the design of intelligent machines which is the highest level of autonomy cited by Iyengar and Kashyap. So far, no one has been able to build a truly autonomous intelligent machine that satisfies the aforementioned requirements. Brady showed a good example that proves this fact [Brady, 1985]. He posed a very simple problem for humans: How does one disconnect a (car) battery? Solving this simple problem requires its decomposition into a number of simple tasks such as: determining the size of the bolt, finding the proper wrench that fits the bolt, grasping the wrench properly, aligning and mating the wrench with the head of the bolt, determining whether there is enough free space for wrench rotation, applying enough force to unscrew the bolt and so on. To date, no robot is able to perform such a task.

Research activities in the field of robotics have been pursued in a number of

discip

comp

engine

tion of

adapti

[Nitz

### 1.2.1

Indus

trol

indiv

this

the

to d

the

the

gen

by

### 1.2

On

rob

sur

to n

is n

disciplines, such as applied physics, applied mathematics, biomedical engineering, computer science, control and systems science, electrical engineering, and mechanical engineering. Nitzan has identified the following research areas in robotics: manipulation of arms, end-effectors, mobility, general sensing, non-contact vs. contact sensing, adaptive control, robot programming languages, and manufacturing process planning [Nitzan, 1985]. Figure 1.1 shows the different areas in robotics research.

### 1.2.1 Kinematics and Dynamics

Industrial robot arms generally consist of several links connected by joints. To control the motion of such arms, one needs to understand the relationship between the individual joint motions and the geometry of arm motions. Kinematics deals with this issue. Suppose a robot hand is to be moved from one position to another. Given the initial and final positions, kinematics tells us *how much* each joint has to move to displace the hand from one position to the other, but it does not tell *how* to move the joint (how fast?, how slow?, when?, etc.). Dynamics fills the gap by providing the answer to these questions. Hence, kinematics and dynamics go hand in hand. A general and systematic method of describing links or joints geometry was proposed by [Denavit and Hartenberg, 1955].

### 1.2.2 Trajectory Planning, Motion Control, and Obstacle Avoidance

One application example that requires trajectory planning is arc welding, where the robot hand has to follow a specified path at a prespecified height above the metal surface. This task can be formulated as follows: How does one control the joints so as to make the hand follow the specified trajectory? In other applications, the trajectory is not specified, but instead the end points are. Here, the robot arm is required to

have obstacle avoidance capability to move the arm from one end point to the other *without* running into any objects. In motion control, the state description model of the robot manipulator is given, and one needs to determine the control laws, derived from the calculated trajectory, to achieve the desired system response. A common approach to controlling the robot arm is to use a closed-loop feedback control.

A number of motion planning algorithms have been proposed by many authors [Leven and Sharir, 1987; Lozano-Pérez and Wesley, 1979; Lozano-Pérez, 1983; Lumelsky and Stepanov, 1987; Sharir, 1989]. Sharir wrote a series of papers on this problem, known as the Piano Movers' problem [Schwartz and Sharir, 1983a; Schwartz and Sharir, 1983b; Sharir and Ariel-Sheffi, 1984; Schwartz and Sharir, 1984]. Most of these algorithms assume that the robot's environment is known. A major breakthrough in robot motion planning was shown in Canny's thesis [Canny, 1987] where he proposed the "roadmap" algorithm that reduces the complexity of the algorithm from double exponential to single exponential.

Khatib approaches the obstacle avoidance problem using the *Artificial Potential Field* method [Khatib, 1986]. Using this approach, the robot is considered to move in a field of forces. The goal position of the robot is considered as an attractive pole and obstacles are repulsive surfaces.

### 1.2.3 Mobility

Hirose and Morishima have identified three fundamental configurations of mobile robots based on their means of locomotion: wheels (with or without a tractor chain), legs, and articulated bodies [Hirose and Morishima, 1990]. Depending on the roughness or irregularity of the terrain, one method of locomotion might be more suitable than the others. The first configuration originated from locomotive vehicles like cars and tractors, whereas the other two configurations originated from animal-like locomotion.

## Wheeled Robots

In 1960's, researchers at John Hopkins University were able to prowl their Hopkins' Beast around the University's corridors. Later in the decade, Stanford Mechanical Engineering Department built the Stanford Cart subsequently acquired by the Stanford Laboratory [Moravec, 1981]. Stanford Research Institute (SRI) International also completed its mobile robots named Shakey I and Shakey II. Some more recent mobile robot projects include: CMU Rover [Moravec, 1983], CMU NavLab [Thorpe *et al.*, 1991b; Thorpe *et al.*, 1991a], JPL's Rocky III, Rocky IV and Robby [Matthies, 1992].

## Legged Robots

Raibert has identified three main motivations for studying legged-locomotion: *mobility*, *animal biology*, and *dynamics* [Raibert, 1990]. Legged-locomotion can be used on rough terrains, such as rocky areas, hilly regions, wooded areas, etc., as well as indoors. Thus, it has higher versatility. Legged vehicles could also be beneficial to biologists who want to study the properties of legged locomotion in people and animals. Since the legs have to support the weight of the robot, dynamics become more complicated. Six-legged vehicles seem to be popular and the following is a list of projects which have used this approach.

- AMBLER [Bares *et al.*, 1989] (Autonomous MoBiLe Exploration Robot) built at Carnegie Mellon University for planetary exploration. The legs are attached to a central pole and are controlled as two sets of three legs (using tripod gait exclusively). Range sensors are attached to each leg to sense the topology of the terrain.
- OSU Hexapod built by Ozguner *et al.* at Ohio State University [Ozguner *et al.*, 1983]. The height of each leg can be adjusted to keep the robot body level

when maneuvering on irregular terrains. Vertical gyroscope and pendulums are also used to keep the body level. Each leg also has a force sensor that can be used to “feel” the terrain.

- Sutherland’s walking machine. This robot weighs 1600 lbs. and uses hydraulic actuators for leg motion [Sutherland and Ullner, 1984]. The on-board computer controls the sequencing of each leg motion.

Besides six-legged robots, biped and quadruped locomotion are also popular. “Quadruped” was built at the Tokyo Institute of Technology by [Hirose *et al.*, 1983]. Self-balanced robots have also been studied by [Miura and Shimoyama, 1983] and implemented as BIPER3 and BIPER4. Raibert *et al.* also experimented with a three-dimensional hopping machine [Raibert *et al.*, 1983] and bipedal locomotion robot. The BLR-G2 biped [Furusho and Sano, 1990] built by Furusho and Sano at Gifu University is another example of a self-balanced robot.

### **Articulated Body Robots**

This novel approach has not been explored as much as the other two configurations. Here, multiple segments (or articulated bodies) are joined one after another to produce snake-like movement. Due to this type of structure and locomotion, articulated body robots have a higher degree of freedom than any other type of robot. Each segment contributes a small degree of freedom and hence, for multiple segments, the total number of degrees of freedom is a multiple of this value plus additional joints connecting the segments. A number of examples of articulated body robots can be found in [Maeda *et al.*, 1985; McGhee *et al.*, 1980; Ozaki *et al.*, 1983; Ohmichi, 1985].

As part of a research project in robots for atomic reactors, Hirose and Morishima have built the robot KR I (Koryu I), which is a hybrid of wheeled and articulated



body configurations [Hirose and Morishima, 1990]. This six-segment articulated body robot is 40 cm high, 20 cm wide, and has a length of 140 cm. Each joint in KR I has two degrees of freedom (rotation and translation along the vertical axis), for a total of 16 degrees of freedom. This design has an advantage over the inflective joint design, because less force is needed for elevation of each segment. Some capabilities that have been demonstrated by KR I are: passing through a right-angle corner, striding over duct obstacles, stepping over a ditch, stair-climbing, and traversing on an inclined surface. In order to maintain stability in the latter case, the segments are configured in a W-shape form so as to obtain a wider footprint and prevent the robot from falling over. The control mechanism of this robot is also of interest. For instance, the robot employs vertical force control to walk over an obstacle of a certain height, or walk on an irregular surface. A thorough analysis is given in [Hirose and Morishima, 1990]. Figure 1.2 shows the driving mechanism of this articulated body robot.

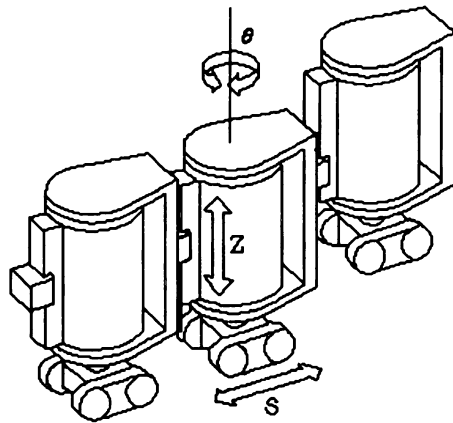


Figure 1.2. Driving mechanism of articulated body mobile robot KR I (reproduced from [Hirose and Morishima, 1990]).

Earlier, Hirose also built ACM III (Active Cord Mechanism), which is comprised

sensor

objects

has the

robots

#### 1.2.4

In build

sensors

a repre

which

depend

depend

in prac

hence

resolu

navig

maps

M

[Mor

The

the c

whic

aroun

appr

R

and

of 20 segments and is about 2 meters long. Each segment is equipped with a tactile sensor. ACM III has shown success in moving within a maze, or even curling around objects. ACM VI, a successor of ACM III, is even more flexible, because each joint has three degrees of freedom. The maximum speed that can be achieved by these robots is 0.6 m/s.

### 1.2.4 Map Building and Navigation

In building a map of the environment, a robot takes several measurements using its sensors from a number of distinct positions and orientations, and incrementally builds a representation of the workspace. Each measurement is used to build a local map which is then combined into a larger global map. The accuracy of the global map depends on how accurately the local maps are registered to each other, which in turn depends on how well the robot stores its internal position and orientation. However, in practice, the true position and orientation of the robot cannot be obtained, and hence the global map is not accurate. One of the major issues in map building is *resolution*, i.e., the granularity of the map in the robot's internal representation. For navigation purpose, the robot can use the information gathered from either the local maps or the more complete global maps.

Moravec and Elfes used ultrasonic sensors to build a map of the robot's workspace [Moravec and Elfes, 1985]. A probabilistic occupancy grid was used as the model. The presence of objects within the workspace is shown as high probability values in the occupancy grid. Initially, each cell in the grid is set to a probability value of 0.5, which denotes that the robot has no knowledge about the cell. As the robot moves around, each cell in the grid is updated with current sensor readings using a Bayesian approach. A similar approach was used by [Elfes, 1989] for robot navigation.

Roth-Tabak and Jain used a 3-D discrete representation of the world [Roth-Tabak and Jain, 1989]. Their volumetric model uses discrete values (void, full, unknown) to

el

the sp

Th

each r

empty

found

Be

in the o

Beck

In th

to desig

used be

is respo

main se

reflecta

in (Sir

Bot

force fi

acting

vector

La

(VFH

used a

sensor

rate, o

an obj

author

label each voxel. The environment was built from *dense* range data, as opposed to the sparse data produced by an ultrasonic sensor.

Thorpe models floor space as a grid of cells containing a suitability measure of each region to be on the robot path [Thorpe, 1984]. High values mean the cell is empty, whereas low values were assigned to cells near obstacles. An optimum path is found by a relaxation algorithm.

Beckerman and Obrow approached the navigation problem by using discrete labels in the occupancy grid, as opposed to probabilistic values used by Moravec and Elfes [Beckerman and Obrow, 1990].

In the Ambler Project [Bares *et al.*, 1989], Bares *et al.* initiated a research project to design a roving explorer capable of operating autonomously. Legged locomotion is used because the robot is supposed to move on a rough terrain. Its perception system is responsible for building and maintaining *terrain maps* and discrete objects. The main sensors used for this purpose are a set of laser range finders that provide both reflectance and range. The control algorithm implemented in Ambler is described in [Simmons and Krotkov, 1991].

Borenstein and Koren used a slightly different approach which they called *virtual force field* (VFF). In this approach the obstacle vectors were viewed as repulsive forces acting on the robot, while the target point attracts the robot with another force. The vector sum of these forces represents the direction the robot should proceed.

Later on, [Borenstein and Koren, 1991b] implemented the *vector field histogram* (VFH) in their obstacle avoidance algorithm. In this approach, a histogram grid was used as the world model. Each cell in the grid keeps a tally of how many times the sensors see an obstacle in the corresponding position. To achieve faster processing rate, only one cell (the target cell) on the sonar axis is updated when a sensor detects an object. The location of the target cell is computed from the sonar reading. The authors also experimented with a different updating scheme [Borenstein and Koren,

1991a], where not only the target cell is updated but also the cells between the sensor and the target cell. In order to determine the safe direction(s) the robot can proceed from its current position (cell)  $P$ , a polar histogram is constructed using the length of *obstacle vectors*. Each obstacle vector has its origin at  $P$  and ends at a cell  $Q$  which lies within a certain distance from the robot's current position. The tally of cell  $Q$  determines the length of the obstacle vector  $\overrightarrow{PQ}$ . The sum of magnitude or length of all obstacle vectors that share a common wedge is used to construct the polar histogram. Valleys in the resulting histogram represent safe directions along which the robot can proceed.

Kweon and Kanade have experimented in building rugged terrain maps from multiple sensors [Kweon and Kanade, 1992]. The final map was constructed by merging a large number of range images. With the knowledge of sensor geometry, locus methods can be used to create elevation maps from range images.

### 1.2.5 Multi- and Micro-Robotics

There have been some experiments conducted on controlling several cooperating robots for solving a common task. This approach is also known as multi-robotics. Compared to the other research areas in robotics, multi-robotics has not received as much attention.

Miller has been working on a team of *microrovers* for applications in exploration [Miller, 1990]. Fujimura used a reactive planning approach for controlling multiple mobile robots [Fujimura, 1991]. A distributed approach to controlling a number of robots was also proposed by [Sugihara and Suzuki, 1990].

Besides searching for new configurations, miniaturization seems to be another promising area of research in robotics. Brooks recently experimented with designing and building micro robots named Attila and Genghis [Flynn *et al.*, 1989; Brooks, 1991; Freedman, 1991].

## 1.3 Formal Models for Robot Navigation

There have been some attempts at formalizing robot behavior using mathematical models. Some models that have been used in the literature are: finite state machines (automata), Schema, and Petri Nets.

### 1.3.1 Finite State Machines

Finite automata (finite state machines) have been used for modeling the behavior of a system with discrete inputs and outputs. The machine has a finite number of internal “states” or configurations. State transitions occur when the machine receives an input. Formally, a finite state machine consists of the following components [Hopcroft and Ullman, 1979]:

- A finite set of *states*,
- A finite set of *input symbols*,
- A finite set of *output symbols*,
- An *initial* state,
- A set of *final* states, and
- A *transition function* mapping that describes how the machine changes state upon receiving an input symbol.

Instead of defining a mathematical function representing state transitions, one can also draw a *transition diagram* using a directed graph in which a node represents a state and an arc represents a state transition. The outputs of a finite state machine can be associated with either its states or its transitions. In Moore machines, outputs are associated with states, while in Mealy machines, outputs are associated with transitions.

g  
in  
n  
be  
ti

tra  
fro  
De  
ent  
sen  
pat  
mo

Gui  
and  
stat



In a navigation task, a robot is given a command to move from an initial to a goal position. The robot starts at an initial state and upon reaching the goal position it is at a final state. While the robot is maneuvering between the two positions, it might change states. Intuitively, one can see that a navigation control program can be modeled by a finite state machine. Table 1.1 shows the correspondences between the two systems.

Table 1.1. Navigation control program vs. finite state machine.

FSM	Navigation Control Program
states	operational modes
inputs	sensor measurements
outputs	robot actions
transition function	behavior activation

In controlling a mobile robot for straight-line motions, Crowley defined the state transition shown in Figure 1.3. In the **Hold** state, the robot waits for a command from a global path planner. Upon receiving a goal, the robot enters the **Decide** state. Depending on the angle difference between the goal and current heading, the robot enters either the **Turn** or **Move** state. The **Blocked** state is entered when the contact sensor on the robot is triggered. In this state, the local path planner determines the path for avoiding the obstacle. The **Wait** state is entered when the robot seems to be moving away from the goal position.

Fok and Kabuka employ a 7-state Moore machine for controlling an Automated Guided Vehicle (AGV) that is required to follow any route in a path network [Fok and Kabuka, 1991]. Nodes in the networks are marked with bar codes. The seven states are associated with the following actions:

- Following an edge,

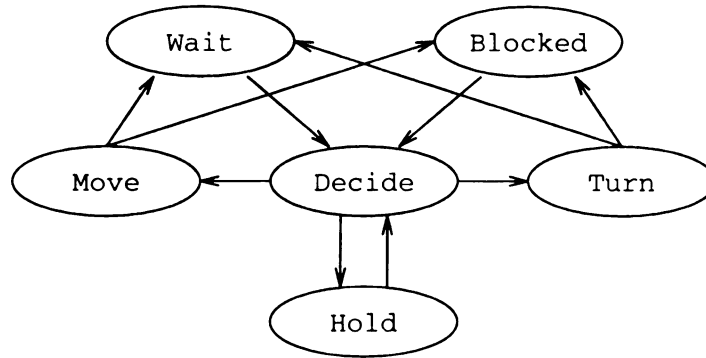


Figure 1.3. State transition diagram used in Crowley's mobile robot (reproduced from [Crowley, 1985]).

- Reaching an expected node,
- Leaving a visited node,
- Encountering an obstacle in the current direction,
- Reaching a nonidentifiable node,
- Reaching a destination node, and
- Reaching an isolated node.

Using a camera facing the floor, scene objects are classified into the following categories: obstacles, expected nodes, visited nodes, required node, destination node, nonidentifiable nodes, edges, and background (floor). The category of the scene object serves as an input for the finite state machine.

### 1.3.2 Robot Schema

Finite State Machines model a system by identifying its states, inputs, and outputs and characterizing how the inputs to the *entire* system affect the states. However, the effects of the input to a particular component of the system cannot be captured

by this model. Lyons and Arbib proposed a formal model for representing robot programs [Lyons and Arbib, 1989]. Their model, *Robot Schemas*, employs a nested network of computing agents called *schemas*. Each schema has a number of internal variables, input and output ports. When a schema is created, one can specify how the internal variables are initialized. A schema  $T$  with input ports  $i_1, i_2, \dots$ , output ports  $o_1, o_2, \dots$  and its internal variables initialized by the expression  $v$ , is written in the following Robot Schema notation:

$$T_v(i_1, i_2, \dots)(o_1, o_2, \dots).$$

Output ports of one schema can be connected to input ports of the other. This connection enables the schema to communicate to one another. A connection map describes how several schemas are interconnected via their ports. For instance, to denote that schemas  $\mathbf{A}_1, \mathbf{A}_2, \dots$  are connected as described in a connection map  $C$  one writes:

$$(\mathbf{A}_1, \mathbf{A}_2, \dots)^C.$$

The connection map  $C$  is written as a list of elements in the form:

$$(\mathbf{X}, p) \rightarrow (\mathbf{Y}, q),$$

which indicates that the output port  $p$  of schema  $\mathbf{X}$  is connected to the input port  $q$  of schema  $\mathbf{Y}$ . Each schema is associated with a *behavioral description* that defines its response during communication.

By connecting several schemas in the above manner, one can create a network of schemas, which is referred to as *assemblage*. An assemblage can be viewed as a single large schema whose behavior is determined by the behavior of the constituent schemas. Moreover, the input/output ports of the constituent schemas can be used

as the input/output ports of the assemblage. Hence, besides a connection map, an assemblage requires a *port equivalence map*  $E$  that specifies the relationships between the ports of the assemblage and those of component schemas. A port equivalence map is a list of elements in the form of

$$(\mathbf{X}, p) \rightarrow (q),$$

which indicates that port  $p$  of schema  $\mathbf{X}$  is equivalent to port  $q$  of the assemblage. With these notations, an assemblage  $S$  is then written as:

$$S_v(i_1, i_2, \dots)(o_1, o_2, \dots) = [A_1, A_2, \dots]^{C,E}.$$

Thus, to the outside world, a basic schema and an assemblage have a similar appearance, i.e., both of them have input and output ports, and their internal variables can be initialized with a given expression. However, a basic schema defines the computing behavior of the schema while an assemblage defines the instantiation procedure of the assemblage (how to create and connect the constituent schemas).

A robot task  $\mathbf{T}$  can then be represented as an assemblage of a sensory schema  $\mathbf{S}$ , a motor schema  $\mathbf{M}$ , and a task schema  $\mathbf{t}$ , and is written as:

$$\mathbf{T} = [\mathbf{S}, \mathbf{t}, \mathbf{M}]^C.$$

### 1.3.3 Petri Nets

Another model mentioned in the robotic literature is Petri Nets. Some researchers have even incorporated temporal information into Petri Nets and called them Timed Petri Nets. So far, Petri Nets have been mostly used for modeling manufacturing systems. Recently, Noreils employed the Pr/Tr Net, which is a Petri Net-based model,

for modeling the coordination of multiple robots [Noreils, 1993].

A Petri Net can be viewed as a graph whose nodes come in two different types: *places* and *transitions*. More specifically, a Petri Net consists of the following components:

- A set of places,
- A set of transitions,
- An input function that defines the input places of each transition, and
- An output function that defines the output places of each transition.

The input and output functions are usually represented by arcs connecting pairs of places and transitions. An input port  $p_i$  is connected to a transition  $t$  with an arc that goes from  $p_i$  to  $t$ . Similarly, an output port  $p_o$  is connected to a transition  $t$  with an arc from  $t$  to  $p_o$ .

We can put a number of *markings* inside each place. If the number of markings in a place  $p$  is at least as many as the number of outgoing arcs, then  $p$  is active. Additionally, a transition can be either *enabled* or *disabled*. A transition  $t$  is enabled if all its input places are active, otherwise the transition is disabled. An enabled transition can “fire”, causing the markings in all its input and output places to change.

## 1.4 Problem Definition and Approach

In this thesis, my interest is in the software subsystem of an indoor mobile robot navigation system. I believe that navigation is an important issue in mobile robotics research. A typical mobile robot application usually involves completion of several subtasks that can be either cooperating or conflicting. For example, when a robot is given a command to move in a straight line but it encounters an obstacle on its



path, it has to modify the current path and avoid the obstacle. Here, the first subtask (straight line motion) and the obstacle avoidance subtask are conflicting in the sense that one subtask gives commands that might be unexpected by the other. In this situation, arbitration of access control to the robot is required. In some cases, the commands for controlling the robot are not atomic, i.e., they consist of a sequence of interruptible computer instructions. Consequently, the current command executed by the robot can be interrupted by another command.

I have designed a control mechanism that will assist in the development of such an application. A task is usually associated with a user-process running on a computer. Thus, one should expect a number of user-processes running concurrently to achieve the common goal of interest.

In a system such as the one described above, process coordination and resource sharing are important issues. My approach is to provide a core system that employs a client-server model. Clients and servers are among the four basic types of processes in a distributed program [Andrews, 1991], the other two kinds are filters and peers. Due to the distributed nature of the processing, this approach is also referred to as *distributed robotics* [Gauthier *et al.*, 1987].

The core of the system consists of a set of servers that manages process coordination, resource sharing, access control, and information sharing. Here, resources refer to the hardware components on the robot. My scheme for controlling these components is to employ at least as many servers as there are sensors or hardware components. Thus, a single server is dedicated to controlling one hardware component. Hardware components are considered independent of each other, thus all servers can run concurrently. There are no dependencies among these servers. By using this scheme, a new component and its server can be added to the system easily. In this thesis, the peripheral-controlling servers will be referred to as *hardware servers*.

Besides the peripheral-controlling servers, this system is also comprised of a number of higher-level, task-specific modules that run concurrently. In relation to the servers, these modules will be referred to as clients or agents. Each higher level module might have a specific “mission” or “goal” to accomplish. Thus, the objective of the entire system is to satisfy multiple subgoals to achieve a common goal of the task being executed.

In a system where multiple concurrent modules are used to carry out the navigation task, the modules might need to communicate with each other. In order to facilitate cooperation among modules, I have designed a communication scheme between the modules and the servers, but not among the servers themselves. If direct intermodule communications were allowed, the number of interconnections among the modules would be numerous. My strategy is to allow only indirect communication among these modules via a common *data server* which accepts all queries or system information updates. To some extent, the data server can be considered as a simplified database server. Before two modules can communicate with each other, they have to agree upon the name and format of the shared data.

It is worth emphasizing that the data server and the hardware servers form the core of the system. The remaining modules in the system will be referred to as *clients*. A typical configuration of the system can be seen in Figure 4.3. All the servers are independent of each other, and so are the clients. The interconnection among the clients and the servers can be modeled by a *bipartite* graph, with one set of nodes containing all the clients and the other set of nodes containing all the server modules. Figure 1.4 shows the client-server interconnections in the system.

The system described above has been designed and implemented for controlling our mobile robot for hallway navigation. In the experiments, I utilize a coarse-level map of the hallway which contains locations of doors, corners, ceiling lights, walls, and elevators. My goal is to command the robot to move from one point to another



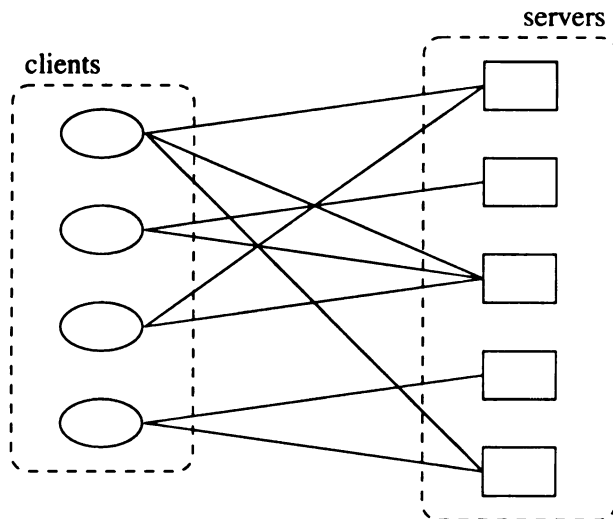


Figure 1.4. Client-server interconnections.

in the Engineering Building on the MSU campus. The initial and final locations can be either on the same floor or different floors of the building. In the later case, it is necessary for the robot to locate the elevator, enter it, and exit when the destination floor is reached. In my experiments, a human operator will help the robot to operate the elevator by pushing the buttons inside and outside the elevator.

I have adapted the *StickRep* representation for constructing our maps. The initial maps were developed by [Walsh, 1992], and I have enhanced and modified them to suit my requirements. A *StickRep* is an attributed edge graph and is a  $1\frac{1}{2}$ -dimensional representation of the robot's environment. An edge in the graph contains information about the type (door, elevator, wall, window), dimensions, and special markings (window openings, fire extinguishers, posters, etc.) of the object it represents. A node holds adjacency information about the incident feature edges, and contains adjacency angle between two edges and the node's identifier. Figure 1.5 shows a segment of the map.

Our mobile robot, RoME (Robotic Mobile Experiment), has evolved from a single-user DOS-based into a multitasking Unix-based autonomous platform. This evolution

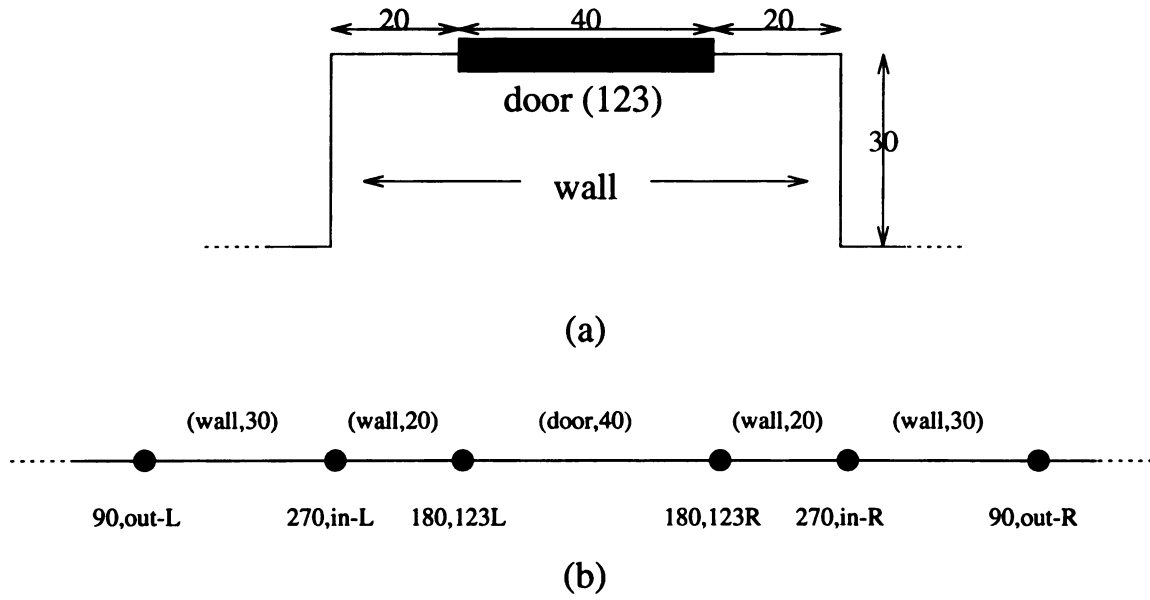


Figure 1.5. A StickRep representation (b) of a building structure (a).

provides a wider range of options for controlling the system as a whole. Currently, hardware components of RoME consist of:

1. A Sun SPARCstation 1 enhanced with 10 serial ports (2 builtins + 8 addons through the SPC board).
2. A 386SX laptop for operator control.
3. TRC LABMATE wheeled locomotion base, which can be controlled from the SPARCstation by sending commands via an RS-232 serial port.
4. Twenty-four ultrasonic sensors and eight infrared sensors attached to the TRC Proximity SubSystem, which is controlled from the SPARCstation via an RS-232 serial port. The maximum detection range of the ultrasonic sensors is 10 m. The infrared sensors return a binary information indicating whether an object is detected within a distance of approximately 76 cm from the sensor.
5. Two Panasonic GP-KR202 CCD color cameras with 6 mm lenses.

1

2

3

**1**

4

5

6

7

8

6. Two Sun VideoPix image grabbers.
7. A pan/tilt carousel for mounting the cameras.

## 1.5 Overview of the Thesis

The rest of this thesis is organized as follows. Chapter 2 discusses the issue of multi-sensor integration and the type of sensors that will be used in our research. Next in Chapter 3 I present a survey of control architectures used in robotics research. In Chapter 4, I discuss in more detail the Client-Server Control Architecture model adapted in this thesis. Chapter 5 describes the system developed for indoor navigation. Chapter 6 describes the experimental results in indoor navigation that I have obtained using the Client-Server Architecture model. Finally, I conclude with what I have learned from this research and mention areas of future research in Chapter 7.

## 1.6 Achievements in Mobile Robotics

Mobile robotics research is multidisciplinary, and the achievements in this area have come from researches in a number of other fields such as: robot vision, geometric planning, locomotion design, control, and sensory interpretation. In this section, the recent achievements in mobile robotics are identified. Most of the published works described below have been cited elsewhere in this thesis.

- Navigation algorithms for autonomous robots in both *known* and *unexplored* terrain of obstacles have been reported in the literature. For unexplored terrains, the navigation algorithm usually creates a map of the environment on the fly [Asada, 1988; Asada, 1990; Borenstein and Koren, 1991a; Bozma and Kuc, 1991; Kweon and Kanade, 1992; Moravec and Elfes, 1985]. The robot navigates using

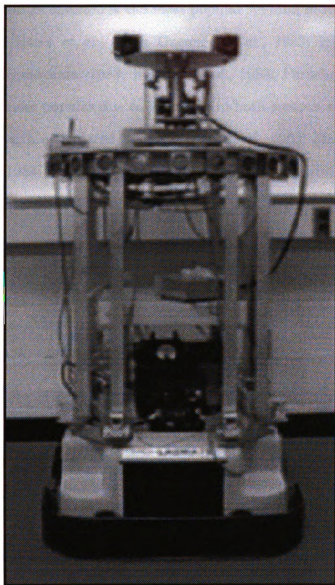


Figure 1.6. Front view of RoME.

a partial information and incrementally updates the map with the most current sensor data.

- Various locomotion methods for mobile robots have been reported in the literature. Wheeled locomotion is the most popular method of locomotion. Legged locomotion [Bares *et al.*, 1989; Ozguner *et al.*, 1983; Hirose *et al.*, 1983; Miura and Shimoyama, 1983; Raibert *et al.*, 1983; Furusho and Sano, 1990] is becoming more popular due to its utility in both smooth and rough terrains. Locomotion with articulated body [Maeda *et al.*, 1985; McGhee *et al.*, 1980; Ozaki *et al.*, 1983; Ohmichi, 1985; Hirose and Morishima, 1990] leads to mobile robots with a large number of degrees of freedom.
- Very small-sized robots have been successfully engineered at the MIT Artificial Intelligence Laboratory [Brooks, 1991; Flynn *et al.*, 1989; Connell, 1987].
- Faster processors and algorithms have enabled mobile robots to maneuver at a high speed. Some examples are: the vision-guided indoor mobile robot that is capable of navigating at a speed of 8-10 m/min [Kosaka and Kak, 1992], and CARMEL with its fast map building algorithm, HIMM, that enables the robot to maneuver at a maximum speed of .78 m/sec [Borenstein and Koren, 1991a]. The latter robot won the AAAI'92 Robot Competition [Kortenkamp *et al.*, 1993]. In the competition, the mobile robots were required to visit 10 poles while avoiding static and dynamic obstacles. Each pole was ten feet tall and three inches wide.
- In the AAAI'93 Robot Competition, a number of mobile robots showed their capabilities in three different events [Nourbakhsh *et al.*, 1993]. In the first event, the robots started inside an office approximately 4 by 5 meters in size. The robot had to find the way out of the office when one of the doors in the

office is opened and then reach a finish line while avoiding obstacles. In the second event, the robot started in an unknown position and orientation inside a realistic office setting. The robot must locate a marked coffeepot and deliver it to a particular office. In the third event, the robot had to rearrange several special boxes in a specified pattern.

- Progress on Autonomous Road Vehicles (ARVs) has also been reported in the literature [Masaki, 1992]. ARVs differ from Autonomous Mobile Robots in the sense that ARVs are computer-controlled cars. Research projects in this field include Harunobu-4 by Yamanashi University (Japan), Martin Marietta Autonomous Land Vehicle, CMU NavLab, ROVA by RARDE (United Kingdom), and OptoPilot of Volkswagen AG (Germany). The fastest ARV was designed and is operated by the Universität der Bundeswehr München (VaMoRs), which was reported to reach a maximum speed of 96 km/h on a newly constructed freeway and 50 km/h on campus roads.

## 1.7 Difficult Problems in Mobile Robotics

Despite the aforementioned achievements in mobile robotics, there are still some difficult problems to be solved. These problems are given in the following:

- **Fast and Robust Navigation.** The existing mobile robot navigation systems work well in only a few types of domain. So far, there is no general navigation algorithm that will perform well in *any* environment.
- **Exploration and Symbolic Map Construction.** Imagine that we have a “smart” mobile robot operating in an indoor environment. The robot is given a finite amount of time to *explore* its environment and to create a *symbolic* representation of it. The task of the robot is to *extract* features from the sensory data,

store them in the symbolic map, and derive the topological relationship among the features. After the map construction is completed, then the robot must be able to navigate to any location using all the information it has accumulated.

- **Nanorobots.** The advent of *nanotechnology* opens up a new era of minuscule mobile robots. Although there have been some successful experiments in fabricating microdevices, these experiments are confined to making simple actuators or mechanical systems. We need a few more leaps before we will manage to build a functional *nanorobot*. Imagine a “teleoperated” nanorobot that can crawl inside your blood vessel! [Lucky, 1990]
- **Household Robots.** At the current stage, mobile robots are not ready to be servants in our homes. Making robots that can do laundry, dishwashing, cooking, lawn mowing, vacuum cleaning, etc. is a non-trivial task.
- **Space Robots.** The Jet Propulsion Laboratory (JPL) has been active in research on using robotics for space missions. The two categories of robot tasks needed in space missions are assembly and sample handling. Several papers describing the progress of the JPL Rover have been published [Dobrotin and Scheinman, 1973; Lewis and Bejczy, 1973; Dobrotin and Lewis, 1977; Miller, 1977]. However, to date, no autonomous robots are being used in any space missions.

In addition to the aforementioned problems in mobile robotics, additional problems can be found in the literature. Brady listed 30 problems that need to be solved in robotics [Brady, 1989]. Specifically, in the context of robot mobility, he mentioned the following four problems:

**P1 System Architecture for Mobility.** This issue relates to the determination of appropriate hardware and software architecture for multisensor, purposive,



mobile vehicle.

**P2 High Bandwidth Control.** As the sensors used in the robot system produce more information at much faster rates, high-bandwidth controllers are required.

**P3 Sensor-based Steering and Foot Placement.** This problem poses the issue of making the robot able to judge the quality of the terrain and thus choose the terrain locally.

**P4 Energy and Legged Locomotion.** For legged robots, how to change the mode of locomotion from one to the other.

Moreover, in the context of system integration, the following two problems were mentioned:

**P5 Communicating Processes.** This issue relates to determining the type of asynchronous software architecture that is most appropriate for a given project.

**P6 Distributed Hierarchical Organization.** Mapping asynchronous software architecture onto networked hardware architecture is a difficult problem.

## 1.8 Contributions of the Thesis

The contributions of this thesis have been:

1. Design, development, and implementation of the Client-Server model using an object-oriented approach. This model provides a framework for controlling a robot in a distributed manner, i.e., using a number of processors connected through a network of computers. To my knowledge, this is the first attempt of applying such a client-server model for robot navigation. The model developed in this thesis can be used for controlling multiple cooperating robots as well.

2. Successful design and implementation of a robot control model that enables resource access sharing, data sharing, and event synchronization in a distributed manner.
3. The implementation of C++ classes for the client-server interaction, low-level access to various components of the robot, process creation and destruction, interprocess synchronization in order to provide high-level abstraction and information consistency.
4. Development of an algorithm for transforming the hierarchical and Colony-style Architectures into the Client-Server Architecture. More specifically, an algorithm for determining the priorities of the modules in a Colony-style network containing both suppressor and inhibitor nodes is presented.
5. A functional indoor mobile robot navigation system with an ability to operate in a large building (approximate size of the floor plan is  $136 \times 116$  square meters). The robot is capable of navigating at 4.7–17.88 m/min and reaching goal positions within a proximity of 1 m as demonstrated by dozens of formal test runs.

I believe that these contributions bring us one step ahead in solving Problem P6 described in Section 1.7. The Client-Server Architecture described in this thesis facilitates the mapping of software architecture onto a networked hardware architecture.

## 1.9 Summary

In this chapter, a brief history of robotics has been discussed with emphasis on a number of milestones in the development of robots from *automata*. The differences between manipulator arms and mobile robots are also examined. Moreover, a number of major research areas in robotics are identified. This list is not at all comprehensive

and there is some overlap among those research areas. The research described in this dissertation focuses on construction of a flexible control strategy for robot navigation tasks. I propose the Client-Server Control Architecture as the standard for such a control strategy.

# CHAPTER 2

## Sensors

**sen·sor** [L sensus, pp. of sentire to perceive]: a device that responds to a physical stimulus (as heat, light, or a particular motion) and transmits a resulting impulse (as for operating a control). \*

When performing a given task, a robot uses its sensors to determine its operational state as well as the state of its environment. For instance, when a mobile robot is given a command to turn counter-clockwise by 90°, the robot controller has to determine if the required amount of rotation has been reached. When the mobile robot must keep a minimum distance to any object in its work space, a means of distance measurement must be incorporated into the system. Grasping objects and moving them from one place to another requires some mechanism for determining if the robot manipulator arm has grasped the object firmly, and if the arm controller has applied enough force to pick up the object of interest.

The above examples show that sensors are important in providing information to the robot when accomplishing a given task. Sensors that are used to obtain the operational state of the robot are usually referred to as *internal sensors*. Sensors that are used to obtain the state of the environment are referred to as *external sensors*. One can also say that internal sensors perform regulatory functions while external

---

\*Electronic version of Webster's 7th Collegiate Dictionary.

sensors perform exploratory functions. In this chapter, the word “sensors” should be interpreted as external sensors, unless noted otherwise. Within this context, the task of a sensor is to *observe* the environment and provide necessary information to the robot.

A partial list of sensors that have been used in robotics applications includes:

- Internal: potentiometer, optical encoder, shaft encoder, resolver, gyroscope, digital compass, slip sensor.
- External:
  - Range: laser range finders, proximity sensors, radar sensors.
  - Acoustic: ultrasound sensors, microphones.
  - Tactile: touch sensors, sensitive “skin” sensors, piezoelectric sensors.
  - Vision: photodetectors, cameras.
  - Other: magnetic sensors, load sensors.

In my experiments, I have used ultrasonic sensors, CCD cameras, infrared proximity sensors, touch sensors (contact-sensitive bumpers on the robot), and wheel encoders on the robot. The main advantage of these sensors is their low cost-to-performance ratio. The ultrasonic sensors are used for local navigation, obstacle avoidance, and building local maps of the hallway. The range information returned from a single scan of the ultrasonic sensors is used for determining the open space in front of the robot. The local maps of the hallway are created by accumulating the ultrasonic range information while the robot is in motion. The two CCD cameras are used for detecting ceiling lights and door number plates in the hallway. The infrared proximity sensors are used for a simple obstacle avoidance capability, i.e., for stopping the robot whenever the infrared proximity sensor detects an obstacle. The contact sensitive bumpers guarantee a safe operation for both the robot and humans. Finally,

on many different occasions, the robot needs to obtain its actual position, heading, and speed.

The obstacle avoidance capability of our robot is achieved by integrating the information from the ultrasonic sensors, infrared proximity detectors, and heading direction obtained from the robot. The ultrasonic data are used to determine the open space “in front of” the robot. The heading direction guides the algorithm in determining which ultrasonic sensors to read while it searches for the open space. The navigation algorithm uses information from the cameras and the robot position and heading for registering the actual robot position within the stored map.

## 2.1 Redundant Sensing

It is very common to employ multiple sensors in a given robotic application. The information obtained from different sources is integrated systematically so as to obtain useful and robust data. Use of multiple sensors is often considered as a requirement for building intelligent and autonomous robots. Sometimes, the term *redundant sensing* is also used to indicate multiple sensing. Moreover, in this context, redundant can mean [Brooks, 1988]: (1) duplicate (more than one sensor of the same type), (2) multi (more than one sensor of different types), or (3) distributed (network of sensors). The information provided by a single sensor is often incomplete and contains a certain amount of uncertainty. By employing multiple sensors, information from a number of different sources might be used to reduce this uncertainty. Of course, a trade-off between complexity and uncertainty has to be considered. When there are too many sensors used in the system, the fusion process becomes more complicated.

Some motivations for using redundant and/or multiple sensors are:

- *Reliability.* The system will not fail if one sensor is malfunctioning.
- *Error Detection.* A faulty sensor can be detected if its output does not conform

with the others.

- *Low Cost-to-Performance Ratio*. Instead of using a few high quality sensors, the system can utilize many less accurate and low-cost sensors and still achieve better performance.
- *Distributed Processing*. Each sensor is controlled independently by a dedicated processor. de Almeida *et al.* showed an example of such a system [de Almeida *et al.*, 1988].

Despite the advantages we gain from redundant sensing, there are still several problems to be addressed in integrating information from multiple sensors. The main issues are: (i) sensor fusion, (ii) conflict resolution, and (iii) information enhancement or update. One aspect of sensor fusion is “polymorphic” communication, in the sense that each sensor produces a piece of information in its own “language”. Before fusion can occur, all this information has to be converted to a common language. Usually, this conversion involves some geometric transformations such as rotation, scaling, etc. Luo and Lin distinguished between sensor fusion and integration by emphasizing this aspect [Luo and Lin, 1988]. For instance, consider the situation where a range detector and a camera are used as sensors, and data points of the range detector are to be registered with the intensity values obtained from the camera. The geometric transformation involved in this example is a change of coordinate systems, i.e., transformation from both the range and camera coordinate system to the global coordinate system.

Tou has cited a number of sensor fusion models [Tou, 1988]:

- *Boolean fusion*. This model simply says that for any two redundant sensors, the fused data value is obtained from the sensor whose reliability index is higher. This scheme is similar to a *winner-take-all* strategy.

- Probabilistic fusion. Evidence from multiple sensors is combined using Dempster-Shafer or Bayesian approach.
- Markov Renewal Analysis. Markov chains are used for modeling the addition or removal of sensors in the system. In the model, a state represents the number of sensors that are functioning properly, whereas a chain represents the state transition probabilities. A sensor is removed from the system if it is not functioning properly. When the faulty sensor is fixed, it is added back to the system.

Hackett and Shah identified several methods of combining multiple data sources [Hackett and Shah, 1990]:

- *Deciding*. Only one source is used during the fusion process. Which sensor to use is based on confidence measures of the sources.
- *Guiding*. Information from one or more sources is used to focus the attention of the others.
- *Averaging*. All sources are used in the fusion process and weights are assigned to each source depending on its reliability measure.
- *Bayesian statistics*. One or more sources provide prior information to the others.
- *Integration*. Each source provides a certain type of information specific to a given subtask which is part of a larger task. The main principle of this method is the delegation of a subtask to a single source.

Tang and Lee proposed a graph-based approach for sensor fusion [Tang and Lee, 1990]. Geometric features obtained from each sensor are represented by a *Geometric Feature Relation Graph* (GFRG). Three types of features are used in the representation: (1) the *sensor* feature, which is the local coordinate frame of each sensor, (2) the *null* feature, which is the global coordinate frame, and (3) the *normal* feature, which



is the actual geometric feature of the object being observed. The nodes of the graph represent the geometric features and the arcs represent the geometric or topological relationship among the features. In Tang and Lee's experiment [Tang and Lee, 1990], only adjacency is used as the topological relation. To facilitate the fusion process, the notion of a feature-associated coordinate frame is used. Using this coordinate frame, information about a particular feature can be fused without first transforming it to a common coordinate frame.

Unlike the other approaches that fuse sensory measurements, Krzysztofowicz and Long's approach fuses the detection probabilities of the sensors [Krysztofowicz and Long, 1990]. The processing model they employed consists of three major components: sensor, forecaster, and decision-maker. Each sensor produces observation vectors to be used by the forecaster. Based on the information obtained from these observation vector(s), the forecaster computes the detection probability, which in turn is used by the decision-maker to determine what action(s) to take. Depending on where the fusion takes place, there are three possible schemes of fusion:

1. *Observation fusion.* A number of observation vectors are fused to a single piece of information to the forecaster.
2. *Decision fusion.* The actions chosen by all the decision-makers are fused together.
3. *Detection probability fusion.* The detection probabilities produced by the forecasters are fused into a single probability measurement and is passed to the decision-maker.

Note that this scheme can only be used for binary sensors, and therefore it is also called a *multisensor detection* scheme.

## 2.2 Sensor Model and Uncertainty

A common approach for modeling sensor measurement is to use Gaussian distributions. The parameters of the distribution are estimated via a calibration process. This is necessary because measurements obtained from a sensor are not always accurate. To account for this inaccuracy, mathematical models are used for describing the behavior of the sensor. Specifically, there are two models used for this purpose. One model describes *how* the sensing process takes place, while the other describes *what* object is being sensed [Hager, 1990]. Modeling sensor uncertainty is also important since it provides the information about the influence of uncertainty on the fused sensor data.

The mathematical tool commonly used in modeling uncertainty is the Dempster-Shafer Theory. The theory differs from the Bayesian statistic in the sense that a probability mass function can be assigned to a set of propositions, as opposed to a single proposition only. Also, the interval of uncertainty can be modeled by the *support* and the *plausibility* functions. In Bayesian approach, these two functions have the same value, and consequently the length of the uncertainty “interval” is zero.

Durrant-Whyte viewed sensing in a multi sensor system as the observations made by the sensors on its environment [Durrant-Whyte, 1988]. Using this approach, an observation ( $\mathbf{z}_i$ ) is modeled as a function ( $\eta_i$ ) of the state of the sensor ( $\mathbf{x}_i$ ), the state of the environment ( $\mathbf{p}_i$ ), and the actions ( $\mathbf{a}_j$ ) taken by all other sensors within the system. Formally, if there are  $n$  sensors in a system, the model for sensor  $i$  can be written as:

$$\mathbf{z}_i = \eta_i(\mathbf{x}_i, \mathbf{p}_i, \mathbf{a}_1, \dots, \mathbf{a}_{i-1}, \mathbf{a}_{i+1}, \dots, \mathbf{a}_n).$$

Moreover, Durrant-Whyte showed how to decompose the sensor model into three components: the *observation*, *dependency*, and *state models*. By using an organization

theory approach, a multi-sensor system is viewed as a team of decision makers in which each team member can exchange information with the others. Sensor uncertainties are modeled by stochastic geometry.

Luo and Lin described a model for estimating fused data from a number of sensors measuring the same object property [Luo and Lin, 1988]. The probability density function associated with each sensor was obtained from a calibration process. Using these probability density functions, the *distance* between any two measurements can be computed. The distance  $d_{ij}$  between sensors  $i$  and  $j$  is defined as the area between measurements  $x_i$  and  $x_j$  under the probability distribution function of sensor  $i$ . The Mahalanobis distance was also used as a metric for computing the distance. The adjacency matrix representing a directed graph of the measurements was obtained by thresholding the distance matrix. Two adjacent nodes in the graph correspond to two *compatible* sensor readings. Once the completely connected subgraph is identified, spurious sensor readings (outliers) can be identified by searching for nodes whose distance to the completely connected subgraph exceeds some threshold. An optimum estimation of sonar readings was calculated from the connected components. Using this approach, a malfunctioning sensor can be detected if its reading is very far from the estimated value.

Hager used the *state space* approach for modeling the information gathering process [Hager, 1990]. At any given time, the information processing system is described by its *information state*. The information or data obtained from the sensors modify the (information) state of the system. The objective of sensing could be either to reach a goal state or to maintain a given state for a period of time. In other words, the sensing is task-oriented. Moreover, he decomposed the sensing model into two major components: sensing geometry and sensing error. The goal is to minimize the error using a task-oriented (active) sensing model. Sensing geometry describes the

relationship between the unknown parameters and the observable values of the environment. Using this model, Hager also described the notion of system observability. The sensing system is observable when the unknown parameters (or states) can be determined from a finite number of observations. These unknown parameters could be: location or size of the object being observed, its color, the lighting conditions, etc. Besides the unknown parameters, sensing geometry could also include: *calibration parameters* and *control parameters* (i.e., parameters for positioning the sensors in an “active sensing” approach). The uncertainty in sensing can be either deterministic or non-deterministic. Given the values of calibration parameters, the effect of deterministic uncertainty (like calibration or statistical error) of the observed values can be accounted for. On the other hand, for non-deterministic uncertainties, only the range of uncertainty could be estimated and the error model is given using tolerance sets.

## 2.3 Computer Vision

One of the important research problems in computer or robot vision is the recovery of 3D information from 2D intensity images. For object recognition applications, one of the goals is recovery of the shape of the object found in the image. Given a single intensity image, computer vision researchers have been approaching this problem by *shape from X* paradigms, such as: shape from shading, shape from texture, shape from contour, shape from focus, etc. For mobile robot navigation applications, the recovery of depth information is more important than exact object shape recovery. This section explores the role of vision in depth recovery.

Depth information can be acquired either by *active* or *passive* sensing model. Some of the popular methods for obtaining depth information via active sensing are: laser range finding, ultrasonic sensing, structured light, etc. Passive sensing methods

are: optic flow, stereo vision, and axial motion stereo.

The name “axial motion stereo” is used because the motion of the observer (the robot) coincides with the optical axis of the camera. In practice, this is not always possible. The depth recovered from axial motion stereo depends on the distance traveled by the robot between the two time instances when the image frames were taken. This distance also determines the accuracy of the depth obtained with this method. Small interframe distances will result in less accurate depth information. But, too large a distance may cause the image matching to be impossible because there is no overlap between successive image frames. Besides, over a longer travel distance, the robot’s wheels are likely to slip, resulting in erroneous odometry readings.

Besides axial motion stereo, stereo vision is also used for estimating depth. In this method, depth information is recovered from multiple (typically two or three) 2D images, which are usually taken by multiple cameras placed in a certain arrangement or by a single camera translated (and rotated) to different positions to obtain multiple views of the same 3D scene. The common practice is to use a pair of cameras and, for this reason, stereo vision is also known as *binocular vision*. Other names that have been used for this approach are: stereopsis, stereo or stereoscopic vision, binocular stereo, binocular vision, and binocular stereoscopic vision. Recently, there have been some attempts to recover depth using three cameras. This approach is known as trinocular stereo [Ayache, 1991; Ayache and Lustman, 1991].

In general, the positions of the two cameras could be arbitrary, in the sense that the relative position and orientation (or pose) of both the cameras can be specified by six degrees of freedom. However, the choice of the relative pose imposes some geometric constraints which might simplify the depth recovery process. For instance, in the special configuration where both the optical axes are parallel, the matching needs to be performed only along the horizontal scan lines. The matching in the non-parallel configuration can be transformed to this special configuration by a process

called *rectification*, which involves a computation of new perspective matrices using the calibration parameters (optical centers and perspective transformation matrices) of the non-rectified configuration.

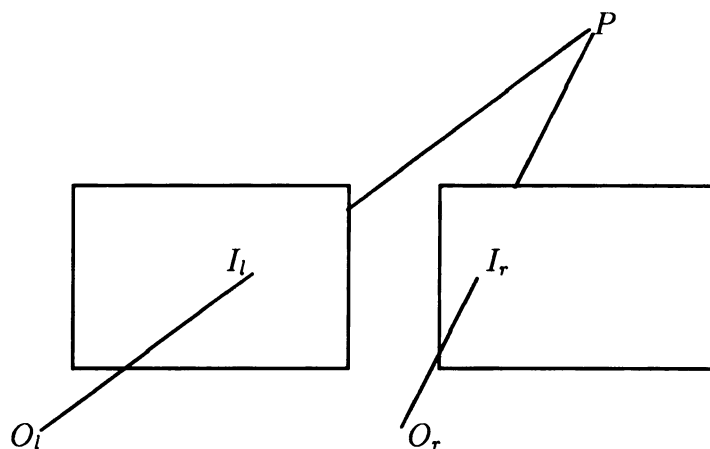


Figure 2.1. Stereo imaging geometry.

### 2.3.1 Stereo Matching Algorithms

In stereo vision problems, there are three major tasks to be solved: *feature extraction*, *feature matching*, and *depth recovery*. In their survey paper, Barnard and Fischler cited three additional steps: *image acquisition*, *camera modeling/calibration* and *interpolation* [Barnard and Fischler, 1982]. Once the correspondence or matching problem is solved, the computation of depth is relatively simple via the triangulation principle. The recovered depth can be either *absolute* or *relative*, depending on the availability of camera imaging parameters recovered through camera calibration. It is worth noting that solving the image correspondence problem is the hardest task of a stereo algorithm.

In general, solving the correspondence problem requires finding the correct match

in a 2D search space. The dimension of this search space can be further reduced by imposing the *epipolar constraint* in the matching process. The epipolar constraint simply says that given a point  $P$  in 3D space (Figure 2.1) whose projection in the left image is  $I_l$ , the match of  $I_l$  can be found along a straight line in the right image. This line is known as the *epipolar line* and is the intersection between the epipolar plane and the image plane. The epipolar plane itself is a plane that spans  $P$  and both the optical centers ( $O_l$  and  $O_r$ ).

### 2.3.2 Stereo Vision in Mobile Robot Applications

Moravec was one of the first researchers to put a stereo vision system on a mobile robot [Moravec, 1980]. Depth information obtained from stereo vision provides an alternative to active range measurement. Nishihara and Poggio cited three types of 3-D information that may be derived from stereopsis [Nishihara and Poggio, 1983]:

- *Volume Occupancy.* Given a specified volume of space, stereo vision can determine whether any object exists in the volume.
- *Surface Location.* Given an estimated position of a surface patch, stereo vision can compute the orientation, range, and size of the patch.
- *Edge Location.* Given an estimated position of an edge, stereo vision can compute its position in space.

The above types of 3-D information are presented in increasing level of detail and complexity. For instance, when a stereo vision system is used for obstacle avoidance applications, information about volume occupancy is sufficient, whereas in object tracking applications, information about edge location might be necessary.

An interesting approach to stereo vision has been proposed in [Storjohann *et al.*, 1990] where an *inverse perspective mapping* was employed. Inverse perspective

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000

2

As  
em  
a w  
re  
tr  
ech



mappings are usually applied to images to undo the perspective distortion with respect to a certain planar surface whose orientation is known. By applying this technique to a pair of images obtained from stereo cameras tilted down to view the floor, the perspective distortions of the floor were eliminated. The resulting images show the floor as tiles of squares as if they were viewed from overhead. By subtracting one image from the other, existence of obstacles on the floor can be easily detected.

In the dynamic stereo vision approach, depth information is usually obtained using knowledge of the observer's motion. However, using an object-centered fusion scheme, Moezzi *et al.* have shown how one can combine several local spatial maps of the environment without using the knowledge of observer's motion parameters [Moezzi *et al.*, 1991]. The main idea in their approach is the computation of the *relative distance* between objects.

Matthies and Okutomi have proposed a *bootstrap* algorithm that combines narrow and wide baseline stereo to utilize the strengths of both approaches [Matthies and Okutomi, 1989]. The depth map obtained from a narrow baseline stereo was used to constrain the matching process in the wider baseline stereo. More formally, the matching problem was formulated using a Bayesian approach where the prior probability was provided by the narrow baseline stereo.

## 2.4 Sonar

Another method for obtaining range measurements from the robot's environment employs ultrasonic sensors (sonar). Range measurements can be obtained by sending a wave train (pulse) of a very high frequency sound wave and then detecting the echo reflected back to the sensor. Usually, the sound wave is generated by a piezoelectric transducer. The time between the transmission of the pulse and the reception of an echo is called the *time of flight*. In practice, only the time to the first echo whose

amplitude exceeds a certain threshold is measured.

The characteristic of the echo varies depending on the type of the surface reflecting the ultrasonic pulse. A surface is considered *specular* (mirror-like) if its roughness is much smaller than the pulse wavelength [Bozma and Kuc, 1991]. When a sound wave hits a specular surface, there will be no strong reflection from the surface to the ultrasonic sensor, thus making it “invisible” to the ultrasonic sensor. Usually, this situation can be overcome by using a higher ultrasonic frequency, and hence a shorter wavelength. Typical Polaroid sensors use a frequency of 50 kHz. The distance  $z$  from the sensor to a non-specular surface is equal to half of the distance traveled by the pulse from the time it is transmitted until the echo is received at time  $t_0$ . More formally,

$$z = \frac{ct_0}{2},$$

where  $c$  is the speed of sound in the medium (typically air or water).

Ultrasonic sensors provide good depth measurements, but poor angular resolution. This is mainly due to the wide beam angle ( $12^\circ$ – $17^\circ$  for Polaroid sensors) used in most ultrasonic sensors. The beam angle is also determined by the wavelength of the sound wave and the diameter of the transducer. If  $\theta$  is half of the beam angle,  $\lambda$  is the wavelength of the sound wave, and  $D$  is the diameter of the transducer, then the following relation holds:

$$\sin \theta = \frac{1.2\lambda}{D}.$$

An echo received from an object at distance  $d$ , might come from *any* point in an arc of radial distance  $d$  from the sensor and of width  $\theta$  degrees. There are a number of different approaches to interpret this data. Moravec and Elfes employed a paraboloid for modeling both the radial and angular sonar readings [Moravec and Elfes, 1985]. Elfes employed a Gaussian model [Elfes, 1989]. Borenstein and Koren used only one reading at a normal angle to the sensor [Borenstein and Koren, 1991a]. One could

of  
are  
Can

also model all the readings in the arc as equally probable [Beckerman and Oblow, 1990].

## 2.5 Infrared Proximity Detectors

In some situations, we need to detect only the presence, not the exact location, of an obstacle in the vicinity of the robot. Proximity detectors give this type of information. A typical infrared proximity sensor uses a light whose frequency is just below the visible light, usually around 880 nanometers in wavelength. The sensor, consisting of an infrared emitter and detector, operates by detecting the reflection of its own light from objects. A proximity detector is a binary sensor, i.e., it returns only “on” or “off” values.

## 2.6 Summary

In this chapter, the role of sensors in robotics applications has been described. In my experiments, I used cameras, ultrasonic sensors, infrared proximity detectors, and the robot’s internal sensors for obtaining the position, heading, speed, and bumper status of the robot. Internal sensors were distinguished from external ones. In order to build a fault-tolerant system, the redundant sensing approach is commonly employed in robotics applications. A number of different approaches to sensor fusion and sensor modeling were also given. The role of vision has been explicitly emphasized because of its extensive use in many robotics research projects. In addition, ultrasonic sensors are also of interest because they provide a low-cost method of range detection that can be used in obstacle avoidance tasks.



# CHAPTER 3

## Control Architectures

Sensing, planning, and control are three main design issues in autonomous robot systems. From one perspective, they can be considered as the input-process-output model of a system. In robotics research, there has been a significant amount of time and effort devoted to each one of these areas.

As the tasks given to a robot become more difficult, the planning algorithm needed to carry out the task becomes more complex accordingly. Managing such a complex planning algorithm to achieve a good working system is an important issue. The common approach in reducing the complexity of a large system is to decompose the system into a number of smaller modules, each one performing a specific task. The method of building such a large system from smaller modules is sometimes referred to as *control architecture*.

The domain of discourse of this chapter is related to the research in control architecture. Control architecture is an area of active research. Some of the early work in this area was published in the late 1980's [Kaelbling, 1987; Wong and Payton, 1987; Kadonoff *et al.*, 1986]. To date, there have been five workshops held to discuss this issue. Three of the workshops, held annually since 1990, were sponsored by the U.S. Department of Defense (DOD) and were aimed at specifying control system architecture for the first and second generations of DOD's unmanned ground vehicles (UGV).

S

T

pt

st

adj

The other two workshops were held in association with the IEEE Conference on Intelligent Control. The first DOD-sponsored workshop was aimed at defining the UGV specifications and identifying its impact on design requirements for intelligent control architectures. The second workshop was aimed at identifying the technical options and viable approaches to real-time intelligent control architecture. The goal of the third workshop was to draft recommendations for specifications of the control system architectures.

Despite the immaturity of the area of control architecture, a number of different architectures have been proposed by robotics researchers. At present, there are no rigorous methodologies for designing a control architecture; design, implementation, and testing are done in non-standard manners. On the other hand, some efforts for formalizing a computational model for robot programming have been undertaken [Lyons and Arbib, 1989; Steenstrup *et al.*, 1983]. Still, no quantitative study which compares the performances of various control architectures is available. Also, since there are no common terminologies used by the researchers in this field, different names might be used for referring to the same entity.

The existing control architectures can be classified into four different categories [Quintero, 1991]: multilevel, connectionist, hybrid, and toolkit-based. In the following sections, these categories are described and some examples of the control architecture in each category are given.

### **3.1 Multilevel Architecture**

The control architectures in this class are characterized by a hierarchical design approach, where a task is performed at a number of levels by decomposing the task into smaller subtasks at each level. Typically, one level can communicate with its two adjacent levels. Commands flow from higher to lower levels, and data flow the other



way around. The highest level typically consists of coarse-grained processes, while the lowest level consists of fine-grained processes.

Some advantages of multilevel architectures are:

- Provide a systematic approach for task executions.
- Top-down refinement is the paradigm used by most computer programmers, therefore writing programs in this system does not require the programmers to think in a different way.

Some disadvantages of multilevel architectures are:

- There are no rigorous methodologies for determining how to decompose a task into its constituent subtasks.
- The number of levels in the system is determined in an ad hoc manner.
- Higher levels depend on the result of operation of the lower level. If at some point lower levels fail, higher levels might not be able to complete the operation successfully. The failing level might become the single-point failure of the system.

Some examples of multilevel architecture which have been implemented are given below.

### 3.1.1 NASREM

NASA and National Bureau of Standards have defined a hierarchical control architecture called NASA/NBS\* Standard Reference Model (NASREM) for the NASA Space Station IOC Flight Telerobot Servicer [Albus *et al.*, 1989]. Each layer in the

---

\*The National Bureau of Standards (NBS) is now known as National Institute of Standards and Technology (NIST).

hierarchy performs a mathematical transformation of its input to its output. The current specification defines six layers of modules partitioned into three sections for controlling a telemanipulator arm. The transformation between physical coordinates and the robot joint coordinates is performed at layer one (the lowest layer). Layer two deals with mechanical dynamics. Layer three handles the obstacle avoidance. At level four, the robot tasks are transformed into movements of manipulator arm. Level five performs the sequencing and scheduling of tasks. Level six, the highest level, performs the higher level of scheduling, resource assignment, part routing, and object grouping.

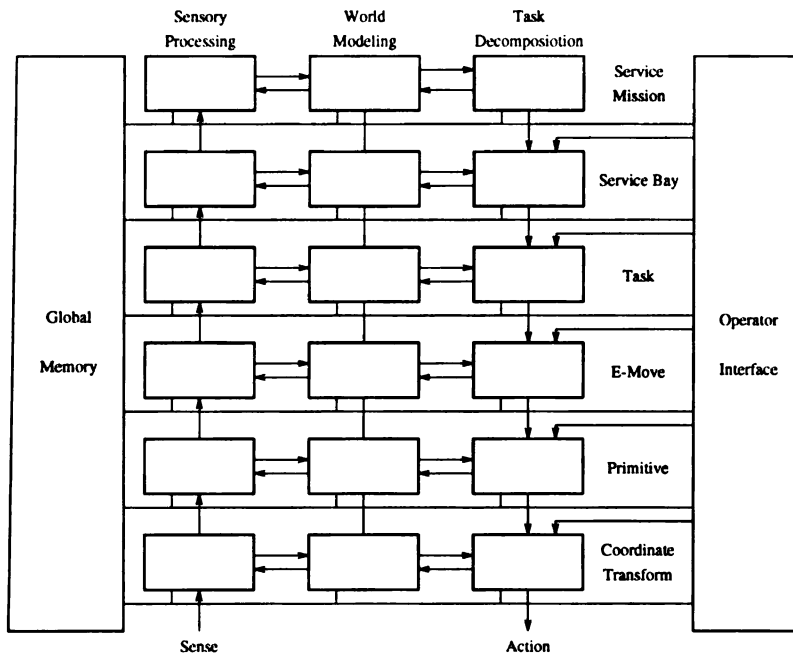


Figure 3.1. The NASREM Architecture for telerobots.

The three horizontal partitions defined in the architecture are: Task Decomposition, World Modeling, and Sensory Processing. Actually, these partitions correspond to the sense-plan-action cycle model commonly used in robotics and hence each layer in the architecture performs the sense-plan-action cycle independently. Figure 3.1

shows the NASREM architecture.

The Task Decomposition modules consist of assignment modules, planner modules, and executor modules. Upon receiving a goal task, the assignment module decomposes the task into both temporal and spatial subtasks to be executed by the planner modules. The spatial decomposition of a task causes a number of planner modules to run concurrently. On the other hand, the temporal decomposition causes each planner module to control a sequence of actions/subtasks, each assigned to an executor module.

The World Modeling modules employ geometric models for representing the robot workspace, update the state variables, generate predictions, and compute evaluation functions based on planned actions. Here, the “world model” refers to the system’s best estimate of the state of the world. The information about the state of the world, maps, lists of objects and events, are stored in a global memory.

The Sensory Processing modules deal with the processing of sensory data such as pattern recognition, event detection, filtering, and integration of sensory data over space and time. These spatial and temporal integration of sensory data can be viewed as a fusion of information from multiple sources over a time interval.

The modules in the architecture can communicate either horizontally or hierarchically. Commands and status feedbacks are communicated hierarchically, while data are shared horizontally. The information flow on the horizontal communication channels is greater than that of the hierarchical channels.

### **3.1.2 ARTICS**

Another standard model proposed by NIST is ARTICS (Architecture for Real-Time Intelligent Control Systems) [Albus *et al.*, 1991]. This model closely resembles the NASREM architecture and is meant to provide guidelines for implementing real-time and intelligent control systems for robotics applications. In addition to the

multi-layer hierarchy architecture defined in NASREM, ARTICS also specifies the communication speed requirement of each layer and describes the possible hardware implementation of each level. Based on the communication speed requirement, the six layers in NASREM (hence in ARTICS) can be organized into three levels. The top level whose communication speed requirement is the slowest, consists of a number of workstations connected via ethernet. The communication requirement at this level is on the order of 1 second. Specifically, this means that the operating system used at this level must be able to give a response time on the order of 1 second. The middle level of ARTICS consists of single-board computers, memory boards whose speed requirement is within the range of 10 ms–1 sec. The lowest level consists of special purpose hardware and has the communication speed requirement on the order of 10  $\mu$ sec–10 msec.

### 3.1.3 Task Control Architecture

Simmons *et al.* proposed a general-purpose process control mechanism for mobile robots. The mechanism, called *Task Control Architecture* [Simmons *et al.*, 1990], provides facilities for managing concurrent tasks that control the robot. Application of the system in controlling a legged robot is discussed in [Lin *et al.*, 1989; Simmons, 1992].

The architecture is designed specifically for robots that have multiple goals to achieve with a variety of strategies to achieve them. Constructed on a layered system, the architecture provides mechanism for [Fedor and Simmons, 1991]:

- Message passing,
- Resource management,
- Task management,

- Event monitoring,
- Exception handling, and
- “Wiretaps” construction.

## Message Passing

The interprocess communication mechanism operates by passing messages through a central server. Using this approach, one can build a number of modules that communicate with each other. Each module in the control system must connect to the central server before communication with the other modules can be established. Also, modules must register the types, message format, and the message handlers defined in the module. The Task Control Architecture provides a mechanism for connecting to the central server [Simmons *et al.*, 1992]. The types of messages supported by the architecture are: *query*, *command*, and *constraint* messages. A module that sends a *query* message expects a response from the receiver of the message. Sending a *command* message to a module makes that module perform an action. *Constraint* messages expect neither response nor action from the receiver. A typical configuration of a system built using the TCA architecture is given in Figure 3.2.

## Resource Management

In TCA terminology, a *resource* is a collection of message handlers that are related to a common task. For instance, one can define a resource that serves all movement-related procedures of a mobile robot. Another way of viewing this idea is to consider a resource as a module. All messages sent to a resource (i.e., to the handlers within the resource) are queued in a common message queue. TCA provides mechanisms for building, locking, and reserving a resource. If a resource is *locked*, no message can be sent to it. On the other hand, if a resource is *reserved* for a module, then that

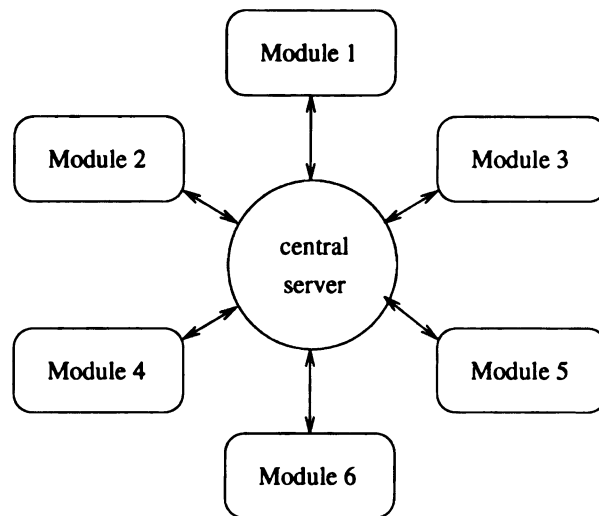


Figure 3.2. An example of a system using the TCA architecture.

resource will accept only the messages from that module.

### Task Management

It is a common practice in software design to decompose a problem into a number of smaller subtasks. TCA provides this facility as well, and the relationship among tasks is represented as a *task tree*. In the tree, each task is represented by a node and task-subtask relationships are represented by parent-child links. In addition to these links, there is another type of link that one can use to impose a *temporal constraint* between two tasks. This type of link is also known as the *precedence relation* in operating system terminology. A temporal constraint linking node A to node B implies that task B cannot start until task A is completed. An example of a task tree is given in Figure 3.3, where the two types of links are shown.

### Monitors

There are two kinds of monitors that TCA provides: *point* and *interval* monitors. The former tests a condition only once, while the latter tests a condition many times

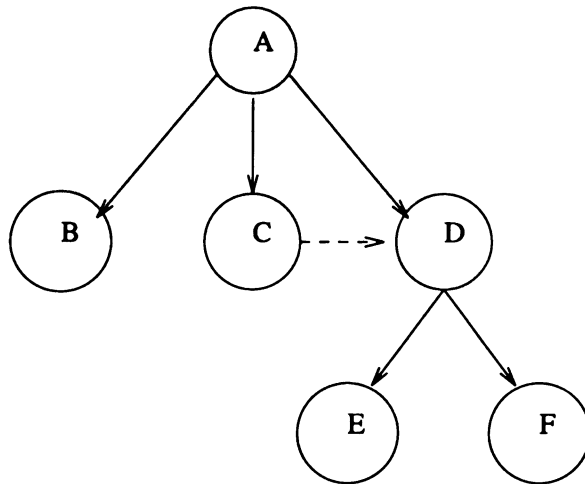


Figure 3.3. Example of TCA task tree with a temporal constraint from task C to D.

over a specified time interval. Monitors behave like a conditional statement in a program, that is, to perform an action if some event of interest occurs.

Another facility similar to monitors provided by TCA is *wiretaps*. This facility is useful for doing incremental system development. A *wiretap* is a facility to receive a copy of a message destined for another module. Thus, using this facility, the existing system does not have to be modified to make the sender duplicate the message.

### Exception Handlers

Usually, one can write a well-structured (sequential) program so that each primitive block in the program has exactly *one entry point* and *one exit point* (Single-In-Single-Out). However, when the program has to deal with exception conditions, some primitive blocks might have more than one exit point. TCA also provides a mechanism for handling exceptions during the execution of a task. The error recovery procedure for a task can be attached to that task, and the structure of the task tree is modified accordingly.

## **Wiretaps**

This is a facility that allows one module to receive a copy of a message destined for another module. Wiretaps are also useful for adding new behavior to the control system without rewriting the main algorithms.

### **3.1.4 HAREMS**

A hierarchical architecture specifically designed for controlling multisensory robots was proposed by [Buttazzo, 1992]. HAREMS stands for Hierarchical Architecture for Robotics Experiments with Multiple Sensors. The main strategy of this architecture is to utilize a bottom-up approach in building the system, i.e., higher level processes are developed from a number of lower level activities. At the highest level, there is a supervisor process that controls all subordinate processes. Hence, this architecture defines a tree of tasks, where each intermediate (internal) node of the tree represents an active computing agent, while the terminal nodes correspond to the sensors used by the robot. Figure 3.4 shows an example of the architecture.

Each computing node possesses a number of acquisition and communication boards that enable the node to handle input/output as well as to communicate with the other nodes of the system. In this architecture, interprocess communication is implemented using message passing mechanism.

### **3.1.5 Layered Architecture**

Crowley has proposed another hierarchical control architecture for robotic devices [Crowley and Causse, 1992]. The main approach in the layered architecture is the use of abstraction, implemented with a hierarchy of state spaces and control loops. A control loop in the hierarchy might have at most three communication paths. The first path goes to the subordinate control loop and is used to set the desired state of



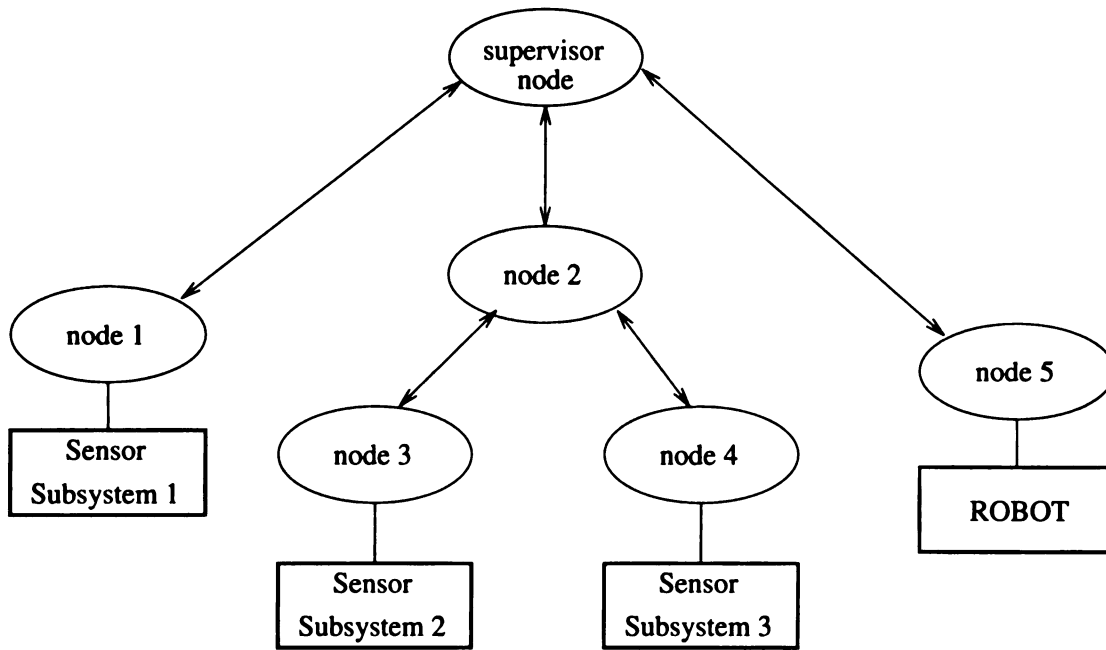


Figure 3.4. HAREMS control architecture.

the subordinate control loop. The other two paths are used for communicating the state of a control loop via an upward path to the control loop directly above it or to the control loop at the same level. This approach reduces the complexity of the control loops because each one of them operates only in a reduced parameter space.

The layered control used by Crowley in the project EUREKA EU 110 is a twin hierarchical control of four layers. This design is based on the observation that for most robotic devices, the control exists at the signal, device, action, and task levels. The first of the two hierarchies controls the locomotion actions while the other controls the perception actions. Figure 3.5 shows this control architecture.

The signal level, the lowest level in the hierarchy, deals with raw signals from the devices and has the fastest response time. On the perception side, the signal level is responsible for converting them into an initial symbolic representation. On the locomotion side, the signal level is responsible for maintaining a specified velocity of the motors and capturing the signal from the internal sensors of the robot.

The device level, the next lowest level, is responsible for representing the information in a higher context. The states controlled at this level are the position and orientation of the robot and the geometric structure of the environment. On the locomotion side, the vehicle controller accepts asynchronous commands to move and turn the robot vehicle and sends the appropriate control commands to the motor control. On the perception side, the sensor signals are projected into a common coordinate system. The projected information is used to update a composite model of the environment.

The action level coordinates a sequence of commands at the device level. In this context, actions can be viewed as procedures whose behavior depends on the timing and position parameters. The sequence of commands is defined in advance while the timing and position parameters are determined in real-time based on perception.

At the topmost level is the intelligent supervisor that has access to a cartographic and object database. This level is responsible for selecting the appropriate actions for each hierarchy to achieve the specified goal given to the system. The goal is expressed as a sequence of tasks to be accomplished by the robot.

## 3.2 Connectionist Architecture

The main characteristic of the architectures in this class is that they employ a network of modules, each performing information acquisition and decision making. Typically, a module is a simple processing element. Incremental development of the system is possible in this approach, i.e., one can first design a system that can handle only simple tasks, and then more capabilities can be added to the system later so that it can cope with more complex tasks.

Some advantages of this system are:

- The system can be built incrementally by incorporating the simple tasks into

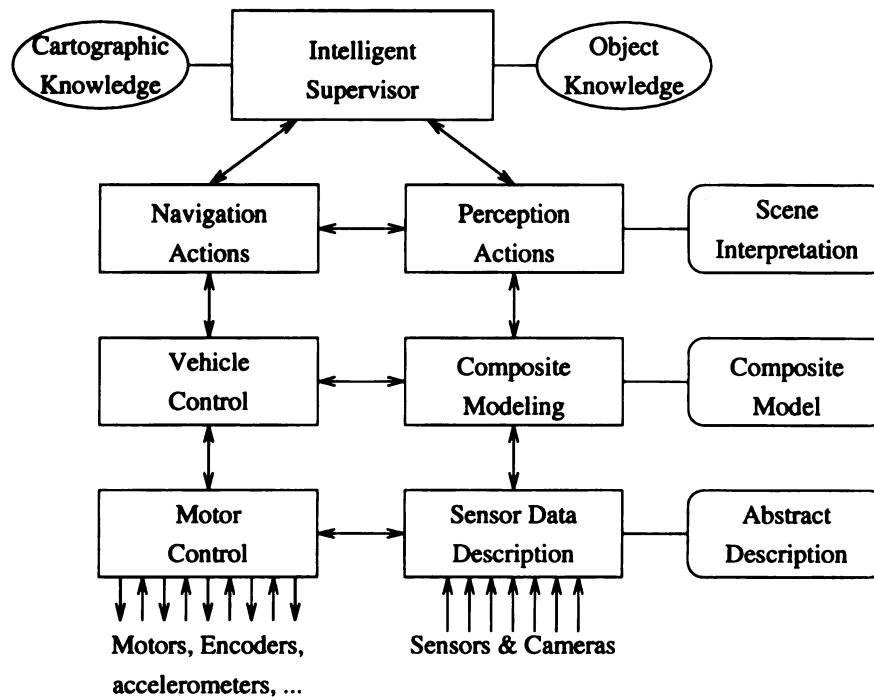


Figure 3.5. Crowley's Layered Architecture.

the system when it is first built. More complex tasks can be incorporated later.

- Unlike the multilevel architecture where higher levels depend on the result of operations at the lower level, each element in the connectionist architecture is almost independent of each other and this implies that single point failure can be avoided.

- The system operates with intrinsically parallel behavior.

Some disadvantages of this system are:

- In the multilevel architecture, determining the number of levels and task decomposition are done in an ad hoc manner without any formal methodology. Likewise, in the connectionist architecture, determining the connections among the elements is done in an ad hoc manner.
- Since the system is intrinsically parallel, the programmers have to think in

terms of co-routines instead of sub-routines. Hence, it forces the programmer to employ an unusual conceptualization of programming.

### 3.2.1 Subsumption Architecture

Brooks described a new architecture known as the *subsumption architecture* for controlling mobile robots with both multiple goals and multiple sensors [Brooks, 1986]. The heart of the architecture is layers of control systems that run asynchronously. Each layer can communicate with the others over a low-bandwidth communication channel (24-bit packets at 300 baud). The entire control system is built up by incrementally adding new levels or layers to the existing system. Each layer is composed of a number of modules, which in turn is an (augmented) finite state machine with a number of input and output lines. A module usually represents a primitive behavior of a mobile robot such as *avoid*, *wander*, *hide*, etc.

The input to a module can originate from either sensors or outputs of other modules. The output of a module can be used to command the operation of actuators or used as input to other modules. Basically, an augmented finite state machine can be viewed as a very small sequential program. Each state in a module can be one of four types: **output**, **side-effect**, **conditional-dispatch**, and **event-dispatch**. Output of one module can inhibit or suppress output of another module at a lower layer. In this manner, higher layers can *subsume* lower layers but lower layers continue to work if the higher levels fail or remain inactive. An example of a module is given in Figure 3.6. In the example, the augmented finite state machine contains four states: **nil**, **plan**, **go**, and **start**. Figure 3.7 shows an example of a network in the subsumption architecture. To use the system for controlling an autonomous robot, each module is implemented in hardware as a single processor board (3 inches by 4 inches) and connection among modules is achieved by actually connecting wires between these modules [Brooks, 1987].

```

(defmodule avoid 1
  :inputs (force heading)
  :outputs (command)
  :instance-vars (resultforce)
  :states
    ((nil (event-dispatch (and force heading) plan))
     (plan (setf resultforce (select-direction force heading)) go)
     (go (conditional-dispatch
(significant-force-p resultforce 1.0)
start
nil))
      (start (output command (follow-force resultforce))
              nil)))

```

Figure 3.6. An example of a module in the Subsumption Architecture.

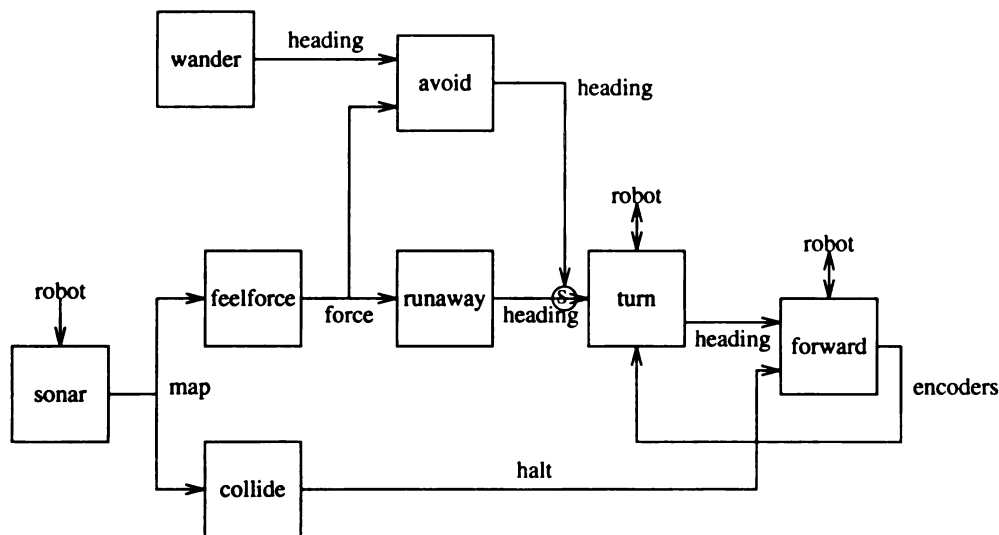


Figure 3.7. An example of module interconnection in the Subsumption Architecture (reproduced from [Brooks, 1986]).

The subsumption architecture uses neither a central representation of the environment nor a control supervisor of the modules. Actually, the architecture is considered both non-representational and distributed. Much of the “intelligence” is found in the module that implements a specific behavior. This is the main characteristic of the system compared to the others. This unconventional approach leads to controversy over whether “insect-like” behavior is sufficient for mobile robots.

This control architecture has been implemented in software using Lisp code and in hardware using custom chips and tested in several mobile robot projects at MIT. Flynn and Brooks have even demonstrated that the hardware implementation can be engineered to a very small size by incorporating it in small sized robots such as toy cars [Flynn and Brooks, 1988; Flynn *et al.*, 1989; Brooks, 1991].

To summarize, some of the features offered by the subsumption architecture are given below:

- Decomposition of a control system into a number of layers (levels of competence).
- Decomposition of a single layer into cooperating modules.
- Connections among modules, suppression and inhibition of output from a module.

Unfortunately, there is no general guideline for implementing a control system in this architecture and one has to use intuition before coming up with a complete implementation of a control system. In my opinion, the interconnections among modules somehow encode the “main control algorithm” of the system.

### 3.2.2 Colony-Style Architecture

As part of his doctoral dissertation, Connell adopted Brooks’ subsumption architecture for controlling a mobile robot that is capable of collecting empty soda cans in a

hallway [Connell, 1990]. Starting from an initial position, the robot wanders around in its environment, looking for empty soda cans. Once the robot finds an empty soda can, it grabs the can, and returns to its initial position to drop the can in a box.

Connell referred to his control architecture as a *colony-style* architecture, since the system is visualized as a colony of locally autonomous agents that coordinate their actions to accomplish the robot's task. The colony-style architecture can be categorized as behavior-based architecture, since it consists of a number of modules, each of which implements some primitive behavior. In Connell's implementation, some modifications were made to the original specification of subsumption architecture.

In the colony-style architecture, a module consists of two major parts. The first part, the transfer function, defines the action that will be performed on the input signal to produce the output signal. The second part of the module, the applicability predicate, determines whether this output of the transfer function should be gated to the output port. Viewed this way, a module can be considered as a production rule. Sometimes the robot is required to possess a reactive behavior by responding to events or situations. When a certain event is triggered, a module might need to set a flag in its own memory to indicate that the event has occurred. For this type of modules, the applicability predicate has to be modified. Figure 3.8 shows the two types of modules which exist in the colony-style architecture.

Unlike the subsumption architecture, where a control system is decomposed into several layers such that a total order is defined, the colony-style architecture defines only a tree-like partial ordering of the layers. Also, the colony-style architecture requires more independence of modules. For instance, the subsumption architecture allows the output of one module connected to the input of another. The colony-style architecture disallows this type of connection, and consequently connections that cross layer boundaries are eliminated. However, the suppression and inhibition of output signals are retained in the colony-style architecture, and in fact these are the only





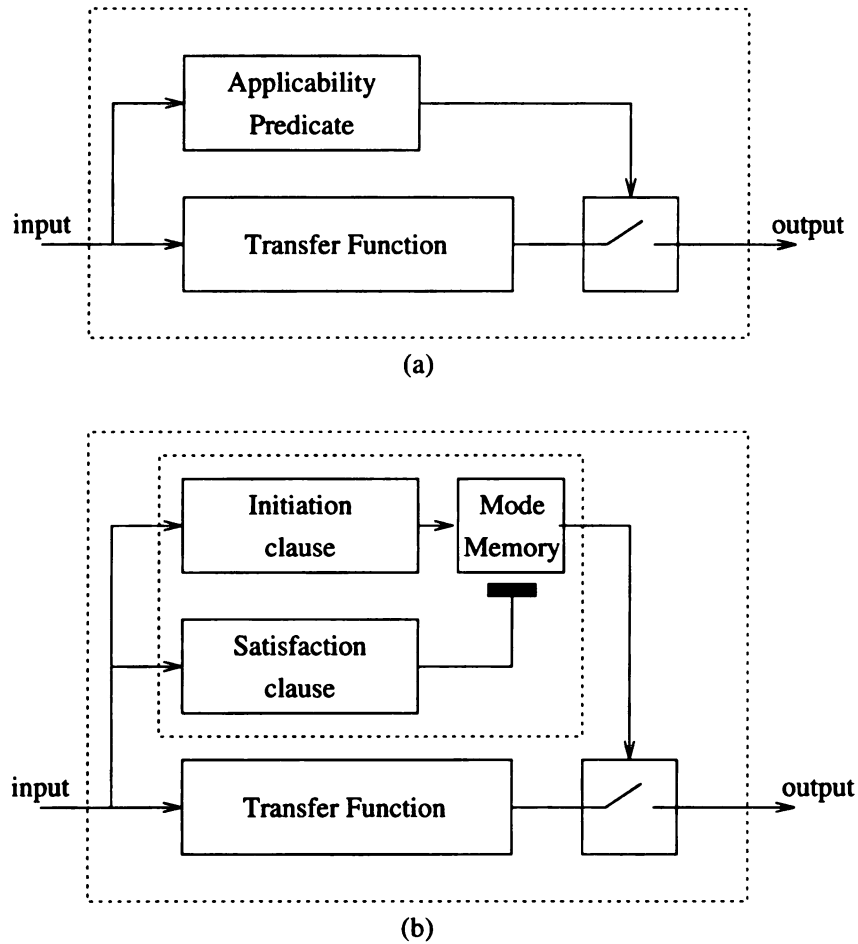


Figure 3.8. Structure of two modules in the Colony-style Architecture. (a) memory-less module; (b) module with memory.

ways of “communication” among modules. Using this construct, a more powerful behavior can take over from general-purpose behaviors. Figure 3.9 shows an example of module interconnection in the colony-style architecture.

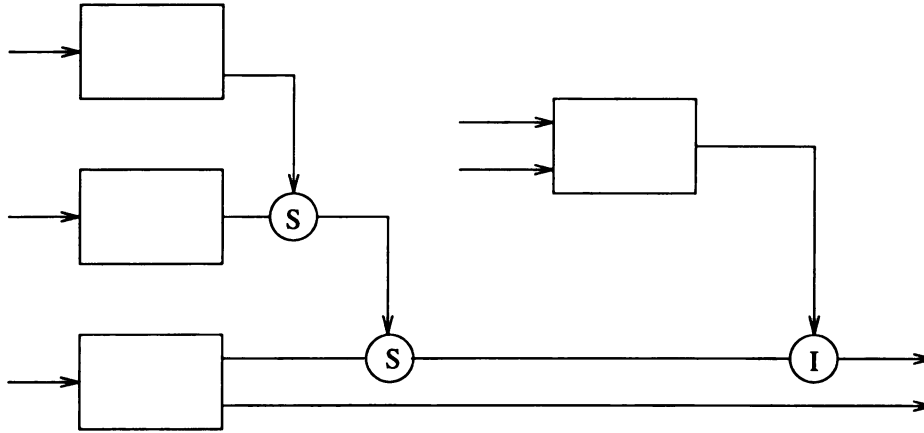


Figure 3.9. An example of a network in the Colony-style Architecture with two suppressor nodes and one inhibitor node.

By modifying the way signals are generated from a module, Connell has eliminated the *lock-out* problem that might occur in the subsumption architecture. In the subsumption architecture, suppressor and inhibitor nodes have a preset time constant associated with them. When a dominant module sends an output pulse to a suppressor (inhibitor) node, this output will be retained for a prespecified length of time. If during this time interval the inferior module connected to the node also sends an output, it will never be observed on the output line of the node, even after the dominant module relinquishes control. In the Colony-style, a module generates a stream of outputs rather than a single-pulse output.

### 3.2.3 Autonomous Robot Architecture

Arkin proposed a motor schema-based control system called the Autonomous Robot

Architecture (AuRA) [Arkin, 1989]. The concept of *schemas* originated from psychology and neurology, in which many different types of schema have been used. AuRA defines two types of schema: *motor* and *perceptual*. A schema can be viewed as a primitive behavior that a robot can perform. Generally, to accomplish a given task, a robot must have a number of distinct schemas. Examples of motor schema are: **move-to-goal**, **avoid-obstacle**, and **move-ahead**, while examples of perceptual schema are: **find-landmark** and **find-intersection**. In this sense, motor schemas are related to actuation tasks, while perceptual schemas are related to sensing or perception tasks. A schema becomes active when it is instantiated by specifying its parameters and activated as a computing agent. For this reason, a schema is also referred to as “a generic specification of a computing agent”. The implementation of schemas in AuRA is mainly influenced by potential fields approach.

AuRA is composed of five major components [Arkin and Murphy, 1990]: perception, cartographic, planning, motor, and homeostatic control system. The perception subsystem is responsible for acquiring sensory data and performing some preprocessing on the raw data. The cartographic subsystem is responsible for navigational planning and model building of the environment. The planning subsystem acts both as a hierarchical planner and a motor schema manager. The motor subsystem acts as the lower-level interface to the mobile robot. Lastly, the homeostatic control subsystem deals with robot survivability in dangerous environment.

During navigation, the planning subsystem will determine which schemas to dispatch based on the output of the perception subsystem. All activated schemas will run concurrently, and the potential fields of each schema will be combined into a single potential field used by the navigation algorithm. If the planning subsystem decides that a particular schema is no longer appropriate, that schema will be deactivated by the schema manager. Therefore, AuRA can be considered a dynamic network of active computing agents. Figure 3.10 shows a navigation system in AuRA.

de  
by

3.

M

R

ten

des

the

type

and

of a

ACB

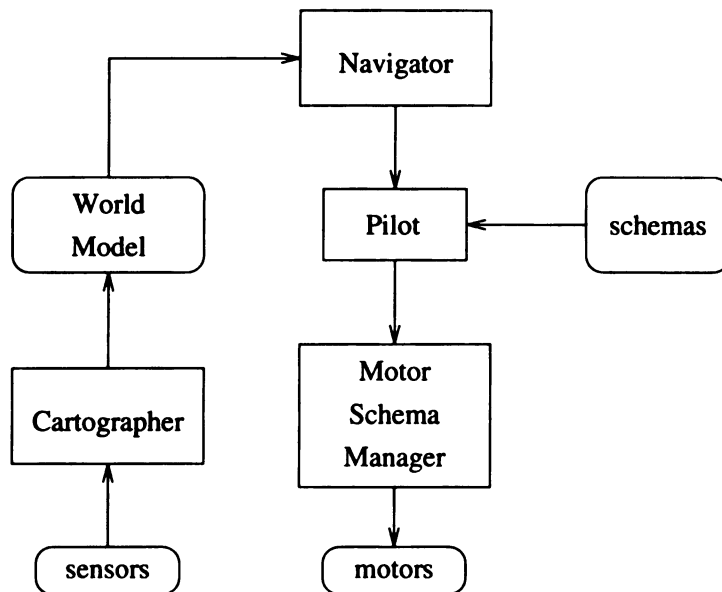


Figure 3.10. Navigation system in AuRA.

As a result of combining several potential fields into a single one, some *dead spots* (i.e., points with no potential), might be encountered. This problem can be eliminated by introducing a noise schema to the current network of active schemas.

### 3.2.4 ACBARR

Moorman and Ram proposed their system, ACBARR – A Case BAsed Reactive Robotics – for autonomous navigation [Moorman and Ram, 1992]. In general, the system has many commonalities with the schema-based approach. However, ACBARR design was based on the Case-Based Reasoning approach in AI. Thus, the system has the capability to modify its current behavior based on the system's immediate past experience. For instance, if a particular behavior is working well, the system will set a higher gain for that behavior. On the other hand, the system might reduce the gain of a particular behavior that is not working well. To facilitate this gain adjustment, ACBARR monitors the feedback for failure situations.

deso

bede

whi

### 3.2

Par

net

et c

dis

ma

bi

ret

at l

le.

dis

leve

mo

ing

for

eg

from

The

h

tion

n sh

The set of behaviors used by ACBARR is organized into a case library. A case describes the desired behavior and the environment suitable for this behavior. The behavior information contains the range of gain values that can be used for this case, while the environment information is used as an index when this case is retrieved.

### 3.2.5 Blackboard-Based Architecture

Paul *et al.* have proposed a distributed control mechanism which incorporates a network of intelligent sensing, action, and reasoning agents for a robot system [Paul *et al.*, 1985]. The framework is used for integrating a number of sensors such as vision, range, touch, force, and motion. The use of a distributed approach is the main characteristic of this architecture compared to others that are designed around a hierarchical structure. A pure hierarchical design precludes interaction with low-level sensors, and processes at higher level totally depend on the output of the processes at lower level. In addition, a hierarchical structure is generally not easily expandable; i.e., addition of a new process might result in a modification of the entire structure.

To overcome the drawbacks of the hierarchical structure, Paul *et al.* proposed a distributed approach that employs a number of sensor agents controlled by a higher level coordinator via a blackboard structure. The coordinator maintains a world model represented as a set of states of the environment and is responsible for integrating the information given by the sensors, making decisions about the task allocation for each agent, and arbitrating between the distributed agents. Additionally, each agent is provided with a significant amount of self-knowledge to allow it to recover from local errors and to decide what information is to be supplied to the coordinator. The structure of this system is shown in Figure 3.11.

In the system, all sensors and actuators run in parallel and supply partial information to the coordinator via a blackboard database. Three types of agents are defined in the system:

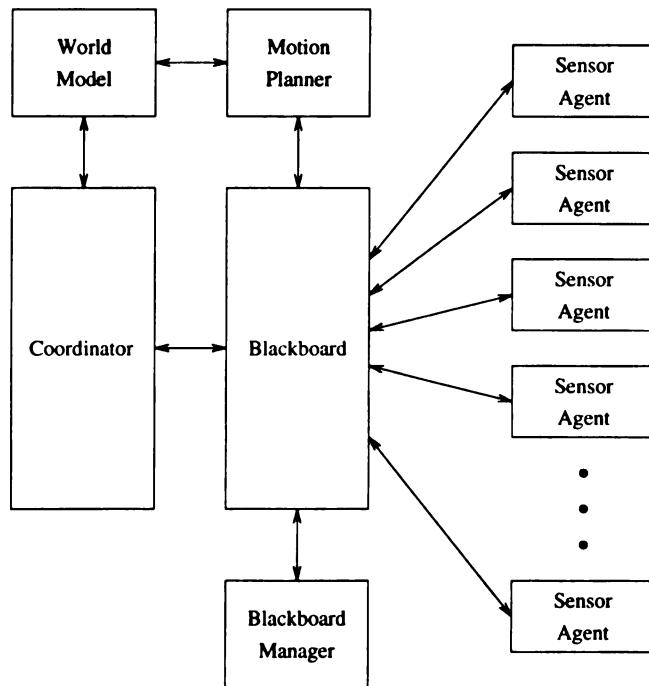


Figure 3.11. Structure of the distributed agent in blackboard-based architecture.

- A sensing agent that provides information about the environment.
- An action agent that performs operations that modify the state of the environment.
- A reasoning agent that provides information based on its reasoning expertise.

An agent in the system can be of more than one type simultaneously. The crucial agents in the system are the manipulator, hand, and vision agents. These are crucial in the sense that without any one of these agents the entire system is considered to be non-operational. On the other hand, other agents can be removed from the system without crippling the entire system.



### 3.3 Hybrid Architecture

Control architectures in this class combine the multilevel and connectionist approaches by merging positive features of the two approaches. At a higher level of abstraction, the hybrid architecture employs the multilevel approach while at a lower level of abstraction, the connectionist approach is used. Thus, in a hybrid control architecture, one will find both task decomposition into subtasks and a network of computing agents executing the subtasks.

#### 3.3.1 HRL/Voting-Based Architecture

Hughes Research Laboratories has developed a hybrid architecture for real-time intelligent control systems. In the architecture, the control system is modeled as an intelligent system driven by a number of goal-directed behaviors. Four types of processes are defined in the architecture: coordinator/planner, behavior system, sensing processes, and control processors. Except for the behavior system, the remaining three types of processes perform the well-known “sense-plan-action” cycle. The inclusion of the behavior system adds a new flavor to the architecture. The behavior system is capable of maintaining goals which are both parallel and interact to various degrees. These concurrent goals may interact with one another in constructive or destructive ways. The behavior system determines the commands sent to the control processes depending on the task goals and the current task status. Since multiple behaviors can be active at the same time, and goals can be conflicting, the control process might receive conflicting commands. This situation is resolved by employing a voting mechanism to check for consistencies in the commands.

In the

base

a co

pre

In t

The

str

con

res

by

Zi

the

is

be

the

res

out

the

lib

and

### 3.3.2 Rational Behavior Model (RBM)

In the RBM, behaviors are internally represented using a backward-chaining rule-based programming language like Prolog. Satisfying a goal is similar to satisfying a corresponding predicate in a rule-based system. In a rule-based representation, a predicate can be satisfied if all the constituent clauses of the predicate are satisfied. In the RBM, each predicate or clause is either a complex or a primitive behavior. Thus, when a goal is triggered and the corresponding predicate is resolved, all constituent clauses have to be resolved. In the same manner, this also shows how a complex behavior is activated. When a primitive behavior is activated during the resolution process, function calls to imperative programming languages will be made for performing the requested behavior.

### 3.3.3 Concurrent Behavior Control Architecture

Zimmerman proposed another behavior-based control architecture. One of the criteria of this architecture design is to allow high reactivity of the robot. The architecture is composed of a number of concurrently operating, multiple independent layers of behaviors. Each behavior layer performs its own perception and control, and conflicts between layers are resolved by a separate behavior fusion block. The conflict resolution mechanism used by the fusion block is a weighted sum of all behavior layer outputs. Behavior layers with larger weights have a higher priority over the other layers. This weighting scheme has the advantage that a behavior layer can be inhibited by assigning a very low weight to it. Figure 3.12 shows the structure of this architecture.

The design of this architecture was based on the objective of building “intelligent” locomotion with insect-like characteristics and on the observation that in biological

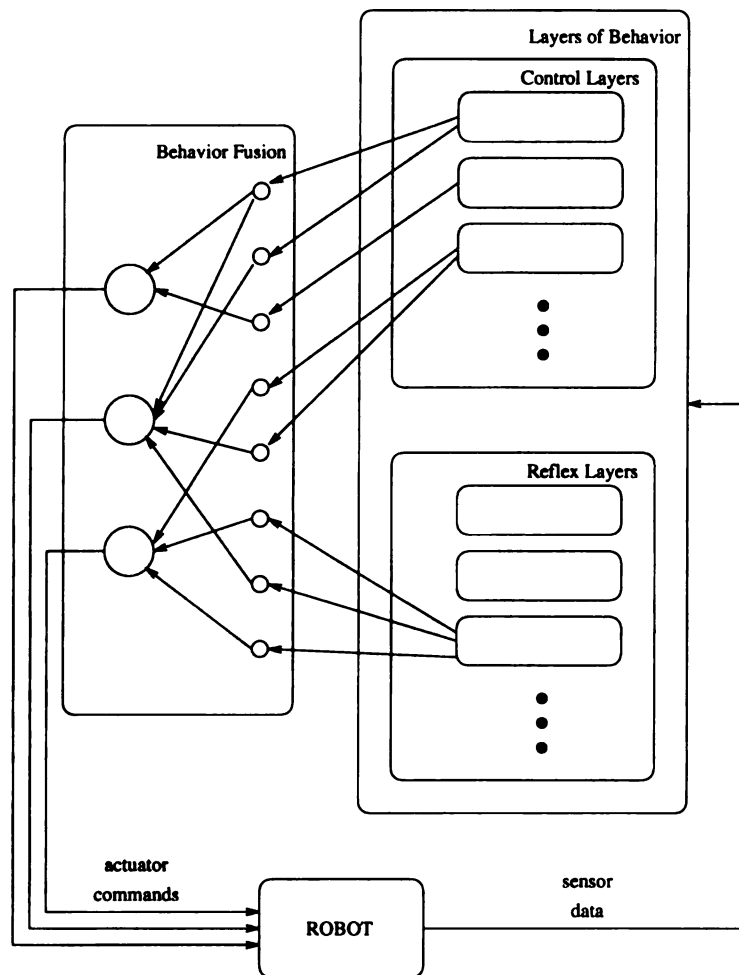


Figure 3.12. The structure of the Concurrent Behavior Control Architecture.

system

recon

cont

cont

trig

1

the

to

det

det

to

3.

At

It

for

se

78

it

at

to

to

to

t

e

to

systems, the overall intelligence emerges from a collection of simple and simultaneously operating “behaviors”. In the architecture, two sets of layers are defined: control and reflex layers. The control layers are always active and generate output continuously, while the reflex layers become active only when a certain context is triggered.

Each layer of behavior performs its own sensing, planning, and action to achieve the goal associated with the behavior. This approach allows each layer to be reactive to the changes in the environment. The input of each layer is real physical sensor data as well as preprocessed virtual sensor data, while the output of each layer is the desired actuation commands necessary to achieve the behavior. The response of each behavior layer is determined using both classical and fuzzy logic rules.

### 3.3.4 Independent Agents Architecture

Another behavior-based architecture is reported in [Yamauchi and Nelson, 1991]. This architecture decomposes control systems into three vertical layers: perception, behavior, and motor control. The behavior layer is also decomposed into independent sensorimotor behaviors (agents). In this architecture, the concept of *stimulus* and *response* spaces is used to describe the role of perception and behavior layers. The stimulus space is the space of all possible processed sensory inputs, while the response space is the space of all possible actions that may be taken by the robot. The perception layer is responsible for mapping the raw sensor into stimuli in the stimulus space, while the behavior layer maps a stimulus into a response which will be used by the motor control layer to decide the proper action the robot has to execute. In other words, the three layers in the Independent Agents Architecture link perception to action. Figure 3.13 shows this link.

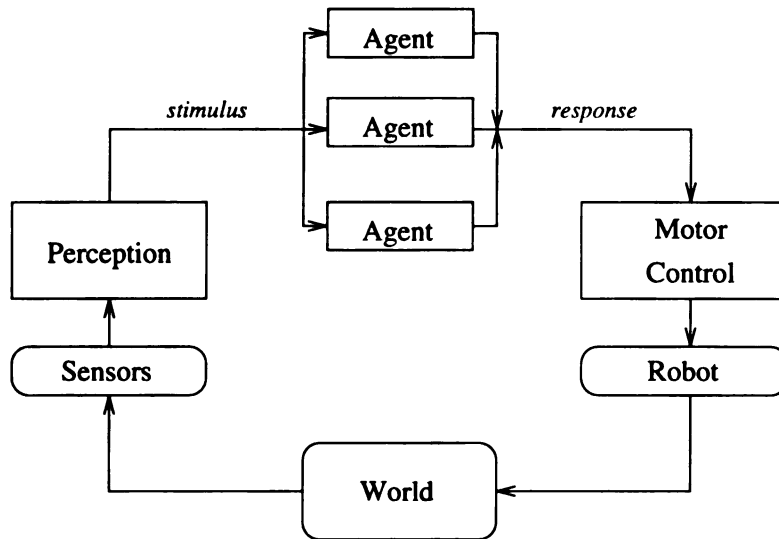


Figure 3.13. A model of Independent Agents Architecture.

### 3.4 Toolkit-Based Approach to Control Architecture

While no specific control architecture has been agreed to best fit a certain type of robotics application, a programming tool that can be used to aid the implementation of a particular architecture would be useful. This is the underlying philosophy of the toolkit-based approach, i.e., not to use an architectural standard, but to construct task-specific architectures with the help of toolkits. By doing this, the architecture implementor can concentrate mainly on the higher level aspects of the architecture. In general, such a toolkit should provide utilities for:

- Interprocess communication,
- Control Flow,
- Shared access to global data,
- Task coordination,

- Resource sharing, and
- Real-time control.

Some advantages of this approach are:

- A toolkit for constructing task-specific architectures can be viewed as a general-purpose programming language for implementing problem-specific algorithms. Thus, using this approach one can construct a control architecture that best fits the task of interest.

Some disadvantages of this approach are:

- When the best architecture for a certain type of application exists, adapting that architecture to fit the task of interest is usually a matter of “filling-in-the-blanks”, i.e., one has to provide routines to be plugged into the architecture so that the architecture functions as desired. Here, it is much easier to use that architecture instead of constructing it using the toolkit approach.
- The toolkit designer has to ensure that the toolkit is complete, so that it provides all the required functionalities for constructing a working control architecture.

### 3.5 Summary

This chapter has reviewed a number of different architectures for controlling the execution of tasks to be performed by a robot. In general, control architectures can be classified into one of the four principle organizations: multilevel, connectionist, hybrid, and toolkit. Due to the immaturity of the control architecture technology, we lack a rigorous method for designing a standard control architecture, common terminologies, and common comparative measures.





The multilevel/hierarchical architecture emphasizes decomposing the given task in a top-down refinement approach, thus breaking down a large task into a number of smaller subtasks. On the other hand, the connectionist architecture deviates from this approach by forcing the programmer to decompose a large task into a number of coroutines. Hierarchical architectures are widely used and seem to be a “natural” choice of system decomposition.

The hybrid architecture takes the positive features of both multilevel and connectionist architectures. In such systems, a multilevel organization is used at the higher levels of abstraction and a connectionist organization is used at the lower levels.

The toolkit-based approach views the design of control architectures in a different way. Instead of using a particular architectural standard, it constructs a task-specific control architecture with the help of a toolkit. Thus, its role is very similar to a general-purpose programming language for constructing a problem-specific algorithm. All control architectures, except the toolkit-based, have been used in various mobile robot applications. The NASREM control architecture and its derivatives have been applied to a number of applications. I believe that NASREM and the Subsumption Architecture are among the most popular architectures.

Table 3.1 summarizes the control architectures described in this chapter.

Table 3.1. Comparison of various control architectures.

Architecture	Advantages	Disadvantages
<b>Multilevel Architecture</b> <ul style="list-style-type: none"> <li>• NASREM</li> <li>• ARTICS</li> <li>• TCA</li> <li>• HAREMS</li> <li>• Layered</li> </ul>	<ul style="list-style-type: none"> <li>• Systematic approach for task executions.</li> <li>• Top-down design is commonly used in programming, so this approach is “natural”.</li> </ul>	<ul style="list-style-type: none"> <li>• No rigorous methodologies for task decomposition.</li> <li>• The number of levels in the system is determined in an ad hoc manner.</li> <li>• Interlevel dependency might become the single point failure of the entire system.</li> </ul>
<b>Connectionist Architecture</b> <ul style="list-style-type: none"> <li>• Subsumption</li> <li>• Colony-Style</li> <li>• AuRA</li> <li>• ACBARR</li> <li>• Blackboard-based</li> </ul>	<ul style="list-style-type: none"> <li>• Incremental system development is possible.</li> <li>• Elements in the connectionist architecture are almost independent of each other and thus single point failure can be avoided.</li> <li>• The system operates in intrinsically parallel behavior.</li> </ul>	<ul style="list-style-type: none"> <li>• No formal methodologies for determining the connections among the elements. It is done in an ad hoc manner.</li> <li>• Due to the system’s parallel behavior, programmers have to think in terms of co-routines instead of sub-routines.</li> </ul>
<b>Hybrid Architecture</b> <ul style="list-style-type: none"> <li>• HRL</li> <li>• RBM</li> <li>• CBC</li> <li>• Independent Agent</li> </ul>	<ul style="list-style-type: none"> <li>• Merges the positive features of both the multilevel and connectionist approach.</li> </ul>	<ul style="list-style-type: none"> <li>• Determining the boundary between multilevel and connectionist implementation is done in an ad hoc manner.</li> <li>• Requires programmers to think in terms of both subroutine and co-routine approaches.</li> </ul>

Table 3.1. (cont'd).

Architecture	Advantages	Disadvantages
Toolkit-based	<ul style="list-style-type: none"> <li>• Using a toolkit, one can construct a control architecture that best fits the task of interest.</li> </ul>	<ul style="list-style-type: none"> <li>• When the best architecture for a certain type of application exists, adapting that architecture to fit the task of interest is preferable to constructing one from scratch using the toolkit.</li> </ul>
Client-Server <sup>a</sup>	<ul style="list-style-type: none"> <li>• Unified framework for distributed processing, resource sharing, resource access control, and synchronization.</li> <li>• Incremental development is possible.</li> <li>• General enough to emulate other control architectures.</li> <li>• Hardware servers can be designed to be small enough to run on processors with limited resources.</li> <li>• Clients can be assigned “execution” priority regardless of the facility of setting execution priority from the underlying operating system.</li> </ul>	<ul style="list-style-type: none"> <li>• Communication time overhead that might be incurred by the client-server interactions.</li> <li>• There is no formal methodology for determining the size of a client module.</li> </ul>

<sup>a</sup>Control Architecture proposed in this thesis. It is described in detail in Chapter 4.

# CHAPTER 4

## Client-Server Control Architecture

Recently, “client-server” has become a buzzword in the commercial market, and a number of software vendors claim that their products support the client-server model. Also, tools for software development in client-server environments and for migration of existing applications to a client-server environment are being routinely mentioned in the media. In this chapter, I will explain the client-server model in general and how this model is applied to a system for controlling mobile robot navigation.

### 4.1 Client-Server Model

In a large software system it is very common to find several programs or processes running concurrently. These processes cooperate and interact with each other with a certain mechanism. Physically, these processes can reside on a single processor, a number of different processors, or even across several computers connected via a network. A program consisting of several cooperating and interacting concurrent processes is sometimes referred to as a *distributed program*. The word “distributed” should not be interpreted as “geographically distributed”, but rather in the sense

that the program execution is distributed across several processes. Andrews described four kinds of processes in a distributed program: filters, clients, servers, and peers [Andrews, 1991]. A filter acts like a data transformer; it transforms input data to output by performing some operations on the input data. A peer is one of a collection of identical processes that interact to provide a service. For instance, in a parallel programming environment several peers interact to solve a problem, each solving a piece of the problem.

A client initiates a request, and a server responds to requests. Thus, a client sends requests to a server which, in turn, reacts to the request by sending the information needed by the client. Client and server types should be viewed in the context of their relationship; for instance, a server process can also be a client if that process sends requests to another server. Also, it should be emphasized that requests flow from the client to the server. This distinguishes client-server from peer-to-peer relationship where requests can flow from either side.

Typically, a client can make requests to several servers and a server can accept requests from a number of clients. Thus, designing a server is more complicated than designing a client. A server has to multiplex requests from a number of clients simultaneously, and a failing client should not prevent the server from accepting and completing requests from other clients. Figure 4.1 shows a typical client-server relationship.

Due to the nature of the division of an application into the “client” and “server” parts, the client-server model is also known as a two-tier model. This characteristic is important to distinguish the client-server model from subroutine calls, where the caller and subroutine being called are parts of a common process.

Some examples of applications that employ the client-server model are Internet computer networking, X-Window systems, and Sun’s Network File System. For instance, programs running under an X-Window environment are considered clients to

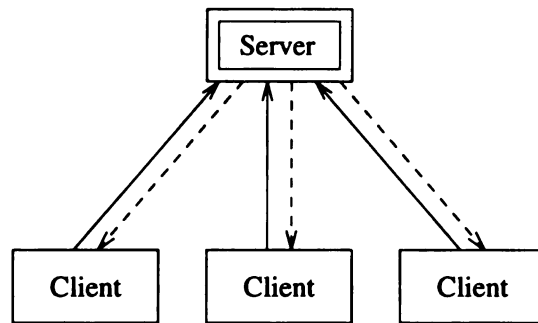


Figure 4.1. Client-server relationship.

the X-Server. These clients do not have direct access to the peripherals (keyboard, mouse, display screen, etc.) of the workstation where the X-Server is running. Whenever a client needs to access one of these peripherals, it sends a request containing the type of operation to be performed on the requested peripheral. In response, the X-server will perform the requested operation on the peripheral and notify the caller of completion of the operation. By directing all requests to the X-Server, the peripherals are accessible from a remote site.

### Client-Server Interaction

Figure 4.1 shows a simplified communication model between clients and server. In practice, the communication involves intermediate steps in the operating system where the client/server runs. Some of these steps are: execution of the operating system's kernel code and transmission of data through the communication network (if the client and the server reside on different machines).

There are a number of issues one has to consider when implementing the interaction between clients and servers. Tanenbaum mentions the following four issues [Tanenbaum, 1992]:

1. **Addressing.** Before a client can communicate with a server, the server's address has to be obtained. There are at least three approaches for resolving

this situation. The first is to hardwire the server's address into the client's codes. For this approach to work, the addresses have to be fixed. The second approach is to use random addresses and let the client determine the address via broadcasting and waiting for a response from the appropriate server. The last approach is to use a symbolic name and resolve the physical address at run time through a name server.

2. **Blocking vs. non-blocking.** The low-level primitives for sending and receiving messages can be either blocking (synchronous) or non-blocking (asynchronous). After a non-blocking send/receive, control returns to the caller immediately. A blocking send/receive does not return the control until the other party completes its corresponding receive/send command.
3. **Buffered vs. unbuffered.** In a buffered receive, the kernel reserves a buffer for storing messages before they are read by the designated process. This mechanism allows the sender to send the message before the client is waiting for the data. In an unbuffered receive, when the above situation occurs, the message is lost.
4. **Reliability.** It is always possible that the messages sent by a process do not actually reach the destination. In this event, there has to be a way of detecting such a situation and restarting the message transfer.

## 4.2 Remote Procedure Call

In using the client-server model, a programmer has to pay attention to the input and output operations for all communication between clients and servers. Hence, the programmer has to think in a slightly different way from writing a centralized (non-distributed) program. Remote Procedure Call (RPC) alleviates this drawback



by allowing a process to call a procedure on a remote site.

RPC provides a way to make the call to a remote procedure look very much like a local procedure call. This transparency is made possible by using *stubs*. A stub is a different version of the procedure with a remote call embedded in it. In a local procedure call, the parameters of the procedure are passed from the caller to the callee through the stack in the same address space. In a remote procedure call, the parameters are passed to the remote machine. The client stub is responsible for packing the parameters into a format acceptable to the server stub. Upon receiving the parameters, the server stub unpacks them and performs a local procedure call on the remote side. After the remote procedure completes execution, the results are passed back to the calling procedure by the server stub. The client stub will unpack the results and return them to the client. Figure 4.2 shows the steps involved in a remote procedure call. The numbers in the figure correspond to the following steps:

1. The client calls the local client stub.
2. After packing the parameters, the client stub passes them to the kernel.
3. The kernel on the client machine sends the request to the remote kernel over the network.
4. The remote kernel passes the message to the server stub.
5. The server stub uses the unpacked parameters to call the server.
6. The result from the server is passed back to the server stub.
7. After packing the results, the server stub passes them to the kernel on the server machine.
8. Over the communication network, the kernel on the remote machine sends the message to the kernel on the client machine.

9. The client stub receives the packed results from the kernel.
10. After being unpacked, the results are returned to the client.

Figure 4.2 shows how the client sees the remote procedure as if it were a local routine.

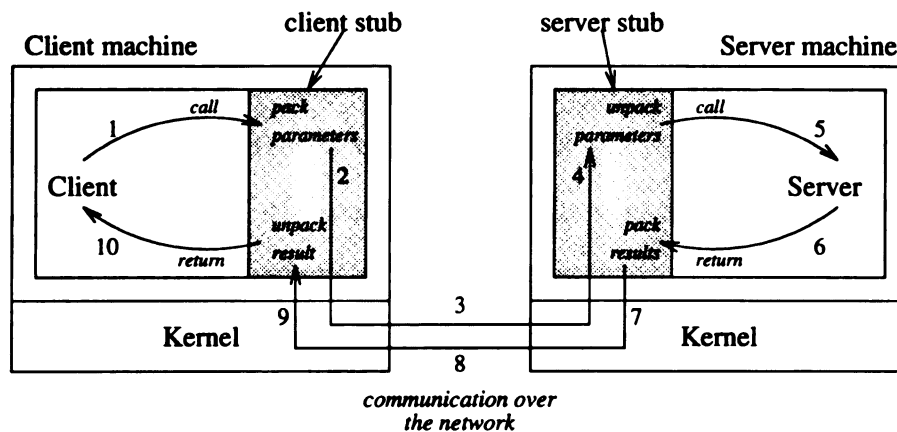


Figure 4.2. An example of a Remote Procedure Call.

Because different hardware platforms might be used on the client and the server sites, the byte ordering of data on both machines might be different as well. Thus, when the parameters and the results are transferred from one machine to the other, a convention must be established to avoid misinterpretation of the data due to the different byte ordering on the two machines. The stubs handle this task and guarantee that data has the same interpretation both to the client and the server.

### 4.3 Client-Server Control Architecture

In my opinion, the client-server model can be appropriately applied to controlling the operation of a mobile robot. It is common to find a number of concurrent processes (modules) running on a mobile robot to accomplish the subtask assigned to the controlling process. It is most likely that these concurrently running processes will acquire data from the peripherals/sensors available on the robot. Thus, a mechanism for controlling resource sharing among these processes is needed. Moreover, since the processes cooperate to achieve a common goal of the robot, they need to interact and exchange information with one another. This situation dictates the need for a common communication medium for the processes.

Based on the above observations, I have designed and implemented a control architecture for mobile robots that employs a number of servers to act as the interface between the processes and the sensors/peripheral devices and a single server that serves as a common communication channel for the modules. For each hardware component accessed by the modules, there is one server dedicated to it. This type of server will be referred to as the *hardware server* or **HServer** for short. All modules that need to use this hardware component have to send a request to the **HServer**. The actual operation is performed by the **HServer** and the module requesting the operation will be notified by the **HServer** when the operation is completed, or if an exception arises.

To provide a common communication medium for all modules, we need to incorporate another server that acts as an information center. This server is referred to as the *data server* (**DServer**). Any two modules can interact with each other with the help of the data server. Thus, the two modules are not exchanging messages directly. A similar mechanism is also used in Linda parallel programming model [Ahuja *et al.*, 1986; Carriero and Gelernter, 1989]. One might think that the role of the data server is

similar to Linda's tuple space. However, the data server does not have an equivalent operator for Linda's operators `out()` and `eval()`.

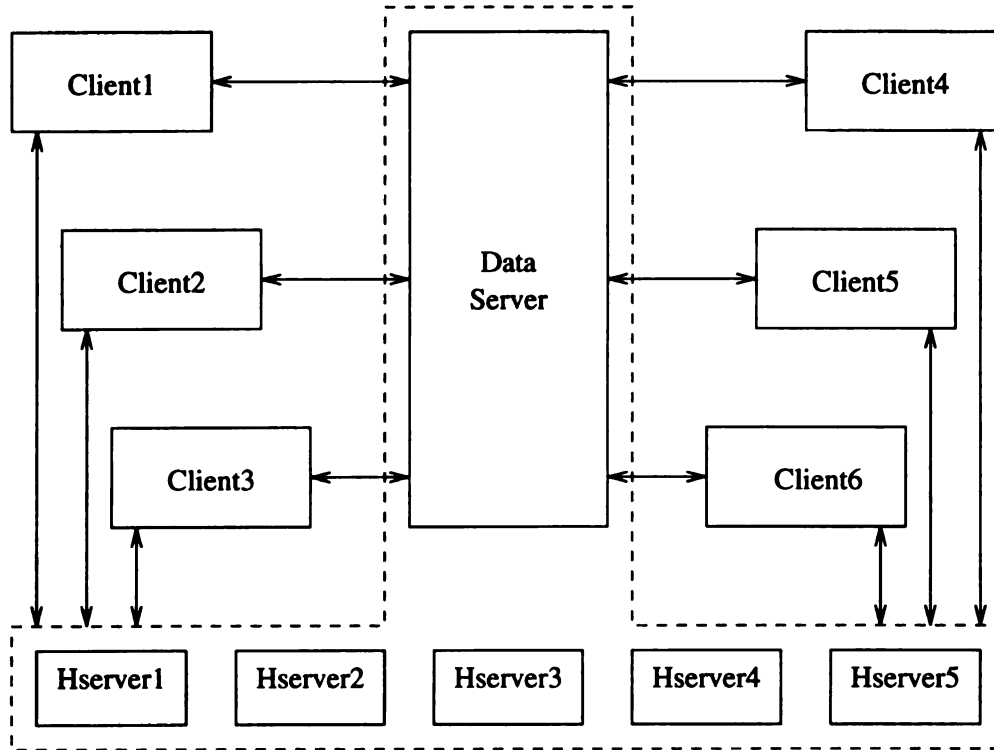


Figure 4.3. Typical configuration of the Client-Server Architecture.

A typical configuration of the system would look like Figure 4.3. In this figure, all client modules can communicate to the *Data Server* as well as to the hardware servers (*HServer*'s). By using this approach, all modules can share data with each other. The hardware and data servers will always exist in a system and they can be considered to form the core of the entire system. However, the existence of the other modules depends on the type of application assigned to the robot. Since the client-server model allows us to distribute the client and server modules across several processors, the physical address of the servers can be assigned on demand. In this situation, a *name server* is needed when a client has to resolve the physical address

of a particular server. Upon initialization, all the other servers must register their physical addresses to the name server.

Also, more than one copy of each server can run simultaneously to provide a more reliable system. When a client thinks that a particular server is not active any more, the client can direct its requests to an alternate server.

It should be emphasized that the servers do not communicate with each other, and neither do the clients have direct communication among themselves. Clients make requests to servers only. Indirect communication among the clients are achieved through the Data Server. Due to the independence of the servers as well as the clients, the module interconnections in the Client-Server Control Architecture can be modeled using a *bipartite graph*. See Figure 1.4. All the client modules belong to one set of nodes and all the server modules belong to the other set of nodes.

A module running in the system described above will have to use a communication scheme to send requests and receive the results. Therefore, the system has to be run on an operating system that supports an interprocess communication facility.

Some advantages of the client-server model as described here are given below:

- Each client-server connection can be assigned a priority value indicating how the requests from the client will be processed by the corresponding server. Thus, a client module might have several priority values depending on the number of open connections to the servers. By assigning priority values to the *connection*, not to the client module itself, the client appears to have a different level of importance to each server. This feature is useful, for instance, when the operation of a client module relies mostly on the data from a particular sensor and needs to obtain a fast response from the sensor server. Also, by using this priority scheme, the priority of execution of a client module can be controlled regardless of the availability of this facility on the underlying operating system.

- By delegating the hardware accesses to the hardware servers, a client module that needs to access a particular hardware component does not have to run on the machine where the hardware resides. Only the hardware server has to run on that particular machine. This situation enables us to use small-sized machines on the robot to run the hardware servers and let the clients run on more powerful machines perhaps at a remote site. This approach also makes the hardware transparent to the clients. When the hardware needs to be replaced or upgraded, the client codes need not be rewritten. Clients can keep sending the same requests to the server and the server takes care of the low-level interface to the upgraded hardware.
- The services offered by the Data Server and the Hardware Servers provide sufficient constructs for information sharing, resource sharing, resource access control, and process synchronization. These facilities are inherently used by all control architectures for robot navigation system. Thus, the client-server model described here can be used as a general control architecture for robot navigation.
- Due to the distributed nature of processing in the model, it is possible to employ a combination of different hardware platforms and operating systems for controlling the robot. Each client or server can run on its own hardware and operating system. Thus, the underlying hardware platforms can be heterogeneous.
- Besides the current two types, “variables” and “semaphores”, data in the Data Server can also be used for storing other types of structured information. For instance, one can store “rules”, “action”, “predicates”, etc.

To summarize, the Client-Server Architecture described in this thesis provides the necessary constructs for writing distributed control programs for mobile robot navigation. What one needs to do is to identify all the clients required in the navigation

program and write a separate module for each client. Usually, a client is associated with a specific subtask of the entire navigation task. The priority assignment to the client-server connection is another unique feature of the Client-Server Architecture. The above facilities were not found in the other control architectures discussed in Chapter 3.

## 4.4 Related Work

Other research projects which have some commonalities to my approach have been reported in the literature. Bagchi and Kawamura employed the blackboard approach for task decomposition and execution [Bagchi and Kawamura, 1992]. Therefore, all the modules in their system are controlled through a central blackboard manager. The blackboard is viewed as a software resource. They also employ *client* and *server* objects. For instance, a module can create a client *arm* object that corresponds to a physical robot arm associated with the server *arm* object. To some extent, this is similar to the hardware server in my implementation. However, their implementation does not have the capability of synchronization, information sharing, and interprocess communication provided by the Data Server in my implementation. Moreover, in the Client-Server Control Architecture, no central manager exists.

The system described in [Chocon, 1992] consists of a number of *functional subsystems*. A functional subsystem corresponds to a set of data, functions, and tasks and implemented as an object. A functional subsystem is allowed to have a direct access to the other subsystems through a client/server mechanism. There is no clear distinction between the client and server modules. It appears that peer-to-peer relationships are employed in the system. In my approach, the distinction between client and server modules is explicit.

TelRIP (TeleRobotics Interconnect Protocol) provides a “virtual” fully connected

network of user programs via a special type of process called *router* [Wise and Ciscen, 1992]. These user programs can either reside on the same processor or distributed across several processors. In the former, intraprocessor communications are handled by the router on the processor. While in the latter, interprocessor communications are directed from the router in one processor to the router on the other processor. Thus, the number of routers in the system will be the same as the number of processors.

To some extent, this approach is similar to the Task Control Architecture described in Section 3.1.3. Although, TelRIP and TCA were developed by two different institutions, I view the former as a precursor of the latter. The routers in TelRIP and the central server in TCA act as the “reflector” for all the message traffic in the system. The design of the Client-Server Control Architecture allows the messages for hardware control to be routed separately from those for interprocess communication.

## 4.5 Implementation

In the following sections, a more detailed description of the interprocess communication, client interface, and server interface functions is given. The initial implementation of the Client-Server Control Architecture is reported in [Dulimarta and Jain, 1993].

### 4.5.1 Interprocess Communication

The communication scheme is implemented using the Interprocess Communication (IPC) facilities available in the UNIX\* operating system. The possible choices of IPC are: (1) *shared memory*, (2) *semaphores*, (3) *message passing*, (4) *sockets*, and (5) *remote procedure calls* (RPC). Among these choices, sockets or RPC seem to be the most preferable mechanisms, since they also support communication across

---

\*UNIX is a trademark of AT&T Bell Laboratories.



different hosts (machines). The first three choices support communication within the same host only.

My implementation employs the socket-based communication mechanism. One of the reasons this approach was chosen is because sockets are widely used on the Internet. Also, socket-based communication has been adapted to run on personal computers as well. Therefore, this approach can be applied to a wide choice of computers.

To relieve the users of the system from writing low-level socket-related system calls, I have designed two C++ classes **SockStr** and **Server** that include methods for opening and closing connections, exchanging data, and processing requests from clients by a server.

#### 4.5.2 Client Interface Functions

Before a client can communicate with a particular server, an object of type **SockStr** has to be created.

```
class SockStr {  
public:  
    open_comm (char *);  
    close();  
  
    /* the following functions are overloaded */  
    send (...);  
    recv (...);  
};
```

Figure 4.4. A segment of **SockStr** class declaration.

Sending or receiving several different data formats is made possible by implementing the interface methods `send()` and `recv()` as overloaded functions<sup>†</sup>. Using this class, a client initiates a communication with a server by first calling `open_comm()` and then exchanging data using `send()` and `recv()`. Before exiting, the client should call `close()` to terminate the communication. A `SockStr` object can be viewed as a duplex communication channel between a client and a single server. Thus, a client that communicates with several servers has to allocate a separate channel to each server. An example of a client that connects to a server named “RServer” is given in Figure 4.5.

```
main()
{
    SockStr S;
    int speed;

    S.open_comm ("RServer");
    S.send (ROBOT_GETSPEED);
    S.recv (speed);
    S.close();
}
```

Figure 4.5. An example of a program communicating to RServer.

As shown in the above example, a client does not need to know the details of how the server obtains information about the robot speed. Instead, the servers are responsible for converting data or commands into and from a generic device-independent format known to the client. Besides controlling the operation of the hardware attached to it, a server can also be assigned an intermediate-level task. For instance, the camera server can be used not only for acquiring images, but also for extracting

---

<sup>†</sup>A *method* is a procedure associated with an object.

image features.

### 4.5.3 Server Interface Functions

The single and multiple process concurrent servers described in [Comer and Stevens, 1993] were used in the system. For these types of servers, the necessary methods needed to *partially* implement a server have been identified, and the **Server** class was designed accordingly. Since the actual interpretation of the requests varies from one server to another, the **Server** class provides a dynamic linking point to invoke the processing or interpretation of a specific request.

```
class Server {
public:
    open (char *);
    close ();
    mainloop (int (*) (int));
    forkloop (int (*) (int));
    exitloop ();
};
```

Figure 4.6. Public interfaces of **Server** class declaration.

Using the method `open()`, a server announces the given service name to the world. A client that wants to communicate with this server must use this name to initiate communication. To process all requests using a single process, a server would call the `mainloop()` method. A multi-process concurrent server can be established via the `forkloop()` method. Earlier, it was mentioned that the above C++ class *partially* implements a server since the processing of the client requests varies from server to server. For this reason, a server that calls either `mainloop()` or `forkloop()` must provide a function name that will process or interpret individual client requests. This function is referred to as the *request handler*. For each request, the request handler

is invoked with the “channel number” in which the request arrives. When the server does not want to serve any more requests, it should call the method `close()`. The method `exitloop()` is provided so that the server can terminate its infinite loop processing inside `mainloop()` or `forkloop()`. Since the last two methods run in an infinite loop, the method `exitloop()` has to be called asynchronously such as through an interrupt handler. An example of a server whose service name is “**XServer**” is given in Figure 4.7

Once a `SockStr` object associated with the communication channel is created, the request handler can use the `send()` and `recv()` methods to exchange data with the clients. These high-level interface functions greatly simplify the design of a server.

All requests from the clients to the sensors must be directed to the appropriate **HServer**. Enforcing this policy has an important effect on the overall sensor data processing. Since a sensor might be utilized by more than one client, access to the sensor has to be serialized and protected. This means that a request from one client should not interfere with the request of another client. This is particularly important when the sensor has the capability of executing asynchronously, i.e., it can execute a new command before the previous command has been completed. Thus, each **HServer** has to be capable of accepting multiple requests concurrently while providing protection to the sensor processing of an individual client. This should be accomplished without complicating the communication protocol used by the clients. In other words, a client should be able to employ a very simple protocol to initiate or terminate a communication session, and to transfer information to/from an **HServer**. Also, the sensor should appear to a client as being owned only by itself, i.e., a client should not have to worry if the sensor it is using is shared among several other clients.

Some advantages of this system are given below:

- The control scheme can be ported to a non Unix-based system as long as it provides multi-tasking and inter-module communication facilities.

```

Server S;

main()
{
    S.open ("XServer");
    S.mainloop (Request_handler);
    S.close ();
}

int Request_handler(int channel_id)
{
    /*--- create a SockStr for this "channel_id" ---*/
    SockStr CommCh(channel_id);
    int rval;
    char cmd;

    while (1) {
        if (CommCh -> dataReady() == 0) /* no data to read */
            break;

        rval = CommCh -> recv (cmd);
        if (rval == 0) {
            delete CommCh;
            return 0;
        }

        switch (cmd) {
            /*
             * following these lines are the codes
             * for interpret requests
             * that are sent thru CommCh.
             */
        }
    }
    return 1;
}

```

Figure 4.7. An example of a request handler in a server.

- The use of *sockets* or RPC for the communication scheme enables us to use a distributed computing approach when additional processors are added to the mobile robot. Also, when a wireless communication method is available, the processing can be distributed to other machines. Of course, when this approach is pursued, the communication bandwidth between the mobile robot and the other hosts must not be a bottleneck in the processing.
- By forcing all clients to communicate through a single **HServer** for each sensor, the system is guaranteed to be free of interference among clients' sensor processing.

As the generic model of a mobile robot, I have adopted the “virtual controller” model described in [Crowley, 1989]. This model is used to command a robot to achieve coordinated motion in a Cartesian coordinate space. Commands can be sent to the controller asynchronously, i.e., a new command can be given at any time without having to wait for the completion of the current command. The controller is composed of independent control of forward displacement and orientation.

The external interface protocol described in Crowley's controller includes the following commands: **Move**, **Turn**, **Stop**, **GetEstPos**, **GetEstSpeed**, **CorrectPos**, **ResetPos** [Crowley, 1989]. The virtual controller proposed by Crowley is organized into three layers. The top layer is the command interpreter. The middle layer is the main control loop that consists of a motor command generator and control parameter (vehicle position, velocity, covariance, incremental displacement) estimator. The bottom layer is the interface to the particular vehicle geometry.

## 4.6 Servers

Section 4.5 describes how to write the client program interface that communicates with the servers. The following sections describe the possible commands accepted

by each server in the system. These servers are: Data Server (**DServer**), Proximity Server (**PServer**), Robot Server (**RServer**), and Camera Server (**CServer**). For each server described here, there is a library of functions that acts as another layer of programming interface that runs on top of the **SockStr** class. The libraries are not part of the **SockStr** class, but they use the **SockStr** class to provide yet a higher level of abstraction. Using these libraries a user does not have to write the sequence of send/receive operations. Figure 4.8 shows the relationship between this layer, the user program, **SockStr** class, and the servers. Functions in the libraries are accessible only if the connection to the corresponding server has been established.

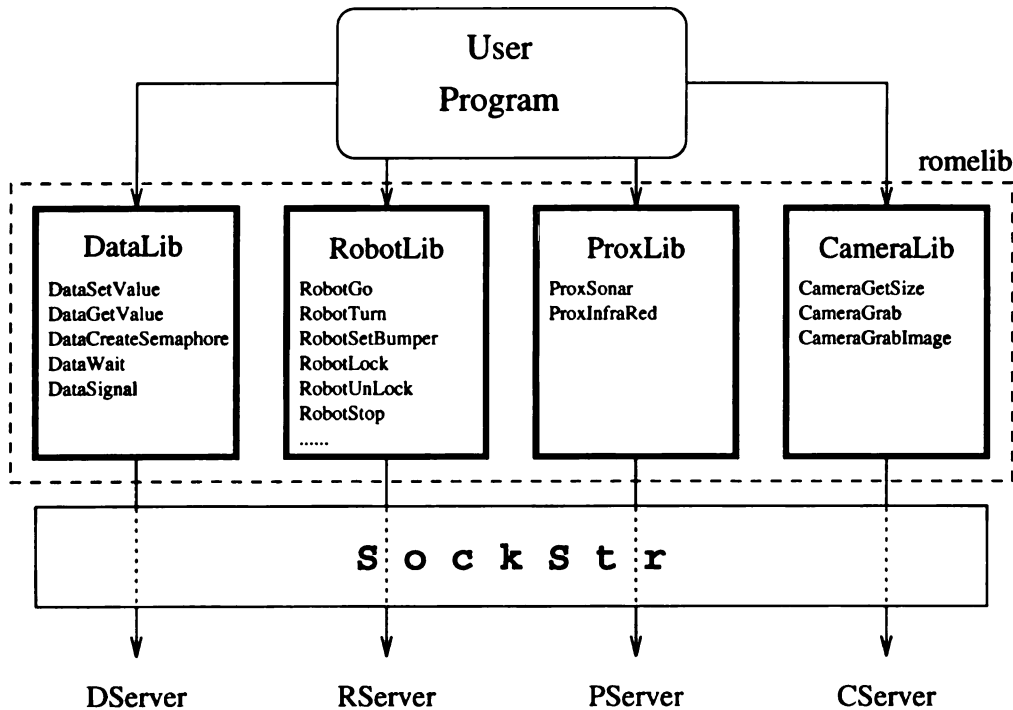


Figure 4.8. Library of functions defined on top of **SockStr** class.

### 4.6.1 DServer

This program implements the Data/Information Server in which all global variables used in the system are stored. Since variable names can be of any length, in the transmission to/from the server, they are preceded by a byte length value. For instance, the name “DRIVE” will be transmitted as six ASCII characters : ENQ (05 octal), ‘D’, ‘R’, ‘I’, ‘V’, and ‘E’. Internally, the server uses a two-level linked list for keeping all variable names. The first level stores the list of variable names defined by the clients to the Data Server. For each variable, there is a “history” of values associated with the variable. For instance, if the length of the history of a variable is  $N$ , only the last  $N$  values will be kept in the list. The user can set the length of the history of each variable independently. The history list is maintained in a FIFO manner, i.e., when a new value arrives, the oldest value will be discarded. Two types of variables in the Data Server are: (i) *semaphores* for synchronization of events among clients, and (ii) *ordinary variables* for sharing information among clients.

In the following sections, it is assumed that the connection to the Data Server has been established by the following code:

```

SockStr DS;

DS.open_comm ("DServer");

```

### Semaphores

For synchronization purposes, the Data Server initializes and maintains all semaphore identifiers used by the system. These identifiers are returned by the system call **semget()** in the SunOS operating system. To enable access by the clients, the Data Server associates each semaphore identifier with a variable name. A client who wants to operate on a semaphore has to send a request to the Data Server and use the variable name to refer to the semaphore. In my implementation, the Data Server



provides three services for manipulating semaphores via the following command bytes: **DATA\_CREATE\_SEM**, **DATA\_WAIT**, and **DATA\_SIGNAL**.

1. The function **DataCreateSemaphore()** can be used by a client to create a semaphore. The client must pass the name and initial value of the semaphore:

```
DataCreateSemaphore (DS, "$XYZ", 1);
```

The above function call creates the semaphore **\$XYZ** on the data server associated with variable **DS** and sets the semaphore's initial value to 1.

2. To increment the value of a semaphore, a user can write:

```
DataSignal (DS, "$XYZ");
```

3. To decrement the value of a semaphore, a user can write:

```
DataWait (DS, "$XYZ");
```

Using these three functions, the Data Server provides *semaphores* to the clients. Physically, the semaphores are defined only inside the Data Server. However, the synchronization using the semaphores can be effective even across several machines. The actual **up()** and **down()** operations on a semaphore are carried out by the Data Server upon requests from clients.

For single-process concurrent servers, there is a potential blocking by a client requesting a **DataWait()** operation on a semaphore, i.e., when the semaphore is not available. To avoid this situation, I have implemented the non-blocking **down()** operation on the semaphores (**down\_nowait()**). By observing the return value of the operation, a decision can be made whether the **down\_nowait()** operation was

successful or would have been blocked. In the latter operation, a list of socket identifier waiting for the specified “event” is maintained in the Data Server.

When a client requests a **DataSignal** operation on a semaphore, the Data Server consults the list of socket identifiers associated with the semaphore. If the list is not empty, the first socket identifier in the list will be notified to indicate that the event associated with the semaphore has occurred. Upon receiving the notification, the blocked client can continue processing. Also, the socket identifier is removed from the list and the actual **down()** operation is performed by the Data Server on behalf of the blocked client.

### Ordinary Variables

In my implementation, only integer variables and the following commands are accepted by **DServer**:

1. **Set Value.** This command is used to modify the value of a variable. Clients can execute this command by sending **DATA\_SETVALUE** request byte using the following function:

```
DataSetValue (DS, "PosX", 4);
```

The effect of the above command is to set the value of variable **PosX** to 4.

2. **Get Value.** This command is used to get the value of a variable from its history list. The format of this command is:

```
DataGetValue (DS, "PosX", -N);
```

3. **Set History.** This command is used to modify the length of the history to be maintained for a variable name. The format of this command is:

```
DataSetHistory (DS, "PosX", N);
```

In the last two commands,  $N$  determines which value of the variable is to be retrieved. If this value is zero, the current content of the variable will be retrieved. A value of  $-1$  denotes the previous value,  $-2$  denotes the value before the previous value, and so on.

#### 4.6.2 PServer

This server handles all requests to the Proximity Subsystem. Currently, two commands are supported: **PROX\_SONAR**, for acquiring all ultrasonic sensor readings, and **PROX\_IR**, for acquiring the proximity infrared detector readings.

- **PROX\_SONAR.** Upon sending a single byte (**PROX\_SONAR**), the client will receive a 4-byte length code and  $24 \times \text{sizeof}(\text{int})$  bytes of sonar readings. Each one of the sonar readings is a distance in millimeters. In the library, this function is implemented in **ProxReadSonar()**.

```
int sonar[24];
```

```
ProxReadSonar (PS, sonar);
```

- **PROX\_IR.** Upon sending a single byte (**PROX\_IR**), the client will receive a 4-byte length code and  $24 \times \text{sizeof}(\text{char})$  bytes of infrared detector readings. Each reading is a binary value and a value of 1 means that the corresponding infrared sensor detects an object within its detection range. Our mobile robot has only

eight infrared sensors, but a 24-element is returned since the controller board supports a maximum of 24 infrared sensors. The library function to be used for this purpose is `ProxReadInfraRead()`;

```
char infrared[24];

ProxReadSonar (PS, infrared);
```

- **PROX\_TIMEOUT.** A client can set the maximum range reading of individual sensors by sending this command byte.

```
int max_r[24], i;

for (i=0; i<24; i++)
    max_r[i] = 5000; /* set maximum range to 5000 mm */
ProxSetTimeout (PS, max_r);
```

### 4.6.3 RServer

This is the Robot Server and it acts as the interface between programs/modules and the TRC Labmate mobile robot. The set of commands accepted by the Robot Server is a subset of all commands understood by the Labmate. The implemented command subset is sufficient for the purpose of robot navigation. Our mobile robot, RoME, employs two independently controlled wheels located near the center of the base. When the two wheels spin at the same speed, the robot will either move in a straight line or turn with respect to its vertical axis, depending on the direction of spin of the two wheels. By applying different speeds (or velocities) to the two wheels, various types of motion configuration can be achieved. The microprocessor that controls the

operation of the TRC Labmate is a Motorola 68HC11 chip. The heading of the robot is calculated from the two wheel encoders.

The Labmate has seven operating modes [TRC, 1991]:

1. **Joystick Mode.** The Labmate can be controlled using a joystick. This is the default mode on power-up and reset.
2. **Go Mode.** The Labmate moves in a straight line at the current velocity setting. While in this mode, the Labmate can accept another command and execute it.
3. **Proportional Go Mode.** Similar to **Go Mode**, but without error integration.
4. **Continuous Turn Mode.** The robot turns a specified number of degrees in a specified radius. Upon completion of this command, the Labmate enters the **Go Mode**.
5. **Point-to-Point Go Mode.** The Labmate moves in a straight line by a specified distance. Upon completion of this command, the Labmate enters the **Go Mode**.
6. **Point-to-Point Turn Mode.** Similar to **Continuous Turn**, but using a trapezoidal control profile.
7. **Jog Mode.** The Labmate superimposes a turning mode on forward motion. This mode continues until the robot receives a **Go Mode**, or another **Jog Mode**.

The commands accepted by the Robot Server are described below and Table 4.1 summarizes all these commands. Most of the commands are for motion control and status enquiry. A command of this type corresponds to a primitive command in the Labmate. In addition, I have identified that in some situations there is a need for gaining exclusive control of the robot by a client. For this reason, a “locking”

Table 4.1. Summary of commands to RServer.

Command	Parameters
Adjust Position	dX, dY, dHeading
Get Bumper Status	-
Get Mode	-
Get Position	-
Get Speed Setting	-
Get Wheel Speed	-
Set GO Mode	Velocity
Set JOG Mode	Jog rate
Move	Distance, Vel, Accel
Set Bumper	Setting
Set Position	X, Y, Heading
Stop	Stop Mode
Turn	Degree, Vel, Accel
Lock	-
UnLock	-

mechanism was also implemented in the Robot Server. The functions related to this are **RobotLock()** and **RobotUnlock()**.

- Adjust Position (**ROBOT\_ADJPOS**). This command adjusts (increments or decrements) the robot position and heading with the given integer values.
- Get Bumper Status (**ROBOT\_GETBMPR**). This command is used to check the bumper status. Upon sending this command, the client will receive an integer value indicating the status of the bumpers. A zero value means that the bumper is not in contact with any object, while a non-zero value indicates whether the front or rear bumper is in contact with any object.
- Get Robot Mode (**ROBOT\_GETMODE**). Our TRC Labmate can operate in several modes. This command is used to enquire which mode the robot is currently in.

- Get Robot Position (**ROBOT\_GETPOS**). Get the robot's current heading and position. On return, the client will receive three integer values representing X-position, Y-position, and Heading in that order.
- Get Speed Setting (**ROBOT\_GETSPEED**). This command is used to enquire the most recent value passed to the robot controller to set its straight-line velocity. This value can be considered as an approximation to the actual straight-line velocity of the robot. The server will respond by sending an integer value indicating the speed setting.
- Get Wheel Speed (**ROBOT\_GETWSPEED**). Unlike the previous command, this command sends the actual speed of the robot. The value sent to the client is obtained by averaging the magnitude of left and right wheel velocities.
- Set Go Mode (**ROBOT\_GO**). This command sets the robot into the GO mode, where the robot will keep moving in a straight line with its current velocity setting. The integer value parameter specifies the velocity to be used in this mode.
- Set Jog Mode (**ROBOT\_JOG**). This command is similar to Set Go mode, except the robot is set into the JOG mode. The integer parameter specifies the jog rate to be used in this mode.
- Move (**ROBOT\_MOVE**). This command is used to make the robot move in a straight line. The only difference between this command and the Set Go mode command is that the robot will stop after it travels a certain distance. The first parameter specifies the distance the robot has to travel, the second parameter specifies its velocity, and the third parameter specifies its acceleration. A negative distance value will make the robot to move backward, while a negative velocity or acceleration means that the robot should use the current velocity and acceleration

settings.

- **Set Bumpers (ROBOT\_SETBMPR).** This command is used to specify whether the robot should ignore or watch for bumper switches. When the bumper switch is ignored, the robot will not stop even when the bumper is in contact with objects. A zero value given to the first parameter will make the robot ignore the bumper switch.
- **Set Position (ROBOT\_SETPOS).** This command is used for setting the internal registers of the robot that keep the position and heading information. The first two parameters are the X and Y positions that should be assigned to the robot. The third parameter is its heading.
- **Stop (ROBOT\_STOP).** This command is used for stopping the motion of the robot. The first parameter determines how the robot will be stopped. A non-zero value means the the robot should be stopped immediately using **emergency\_stop()** Labmate command. Otherwise, the robot will be stopped gradually using **pause()**.
- **Turn (ROBOT\_TURN).** This command makes the robot to change its heading by physically turning by a specified amount (in degrees). The first parameter determines the number of degrees the robot has to turn (positive value means clockwise turning). The second parameter determines the velocity to be used during the turn, and the third parameter is its acceleration. Negative values can be specified for the velocity and acceleration to indicate that the robot should use the current velocity and acceleration settings.

## Locking Mechanism

The locking mechanism used by the Robot Server is a token-based approach, i.e., to lock the Robot Server, a client has to obtain a “token” first. When a client requests



a lock through `RobotLock()`, the Robot Server will grant it if the lock is not being held by another client. When a client sends a request while the lock is being held by another client, the request will be postponed until the lock is available.

### Virtual X-Y Coordinate

The internal X-Y positions in the Labmate are stored as 16-bit signed integers. Using these values, the area in which the 2D positions can be correctly reported by the robot, is limited to  $65536 \times 65536$  millimeter squares. RServer makes this area virtually much larger. By anticipating the underflow/overflow in the 16-bit registers, RServer returns 32-bit signed integer values, thus enlarging the area by a factor of  $2^{32}$  ( $2^{16}$  on each side).

The underflow/overflow is detected only when a `ROBOT_GETPOS` request is received. Each time a new position is acquired from the robot, it is compared with the last acquired position. If the two values have different signs, it is a necessary condition for an overflow/underflow. For the sufficient condition, I assume that the magnitude difference of the two values exceeds 32768 ( $2^{15}$ ) millimeters. Theoretically, this assumption is incorrect, but in practice the time interval between two `ROBOT_GETPOS` requests is much shorter than the time needed for the robot to travel 32768 millimeters. Specifically, an overflow from  $X_{\text{old}}$  to  $X_{\text{new}}$  is detected when the following condition holds:

$$\text{sgn}(X_{\text{old}}) = +1 \wedge \text{sgn}(X_{\text{new}}) = -1 \wedge |X_{\text{new}} - X_{\text{old}}| > 2^{15}.$$

Similarly, an underflow is detected when the following condition holds:

$$\text{sgn}(X_{\text{old}}) = -1 \wedge \text{sgn}(X_{\text{new}}) = +1 \wedge |X_{\text{new}} - X_{\text{old}}| > 2^{15}.$$

#### 4.6.4 CServer

The Camera Server (**CServer**) controls the image grabbing operation of a single camera. When multiple cameras are used, multiple CServer processes have to be activated with different service names. In my implementation, the Camera Server periodically grabs a new image approximately every 0.5 seconds. The objective of this approach is to reduce the processing time when a client requests an image, because the Camera Server will just return the image that was grabbed previously. However, the client might not receive the most up-to-date image. The implemented commands in the Camera Server provide some basic interfaces for camera operation which include:

1. **CameraGetSize()** for acquiring the size of the image grabbed by the Camera Server.
2. **CameraGrab()** for obtaining the full image grabbed by the Camera Server.
3. **CameraGrabRow()** for obtaining a particular row in the image. This function is provided because, in some situations, a client is interested in only a small region of the entire image. Using this command, a client can select the image rows of interest.
4. **CameraLock()** and **CameraUnLock()** provide a mechanism for “freezing” the Camera Server. When a lock request is granted, the Camera Server will not update the image periodically. Both the Camera Server and the Robot Server use the same locking mechanism.

### 4.7 Controlling Multiple Robots

The client-server model enables distributed control of robot systems. This implies that the execution of all clients and servers can be distributed across a number of

processors connected to a computer network. The Client-Server Control Architecture described in this thesis does not require any particular media or method of computer communication. It only requires that socket-based communication is available between the two parties. The physical connection can be an ethernet cable, wireless modem, packet radio, or any other communication technology. This advantage provides us capability of controlling multiple robots using the Client-Server Control Architecture.

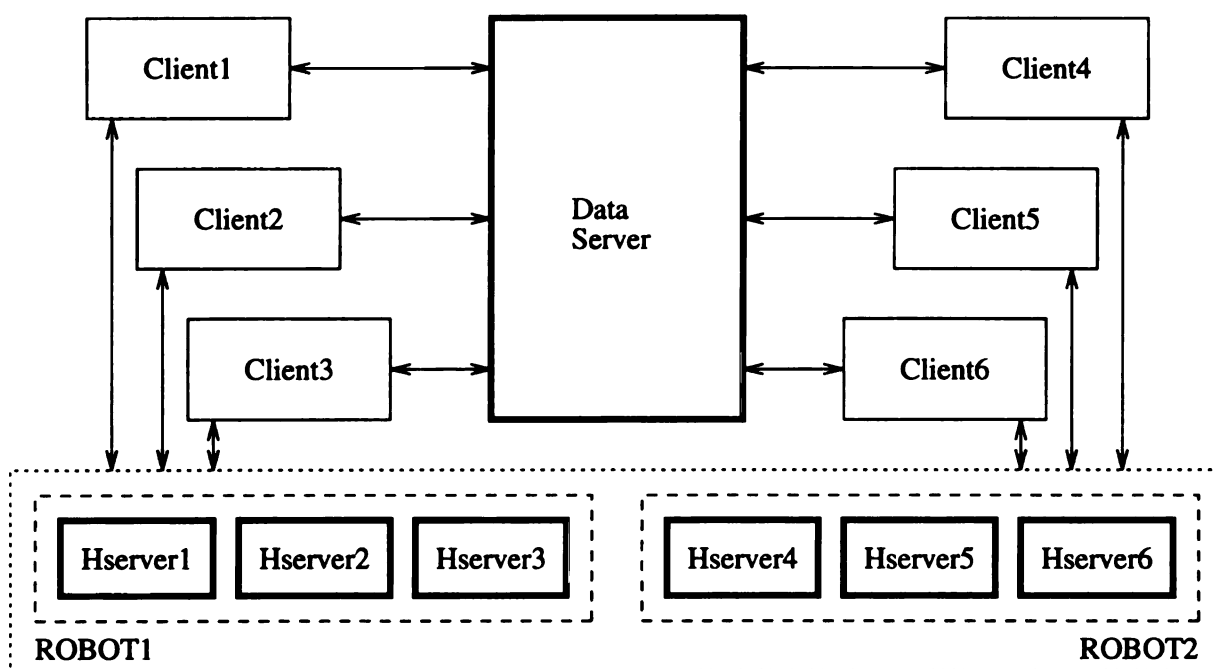


Figure 4.9. An example of a multiple Client-Server Control Architecture.

It was mentioned that the collection of servers in the Client-Server Control Architecture forms the core of the model. This core consists of a single Data Server (**DServer**) that serves as the common communication medium among the clients, and a number of hardware servers (**HServer**) that serve as interfaces to the various hardware/peripherals on the robot. When our client-server model is used for controlling multiple robots, each robot must be equipped with a set of **HServers** that acts as

the interface to its own hardware. However, the **DServer** is shared by *all* clients on *all* robots. For instance, when we have two identical robots, the core of the Client-Server Control Architecture will consist of a single Data Server and two replicas of **HServers**, each controlling a single robot. This model will be referred to as *multiple client-server model*. Figure 4.9 shows the configuration of a “double” client-server architecture. In the figure, the core servers are shown in thick boxes.

In this situation, every hardware server must have a unique identity. This is necessary to guarantee the clients will access the proper server. In general, the number of requests sent to the Data Server is less than those to the Hardware Servers. This is due to the fact that Hardware Servers serve as the gateway to the resources on the robot and resources are frequently needed and consulted by the client modules. Due to the relatively small number of requests to the Data Server, I claim that a single Data Server that acts as the common communication media for all the clients on multiple robots is sufficient. However, if a high bandwidth to the Data Server is necessary for a particular system, *local* Data Servers can be incorporated in the system. The attribute “local” is used to denote that the Data Server is used locally on a single mobile robot. This can be used for managing the facilities provided by the Local Data Server to all clients in a single mobile robot. Facilities intended for system-wide use should be directed to the “global” Data Server. It should be noted that the independence among the server modules should not be violated by the addition of local Data Servers. All servers are to be kept independent of each other.

## 4.8 Emulating Other Control Architectures

In this section, we will show how the other control architectures which have been reported in the literature can be emulated by the Client-Server Architecture. We will focus our attention to hierarchical architectures due to its wide use and to the

Colony-style architecture due to its uniqueness in its subsumptive property.

### 4.8.1 Emulating Hierarchical Systems

A hierarchical system consists of several modules configured in a number of layers. These modules exchange information either vertically with the modules in different layers or horizontally with the modules in the same layer. In the Client-Server architecture, communication between two modules can always be accomplished via the Data Server (See Figure 4.10). Therefore, for any hierarchical system configuration, its functionality can always be emulated in the Client-Server Control Architecture.

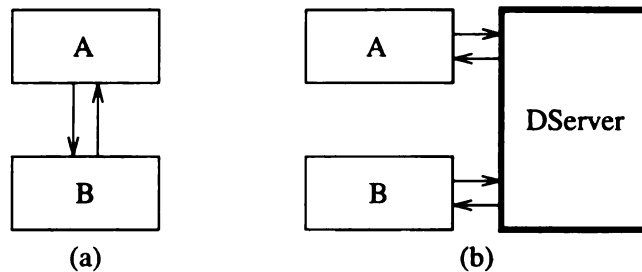


Figure 4.10. Intermodule communication in: (a) hierarchical systems, and (b) its equivalent construct in the Client-Server Architecture.

### 4.8.2 Emulating the Colony-Style Architecture

The unique characteristic of the Colony-Style Architecture lies in its subsumptive behavior, where higher level modules can take over the lower level modules. A more comprehensive description of this architecture is given in Section 3.2.2. Connell has showed that a pure suppression network defines a total ordering on the priority of the modules [Connell, 1990]. In this dissertation I present a method for determining the priorities of the modules even for a network containing inhibitor nodes. By assigning

each module a fixed priority value, Colony-style networks can be emulated in the Client-Server Architecture. A module in the Colony-style network becomes a client module in the Client-Server Architecture. Client modules with higher priority will be served before the other client modules. Thus a high-priority module subsumes the lower-priority ones.

Each module in the Colony-Style architecture can be modeled by Figure 4.11. The transfer function  $\varphi$  outputs a command using the information from the sensory input  $x$ , while the applicability predicate  $P$  decides whether the command should be gated as output  $y$ . We will denote a module in such a model with the following notation:

$$M(P_M, \varphi_M).$$

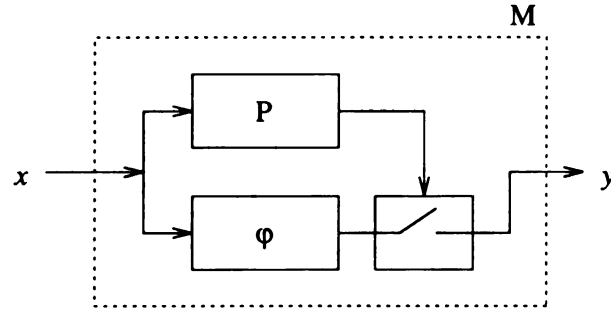


Figure 4.11. The internal structure of a module in Colony-Style architecture.

The semantics or functionality of a module in a Colony-Style architecture can be described by the following semantically equivalent block of statement<sup>†</sup>

---

<sup>†</sup>the subscript  $M$  is used to denote the entity belonging to module  $M$ .

```

if  $P_M(x)$  then
   $y := \varphi_M(x);$ 
fi

```

Modules in the Colony-style architecture are connected by either suppressor or inhibitor nodes. In the following sections, the semantics of networks that contain modules connected by inhibitor or suppressor nodes is described. Examples for two-module networks are presented, but the method given here can be iteratively applied to networks containing more than two modules.

### Inhibition Networks

In a Colony-style network, a module can inhibit the output of the other modules through an inhibitor node. The inhibition occurs when the dominant module is producing outputs.

Given two modules  $A(P_A, \varphi_A)$  and  $B(P_B, \varphi_B)$ , the inhibition of  $B$  by module  $A$  as shown in Figure 4.12 will be denoted by:

$$A(P_A, \varphi_A) \otimes B(P_B, \varphi_B).$$

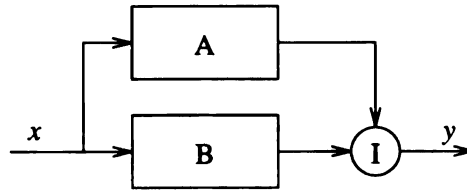


Figure 4.12. Inhibition network in the Colony-style architecture.

**Lemma 4.1**  $A(P_A, \varphi_A) \otimes B(P_B, \varphi_B)$  can be replaced by  $C(\neg P_A \wedge P_B, \varphi_B)$ .

**Proof:** When the module  $A$  inhibits the output of module  $B$ , the transfer function  $\varphi_A$  of  $A$  does not affect the output of an inhibition network. Only the applicability predicate  $P_A$  of module  $A$  is effective in determining the final output of the network. If the applicability predicate  $P_A$  is true, then the network will not produce any output. Outputs of the transfer function  $\varphi_B$  will be produced when the applicability predicate  $P_B$  is true and the applicability predicate  $P_A$  of the inhibiting module is false. It should be clear now that the behavior of an inhibition network can be described by the following code segment:

```

if  $\neg P_A(x) \wedge P_B(x)$  then
   $y := \varphi_B(x);$ 
fi

```

In fact, the above code segment describes the semantics of a module whose applicability predicate is  $\neg P_A \wedge P_B$  and whose transfer function is  $\varphi_B$ .  $\square$

Using the above lemma, any Colony-style network can always be transformed into an equivalent network without inhibition nodes.

### Suppression Networks

Given two modules  $A(P_A, \varphi_A)$  and  $B(P_B, \varphi_B)$ , the network containing a suppression of  $B$  by module  $A$  as shown in Figure 4.13 will be denoted by:

$$A(P_A, \varphi_A) \oplus B(P_B, \varphi_B).$$



Here, we will say that  $A$  *suppresses*  $B$  or  $A \downarrow B$  for short. This relation is transitive, *i.e.*, if  $A \downarrow B$  and  $B \downarrow C$ , then  $A \downarrow C$ <sup>§</sup>. It is also associative, *i.e.*,  $(A \downarrow B) \downarrow C = A \downarrow (B \downarrow C)$ . The expression  $(A \downarrow B) \downarrow C$  can be interpreted as a subnetwork containing  $A$  and  $B$  that suppresses  $C$ .

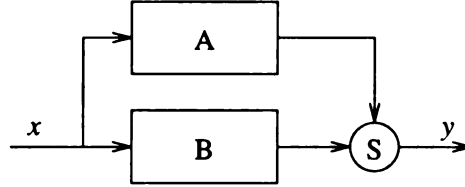


Figure 4.13. Suppression network in the Colony-style architecture.

When the output of a module  $A$  is connected to the output of module  $B$  via a suppressor node, then module  $A$  dominates the output if its applicability predicate returns a logical TRUE value and the result of the transfer function  $\varphi_A(x)$  is observed on the output line  $y$ . On the other hand, if module  $A$  is not suppressing the output of  $B$ , then  $B$  continues to operate as if module  $A$  does not exist. The behavior described above can be depicted by the following code segment:

```

if  $P_A(x)$  then
   $y := \varphi_A(x);$ 
else
  if  $P_B(x)$  then
     $y := \varphi_B(x);$ 
  fi
fi
  
```

---

<sup>§</sup>I do not distinguish the configuration where  $A$  “directly” suppresses  $B$  vs.  $A$  suppresses  $B$  via a number of intermediate suppressor nodes.

The above code segment indicates that the applicability predicate  $P_A$  will always be checked prior to  $P_B$ . In other words, the module  $A$  has a higher priority than module  $B$ . Intuitively, the “higher” the position of a module in a suppression network, the higher is its priority.

### Determining Module Priorities in Suppression Networks

In the following section, an algorithm for determining the priority of each module in a suppression network is presented.

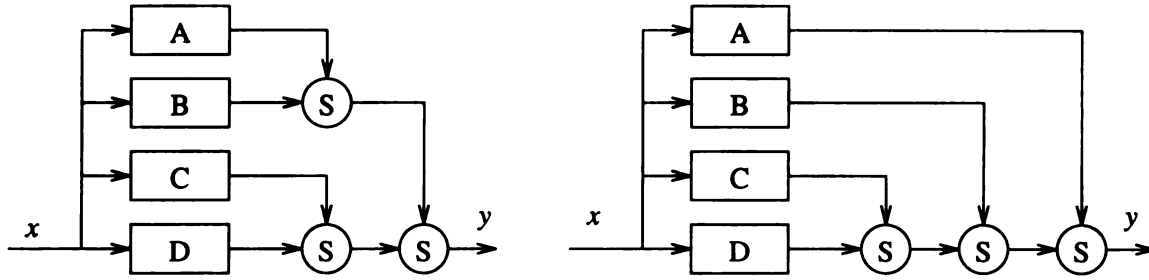


Figure 4.14. Two different suppression networks with the same priority assignment.

Figure 4.14 shows two networks with the same priority values assigned to each module. The priorities in a suppression network can be obtained by first constructing a binary tree that represents the intermodule connection in the network, and then assigning priority from the order of visit of the node by a *forward* traversal algorithm in the binary tree (See Definition 4.1). Since inhibitor nodes can be removed from the network by the method described in Section 4.8.2, the approach presented here can also be applied to a general Colony-style network.

In the following sections,  $T(V, E_l, E_r, v)$  denotes a binary tree, where  $V$  is the set of nodes,  $E_l$  ( $E_r$ ) is the set of links to left (right) children, and  $v \in V$  is the root of the tree. A suppression network will be denoted as  $N(M, S, L_d, L_i, \sigma)$ , where  $M$  is the set

of all modules in  $N$ ,  $S$  is the set of its suppressor nodes,  $L_d, L_i \subset (M \times S) \cup (S \times S)$ , are the set of dominant and inferior links, respectively, and  $\sigma \in S$  is the distinguished suppressor node connected to the output of the network. For instance, the network on the left side in Figure 4.14 can be represented by  $N(M, S, L_d, L_i, s_3)$ , where:

$$\begin{aligned} M &= \{A, B, C, D\} \\ S &= \{s_1, s_2, s_3\} \\ L_d &= \{(A, s_1), (C, s_2), (s_1, s_3)\} \\ L_i &= \{(B, s_1), (D, s_2), (s_2, s_3)\}. \end{aligned}$$

**Binary Tree Construction** The following paragraphs show how to construct a binary tree  $T$  from a suppression network  $N$ . The number of interior nodes in  $T$  is equal to the number of suppressor nodes in  $N$ , and the number of leaves in  $T$  is equal to the number of modules in  $N$ . On a suppressor node, the output of the suppressing module will be referred to as *dominate* arrow, while the output of the suppressed module will be referred to as *inferior* arrow. The binary tree  $T$  is constructed from  $N$  using the replacement rules given in Table 4.2.

Table 4.2. Replacement rules for binary tree constructions.

Suppression Network $N(M, S, L_d, L_i, \sigma)$	Binary Tree $T(V, E_l, E_r, v)$
$\sigma$ Suppressor nodes $S$ Dominate arrows $L_d$ Inferior arrows $L_i$ Module $M$	$v$ Interior nodes of $T$ Links to right child $E_r$ Links to left child $E_l$ Leaf nodes

It is obvious that in Table 4.2,  $V = M \cup S$ . Figure 4.15 shows some examples of binary trees constructed from their corresponding suppression networks.

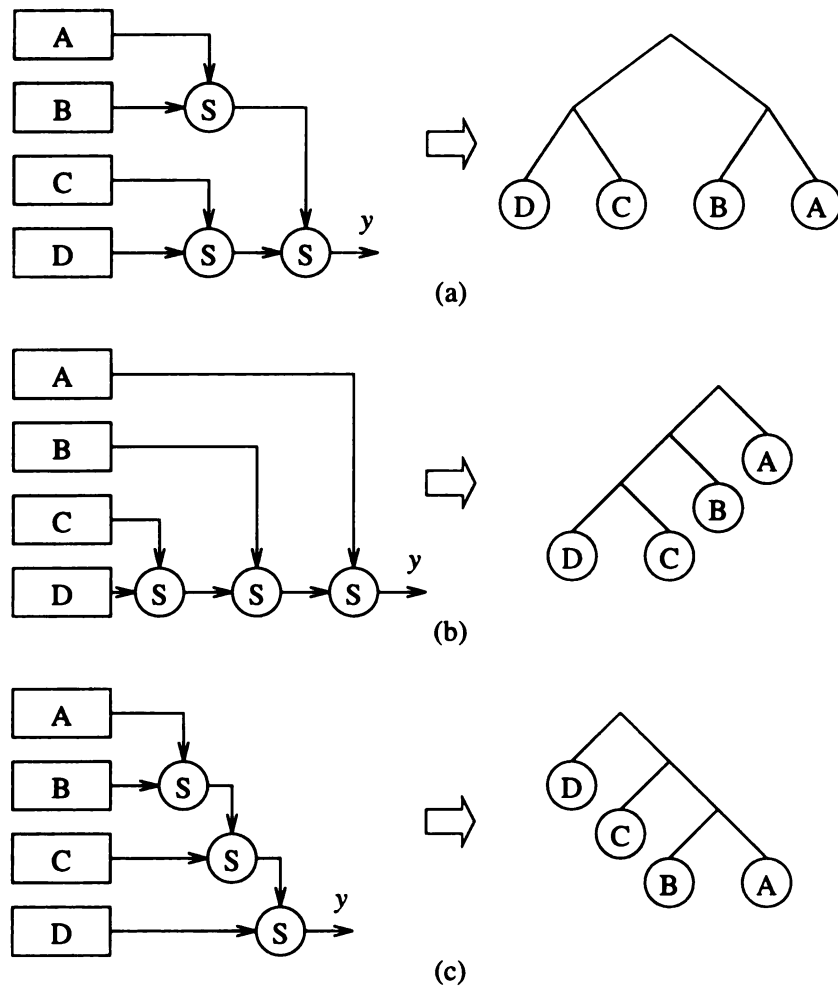


Figure 4.15. Binary trees constructed from suppression networks.

**Binary Tree Traversal** For clarification of presentation, the following definition will be used in this thesis.

**Definition 4.1** *A forward binary tree traversal algorithm is a tree traversal algorithm that visits the left subtree prior to visiting the right subtree.*

**Definition 4.2** *The order of visit of a traversal algorithm on a tree  $T(V, E)$  with  $K$  nodes is a one-to-one mapping:*

$$\mathcal{O} : V \longrightarrow \{1, 2, \dots, K\},$$

*where for any  $x$  and  $y$  in  $V$ ,  $\mathcal{O}(x) < \mathcal{O}(y)$  if and only if node  $x$  is visited before node  $y$ .*

By using a forward algorithm for traversing the tree constructed from a suppression network, the order of visit of the leaf nodes can be used to determine the priority of the corresponding module in the suppression network. The node visited first has the lowest priority, while the node visited last has the highest priority. The order of visit of a forward algorithm applied to the examples in Figure 4.15 is D–C–B–A.

**Lemma 4.2** *Given a tree  $T$  constructed from a suppressor network  $N$  using the replacement rules in Table 4.2, and two modules  $A$  and  $B$ , then*

$$\mathcal{O}(A) > \mathcal{O}(B) \iff A \downarrow B.$$

**Proof:** Let  $s$  be the closest common ancestor of  $A$  and  $B$ , then  $s$  must be a suppressor node in  $N$ . First we want to show that  $A \downarrow B \implies \mathcal{O}(A) > \mathcal{O}(B)$ . Since module  $A$  can suppress module  $B$ , then  $A$  must be within a subnetwork connected through the dominate arrow of the suppressor node  $s$ , and module  $B$  is within a subnetwork connected to the inferior arrow of  $s$ . In  $T$ , node  $A$  will be on the right

subtree of interior node  $s$  and node  $B$  will be on its left subtree. Therefore, using a forward traversal algorithm,  $\mathcal{O}(B) < \mathcal{O}(A)$ .

Now we want to show that  $\mathcal{O}(A) > \mathcal{O}(B) \implies A \downarrow B$ .  $\mathcal{O}(A) > \mathcal{O}(B)$  implies that the forward traversal algorithm visits  $B$  before it visits  $A$ . This also implies that  $B$  resides on the left subtree while  $A$  resides on the right subtree of  $s$ . Four different configurations will be considered here:

1. Both  $A$  and  $B$  are immediate children of  $s$ . Here, the outputs of  $A$  and  $B$  are directly connected to the suppressor node. Moreover,  $A$  is connected to the dominate arrow and  $B$  is connected to the inferior arrow.
2. Only  $A$  is the immediate child of  $s$ . In the suppression network  $N$ ,  $A$  suppresses the subnetwork containing  $B$ .
3. Only  $B$  is the immediate child of  $s$ ;  $A$  is inside a subnetwork whose output is connected to the same suppressor node as that of  $B$ . When the output of  $A$  reaches the output of the subnetwork, it suppresses the output of  $B$ .
4. Neither  $A$  nor  $B$  are immediate children of  $s$ ;  $A$  resides in a subnetwork  $N_A$  and  $B$  resides inside another subnetwork  $N_B$ . When the output of  $A$  reaches the output of  $N_A$ , it will suppress the output of  $N_B$ , hence it also suppresses the output of module  $B$ .

□

## 4.9 Summary

In this chapter the client-server model for coordination of a distributed program was described. In addition, the other two types of processes, peers and filters, that exist in

distributed programs were also discussed briefly. Some major characteristics of client and server processes were also explained. In addition, some of the important issues in the client-server interaction were also discussed. Although my implementation of the client-server model employs a socket-based communication, the Remote Procedure Call mechanism was briefly described. In Section 4.3 the details of the Client-Server Control Architecture used in my experiments were given by describing how the interprocess communication, server interface functions, and client interface functions are implemented using C++ classes. The Client-Server Control Architecture has been implemented on a SunOS 4.1 operating system. The size of the server source code is approximately 2500 lines. This small size is possible due to the use of C++ in my implementation. We also discussed the internal implementation of the Data Server, Proximity Server, Robot Server, and Camera Server. Finally, we showed that it is possible to use the Client-Server Control Architecture for controlling multiple robots and presented an algorithm for converting hierarchical systems and Colony-style networks to the Client-Server Architecture.

# CHAPTER 5

## Indoor Navigation

The generic structure of the Client-Server architecture was explained in Chapter 4. This chapter describes how the architecture was employed in controlling our mobile robot for indoor navigation. The initial implementation of the navigation system can be found in [Dulimarta and Jain, 1993]. In my experiments, the robot was commanded to move from one place to another in the Engineering building by giving the room numbers of the initial and final positions. Initially, the robot is provided its true heading. I have used the convention that North is 0 degrees, West is 90 degrees, South is 180 degrees, and East is 270 degrees. This choice follows the coordinate frame used by TRC in its LABMATE mobile robots [TRC, 1991].

A common approach to robot navigation within a dynamic environment is to use both global and local navigation plans. Global navigation is more likely to be a symbolic approach while local navigation is not. Humans use a similar strategy; for global navigation, we think in terms of coarse directions. For instance, one possible route to go from city X to city Y is via freeways I-xx South, I-yy South, I-zz West, and so on. On the other hand, during the local navigation on each freeway, we think in terms of how to control the steering wheel so that our car stays in the proper lane of the freeway at all times.

When applied to a mobile robot, global navigation deals with establishing its



coarse path, while the local navigation controls the robot so that the given path is followed as closely as possible and the robot does not run into any obstacles. In my indoor navigation program, the following modules were developed:

- Path Planner
- Navigator
- Ceiling Light Tracker
- Local Mapper
- Door Number Plate Detector

In the context of the client-server control architecture, the above modules should be considered as clients of the servers discussed in Chapter 4. Figure 5.1 shows the above clients and the data shared among the client modules.

## 5.1 World Representation

I have chosen to use the *StickRep* attributed graph representation developed by [Walsh, 1992], because it is well-suited for representing the hallway structure of most large buildings. Usually, wall segments and doors form a closed contour, therefore, the attributed graphs have a ring-like structure. An arc in the graph represents either a wall segment or a door. A node stores information about the topological connectivity of the arcs. Two entities are stored in a node: adjacency angle and node name. Physically, nodes in the graph correspond to vertical edges in the building, such as junctions between wall segments and door jambs. An arc contains information about edge type, length of the edge, attributes of the edge, and the projected positions of the ceiling lights in the hallway to the edge.





Figure 5.2 shows a partial map of the third floor of the MSU Engineering Building. A complete map is given in Appendix A.

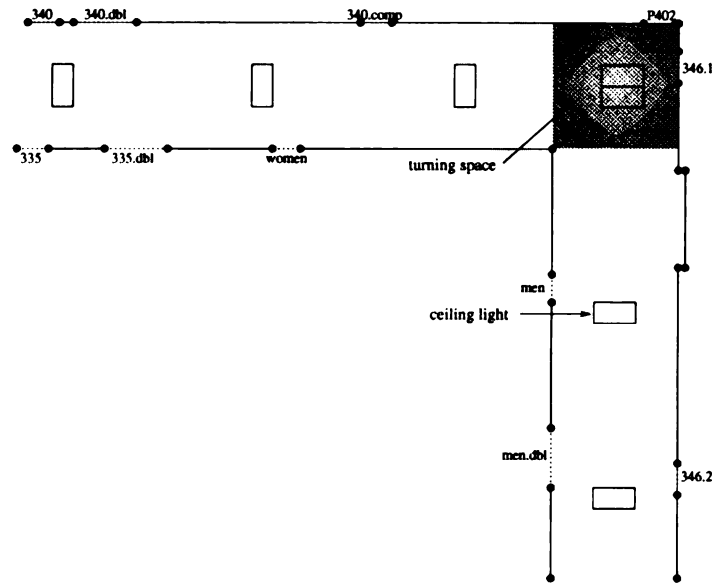


Figure 5.2. A partial map of the third floor of the MSU Engineering Building.

### 5.1.1 Map Construction

In the following sections, the information stored in the maps and how it was obtained from the third floor of the MSU Engineering Building are described in more detail.

#### Vertical Edges

A map is constructed as if a human operator were tracing the walls and doors on his/her left side. Each time a door or a wall segment is encountered, an edge will be created in the map and the appropriate information will be filled in. At the same time, a node will be created to represent the junction between two wall segments or between a door and a wall. To clarify the presentation, consider a hypothetical

structure shown in Figure 5.3.

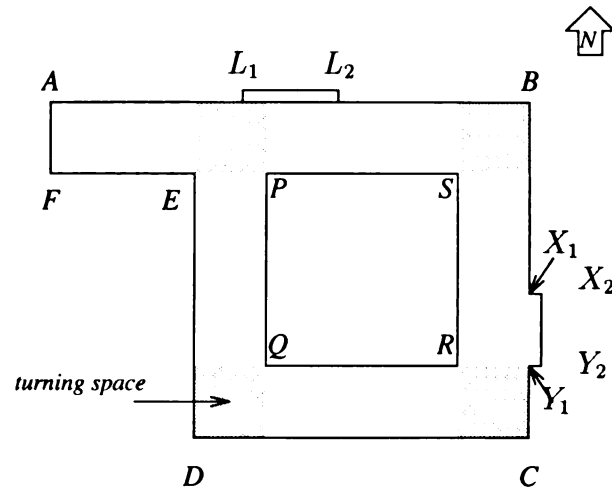


Figure 5.3. A hypothetical building structure.

In this example, there are two closed contours: A-B-C-D-E-F and P-Q-R-S. This structure is represented as two rings in the StickRep representation. The first graph (ring), which represents the outer contour, will have twelve nodes and twelve edges and the second ring will have eight nodes and eight edges. Using the convention given above, the outer ring will be stored in the following order: A,  $L_1$ ,  $L_2$ , B,  $X_1$ ,  $X_2$ ,  $Y_2$ ,  $Y_1$ , C, D, E, and F while the inner ring will be stored in the following order: P, Q, R, and S. On North side of the outer ring, there is a door labeled  $L_1L_2$ , and on the east side of the outer ring there is a small indentation labeled  $X_1X_2Y_2Y_1$ .

The adjacency angle at a node is determined as follows. Consider node  $L_1$  and the two adjacent edges  $AL_1$  and  $L_1L_2$ . Here, with respect to node  $L_1$ , the edge  $L_1L_2$  is considered as the “next” edge and the edge  $AL_1$  as the “previous” edge. Hence, the adjacency angle at  $L_1$  is the positive angle from  $L_1L_2$  to  $AL_1$  which is  $180^\circ$ . Using this convention, the magnitude of the adjacency angles of nodes A, B,  $X_2$ ,  $Y_2$ , C, D, and F is  $270^\circ$ , and the magnitude of the adjacency angles of nodes E,  $X_1$ , and  $Y_1$

is  $90^\circ$ . Nodes representing corners in the actual building are encoded by a negative value in the adjacency angle; the adjacency angle of node  $E$ , for instance, is  $-90^\circ$ . On the other hand, nodes  $X_1$ ,  $X_2$ ,  $Y_1$ , and  $Y_2$  are not considered as corner nodes, so their adjacency angles are all positive. The two rings of the structure shown in Figure 5.3 are given in Figure 5.4. This representation has a drawback in that the relative position between items across two rings cannot be determined. This drawback can be overcome by adding entries that link two rings in some predetermined positions.

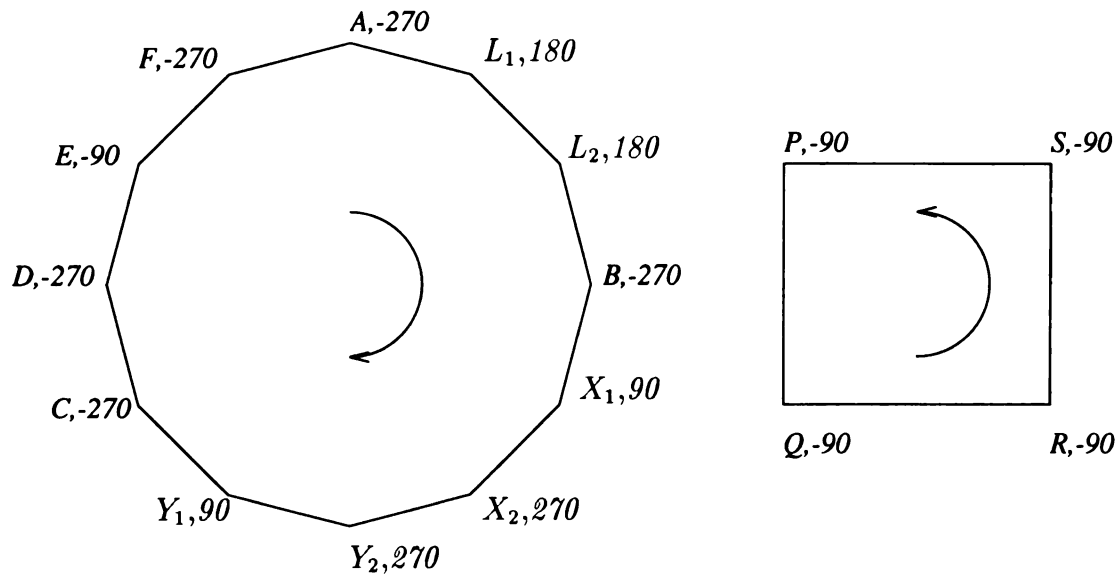


Figure 5.4. A StickRep representation of the structure shown in Figure 5.3.

### Ceiling Lights

In the navigation process, ceiling lights are used as a landmark to be tracked by the robot. For this purpose, the positions of the ceiling lights are encoded into the maps by projecting the “reference point” of the lights to the edge closest to the light. Here, the “reference point” is the axis of symmetry of the ceiling light which is perpendicular

to the edge.

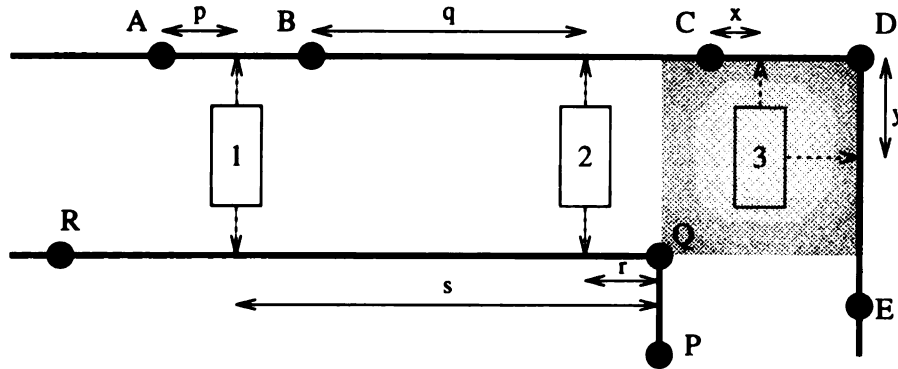


Figure 5.5. Ceiling light projection for three lights (1,2,3) in a hallway.

Figure 5.5 shows a situation where two ceiling lights are projected onto two different rings and a ceiling light residing in a turning space. A “turning space” is the intersection of one or more corridors. The outer ring is traced in the direction A–B–C–D–E, while the inner ring is traced in the direction P–Q–R. In the given figure, edges AB and BC have one ceiling light each, and edge QR has two ceiling lights. In some cases, ceiling lights that reside in a turning space have to be encoded differently in the map. For instance, ceiling light 3 is projected twice, i.e., onto edges CD and DE. To indicate that the two projections come from the same ceiling light, the offsets are given as negative values. Table 5.1 shows the value of  $\lambda$  and the light offsets  $d_1, d_2, \dots, d_\lambda$  for each edge.

Table 5.1. Encoding of ceiling positions in the StickRep.

Edge	$\lambda$	$d_1, d_2, \dots, d_\lambda$
AB	1	$p$
BC	1	$q$
CD	1	$-x$
DE	1	$-y$
QR	2	$r \quad s$

## 5.2 A Model of Robot Navigation

Using global and local navigation as described earlier, we now formalize a model for sensor-based robot navigation. I claim that the navigation system is composed of the following components:

1.  $\mathcal{W}$ : the subspace where the robot environment is defined. Typically, this space is either two- or three-dimensional.
2. A set of static obstacles  $\{S_1, S_2, \dots, S_m\}$ , each  $S_i$  is a subset of  $\mathcal{W}$ . The *static free space*  $\mathcal{F}$  can be represented by  $\mathcal{F} = \mathcal{W} - (\cup_{i=1}^m S_i)$ .
3. A set of dynamic obstacles  $\{D_1, D_2, \dots, D_n\}$ , with their unknown trajectories  $\mathbf{x}_1(t), \mathbf{x}_2(t), \dots, \mathbf{x}_n(t)$ .
4. The initial position ( $X_i$ ) and the final position ( $X_f$ ) of the robot, where  $X_i, X_f \in \mathcal{F}$ .
5. The desired path  $\Pi_{X_i, X_f}(\mathbf{x})$  from  $X_i$  to  $X_f$  to be followed by the robot.
6. Controllable parameters of the robot: orientation  $\theta(t)$  and position  $\xi(t)$  at time  $t$ . Using these parameters, the footprint of the robot at time  $t$ ,  $Fp(t)$ , can be defined as a function of  $\theta(t)$  and  $\xi(t)$ .



7. If there are  $s$  sensors in the system, then let  $V_i(t), i = 1, 2, \dots, s$  denote the subspace of  $\mathcal{W}$  observed by sensor  $i$  at time  $t$ .
8. The neighborhood of the robot:  $\mathcal{N}(t) = \cup_{i=1}^s V_i(t)$ . Thus,  $\mathcal{N}(t)$  represents the subspace that lies within the range of the robot sensors. The free subspace of  $\mathcal{N}(t)$  can be denoted by  $(\mathcal{N}(t) \cap \mathcal{F}) - (\cup_{i=1}^n \mathbf{x}_i(t))$ . Local navigator uses this free subspace to guide the robot so as to make it follow the global path  $\Pi_{X_i, X_f}(\mathbf{x})$  as closely as possible.

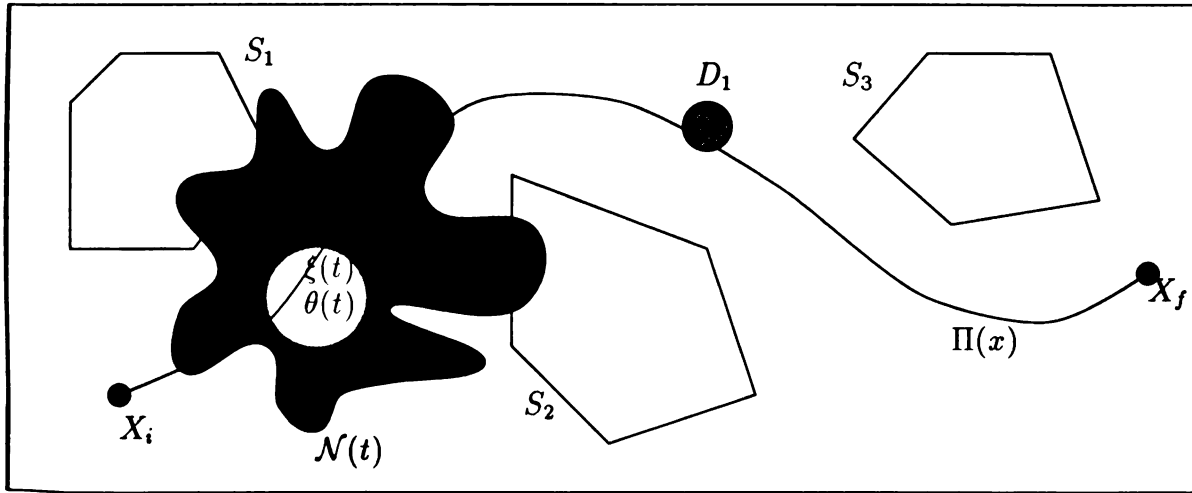


Figure 5.6. Navigation model.

An obstacle  $X$  (static or dynamic) is visible if it lies within the current sensor range of the robot, i.e:

$$Visible(X) \equiv X \cap \mathcal{N}(t) \neq \emptyset. \quad (5.1)$$

Figure 5.6 depicts the navigation model where  $\mathcal{W}$  is a two-dimensional robot workspace. In the figure, the static obstacles  $S_1$  and  $S_2$  are visible to the robot, while  $D_1$  and  $S_3$  are not.

Using the above model, the problem of robot navigation “along” the path  $\Pi_{X_i, X_f}(\mathbf{x})$  can be formulated in terms of the following goals:

1. The destination point  $X_f$  should be reachable in a finite amount of time. Expressed in a predicate calculus expression, this goal becomes:

$$(\exists t_f) \ni [(t_f < \infty) \wedge (\xi(t_f) = X_f)]. \quad (5.2)$$

2. Another capability that the robot must have is *obstacle avoidance*. This can be expressed as the following predicate:

$$(\forall t)(\forall X) [Visible(X) \implies Fp(t) \subset (\mathcal{N}(t) - X)], \quad (5.3)$$

which translates to the interpretation that at any given time, the robot does not touch any visible obstacles.

3. The robot should follow the path  $\Pi(\mathbf{x})$  as closely as possible. This implies that the actual position of the robot should not be too far from the planned path and the orientation of the robot should be about the same as the tangent vector to the path at the closest point. More formally, first we would like to minimize

$$\|\xi(t) - \Pi\|, \quad (5.4)$$

which is the distance between the robot and the desired path. Now, let us denote  $\hat{\xi}_t$  to be the closest point on the path to the robot, i.e.,

$$(\forall \bar{\xi} \in \Pi) [ \|\hat{\xi}_t - \xi(t)\| \leq \|\bar{\xi} - \xi(t)\| ].$$

The orientation constraint can be accomplished by minimizing

$$\theta(t) \cdot \nabla \Pi(\mathbf{x})|_{\mathbf{x}=\hat{\xi}_t}. \quad (5.5)$$

It is possible to assign each goal to a separate computing agent or client and let them run concurrently. Each client should then just concentrate on satisfying the goal assigned to it.

### 5.3 Path Planner

The Path Planner implements the global navigation routine in the system. Given the initial position  $X_i$  and final position  $X_f$  of the robot, Path Planner locates these positions in the map and determines the “optimal” path between the two positions. The path is segmented at corner points into a number of straight line subpaths. Suppose the path  $\Pi$  between  $X_i$  and  $X_f$  has been segmented into a sequence of subpaths  $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n$ . This will be denoted as

$$X_i [\Pi] X_f = (\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_n).$$

Using this strategy, on a path containing some corner points, the robot is commanded to move from the initial position  $X_i$  to the first corner, then to the second corner, and so on until it finally reaches the last corner and proceeds to the final position  $X_f$ . Otherwise, if no corners exist in the path, the robot will be commanded to move directly from the initial to the final position.

Furthermore, a straight subpath  $\mathcal{S}_i$  can be decomposed into at most three segments  $\mathcal{S}_i^l$ ,  $\mathcal{S}_i^d$ , and  $\mathcal{S}_i^c$ , in which the robot operates in different modes. Also, to denote that the robot starts at location  $X$  and ends at location  $Y$  in subpath  $\mathcal{S}_i$ , the following

notation will be used:

$$\mathcal{S}_i \equiv X [\mathcal{S}_i^l] I_1 [\mathcal{S}_i^d] I_2 [\mathcal{S}_i^c] Y,$$

where  $I_1$  and  $I_2$  are two intermediate points along the subpath  $\mathcal{S}_i$ . When the intermediate points are not of interest, the above notation will be written in a simplified notation as:

$$\mathcal{S}_i \equiv X [\mathcal{S}_i^l; \mathcal{S}_i^d; \mathcal{S}_i^c] Y.$$

In segment  $\mathcal{S}_i^l$ , the robot moves from the starting point  $X$  to the intermediate point  $I_1$  while tracking the ceiling lights. Point  $I_1$  corresponds to the location where the last ceiling light in subpath  $\mathcal{S}_i$  is seen by the robot. The superscript  $l$  is used to indicate that the robot is operating in “light-tracking” mode. Along the segment  $\mathcal{S}_i^d$ , the robot operates in “dead-reckoning” mode by relying only on the *approximate* distance between the last ceiling light in subpath  $\mathcal{S}_i$  to the center of the “turning space”  $\mathcal{T}_{i,i+1}$  between subpaths  $\mathcal{S}_i$  and  $\mathcal{S}_{i+1}$ . Here, the turning space of the two paths is defined as the intersection between the two corridors covering the path. More formally,

$$\mathcal{T}_{i,j} = \{C_i \cap C_j | \mathcal{S}_i \in C_i \wedge \mathcal{S}_j \in C_j\},$$

where  $C_i$  and  $C_j$  are corridors that cover the subpaths  $\mathcal{S}_i$  and  $\mathcal{S}_j$ , respectively. In the turning space, the robot stops at point  $I_2$  and then leaves the turning space and enters the initial part of subpath  $\mathcal{S}_{i+1}$  and stops at  $Y$ . Within this context, the corridor covering  $\mathcal{S}_i$  will be denoted as the “incoming” corridor, while the corridor covering  $\mathcal{S}_{i+1}$  will be denoted as the “outgoing” corridor. The superscript  $c$  in  $\mathcal{S}_i^c$  indicates that the robot operates in the “leave-corner” mode while traveling along the segment. Figure 5.7 shows the relation between a path with one corner and the operating modes of our robot.

Using this notation, a path  $\Pi_{AB}$  from  $A$  to  $B$  consisting of two corners and three

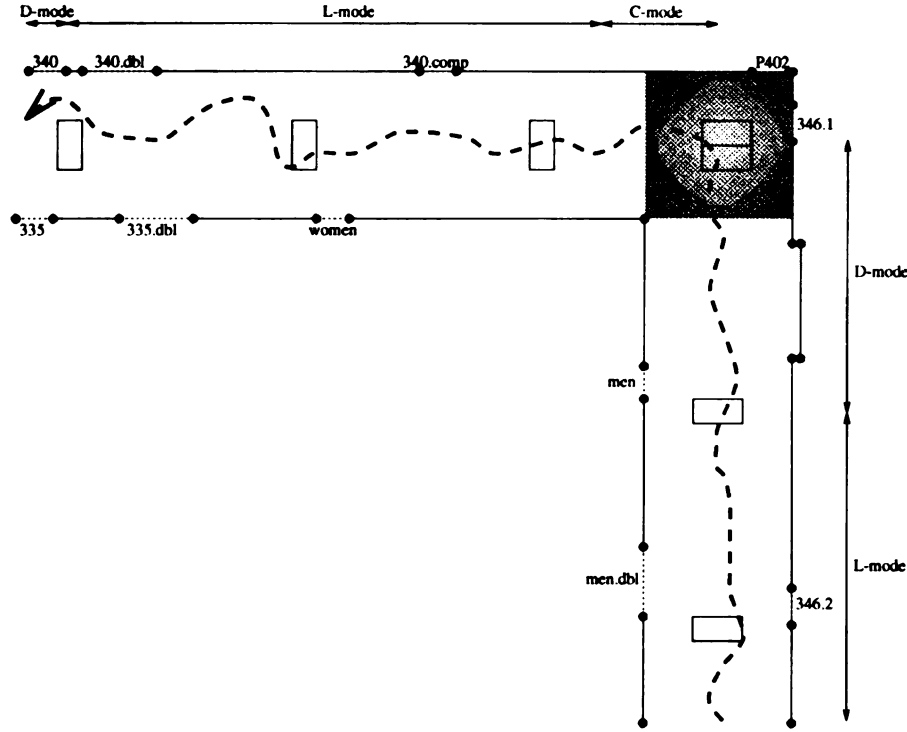


Figure 5.7. Different operating modes of the robot.

subpaths can be written as:

$$A[\Pi_{AB}]B = (A [\mathcal{S}_1^l; \mathcal{S}_1^d; \mathcal{S}_1^c] U_1, U_1 [\mathcal{S}_2^l; \mathcal{S}_2^d; \mathcal{S}_2^c] U_2, U_2 [\mathcal{S}_3^l; \mathcal{S}_3^d; \mathcal{S}_3^c] B),$$

where  $U_1$  is a point in the turning space between  $\mathcal{S}_1$  and  $\mathcal{S}_2$ , while  $U_2$  is a point in the turning space between  $\mathcal{S}_2$  and  $\mathcal{S}_3$ .

The Path Planner does not actually control the movement of the robot. This is the responsibility of the Navigator module described in Section 5.4. Thus, the two modules must be able to communicate with each other. For this reason, the variables **GoalType**, **GoalDistance**, and **GoalDirection** are declared to the Data Server (DServer). The variable **GoalType** indicates the mode in which the robot operates when traversing a segment. In my implementation, 'C' is used for segments  $\mathcal{S}^c$ , 'D' or 'E' for segments  $\mathcal{S}^d$ , and 'L' for segments  $\mathcal{S}^l$ . The variable **GoalDistance** has different

interpretations depending on the operating mode of the robot. For the light-tracking mode, this variable contains the number of ceiling lights to be tracked by the robot. For the “dead-reckoning” and “leave-corner” modes, this variable contains the distance (in millimeters) that the robot should travel. Lastly, **GoalDirection** contains the heading (in degrees) along which the robot should proceed when operating in each mode. To synchronize the data transfer between the Path Planner and the Navigator, two semaphores, **\$GoalData** and **\$RobotStatus** are declared to the Data Server. When the three variables described above have been set, the Path Planner will increment the **\$GoalData** semaphore. Before the Path Planner can assign a new set of data, it has to wait until the **\$RobotStatus** semaphore is incremented by Navigator. In this context, there is a producer-consumer relationship between the Path Planner and the Navigator.

## 5.4 Navigator

This client implements the local navigator described above. Here, the word “local” implies that the Navigator employs only the most recent sensor data for controlling the robot. This approach differs from what was implemented in my earlier experiment on the mobile robot Sparta [Schneider *et al.*, 1989].

The purpose of the Navigator module is to control the robot, so that plans generated by the Path Planner can be executed. At the same time, the Navigator is also responsible for obstacle avoidance capability. Plans from the Path Planner are communicated to the Navigator via the three variables described in Section 5.3.

In controlling our mobile robot, the Navigator module employs a deterministic finite state machine approach. Each state corresponds to a primitive behavior of the robot such as Move-to-Hallway-Center, Avoid-Trap, and Leave-Corner. It is worth noting that the finite state machine is deterministic, which implies that at any given

time, the program can only be in a single state. The program changes from one state to another when certain events arise, as determined from sensor observations. Based on this characteristic, each behavior can be executed in a common control loop and given the same type of sensor data, which includes the robot position, robot heading, and ultrasonic and infrared sensor readings. In addition, the control loop requires two other parameters, i.e., the goal checking function and the current state of the finite state machine. The system exits the loop when the goal checking function returns a logically true value. In Figure 5.8, these two parameters are shown in the shaded boxes. Even though the structure of the loop is fixed, additional states and goal checking functions can be added to provide new behaviors to the robot.

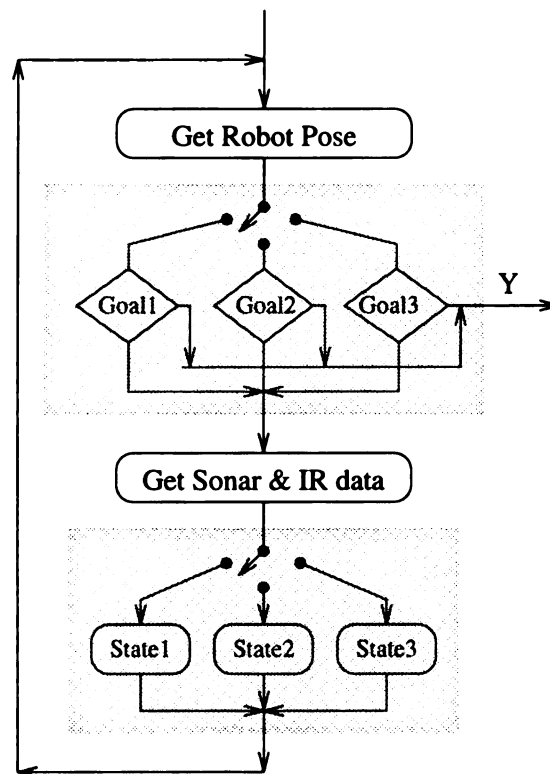


Figure 5.8. Control loop in the Navigator.

The Navigator also employs some checking functions that are activated periodically via a timer interrupt. The purpose of these functions is to complement both the local navigation and the obstacle avoidance capabilities of the robot as carried out by the control loop described above. These functions are activated only once every 3–5 seconds. Since they are activated through alarm interrupt handlers, calling these functions too often might cause the main control loop to be preempted frequently, causing the robot to be less responsive to the external events or sensor readings.

In Section 5.3 it was described how a path is decomposed into straight-line sub-paths which are further decomposed into segments  $S_i^l$ ,  $S_i^d$ , and  $S_i^c$ , in which the robot operates in different modes. It is the responsibility of the Navigator to control the robot so that it operates in the proper mode for each segment. The Navigator establishes a mode by selecting a proper combination of state and goal functions as described above.

In the hallway navigation system, there are two states and three goal checking functions. The states are **Center-Hallway** and **Corner**, while the goals are **L-Goal**, **D-Goal**, and **E-Goal**. Table 5.2 shows the relationships between the modes and their corresponding states and goal functions. Each one of these states and goal checking functions is implemented as a C++ function in the program.

Table 5.2. Relationship between modes, states, and goal functions.

Mode	State	Goal Function
Ceiling-Light Tracking	<b>Center-Hallway</b>	<b>L-Goal</b>
Dead-Reckoning	<b>Center-Hallway</b>	<b>D-Goal</b>
Leave-Corner	<b>Corner</b>	<b>D-Goal</b>
Enter-Corner	<b>Corner</b>	<b>E-Goal</b>

Besides the above operating modes, the Navigator is also responsible for controlling the robot to enter and exit elevators. This function is provided as a step towards



making the robot navigate on different floors of the MSU Engineering Building. The states, goals, and the control needed for the robot to enter and exit elevators are described in the following sections.

#### 5.4.1 Center-Hallway state

The behavior of the robot to be accomplished in this state is to move in the hallway while maintaining its position approximately in the center of the corridor. The main sensor data used in this state is the ultrasonic sensor readings. The configuration of the 24 ultrasonic sensors attached to RoME and the coordinate frame used by the TRC LABMATE is shown in Figure 5.9. In the figure, the  $y$ -axis indicates the forward direction of the robot.

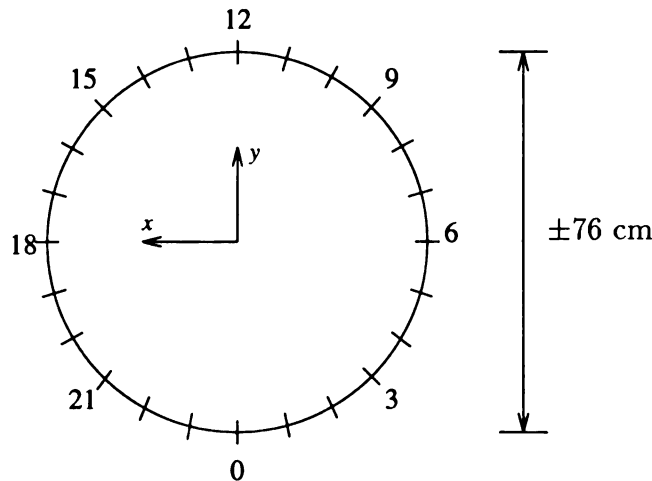


Figure 5.9. Sonar configuration.

The structure of most hallways usually consists of long parallel walls. When our robot is placed in such a hallway, the walls constrain the robot maneuvers and this information can be used to guide the robot to move along the mid axis of the hallway.

The sonar readings can be used to estimate the direction of the hallway axis

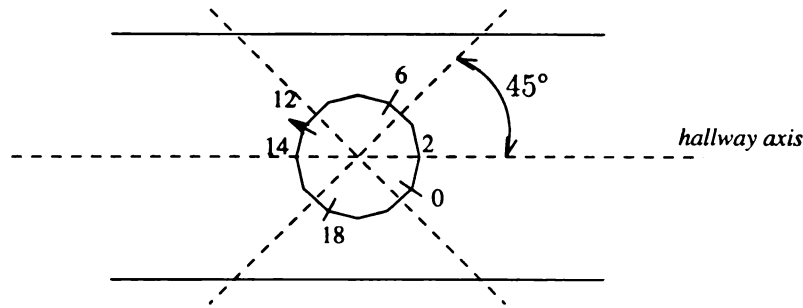


Figure 5.10. *Far* and *near* sonars.

relative to the robot's heading. At any given time while the robot is maneuvering in the hallway, each ultrasonic sensor can be classified into two types: *far* and *near*. An ultrasonic sensor is labeled *far* if its axis creates an angle less than  $45^\circ$  with the hallway axis. Otherwise, the sensor is labeled *near*. In Figure 5.10, sonars 0 and 12 are *far*, while sonars 6 and 18 are *near*.

Normally, *far* sensors will have longer distance readings than those of *near* ones. Also, sonars which make the smallest angle with the hallway axis should return the maximum reading. This approach can be used to determine which direction is "open" or "closed". The "open" direction should be identified by at least one of the *far* sonars. It is obvious that there are two possible "open" directions. In Figure 5.10 these directions correspond to sonar 14 and sonar 2. To control the robot to move forward in the correct "open" direction only, the other "open" direction is ignored. In my implementation, only sonars 6 through 18 are used in determining the "open" direction.

Using this approach, controlling the robot to move along the hallway axis can be accomplished by making sonar 12 point in the "open" direction. However, this might not be sufficient to keep the robot away from the walls. In addition to controlling sensor 12 to point in the open direction, the **Center-Hallway** state must check if the robot is approaching the walls. This safety check is carried out by always monitoring

the *near* sonar readings. Ideally, sonars 6 and 18 should be *near* and should indicate the “closed” direction, i.e., their readings are the minimum among all the sonar readings. If other sonar readings besides those of sonar 6 and sonar 18 become too low, it is an indication that the robot might be approaching a wall.

Combining the safety check and controlling sonar 12 to point in the “open” direction, the **Center-Hallway** state operates in two different modes. In the first mode, when the minimum sonar reading is very low (below a threshold), the state changes the robot heading so that, in the new heading, the “closed” direction aligns with either sonar 6 or sonar 18 (depending on the goal direction of the next target set by the Planner). When this situation occurs, the robot is given a command to make an in-place turn and the speed is decreased by 25%.

In the second mode, i.e., when the minimum sonar reading is “safe” (above a threshold), the robot is controlled so that the “open” direction aligns with sonar 12. Unlike in the first mode, where an in-place turn is used, this mode uses a superimposed turn command. This command has the advantage that the robot does not have to stop from moving before it turns. In the LABMATE, the superimposed turn is implemented as the **jog** command. The turning rate is determined by the following non-linear function:

$$\text{turning rate} = \text{sgn}(\delta) * 4,$$

where  $\delta$  is the deviation (in degrees) of the current robot heading from the “open” direction. To reduce false detections of the “open” direction from the sonar readings, the control algorithm first estimates the direction of the hallway axis from the current robot heading and the goal direction. This estimate is then used to determine the sonar whose deviation angle from the hallway axis is the smallest. If this sonar is numbered  $M$ , then “open” direction is determined by searching the largest reading among sonars  $\{M - 2, M - 1, M, M + 1, M + 2\}$ . A modulo-24 arithmetic is used to

resolve the “wrap-around” effect.

The threshold for determining the “safe” mode varies according to the behavior of the robot. Initially, the threshold is set to 1 m. When the robot makes too many turns in a short amount of time, the threshold is decreased by 50%. Otherwise, when the robot does not make any turn in a certain amount of time, the threshold is increased by 25%. This approach proved to be useful for the adaptation of the navigation program in corridors having different widths.

Using the above method, the robot is able to move in the hallways while maintaining its position near the center of the hallway. It is worth noting that the **Center-Hallway** state does not *explicitly* control the robot to move to the next target position; the main control scheme used in this state simply looks for an open direction. However, in addition to the control loop described in Section 5.4, the Navigator also uses two checking functions which are activated via an alarm interrupt handler. These functions are **Check-Bumper** and **Check-Goal**.

The **Check-Bumper** function is needed because in some situations, the sonar cannot detect an obstacle that stands in front of the robot. For instance, an object whose height is below the sonar scan level will not be “seen” by the sonar. Fortunately, the bumper on our robot can be used to detect if the robot touches an obstacle. This function periodically checks for bumper contact and maneuvers the robot to keep the bumper away from the obstacles. The **Check-Goal** function is activated periodically to check if the robot is moving in the specified heading. If it is not, corrective actions will be taken to turn the robot to the correct heading.

#### 5.4.2 Corner state

This state is invoked when the robot needs to make a turn at a corner. More precisely, this state is activated when the Navigator is controlling the robot along segment  $\mathcal{S}_i^c$  (after the robot is in a turning space and has to enter the next hallway in the specified

path). The **Center-Hallway** state cannot be used for this purpose, since that state looks for an “open” direction in too wide a region (sonars 6 through 18). This might result in the detection of an “open” direction in the “incoming” corridor. To avoid this situation, the **Corner** state interrogates a narrower range of sonars, where the range is determined dynamically depending on the current heading of the robot and the heading of the target point. By imposing this constraint, only an “open” direction in the “outgoing” corridor is detected.

In the **Center-Hallway** state, obstacle avoidance is implemented using the sonar data only. Since one of the objectives of the **Corner** state is to avoid detecting an “open” direction in the “incoming” corridor, we want to keep the robot moving in a straight line as much as possible. Consequently, the obstacle avoidance capability in this state is implemented via the infrared sensor readings in a stop-and-go fashion. Whenever any one of the forward-facing infrared sensors detects an object, the robot is stopped and put into jog mode with a small turning rate. The direction of the turn is determined by the deviation of the current heading from the goal direction. The infrared sensors are rechecked in the next Navigator control loop after a delay of 1–2 seconds.

### 5.4.3 L-Goal

Together with the **Center-Hallway** state, this goal function controls the robot while it operates in the “ceiling-light-tracking” mode. **L-Goal** is invoked when the robot is in segment  $S^l$ . This function returns a logically true value when the robot camera has counted a certain number of ceiling lights. This goal function does not perform the actual ceiling light tracking, but instead it continually requests Data Server for the current value of the variable **Ceiling\_Light**. This variable is modified by the ceiling light tracker module described in Section 5.6. Whenever the value of **Ceiling\_Light** matches the specified number of ceiling lights, this function returns a logically true

value causing the control loop in the Navigator to exit.

#### 5.4.4 D-Goal

This goal function checks whether the robot has moved a certain distance in segment  $\mathcal{S}^d$ . In this segment, there are no landmarks to use for position registration, therefore the robot has to rely on dead-reckoning navigation. This function simply gets the current robot position from the Robot Server (RServer) and calculates the distance traveled since the beginning of the segment  $\mathcal{S}^d$ .

As shown in Table 5.2, dead-reckoning navigation is implemented in combination with the **Center-Hallway** state. Since the **Center-Hallway** state constrains the robot to move in one direction of the hallway, there is no need to check the actual final *position* of the robot; only the distance traversed needs to be checked.

#### 5.4.5 E-Goal

This goal function is activated when the robot is entering a turning space in a corner. Also, it is very similar to **D-Goal**, except for the test used to determine when this function returns a logically true value. In **D-Goal** a logically true value is returned when the robot has traveled the desired distance. Besides this test, **E-Goal** also checks if the range reading of the sonar facing towards the goal direction is less than a threshold. If this is true and the distance to the middle of the turning space is less than a threshold, then a logically true value is returned.

#### 5.4.6 Entering and Exiting Elevators

When the Planner specifies an elevator as the destination point, the Navigator will control the robot to enter and exit the elevator upon reaching the destination point. Here, the robot must perform the following tasks:

1. Locate and approach the elevator doors,
2. Wait until the elevator doors are open after a human operator presses the elevator button,
3. Enter the elevator,
4. Turn around when the robot reaches the back wall inside the elevator,
5. Wait for the door to reopen after a human operator tells the robot that the destination floor is reached, and
6. Exit the elevator.

Using sonar for controlling the robot inside the elevator is not a reliable approach. Due to the specular surface of the elevator walls, spurious sonar readings are unavoidable. For this reason, the sonar data are used only for detecting whether the elevator doors are open when the robot stands in front of the elevator. Only the infrared proximity detectors are used for controlling the robot inside the elevator. However, in my initial experiments I observed that the metallic surface of the elevator walls also caused strong reflections of the infrared light, resulting in spurious readings of the infrared proximity detectors. To overcome this situation, the inner elevator walls were covered with either newspapers or manila cartons.

Locating the elevator doors is carried out mostly by visual sensing. Initially, the locations of vertical edges on the left and right sides of the elevator doors were used to determine the center of the door. For this purpose, one of the cameras was panned to the left and right until a strong edge appears in the center field of view. However, false edges are likely to be detected, especially when the camera was not initially facing the elevator doors. To overcome this situation, a barcode-like marker was pasted on the elevator doors. Figure 5.11 shows the design of the barcodes. A barcode consists of  $N$  bits interspersed within  $N + 1$  black separators. A barcode

always starts and ends with a separator. A black bar on the bit field represents a 1 while white bar represents a 0. No consecutive 1's can exist in a code. Thus, the code in Figure 5.11 (b) represents a binary value 100. Separators and bits have the same width ( $p$ ) and height ( $q$ ). When a binary 1 exists in a code, a black rectangle of width  $3p$  will exist in the code. In the experiment, we set  $N = 3$ ,  $p = 3.5$  inches, and  $q = 6$  inches.

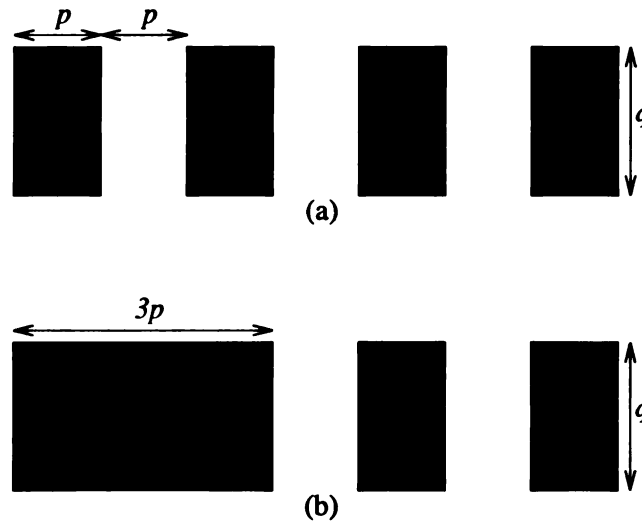


Figure 5.11. Barcode-like Markers representing binary values: (a) 000 and (b) 100.

The above barcode design reduces the possibility of false positive detections and yet provides easy detection by a computer program. This design was adapted from the barcode design used by the University of Michigan mobile robot, CARMEL, during the AAAI'92 Robot Competition [Kortenkamp *et al.*, 1993]. The barcodes used by CARMEL are vertically oriented. The barcodes used in my experiments were horizontally oriented to take advantage of the `CameraGetRow()` function in the Camera Server. By scanning each row in the image and computing the black and white transitions, the program collects black and white patterns starting with a white-to-black transition. Every detected black and white pattern is verified by checking the



ratio of the width of each stripe to the width of the smallest stripe. A pattern is accepted if this ratio is either 1 or 3. Accepted patterns are stored and merged with previous patterns within the neighboring rows/columns in the image. A valid barcode is detected when the number of rows in the barcode exceeds some threshold.

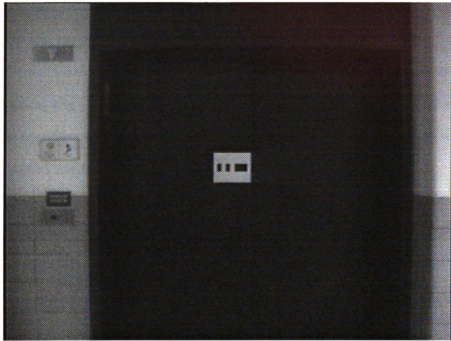


Figure 5.12. An elevator door pasted with a barcode-like marker.

Since the robot must approach the marker on the elevator doors, the marker was placed at the camera height above the floor. In our setup, this height is approximately 53 inches. The purpose of this placement is to make the marker visible in the rows around the center of the image. Thus, when the robot approaches a barcode marker, it will always be in the field of view of the camera. To locate a barcode marker, the middle half of the input image is scanned while the robot turns to the left or right at a sweeping angle of approximately 45 degrees on each side. Once the marker is found, only the rows confining the marker are scanned. Figure 5.12 shows a picture

of an elevator door pasted with a barcode-like marker.

## 5.5 Local Mapper for Heading Correction

The internal heading information of the LABMATE mobile robot is determined by encoders on the two drive wheels. When the encoders record the same distance traversed by the respective wheels, the robot heading remains the same between the initial and final locations. On the other hand, if the right wheel encoder records a larger distance, the robot heading changes as if it is turning counter clockwise. In reality, the encoders cannot record the actual distance traversed by the wheels, due to errors caused mainly by wheel slippage.

In a situation where an absolute heading is required, the internal robot heading is not a reliable source of such information. Another method of estimating the heading information must be used. As an alternative, we can use an external source of heading information such as a gyro or a digital compass, but adding such an equipment to the robot means an additional hardware investment.

In my experiments, I have successfully developed a method for registering the map constructed by the sonars and the stored StickRep map. The map created from the sonar readings is “matched” to the preconstructed StickRep map. This method is used to reset and correct the erroneous internal heading of the robot after it has traveled a considerably long distance. While the robot is moving in segment  $S^l$ , one or more local maps of the environment are created from the sonar readings. In the experiments, a local sonar map is created for every three ceiling lights tracked by the robot. Each local map covers a distance of approximately 10–12 meters. This process is repeated until the robot reaches the last ceiling light in segment  $S^l$ .

The Histogrammic In-Motion Mapping (HIMM) method [Borenstein and Koren, 1991a] has been adapted for the above purpose. The code was first implemented on

our mobile robot by Courtney [Courtney, 1993]. Figure 5.13 shows an example of a map created by this method. The black dots are the accumulated sonar readings and the grey lines are the lines detected by the Hough transform. While the robot moves in the hallway, sonar data are collected and the inverse transformation of the current robot pose relative to the initial pose of the map is determined. This inverse transformation is used for plotting the sonar data on the map.

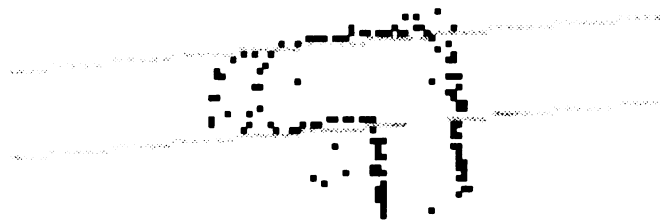


Figure 5.13. An HIMM map created from a corner in the hallway.

To estimate the relative orientation of the hallway from the sonar map created by HIMM, the Hough transform was used for straight line detection. Only the two parallel lines that correspond to the hallway walls need to be located. Assuming that at least one of these lines will be mapped to a global maximum in the Hough space, a guided search for the other line(s) is performed. This approach seems to be robust and the relative orientation of the hallway can be determined with the accuracy needed for indoor navigation. By comparing the actual heading of the robot with the hallway orientation estimated by HIMM, a correction angle for registering the robot heading with the hallway axis can be determined.

One minor problem with this approach is that the relative hallway orientation is “directionless.” Suppose that the relative hallway orientation is  $\alpha$ . There are two

possibilities for aligning the robot heading to the hallway axis,  $\alpha$  and  $(180^\circ + \alpha)$ . In this case, the correct hallway orientation is the one closer to the goal heading direction. This can be easily determined by calculating the difference between the destination angle and the two possible choices. Suppose the destination angle is  $\Gamma$ , then the following quantities can be computed:

$$\theta_1 = (\Gamma - \alpha) \bmod 180 \quad \text{and} \quad \theta_2 = (\Gamma - \alpha - 180) \bmod 180.$$

Then, the correct orientation  $\gamma$  is determined by:

$$\gamma = \begin{cases} \alpha, & \text{if } |\theta_1| < |\theta_2| \\ \alpha + 180, & \text{otherwise.} \end{cases}$$

## 5.6 Ceiling Light Tracking

It is a well-known fact that in *any* kind of navigation, we cannot rely solely on odometry information; some external information must be used. It was mentioned earlier that one of the modes of operation of our mobile robot is the ceiling-light-tracking mode. The objective of this mode is to register the robot position with physical reference points in the robot workspace. I chose to utilize ceiling lights as the reference points because they are easy to detect due to the high contrast between the lights and the ceiling. Figure 5.14 shows a picture of our robot in the hallway of the MSU Engineering Building.

For efficiency, the ceiling light tracking module, **CTracker**, uses only three scan lines from the entire image ( $240 \times 120$  pixels). These lines are taken at distances associated with 1/4th, 1/2, and 3/4th of the image height. The image is periodically grabbed by the Camera Server every 0.5 seconds and then the **CTracker** requests these three scanlines for further processing. By using only a small number of scan lines, the data transfer between the Camera Server and **CTracker** is minimized. The



Figure 5.14. RoME in the hallway of the MSU Engineering Building.

ceiling light detection algorithm simply looks for strong leading edges of the ceiling lights in the three scan lines. An edge is detected if the gradient of a pixel exceeds some threshold. Since this information is used by other modules in the system, the number of detected ceiling lights is stored into the variable `CeilingLight` in the Data Server.

The parameters of our camera for ceiling light detection are given in Table 5.3. Figure 5.15 shows how the two cameras are positioned. The camera facing up is used for detecting ceiling lights and the other camera facing sideways is used for detecting door number plates. With the given setup, the surface area on the ceiling that can be captured by a single image is approximately  $1.2 \times 1.5$  square meters. The maximum width of our hallway is approximately 3.66 meters and the dimension of a ceiling light is approximately 0.6 meters by 1.2 meters.

The ceiling light tracking module consists of a number of concurrent child processes

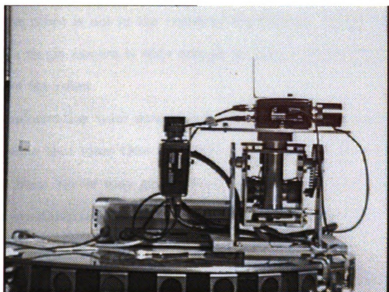


Figure 5.15. Camera setup for detecting ceiling lights (facing up) and door number plates (facing sideways).

Table 5.3. Camera setup for ceiling light tracking.

Parameter	Value
Lens focal length	6 mm
Camera field of view	$\pm 60^\circ$
Distance from ceiling	$\pm 135\text{cm}$
Pixel size	$512 \times 482$

for detecting ceiling lights and a parent process that fuses information from the child processes. Each child process receives input images from a separate camera. Using several cameras is necessary if the ceiling lights are outside the field of view of a single camera when the robot is not in the center of the hallway. In my experiments, the field of view of a single camera is wide enough to detect the ceiling lights regardless of the position of the robot.

Combining information from several different sources is not always straight forward. It is possible that more than one child process detects the same light, so the variable in the Data Server may get updated several times. To avoid this, I have implemented a synchronization mechanism that combines a semaphore and a shared variable used by the child processes. The synchronization policy used in my algorithm forces the processes to acquire a “token” before it can update the information in the Data Server. The “token” can be acquired by a process if it is not already owned by any other process. When a process detects a transition from “no-light” to “light”, it first tries to acquire the token and then update the global variable if the “token” is available. On the other hand, when a transition from “light” to “no-light” is detected, a process holding the “token” has to release it.

To avoid misdetections or false positives, the response from a child process is verified by a parent process. Whenever the parent process receives a response from the light detecting processes, it checks the distance the robot has traveled since the last detected light. If this distance agrees with the actual distance between the two lights computed from the stored map, then the response is valid. Otherwise, it is considered invalid. Only valid responses are passed to the Data Server.

## 5.7 Door Number Plate Detection

In addition to ceiling lights, door number plates are also used as secondary landmarks to guide navigation. In our domain, door number plates are found two or three times more frequently than ceiling lights. Using a camera facing one side of the hallway, the detection routine grabs images while the robot is moving. Since all the door number plates in our Engineering building are located at about the same height from the floor, only a small number of scan lines (40) need to be processed. The height of the camera from the floor is adjusted so that the plates are located near the middle row of the image. For a fast response, my plate detection algorithm searches for a closed contour in a thresholded intensity image and calculates the curvature of each contour point. The bounding box of the door number plate is determined by identifying contour points with high curvature. This bounding box can then be used to obtain the door number on the plates. Figure 5.16 shows the result of this detection.

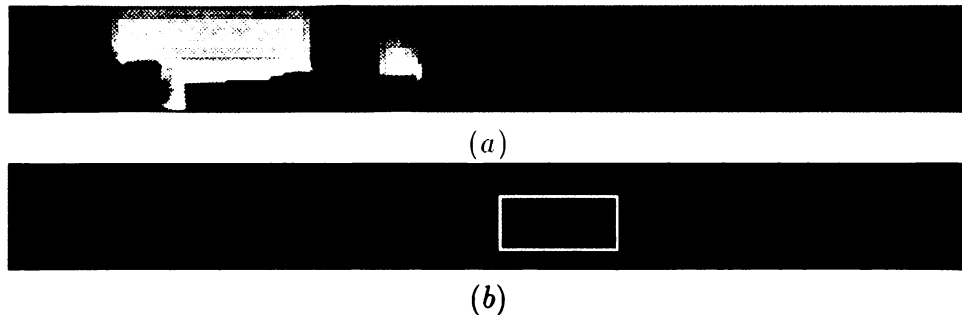


Figure 5.16. Door number plate detection result. (a) input image, (b) door number plate detected.

Since the input image to the number plate detector is captured while the robot is moving, the “lock” facility offered by the Camera Server has to be used. If this facility is not used, the input images to plate detector module might contain two



halves, each of a different portion of the wall/plate number.

All number plates in the MSU Engineering Building have a black background and white letters. To “highlight” the number plates, the input image is preprocessed by computing the negative image and then stretching the graylevel values using the following non-linear function:

$$x' = \begin{cases} (x/M)^{e_1} M & 0 < x \leq M \\ \left(\frac{x-M}{255-M}\right)^{e_2} (255 - M) + M & M < x \leq 255 \end{cases}$$

In my experiments, I used  $M = .8 \times 255$ ,  $e_1 = 5.0$ , and  $e_2 = .5$ . The purpose of this stretching operation is for contrast enhancement. After the graylevel stretching, the image is thresholded to obtain a binary image.

The algorithm searches for dark to bright transition, first along the middle scan line, and then in the upper half of the input image if such a transition could not be found. In binary images, this dark to bright transition corresponds to a pixel on the border of a region. Once such a transition is found, the algorithm will trace the border and store the pixel’s row and column positions in a linked list. At the same time, the chain code of the curve that is constructed by these pixels is determined. If a closed border is found, the “curvature” of each pixel on the border is determined and corner points can be located from the curvature values. I have used the corner detection algorithm from chain code representation of curves described in [Beus and Tiu, 1987]. The detected corner points are then used for determining the bounding box of the number plate. The size of the bounding box is used for rejecting or accepting the box as a door number plate.

In order to avoid multiple detection of the same plate from several different positions and orientations of the robot, the algorithm has to determine whether two image frames overlap. For this purpose, the geometries in Figures 5.17 and 5.18 have been developed. Figure 5.17 shows how the length of left and right projections of the

field of view of the camera to the wall are calculated, given the heading angle of the robot.

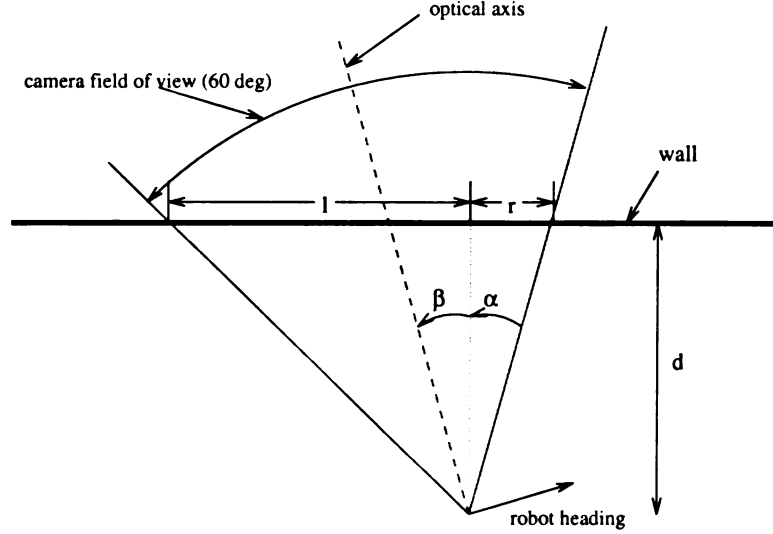


Figure 5.17. Number Plate Detection Geometry.

In Figure 5.17,  $\beta$  is the angle between the optical axis and the wall normal (measured from the wall normal). Since the optical axis of the side-looking camera coincides with the axis of sensor 18,  $\beta$  is also the angle between the robot heading and the hallway axis (measured from the hallway axis);  $\alpha$  is the angle between the right ray of the camera field of view and the wall normal (measured from the right ray);  $d$  is the sonar reading of the sensor approximately perpendicular to the wall;  $l$  and  $r$  are the distances between the point of projection of the robot center and points of projection of the left and right rays;  $l < 0$  when  $\beta < -30^\circ$ , and  $r < 0$  when  $\beta > 30^\circ$ . The following equations hold for the above entities:

$$\alpha = 30^\circ - \beta,$$

$$l = d \tan(\beta + 30^\circ),$$

$$r = d \tan(\alpha).$$

Since the above equations are valid only when  $|\beta| < 60^\circ$ , and the error in the computation is very large when  $|\beta| \approx 60$ , the door number plate detector performs no action when  $|\beta| \geq 50^\circ$ .

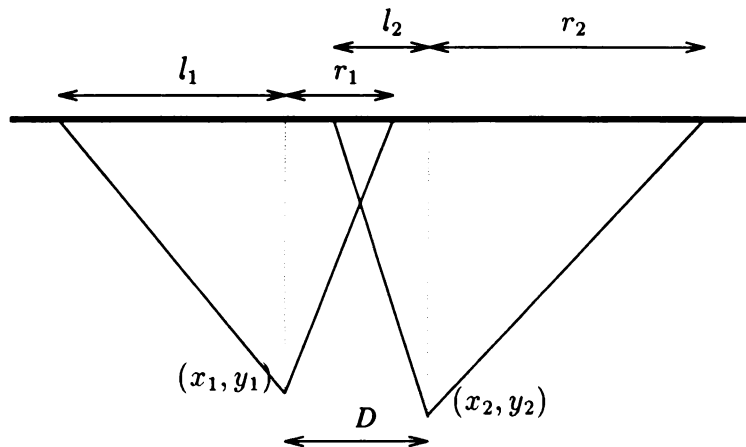


Figure 5.18. Two overlapping snapshots.

The amount of overlap between two successive image frames can be determined using the geometry in Figure 5.18. In the figure, the first image frame is captured when the robot is at position  $(x_1, y_1)$ . The distance between the two wall normals is  $D$ . An overlap occurs when  $D < (r_1 + l_2)$  and  $(r_1 + l_2 - D)$  denotes the length of the overlapping segment. This length also determines the starting column number for locating the number plate in the second image frame. For fast computation of  $D$ , this value is approximated by the distance between  $(x_1, y_1)$  and  $(x_2, y_2)$ .

## 5.8 Information Sharing Among Clients

Table 5.4 summarizes the information shared among the client modules. In the table, U/D entries mean `up()`/`down()` operations by the clients to a semaphore and R/W entries mean read/write data to a variable.

Table 5.4. Information sharing via the Data Server.

	Planner	Navigator	CTracker	HIMM	PIFinder
<b>\$StartMap</b>	U			D	
<b>\$LightDet</b>			U	D	
<b>\$GoalData</b>	U	D			
<b>\$RobotStatus</b>	D	U			
<b>\$TrackData</b>	U		D		
<b>\$EndLMode</b>	D		U		
<b>GoalDirection</b>	W	R		R	R
<b>GoalDistance</b>	W	R	R	R	
<b>GoalType</b>	W	R			
<b>LightOffset</b>	W		R		
<b>CeilingLight</b>		R	W	R	
<b>Overshoot</b>	R		W		

- The semaphore **\$StartMap** synchronizes the Local Mapper (HIMM) with the Planner. The Planner does an `up()` operation to notify the HIMM that the ceiling light tracking mode is activated.
- The Navigator notifies the HIMM via semaphore **\$LightDet** when a ceiling light is detected. This event is used by the HIMM to determine when to start/stop building a local map.
- The semaphores **\$GoalData** and **\$RobotStatus** are used for the producer-consumer interaction between the Planner and the Navigator. Here, the Planner

“produces” goal points and the Navigator “consumes” them. When the Navigator is ready to “consume” a new goal point, it notifies the Planner through **\$RobotStatus**.

- The producer-consumer relationship also exists between the Planner and the CTracker. The two clients synchronize themselves using **\$TrackData** and **\$EndLMode**. When the CTracker completes its light detection cycle, it notifies the Planner (via **\$EndLMode**) of the overshoot distance as a result of detecting the last ceiling light.

## 5.9 Summary

In this chapter the details of each client module for controlling the robot in hallway navigation have been described. First, we described how the robot environment is represented in a map using the StickRep representation. We also showed how the maps are constructed from data (the locations of vertical edges and ceiling lights) collected from the third floor of the MSU Engineering building. Next, the Path Planner module was presented by describing how the path segmentation is performed, and which robot operating mode corresponds to each subpath. Also, we described how the Path Planner communicates its goal to the Navigator module. The internal details of the Navigator module were discussed by describing its various states, the goal functions, and the flexible control loop used in it. The implementation details of the obstacle avoidance capability in the Navigator module were also described. Next, the module that continuously performs heading correction was presented. Finally, the two landmark tracking modules (ceiling light and door number plate modules) were described.

# CHAPTER 6

## Experimental Results

In this chapter, the experimental results of indoor navigation are presented. Based on the results, a probabilistic automata for analyzing the confidence level for the robot to reach the goal position from the initial specified robot position is derived.

### 6.1 Indoor Navigation Experiments

In this section, the results of indoor navigation of the robot both for straight line paths and paths with corners are presented. The robot has been tested at several different locations on the third floor of the MSU Engineering Building. Refer to Figure 6.1 for region notations. The MSU Engineering Building was built in two time periods. The old wing consists of the corridors labeled A, B, C, and M. As can be seen from Figure 6.1, the corridors in the new wing are narrower than those in the old wing.

Based on the experiments, I have determined that the corridors labeled G, N, and J are the “trouble” spots for the robot. In each of these corridors, there are only two ceiling lights and this often confuses the robot due to incorrect heading information. In my implementation, the HIMM module corrects the robot heading at every other light. This implies that in a corridor with less than three ceiling lights, the heading correction will not be performed. If when entering one of these corridors the heading

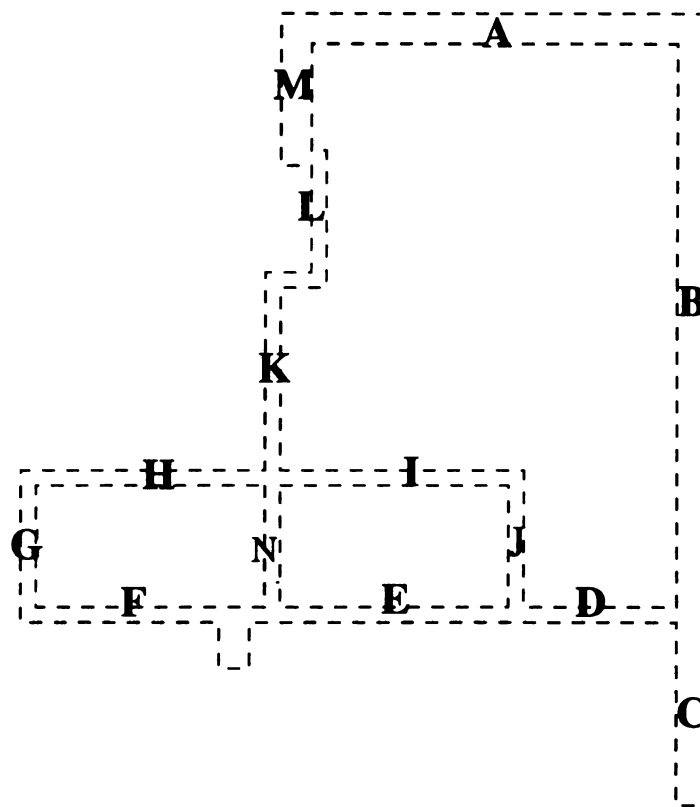


Figure 6.1. Regions in the map.

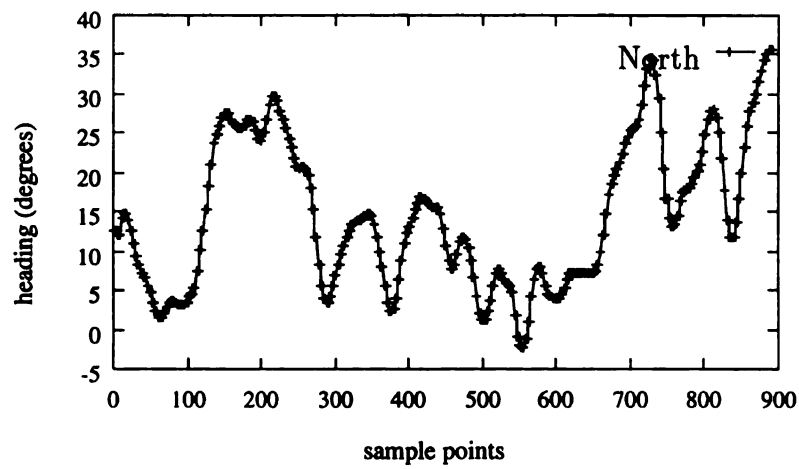


Figure 6.2. Output produced by the digital compass while the robot is moving north.

information is not accurate, the true heading of the robot cannot be determined by the time the robot reaches the end of the corridor. If the goal location resides outside these corridors and the robot has to pass through one of these corridors, the robot might not orient itself correctly to navigate in the next corridor. I tried to fix this problem by using a digital compass that works by detecting the Earth's magnetic field. However, after testing the compass inside the MSU Engineering Building, it turns out that the heading information produced by the compass is not accurate either. Figure 6.2 shows a plot the heading produced by the compass as the robot moves in a straight line.

Tables 6.1 and 6.2 show the results of the indoor navigation on the third floor of the MSU Engineering Building. The results dated after Dec 22, 1993 were obtained after the addition of the overshoot compensation in the Navigator algorithm. Consequently, the results of test runs using the new algorithm should show smaller values in the column labeled "Dist. to Goal".

In Section 5.4, four different modes of operation of the robot were defined: Ceiling-Light Tracking, Dead-Reckoning, Leave-Corner, and Enter-Corner mode. The last three columns of Tables 6.1 and 6.2 show the number of occurrences of each mode in the path. An entry of the form  $m/n$  in these columns means that in the given path, the robot must enter the mode  $n$  times for a successful completion of the path and during the experiment the robot entered the given mode  $m$  times. Some entries are marked with a star to indicate that a failure occurred in the corresponding mode. Since the Leave-Corner mode is always followed by Enter-Corner mode, these two modes are shown as a single column labeled 'C'. The other two columns are labeled 'L' for the Ceiling-Light Tracking mode, and 'D' for the Dead-Reckoning mode.

The results of indoor navigation show that in the old wing of the MSU Engineering Building, the range of the robot average speed is 7.0-17.875 m/min. In the new wing, this range is 4.7-10.25 m/min. This result is to be expected since the



Table 6.1. Results of the indoor navigation in the old wing of the MSU Engineering Building.

No.	Path	Date and Time	Actual Dist. (m)	Dist. to Goal (cm)	Run Time (mm:ss)	Modes		
						L	C	D
1	350R to 376L	Oct 31 1993	78.181	-13	5:26	1/1	0/0	1/1
2	350R to 376L	Oct 31 1993	78.181	+46	5:57	1/1	0/0	1/1
3	350R to 376L	Oct 31 1993	78.181	-15	6:19	1/1	0/0	1/1
4	350R to 376L	Oct 31 1993	78.181	+20	6:09	1/1	0/0	1/1
5	350R to 376L	Oct 31 1993	78.181	+43	5:43	1/1	0/0	1/1
6	350R to 376L	Oct 31 1993	78.181	+61	5:14	1/1	0/0	1/1
7	350R to 376L	Oct 31 1993	78.181	+53	5:45	1/1	0/0	1/1
8	381L to 343R	Oct 31 1993	73.863	+81	5:02	1/1	0/0	1/1
9	381L to 343R	Oct 31 1993	73.863	+30	5:01	1/1	0/0	1/1
10	381L to 343R	Oct 31 1993	73.863	+66	5:11	1/1	0/0	1/1
11	381L to 343R	Oct 31 1993	73.863	+41	5:34	1/1	0/0	1/1
12	381L to 343R	Oct 31 1993	73.863	+25	4:59	1/1	0/0	1/1
13	381L to 343R	Oct 31 1993	73.863	Fail		0*/1	0/0	0/1
14	346L to 388R	Nov 10 1993	107.95	-45	6:46	1/1	0/0	1/1
15	341L to 311R	Nov 15 1993	91.795	+60	9:40	3/3	2/2	1/1
16	303.second.R to 350R	Nov 15 1993	106.984	+100	8:40	3/3	2/2	1/1
17	345R to 311.double.R	Nov 22 1993	105.714	+36	12:36	3/3	2/2	1/1
18	341L to 311L	Nov 22 1993	89.966	+50	12:51	3/3	2/2	1/1
19	303.second.L to 350R	Nov 22 1993	107.899	+62.5	11:53	3/3	2/2	1/1
20	345R to 311R	Dec 5 1993	96.774	+48	13:31	3/3	2/2	1/1

Table 6.1. (cont'd).

No.	Path	Date and Time	Actual Dist. (m)	Dist. to Goal (cm)	Run Time (mm:ss)	Modes		
						L	C	D
21	303.second.L to 350L	Dec 5 1993	106.070	+75	10:40	3/3	2/2	1/1
22	345R to 311.doubleL	Dec 13 1993	103.886	Fail <sup>a</sup>		1/3	1*/2	0/1
23	NE.men.R to 302R	Dec 13 1993	120.650	+50	11:43	3/3	6/6	1/1
24	301R to 350R	Dec 13 1993	154.686	+60 <sup>b</sup>	15:52	3/4	6/6	1/1
25	350R to 372L	Dec 21 1993	59.740	+80	3:26	1/1	0/0	1/1
26	381R to NE.men.R	Dec 21 1993	92.354	+50	5:10	1/1	0/0	1/1
27	acs.elev.R to 376R	Jan 5 1994	89.052	+20	6:11	1/1	0/0	1/1

<sup>a</sup>Glass door near the ACS laboratory was not detected by the sonars. The robot ran into it causing a very large wheel slip.

<sup>b</sup>The robot missed the first ceiling light in front of EB303 causing it to come too close to the wall to the left of EB312

Table 6.2. Results of the indoor navigation in regions of the new wing of the MSU Engineering Building.

No.	Path	Date and Time	Actual Distance (m)	Dist. to Goal (cm)	Run Time (mm:ss)	Modes		
						L	C	D
1	302L to A340R	Nov 12 1993	35.585	-7	7:33	1/1	1/1	1/1
2	A392L to A340R	Nov 12 1993	15.113	+7	3:09	1/1	1/1	1/1
3	311L to 381R	Nov 13 1993	171.754	-36	23:56	6/6	8/8	1/1
4	323L to 381R	Dec 6 1993	197.358	Fail <sup>a</sup>		4*/7	5/9	0/1
5	341R to 381R	Dec 6 1993	260.807	+75	32:31	8/8	10/10	1/1
6	345R to 345L <sup>b</sup>	Dec 6 1993	337.007	Fail <sup>c</sup>		7/9	8*/10	0/1
7	A313L to 345L	Dec 8 1993	98.958	+95	9:39	2/2	2/2	1/1
8	NE.women.L to A314R	Dec 9 1993	203.911	+59	22:08	6/6	7/7	1/1
9	A314R to 345R	Dec 9 1993	101.904	+24	9:52	2/2	2/2	1/1
10	NW.men'sL to A314R	Dec 28 1993	142.748	+35	18:56	5/5	6/6	1/1
11	A342L to A323R	Dec 28 1993	38.633	+40	5:36	1/1	0/0	1/1
12	A347L to A385R	Jan 5 1994	103.530	Fail <sup>d</sup>		2/3	1*/2	0/1
13	A365L to 301R	Jan 5 1994	69.900	+48	9:41	2/2	1/1	1/1

<sup>a</sup>Incorrect heading due to wheel slippage on the metal junction on the floor.<sup>b</sup>A complete cycle of the building.<sup>c</sup>Trouble spot in region J.<sup>d</sup>Trouble spot in region G.

corridors in the new wing are narrower than those in the old wing. As a comparison, human normal walking speed is approximately 50–60 m/min. Recently, a mobile robot that is capable of navigating at an average speed of 8–10 m/min was reported in [Kosaka and Kak, 1992]. Our system can achieve a higher average speed because its operation does not heavily rely on the visual processing. The results also show that the implemented navigation algorithm tends to overshoot the goal point by a maximum distance of 1 meter. Most of the results reported here were obtained before the overshoot compensation capability was implemented in the Navigator.

## 6.2 Door Number Plate Detection

Table 6.3 shows the results of door number plate detection by the PlFinder. The table shows the paths on which the program was tested, both detected and misdetected plates, and the false positives detected by the program. In Section 5.7 it was mentioned that the equations derived from the geometries used for detecting overlapping image frames are valid only for  $|\beta| < 60^\circ$  ( $\beta$  is the deviation of the robot heading relative to the hallway axis). To avoid large inaccuracies in the computations, the PlFinder performs no action when  $|\beta| > 50^\circ$ . Thus, when the deviation angle is too large, door number plates are not detected. This situation can be remedied by having more cameras whose fields of view overlap. These cameras must be arranged such that when the angle between the optical axis of one camera and the wall normal is approaching the  $\beta$ -constraint described above, the input image to the PlFinder will be taken from the other camera whose  $\beta$  angle is small(er). Most of the misdetections shown in Table 6.3 were caused by this phenomena, while other misdetections were caused by background interference, occlusion by people moving in the hallways, and the 0.5-second sampling rate of the Camera Server. Initially, the pan tilt carousel was used to control the camera so it always faces perpendicular to the wall, but the

response of the carousel is too slow compared to the speed of rotation of the robot.

The PlFinder module was not integrated with the ceiling light tracking module, so the results shown in Tables 6.1 and 6.2 do not include door number plate detection. Also, the false positives detected during the experiments can be reduced if the approximate locations of the door number plates were stored in the map.

### 6.3 Confidence Levels of Indoor Navigation

Based on the results shown in Tables 6.1 and 6.2, two finite state automata (FSA) for analyzing the performance of my indoor mobile robot navigation system were constructed. One finite state automaton models the confidence level of the robot for reaching the goal position from the specified initial position using ceiling lights. The other finite state automaton models the reliability of detecting all the door number plates in a given path.

Let us consider a situation where our robot navigates from point  $A$  to point  $B$  by using only ceiling lights as landmarks and operates in the three modes mentioned in Section 6.1. Furthermore, suppose that from  $A$  to  $B$  the robot detects  $(m + n)$  occurrences of ceiling lights. Here, the robot is supposed to detect the first  $m$  ceiling lights, enter the C-mode, and then reenter the L-mode to detect the remaining  $n$  ceiling lights, and finally enter the D-mode. The above situation can be represented by a finite automaton with  $(m + n + 3)$  states as shown in Figure 6.3. Initially, the robot is in state  $S$ , and it changes its states to  $l_{11}, l_{12} \dots l_{1m}$  upon detecting the first  $m$  ceiling lights. In the state  $l_{1m}$  the robot operates in the C-mode until it turns around the corner and then the robot enters the state  $C$ . After detecting an additional  $n$  ceiling lights, the robot reaches its goal state  $G$ .

From the experiments, I have learned that when the robot is operating in the L-mode, once the first ceiling light is correctly detected, then the remaining lights



Table 6.3. Door number plate detection results.

	Correctly Detected	Missed	False Detections
352L to 384R	352, 354, 356, 360, 364, 368, elev, 372, 376, 380, 382, 384	358, 364.double	Black poster between 364.double and 364
381L to NE.men.L	381, 379, 377, 375, 373, 371, 369, 367, 363, 361, 359, 357, 355, 349, 347, 345, 343, 339	353, 351, 341	—
346R to 376R	350.1, 350, 352, 354, 356, 358, 360, 364.double, 364, 368, old.elevator, 372, 376	—	356, 364
381R to NE.men.R	381, 379, 377, 375, 371, 369, 367, 361, 359, 357, 355, 353, 351, 349, 347, 345, 34	373, 363, 343, 339	—
NE.women.L to V317R	335, 329, 327, 325, 323, 317.1	317.2	between 335 and 329
314L to 340.cmptr.room.R	314, 320, 320, 326, 340.cmptr.room	324, 340	—
A342L to A323R	A342, A340, A339, A338, A336, A335, A334, A332, A331, A330, A328, A326, A325, A324, A323	A327	A342, A340
acs.elevator.R to 376R	364, 350.1 350.2, 352, 354, 356, 358, 360, 364.1, 364.2, 368, 372, 376	—	356, between 364.1 and 364.2
381R to NE.men.R	381, 379, 377, 375, 373, 371, 369, 367, 363, 361, 359, 357, 355, 353, 351, 347, 345, 343, 339	349, 341	poster on 377, 371
NE.women.L to 311.doubleR	335, 329, 327, 325, 323, 317.1, 317.2, 311	—	—
A365L to 301R	A365, A367, A368, A371, A372, A373, A377, A378, A379, A380, A381, A383, A385, A391, A393, A395, 301	A369, A375, A382, A397	9 false labels from the glass bricks

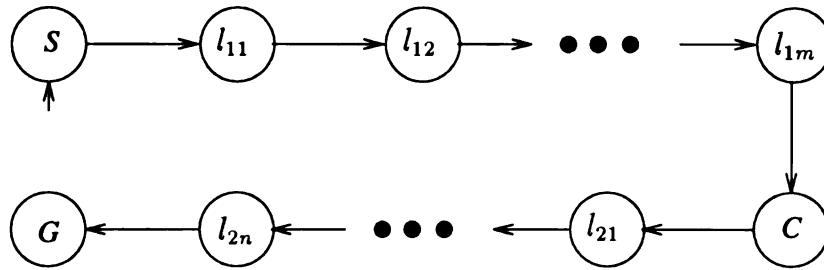


Figure 6.3. A finite state automaton representing a landmark-based navigation task.

will be located correctly. This is possible because the ceiling light detector module uses information from both the camera and the stored maps. Therefore, instead of representing each detection of a ceiling light by a single state, the entire L-mode will be represented by a single state. Figure 6.4 shows a more concise representation of the state transition diagram in Figure 6.3.

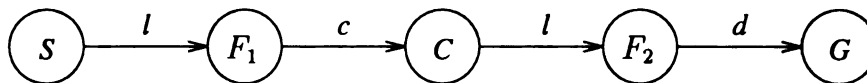


Figure 6.4. A simplified representation of Figure 6.3.

In Figure 6.4 it is assumed that the Planner generates the strings from the set of symbols  $\{c, d, l\}$ . Each symbol corresponds to the mode (C-mode, D-mode, L-mode) in which the Navigator operates during the navigation. Hence, the strings generated by the Planner become the input strings to the finite state automaton. For any path with  $k$  corner points, the state transitions of the finite state automaton can be depicted in Figure 6.5. By eliminating the states that correspond to the same behavior, a more general and concise finite state automaton that represents the internal behaviors of the Navigator can be obtained. This automaton is given in Figure 6.6, where a transition probability is also attached to each arc. The interpretation of each state is



given in Table 6.4.

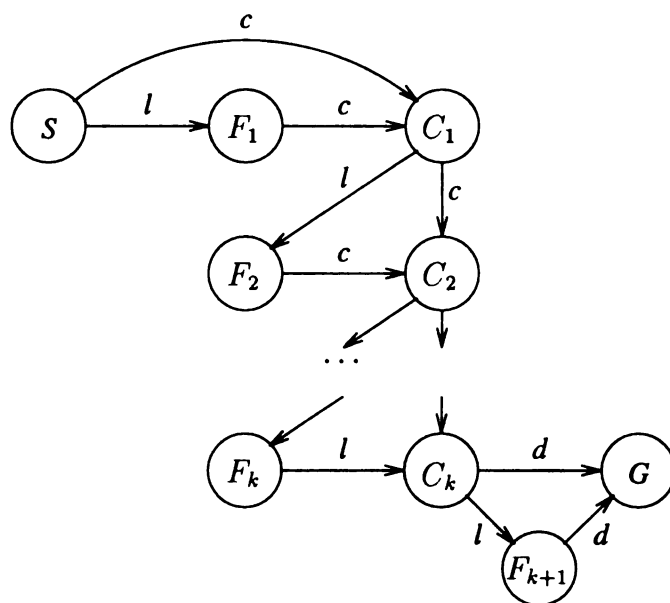


Figure 6.5. A finite state automaton for a path with  $k$  corners.

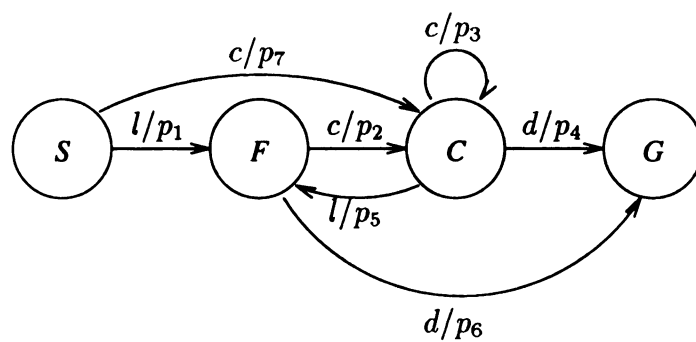


Figure 6.6. A generalized representation of the FSA in Figure 6.5.

The calculation of the transition probabilities in Figure 6.6 is performed in the following manner.

Table 6.4. Interpretation of states in Figure 6.6.

State	Meaning
$S$	Initial state.
$F$	The first ceiling on the corridor is detected.
$C$	The robot is in Corner mode.
$G$	The robot reaches the goal.

1. Probability  $p_1$ . This is the probability of detecting the first ceiling light immediately after the robot starts navigating (i.e., the first ceiling light in the first corridor on the path). We assume that the position of the robot in the map and the distance to the first ceiling light are known exactly. Consequently, the robot can rely on dead-reckoning to put itself under the first ceiling light in the first corridor, even if the light is off. This was also supported by the results of the experiments conducted approximately 100 times. Thus, the estimated probability value of  $p_1$  is effectively 1.0.
2. Probability  $p_2$ . This is the probability of successful completion of the L-mode. In my experiments, there were 2 failures out of 102 test runs. Therefore, the value of  $p_2$  is estimated to be  $1 - 2/101 = 99/101 \approx .9802$ . The probability  $p_6$  is the same as  $p_2$ .
3.  $p_3$ ,  $p_4$ , and  $p_7$  are the probabilities of successful completion of C-mode. In my experiments there were 3 C-mode failures out of 79 test runs. Therefore,  $p_3 = p_4 = p_7 = 76/79 \approx .9620$ .
4.  $p_5$  is the probability of successful completion of C-mode *and* then detecting the first ceiling light on the outgoing corridor after the robot makes a turn. In my experiments there was 1 failure of the latter event out of 57 runs. Thus, the estimated probability of the latter event is  $56/57$ . Consequently,  $p_5$  has a value of  $p_3 \times (56/57) = 4256/4503 \approx .9451$ .

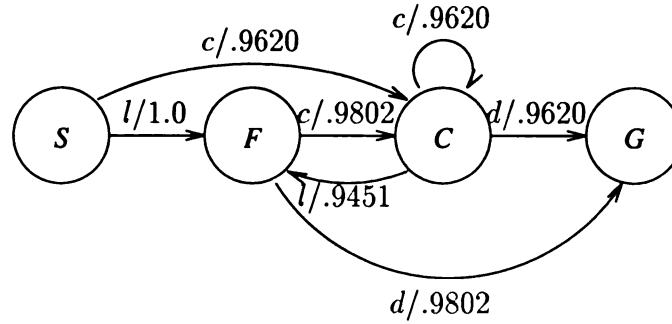


Figure 6.7. The FSA in Figure 6.6 after the estimation of the probability values.

If we assume that  $S = s_1$ ,  $F = s_2$ ,  $C = s_3$ , and  $G = s_4$ , then we can define a  $4 \times 4$  matrix  $P(x) = [p_{ij}(x)]$ , where  $p_{ij}$  is the probability of transition from state  $s_i$  to state  $s_j$  on input symbol  $x$ . In our situation the input symbols come from the set  $\{c, d, l\}$ , therefore we have three  $4 \times 4$  transition probability matrices as follows:

$$P(c) = \begin{bmatrix} 0 & 0 & .9620 & 0 \\ 0 & 0 & .9802 & 0 \\ 0 & 0 & .9620 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad
 P(d) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & .9802 \\ 0 & 0 & 0 & .9620 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad
 P(l) = \begin{bmatrix} 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & .9451 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Using the finite state automaton in Figure 6.7, the probability of a successful navigation from an initial point  $X_i$  to a final point  $X_f$  can be determined by calculating the *joint* probability of reaching state  $G$  from  $S$ . Table 6.5 presents the confidence levels of some paths excerpted from Tables 6.1 and 6.2.

The results in Table 6.5 show that confidence level decreases as path length increases. Here, the path length is measured as the number of mode changes in the local navigator. It is worth comparing this behavior with the solution to the “Where-Am-I” problem studied by Walsh. In his formulation of the problem, a navigator starts in an unknown position, and as it wanders around it has to determine, to a certain level

of confidence, its true position based on a sequence of detected landmarks [Walsh, 1992]. As the navigator wanders around, the uncertainty of the true position decreases because the navigator collects more evidence. On the contrary, the navigator in my experiment starts in a known position, and as it wanders around uncertainty of arriving at the goal position increases because the navigator encounters more “risks”. Thus, the interpretation of “uncertainty” differs in the two experiments.

Figure 6.7 shows that the completion of the C-mode has the lowest probability. The results of the test runs show that the two failures in the C-mode were caused by the “trouble” spots mentioned earlier. If we know in advance that the path does not include any of the “trouble” spots, then the values of probabilities  $p_3$ ,  $p_4$ , and  $p_7$  can be increased to  $78/79 \approx .9873$ . Consequently, the value of probability  $p_5$  (from state  $C$  to state  $F$ ) can be increased to .97.

Table 6.5. Confidence levels of reaching goal positions.

Path	Input String	Confidence
350R to 376L	<i>ld</i>	.962
341L to 311R	<i>lclcl</i>	$(1)(.9802)^3(.9451)^2 = .8412$
NE.men.R to 302R	<i>clclclccd</i>	$(.962)^4(.9451)^3(.9802)^3 = .6809$
341R to 381R	<i>lclclclclclclclcd</i>	$(.9451)^7(.9802)^8(.962)^3 = .5110$

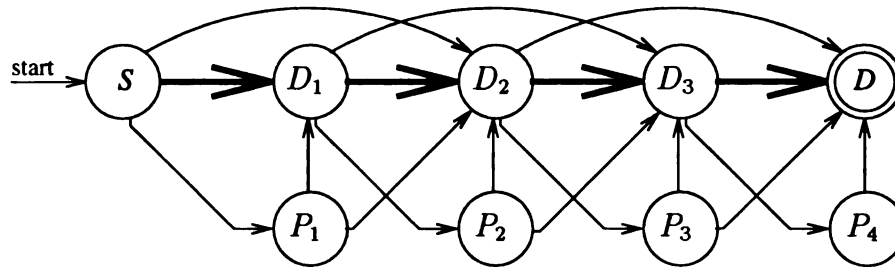


Figure 6.8. A probabilistic automaton for the PlFinder.

# F



# F

After elimination of equivalent states and attaching a probability value to each transition, the resulting finite state automaton is shown in Figure 6.9. For any transition  $p_i$  in the figure, we associate an event  $E_i$  as given in Table 6.6. In the table, “detecting true landmark” means that the system decides “yes” when the input image contains some landmarks. In the analysis, we consider only the input images that contain landmarks. However, when the input image contains no landmarks the finite state automaton remains in the same state.

Table 6.6. Transition probabilities in Figure 6.9.

Event	Description	Prob.
$E_1$	false negative at the beginning of a detection sequence.	0/11
$E_2$	false negative in the middle of a detection sequence.	19/162
$E_3$	a true landmark after the system decides a false negative. Due to the assumption that the system missed <i>at most</i> one landmark, the estimated probability value $p_3$ is 1.0.	1.0
$E_4$	a true landmark at the beginning of a detection sequence.	11/11
$E_5$	a true landmark in the middle of a detection sequence.	132/162
$E_6$	a false negative after the system encounters a false positive.	1/11
$E_7$	a false positive at the beginning of a detection sequence.	0/11
$E_8$	a true landmark after the system decides a false positive.	10/11
$E_9$	detecting a false positive.	11/162

From Table 6.6 it is evident that the transitions  $S \rightarrow M$  and  $S \rightarrow P$  are impossible. The probability of detecting all the door number plates in a corridor with  $N$  plates is:

$$(p_5)^N = .8148^N.$$

The low probability of this result was caused by the high number of occurrences of false negatives and false positives. In my implementation, the locations of door number plates are not encoded in the maps, therefore the detection algorithm cannot eliminate the false positives and false negatives. This also supports the necessity of information integration from several different sources. For instance, the ceiling light detection module integrates information from the camera, the map, and the robot. Moreover, the above result also indicates that this method should not be used solely for guiding the robot in the navigation.

The false negatives were caused mainly by the uncontrollable external events such as  $\beta$ -constraint violation, background interference, and occlusion. However, if these events do not occur, the door number plate detection is robust enough. Therefore, with the addition of digit recognition capability, this module can be used for verifying the destination point and for adjusting the final position of the robot so it stops very close to the destination point.

## 6.4 Summary

The results of indoor navigation, including ceiling light detection and door number plate detection are presented in this chapter. I have also designed two probabilistic finite state automata which have been used for describing the confidence level of operation of the robot. The transition probabilities in the finite state automata are computed from the empirical results of the experiments. The transition probabilities show that the confidence level of reaching the goal position using ceiling lights as

the landmarks is quite high. However, the confidence level of detecting all the door number plates on any path is not sufficiently high. The latter is caused by a number of false positives in the door number plate detection. This problem can be reduced if the door number plate locations are incorporated into the maps. The average speed of the robot during the navigation is approximately 4.7–17.875 m/min. The highest average speed was attained in the old wing and the lowest average speed was attained in the new wing of the Engineering Building. The navigation algorithm tends to overshoot the goal destination by a distance bounded by 1 meter.



# CHAPTER 7

## Conclusions and Future Research Directions

### 7.1 Conclusions

In this dissertation, I have presented my research in developing a control architecture for mobile robot navigation using the Client-Server model. Using this model, the navigation task of interest is decomposed into a number of client and server modules running concurrently. The model enables us to run the modules across several different processors or machines connected through a computer network.

The core of the Client-Server Architecture is a collection of server modules that provide resource sharing, access control, process synchronization, and information sharing among the client modules. These server modules are independent of each other in the sense that a server does not rely on the results of operation of any other server. There are two types of server modules: *Data Server* (DServer) and *Hardware Server* (HServer). This configuration is applicable regardless of the navigation task to be solved using the Client-Server model. The Data Server acts as the common communication media among all the clients in the system. It also provides a synchronization mechanism for the client modules. The other type of server, HServer,

provides access control to the hardware and peripherals on the robot. In the system that has been designed, there are three hardware servers: Robot Server, Proximity Server, and Camera Server. During the robot operation, two copies of Camera Server are running, one for each camera attached to the robot.

Depending on the navigation task at hand, the configuration of the client modules varies. A user has the choice of writing a single large and complicated client module or designing a number of smaller and simpler client modules. At present, we do not have a rigorous methodology for determining how big or small a module should be. The situation is similar to subdividing a large program into subroutines. However, software engineering provides guidelines for decomposing a program into a number of subroutines. Those guidelines can also be applied to the problem of decomposing a navigation task into a number of client modules.

It is important to note that there are no direct communication links among the client modules as well as among the server modules. Indirect communication links between two clients can be accomplished through the Data Server. Due to the independence of the client and the server modules, the interconnection among them can be modeled by a bipartite graph, where one set of nodes contains all the clients and the other set of nodes contains all the server modules.

Most of the existing control architectures fix the *topology* and/or the *functionality* of the constituent modules. The Subsumption Architecture, Colony-style Architecture as well as the Client-Server Architecture described in this thesis *neither* fix the topology *nor* the functionality of the modules. Instead, these architectures specify the “rules” for connecting one module to the other. The “rules” used in the Client-Server Architecture allow connections only between clients and servers and disallow interconnections among the clients or the servers.

It is also claimed that the Client-Server Control Architecture developed in this thesis provides an incremental development of a task. From my own experience, this

is typically how a navigation task is approached. For example, I started by developing a single client module for wandering around the hallway, then I modified the module to make it “wander with a goal”. Later, I learned that another client module to “direct” the first module with goal points is needed. The landmark detectors and a local mapper were then incorporated to make the navigation algorithm more robust.

In the indoor navigation experiments, the only processor available on the robot is a SPARCstation 1 for running all the client and server modules; however, the Client-Server Control Architecture described in this dissertation can be run on a number of processors. In the current configuration, the number of processes running concurrently and competing for CPU time is of the order of 10.

In conclusion, the Client-Server Architecture can be viewed as a standard framework for distributed control of mobile robot navigation systems. The decomposition of the navigation system into several client and server modules has a significant advantage in the sense that the hardware servers and the data server can be made small enough (size of the code) to run on a system with limited resources. I have also showed how some of the existing control architectures can be mapped onto the Client-Server Architecture. In that sense, this control architecture can be considered as a more general model.

All the existing control architectures that provide a distributed environment employ a “central” server for coordinating the constituent modules of the system. Using this approach, the “central” server manages a large number of activities in the system. In the Client-Server Control Architecture, some of the functionalities of this “central” server are delegated to a number of hardware servers. I believe that a client module generally performs a large amount of sensing and actuating the hardware on the robot. By providing hardware servers, in addition to the data server, and letting the client access each hardware server independently, the bandwidth of communication to the hardware is increased. I consider this as an improvement to the existing

control architectures.

## 7.2 Limitations and Suggestions for Improvements

In this section, I describe the limitations of my approach and suggest what can be done to improve the system.

- The ceiling light tracker module does not perform well when there is not enough contrast between the ceiling light and its surroundings.
- When the robot stops inside a “turning space” before it enters the ceiling light tracking mode in the outgoing corridor, its position cannot be determined accurately and hence the approximate distance to the first ceiling light in the outgoing corridor cannot be determined accurately. For this reason, I had to assume in my analysis that the first ceiling light in the outgoing corridor is always correctly detected. This situation can be improved by detecting the relative position of the robot to the corner walls. To do this, the robot has to collect sonar data and build a map while it is turning at a corner. Strong edges corresponding to the walls in the corner can then be detected in the map.
- When there is an open door in the hallway from which the ultrasonic sensors can pick up strong reflections, HIMM might detect the orientation of the door as the orientation of the hallway.
- When the robot travels at a speed of greater than 500 mm/sec, the maps created by HIMM are very sparse and the hallway orientation might not be detected reliably.

- Due to the communication time overhead, the client-server control architecture cannot be used for applications requiring a response time smaller than the communication time overhead. However, it is possible to use the Client-Server Control Architecture for real-time applications\* as long as the real-time constraints of each client can be scheduled properly. Mutka and Li have developed a tool for allocating a set of real-time tasks in a distributed environment such that the real-time constraints are guaranteed [Mutka and Li, 1993].
- The proximity of the final robot location to some goal position that lie in a turning space cannot be controlled. This is because the robot operates in a dead-reckoning mode while moving from the last ceiling light to the goal position.
- The current indoor navigation system can be used only for navigation between two locations on the *same* ring. In order to enable the robot to navigate between two locations on any ring, the map representation must include information on the relative positions of the rings.
- The current indoor navigation system assumes that the robot domain is composed of hallways. The parallel walls in the hallway constrain the motion of the robot and aid the robot to “flow” in the constrained space. This facilitates the task of the local navigator module in controlling the robot. To enable the robot to maneuver in a different building structure, the local navigator module must be modified and adapted to the new structure.
- In “short” hallways, *i.e.*, hallways containing less than three ceiling lights, heading correction cannot be made. This situation caused the robot to be “directionless” when arriving at the end of a “short” hallway. There are two possible

---

\*Due to common misconceptions about “real-time” computing, it is worth emphasizing that real-time computing is not identical to *fast* computing. The correctness of a program in a real-time computing system depends not only on the result of computation but also on the time at which the results are produced [Stankovic, 1988].

solutions to overcome this situation. A hardware solution is to provide a reliable source of heading on the robot. Another solution is to let the robot wander around from its current position, and let it figure out where it is by solving the “Where-Am-I” problem. Walsh has studied this problem, and conducted experiments on our mobile robot. The result of his simulation showed that recovery from positional error using vertical ribbons as landmarks is possible.

- Specular reflections from glass doors make the doors “invisible” to the sonars. Depending on the smoothness of the glass doors, using a higher frequency sonar will solve this problem.

### 7.3 Contributions of the Thesis

The contributions of this thesis have been:

1. Design, development, and implementation of the Client-Server model using an object-oriented approach. This model provides a framework for controlling a robot in a distributed manner, i.e., using a number of processors connected through a network of computers. To my knowledge, this is the first attempt of applying such a client-server model for robot navigation. The model developed in this thesis can be used for controlling multiple cooperating robots as well.
2. Successful design and implementation of a robot control model that enables resource access sharing, data sharing, and event synchronization in a distributed manner.
3. The implementation of C++ classes for the client-server interaction, low-level access to various components of the robot, process creation and destruction, interprocess synchronization in order to provide high-level abstraction and information consistency.

4. Development of an algorithm for transforming the hierarchical and Colony-style Architectures into the Client-Server Architecture. More specifically, an algorithm for determining the priorities of the modules in a Colony-style network containing both suppressor and inhibitor nodes is presented.
5. A functional indoor mobile robot navigation system with an ability to operate in a large building (approximate size of the floor plan is  $136 \times 116$  square meters). The robot is capable of navigating at 4.7–17.88 m/min and reaching goal positions within a proximity of 1 m as demonstrated by dozens of formal test runs.

## 7.4 Future Research Directions

In this section I give some suggestions for future research.

- Investigate the effect of adding more processors and use the `AF_INET` socket interface. This socket interface enables us to run the client-server model across a number of hosts.
- Implementation of a priority scheme in the servers that allows a client to set the priority of its request to a server. A request with a higher priority will be served before the other requests. This facility is particularly useful for scheduling the clients performing real-time tasks.
- Analyze the performance of the architecture in terms of the average completion time of client request. This can be performed by running a simulation of the architecture and to do this, the following parameters must be determined:
  - The patterns of the request transmissions by the client modules.
  - The completion time of each request on each server.

- o Number of client and server modules.
- The strategy employed by the clients in the current indoor navigation system is “sense and act as fast as possible”. This brute-force approach might overburden the servers due to the high traffic of “useless” requests from the clients. A particular client might require a lower sensor scanning rate in order to function properly. A study and analysis of sensing rate requirements of each client needs to be done to better utilize the available computational resources. Once these requirements are determined, the requests from the clients can be scheduled accordingly.
- The accomplishment of a functional mobile robot for indoor navigation resulting from this research opens the possibility for performing vision- and motion-related research, such as depth from motion, structure from motion, and object tracking.
- Experiment with a higher level capability of the Camera Server in order to provide services for *feature extraction*, instead of simply an intensity image grabber.
- Explore the utilization of wireless communication as the initial step towards the development of a navigation system for multiple robots. How to establish a secure system, i.e., to enforce authentication, authorization, and protection in the servers when wireless communication is used, needs to be studied.
- The current core of the Client-Server Architecture is a collection of a single Data Server and a number of Hardware Servers. This core can be extended to include a number of *special* clients that perform, for instance, event monitoring, failure detection and recovery, or resource usage monitoring.



# APPENDICES

# APPENDIX A

## A Map Generated from StickRep

The map of the third floor of the MSU Engineering Building is shown in Figure A.1. The StickRep representation of the map consists of four rings. The first ring corresponds to the outer ring enclosing the other three smaller rings; the second ring encloses region A; the third ring encloses region B, and the last ring encloses region C. The tick marks drawn on the map show the locations of the ceiling lights recorded on each ring. Physically, the floor plan of the MSU Engineering Building covers an area of approximately  $136 \times 116$  square meters.

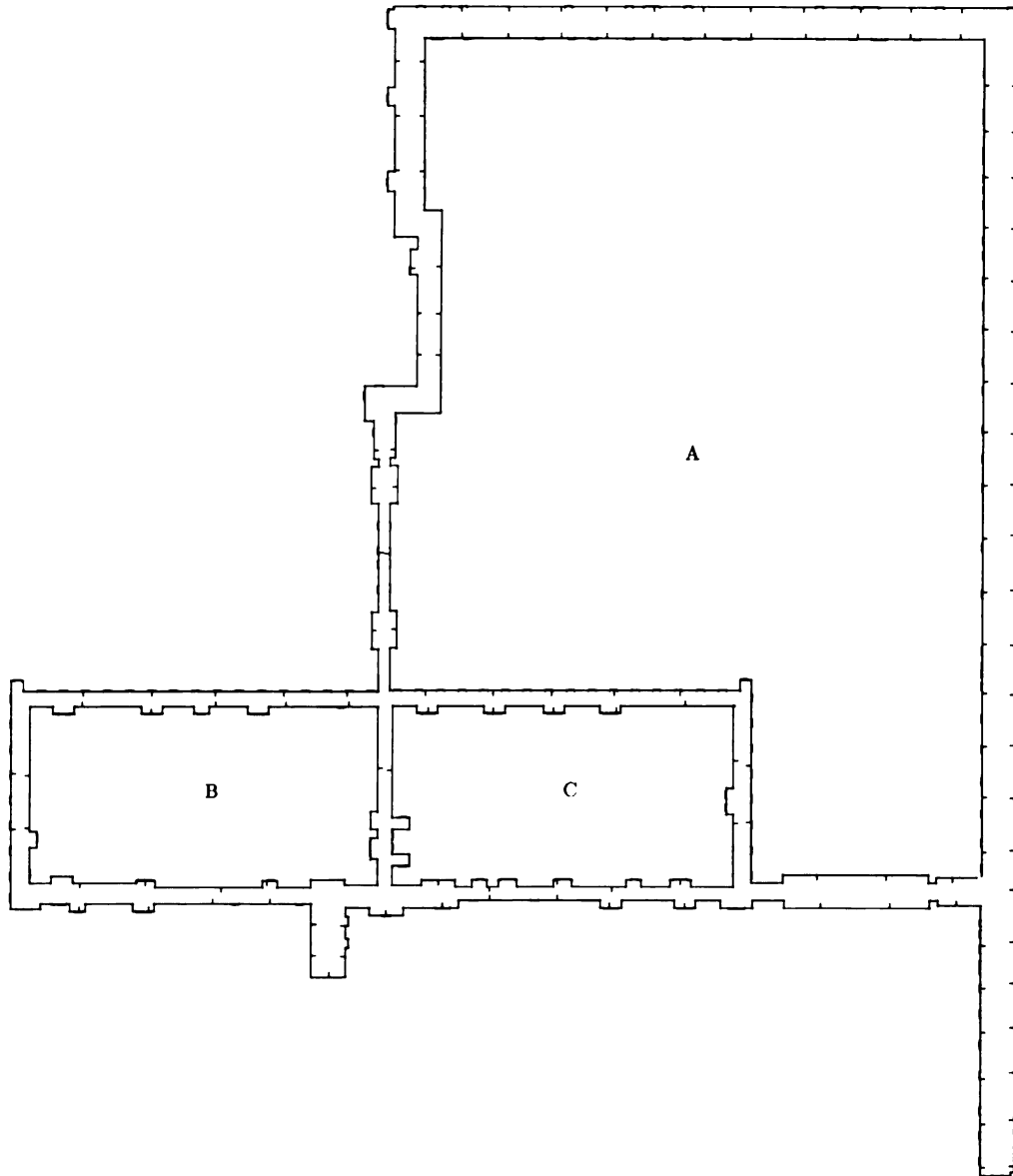


Figure A.1. A map of the third floor of the MSU Engineering Building.

# **BIBLIOGRAPHY**

## BIBLIOGRAPHY

- [Ahuja *et al.*, 1986] Sudhir Ahuja, Nicholas Carriero, and David Gelernter. Linda and friends. *IEEE Computer*, 19(8):26–34, August 1986.
- [Albus *et al.*, 1989] James S. Albus, Harry G. McCain, and Ronald Lumia. NASA/NBS Standard reference model for telerobot control system architecture (NASREM). Technical Report 1235, National Institute of Standards and Technology, April 1989.
- [Albus *et al.*, 1991] James Albus, Richard Quintero, Ronald Lumia, Martin Herman, Roger Kilmer, and Kenneth Goodwin. Concept for a reference model architecture for real-time intelligent control systems (ARTICS). Technical Report 1277, National Institute of Standards and Technology, April 1991.
- [Andrews, 1991] Gregory R. Andrews. Paradigms for process interaction in distributed programs. *ACM Computing Surveys*, 23(1):49–90, March 1991.
- [Arkin and Murphy, 1990] Ronald C. Arkin and Robin R. Murphy. Autonomous navigation in a manufacturing environment. *IEEE Transactions on Robotics and Automation*, 6(4):445–454, August 1990.
- [Arkin, 1989] Ronald C. Arkin. Motor schema-base mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, August 1989.
- [Asada, 1988] Minoru Asada. Building a 3-d world model for a mobile robot from sensory data. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 918–923, Philadelphia, Pennsylvania, 1988.
- [Asada, 1990] Minoru Asada. Map building for a mobile robot from sensory data. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6), November/December 1990.
- [Asimov and Frenkel, 1985] Isaac Asimov and Karen A. Frenkel. *Robots. Machines in Man's Image*. Harmony Books, New York, New York, 1985.
- [Ayache and Lustman, 1991] N. Ayache and F. Lustman. Trinocular stereo vision for robotics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(1):73–85, January 1991.

- [Ayache, 1991] Nicholas Ayache. *Artificial Vision for Mobile Robots*. The MIT Press, 1991.
- [Bagchi and Kawamura, 1992] Sugato Bagchi and Kazuhiko Kawamura. An architecture of a distributed object-oriented robotic system. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, North Carolina, July 1992.
- [Bares *et al.*, 1989] John Bares, Martial Hebert, Takeo Kanade, Eric Krotkov, Tom Mitchell, Reid Simmons, and William Whittaker. Ambler. An autonomous rover for planetary exploration. *IEEE Computer*, 22(6):18–26, June 1989.
- [Barnard and Fischler, 1982] Stephen T. Barnard and Martin A. Fischler. Computational stereo. *Computing Surveys*, 14(4):553–572, December 1982.
- [Beckerman and Oblow, 1990] Martin Beckerman and E.M. Oblow. Treatment of systematic errors in the processing of wide-angle sonar sensor data for robotic navigation. *IEEE Transactions on Robotics and Automation*, 6(2):137–145, April 1990.
- [Beus and Tiu, 1987] H. Lynn Beus and Steven S.H. Tiu. An improved corner detection algorithm based on chain-coded plane curves. *Pattern Recognition*, 20(3):291–296, 1987.
- [Borenstein and Koren, 1991a] Johann Borenstein and Yoram Koren. Histogrammic in-motion mapping for mobile robot obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 7(4):535–539, August 1991.
- [Borenstein and Koren, 1991b] Johann Borenstein and Yoram Koren. The vector field histogram—fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.
- [Bozma and Kuc, 1991] Ömür Bozma and Roman Kuc. Building a sonar map in a specular environment using a single mobile robot. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(12):1260–1269, December 1991.
- [Brady, 1985] Michael Brady. Artificial intelligence and robotics. *Artificial Intelligence*, 26:79–121, 1985.
- [Brady, 1989] Michael Brady. *Robotics Science*. The MIT Press, Cambridge, Massachusetts, 1989.
- [Brooks, 1986] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
- [Brooks, 1987] Rodney A. Brooks. A hardware retargetable distributed layered architecture for mobile robot control. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 106–110, Raleigh, North Carolina, 1987.

- [Brooks, 1988] Martin Brooks. Highly redundant sensing in robotics—analogy from biology: Distributed sensing and learning. In Julius T. Tou and Jens G. Balchen, editors, *Highly Redundant Sensing in Robotic Systems*, pages 35–42. Springer Verlag, 1988.
- [Brooks, 1991] Rodney Brooks. Small autonomous mobile robots: Sensing and action. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, page 452, Lahaina, Maui, Hawaii, 1991.
- [Buttazzo, 1992] Giorgio Buttazzo. HAREMS: Hierarchical architecture for robot experiments with multiple sensors. In Raja Chatila and Scott Y. Harmon, editors, *Workshop On Architecture For Intelligent Control Systems*, Nice, France, May 1992.
- [Canny, 1987] John F. Canny. *The Complexity of Robot Motion Planning*. The MIT Press, 1987.
- [Carriero and Gelernter, 1989] Nicholas Carriero and David Gelernter. Linda in context. *Communication of Association for Computing Machinery*, 32(4):444–458, April 1989.
- [Chocon, 1992] Hélène Chocon. Object-oriented design and distributed implementation of a mobile robot control system. In Raja Chatila and Scott Y. Harmon, editors, *Workshop On Architecture For Intelligent Control Systems*, Nice, France, May 1992.
- [Comer and Stevens, 1993] Douglas E. Comer and David L. Stevens. *Internetworking with TCP/IP. Volume III: Client-Server Programming and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.
- [Connell, 1987] Jonathan H. Connell. Creature design with the subsumption architecture. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1124–1126, Milan, Italy, 1987.
- [Connell, 1990] Jonathan H. Connell. *Minimalist Mobile Robotics. A Colony-style Architecture for an Artificial Creature*. Academic Press, 1990.
- [Courtney, 1993] Jonathan D. Courtney. Mobile robot localization using pattern classification techniques. Master's thesis, Michigan State University, 1993.
- [Crowley and Causse, 1992] Jim Crowley and Olivier Causse. Layered control of intelligent robotic devices. In Raja Chatila and Scott Y. Harmon, editors, *Workshop On Architecture For Intelligent Control Systems*, Nice, France, May 1992.
- [Crowley, 1985] James L. Crowley. Navigation for an intelligent mobile robot. *IEEE Transactions on Robotics and Automation*, RA-1(1):31–41, March 1985.
- [Crowley, 1989] James L. Crowley. Asynchronous control of orientation and displacement in a robot vehicle. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1277–1282, Scottsdale, Arizona, 1989.

- [de Almeida *et al.*, 1988] A. de Almeida, H. Araujo, J. Dias, L. de Sa, M. Criscosotomo, U. Nunes, and V. Silva. A multi-sensor distributed system for a flexible assembly cell. In Julius T. Tou and Jens G. Balchen, editors, *Highly Redundant Sensing in Robotic Systems*, pages 311–320. Springer Verlag, 1988.
- [Denavit and Hartenberg, 1955] J. Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanism based on matrices. *Journal of Applied Mechanics*, 77:215–221, 1955.
- [Dobrotin and Lewis, 1977] Boris Dobrotin and Richard Lewis. A practical manipulation system. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Massachusetts, 1977.
- [Dobrotin and Scheinman, 1973] Boris M. Dobrotin and Victor D. Scheinman. Design of a computer controlled manipulator for robot research. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 291–297, Stanford, California, 1973.
- [Dulimarta and Jain, 1993] Hansye Dulimarta and Anil K. Jain. Modular agents for robot navigation. In *Proceedings of the SPIE Conference on Sensor Fusion VI*, volume 2059, Boston, Massachusetts, September 1993.
- [Durrant-Whyte, 1988] Hugh F. Durrant-Whyte. *Integration, Coordination and Control of Multi-Sensor Robot Systems*. Kluwer Academic Publishers, 1988.
- [Elfes, 1989] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–58, June 1989.
- [Fedor and Simmons, 1991] Christopher Fedor and Reid Simmons. *Task Control Architecture User Manual*. Carnegie-Mellon University, July 1991.
- [Flynn and Brooks, 1988] Anita M. Flynn and Rodney A. Brooks. MIT mobile robots—What’s next? In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 611–717, Philadelphia, Pennsylvania, 1988.
- [Flynn *et al.*, 1989] Anita M. Flynn, Rodney Brooks, W.M. Wells III, and D.S. Barrett. The world’s largest one cubic inch robot. In *Proceedings: IEEE Micro Electro Mechanical Systems*, pages 98–101, Salt Lake City, Utah, 1989. IEEE.
- [Fok and Kabuka, 1991] Koon-Yu Fok and Mansar R. Kabuka. An automatic navigation system for vision guided vehicles using a double heuristic and a finite state machine. *IEEE Transactions on Robotics and Automation*, 7(1):181–189, February 1991.
- [Freedman, 1991] David H. Freedman. Invasion of the insect robots. *Discover*, pages 42–50, March 1991.
- [Fujimura, 1991] K. Fujimura. A model of reactive planning for multiple mobile agents. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1503–1509, Sacramento, California, 1991.



- [Furusho and Sano, 1990] J. Furusho and A. Sano. Sensor-based control of a nine-link biped. *International Journal of Robotics Research*, 9(2):83–98, April 1990.
- [Gauthier *et al.*, 1987] David Gauthier, Paul Freedman, Gregory Carayannis, and Alfred S. Malowany. Interprocess communication for distributed robotics. *IEEE Transactions on Robotics and Automation*, RA-3(6):493–504, December 1987.
- [Hackett and Shah, 1990] Jay K. Hackett and Mubarak Shah. Multi-sensor fusion: A perspective. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1324–1330, Cincinnati, Ohio, 1990.
- [Hager, 1990] Gregory D. Hager. *Task-Directed Sensor Fusion and Planning*. Kluwer Academic Publishers, 1990.
- [Hirose and Morishima, 1990] Shigeo Hirose and Akio Morishima. Design and control of a mobile robot with articulated body. *International Journal of Robotics Research*, 9(2):99–114, April 1990.
- [Hirose *et al.*, 1983] S. Hirose, M. Nose, H. Kikuchi, and Y. Umetani. Adaptive gait control of a quadruped walking machine. In Michael Brady and Richard Paul, editors, *Robotics Research: The First International Symposium*, Brentton Woods, New Hampshire, 1983.
- [Hopcroft and Ullman, 1979] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
- [Iyengar and Kashyap, 1989] S. Sitharama Iyengar and Rangasami L. Kashyap. Special issue on autonomous intelligent machines. *IEEE Computer*, 22(6):14–15, June 1989.
- [Kadonoff *et al.*, 1986] Mark B. Kadonoff, Faycal Benayad-Cherif, Austin Franklin, James F. Maddox, Lon Muller, and Hans Moravec. Arbitration of multiple control strategies for mobile robots. In W. Wolfe and N. Marquina, editors, *Proceedings SPIE. Mobile Robots*, volume 727, pages 90–98, 1986.
- [Kaelbling, 1987] Leslie Pack Kaelbling. An architecture for intelligent reactive systems. In P. Georgeff and A. Lansky, editors, *Reasoning About Plans and Actions*, pages 395–410. Morgan Kaufman, 1987.
- [Khatib, 1986] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, Spring 1986.
- [Kortenkamp *et al.*, 1993] David Kortenkamp, Marcus Huber, Charles Cohen, Ulrich Raschke, Clint Bidlack, Clare Bates Congdon, Frank Koss, and Terry Weymouth. Integrated mobile-robot design. *IEEE Expert*, pages 61–73, August 1993.
- [Kosaka and Kak, 1992] Akio Kosaka and Avinash C. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. *CVGIP: Image Understanding*, 56(3):271–329, 1992.

- [Kryzysztowicz and Long, 1990] Roman Kryzysztowicz and Dou Long. Fusion of detection probabilities and comparison of multisensor systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(3):665–677, 1990.
- [Kweon and Kanade, 1992] I.S. Kweon and Takeo Kanade. High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):278–292, February 1992.
- [Leven and Sharir, 1987] D. Leven and M. Sharir. An efficient and simple motion-planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. *Journal of Algorithms*, 8:192–215, 1987.
- [Lewis and Bejczy, 1973] Richard A. Lewis and Antal K. Bejczy. Planning considerations for a roving robot with arm. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 308–316, Stanford, California, 1973.
- [Lin *et al.*, 1989] Long-Ji Lin, Tom M. Mitchell, Andrew Philips, and Reid Simmons. A case study in robot exploration. Technical Report CMU-RI-TR-89-1, The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, 1989.
- [Lozano-Pérez and Wesley, 1979] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communication of Association for Computing Machinery*, 22(10):560–570, October 1979.
- [Lozano-Pérez, 1983] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Systems, Man, and Cybernetics*, C-32(2):108–120, February 1983.
- [Lucky, 1990] Robert W. Lucky. An EE in the land of lilliput. *IEEE Spectrum*, page 6, March 1990.
- [Lumelsky and Stepanov, 1987] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.
- [Luo and Lin, 1988] Ren C. Luo and Min-Hsiung Lin. Hierarchical robot multi-sensor data fusion system. In Julius T. Tou and Jens G. Balchen, editors, *Highly Redundant Sensing in Robotic Systems*, pages 67–86. Springer Verlag, 1988.
- [Lyons and Arbib, 1989] Damian M. Lyons and Michael A. Arbib. A formal model of computation for sensory-based robotics. *IEEE Transactions on Robotics and Automation*, 5(3):280–293, June 1989.
- [Maeda *et al.*, 1985] Y. Maeda, S. Tsutani, and S. Higihara. Prototype of multifunctional robot vehicles. In *Proceedings of '85 International Conference on Advance Robotics*, pages 421–428, Tokyo, Japan, September 1985.

- [Masaki, 1992] Ichiro Masaki, editor. *Vision-based Vehicle Guidance*. Springer-Verlag, 1992.
- [Matthies and Okutomi, 1989] Larry Matthies and Masatoshi Okutomi. Bootstrap algorithms for dynamic stereo vision. In *Sixth Multidimensional Signal Processing Workshop*, page 12, 1989.
- [Matthies, 1992] Larry Matthies. Passive stereo range imaging for semi-autonomous land navigation. *Journal of Robotics Systems*, September 1992.
- [McGhee *et al.*, 1980] R.B. McGhee, K.W. Olson, and R.L. Briggs. Electric coordination of joint motion for terrain-adaptive vehicles. In *Proceedings of 1980 SAE Automotive Engineering Cong.*, pages 1-7, Detroit, Michigan, 1980.
- [Miller, 1977] J.A. Miller. Autonomous guidance and control of a roving robot. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 759-760, Cambridge, Massachusetts, 1977.
- [Miller, 1990] D.P. Miller. Multiple behavior-controlled micro-robots for planetary surface missions. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 4-7, Los Angeles, California, 1990.
- [Miura and Shimoyama, 1983] H. Miura and I. Shimoyama. Dynamical walk of biped locomotion. In Michael Brady and Richard Paul, editors, *Robotics Research: The First International Symposium*, Brentton Woods, New Hampshire, 1983.
- [Moezzi *et al.*, 1991] Saied Moezzi, Sandra L. Bartlett, and Terry E. Weymouth. The camera stability problem and dynamic stereo vision. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 109-114, Lahaina, Maui, Hawaii, 1991.
- [Moorman and Ram, 1992] Kenneth Moorman and Ashwin Ram. A case-based approach to reactive control for autonomous robots. In *AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, Cambridge, Massachusetts, October 1992.
- [Moravec and Elfes, 1985] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 116-121, St. Louis, Missouri, 1985.
- [Moravec, 1980] Hans P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*. PhD thesis, Stanford University, Stanford, CA, September 1980.
- [Moravec, 1981] Hans P. Moravec. *Robot Rover Visual Navigation*. UMI Research Press, Ann Arbor, Michigan, 1981.
- [Moravec, 1983] Hans P. Moravec. The Stanford Cart and the CMU Rover. *Proceedings of the IEEE*, 71(7):872-884, 1983.

- [Mutka and Li, 1993] Matt W. Mutka and Jong-Pyng Li. A tool for allocating periodic real-time tasks to a set of processors. Technical report, Department of Computer Science, Michigan State University, East Lansing, Michigan 48824, 1993.
- [Nishihara and Poggio, 1983] H. Keith Nishihara and Tomaso Poggio. Stereo vision for robotics. In Michael Brady and Richard Paul, editors, *Robotics Research: The First International Symposium*, pages 489–505, Brentton Woods, New Hampshire, 1983.
- [Nitzan, 1985] David Nitzan. Development of intelligent robots: Achievements and issues. *IEEE Transactions on Robotics and Automation*, RA-1(1):3–13, March 1985.
- [Noreils, 1993] Fabrice R. Noreils. Toward a robot architecture integrating cooperation between mobile robots: Application to indoor environment. *International Journal of Robotics Research*, 12(1):79–98, February 1993.
- [Nourbakhsh *et al.*, 1993] Illah Nourbakhsh, Sarah Morse, Craig Becker, Marko Balabanonic, Erann Gat, Reid Simmons, Steven Goodridge, Harsh Potlapalli, David Hinkle, Ken Jung, and David Van Vactor. The winning robots from the 1993 robot competition. *AI Magazine*, pages 51–62, Winter 1993.
- [Ohmichi, 1985] T. Ohmichi. Development of the multi-function robot for the containment vessel of the nuclear plant. In *Proceedings of '85 International Conference on Advance Robotics*, pages 371–378, Tokyo, Japan, September 1985.
- [Ozaki *et al.*, 1983] N. Ozaki, M. Suzuki, and Y. Ichikawa. Tele-operated mobile robot for remote maintenance in nuclear facilities. In *Proceedings of '83 International Conference on Advance Robotics*, pages 371–378, Tokyo, Japan, September 1983.
- [Ozguner *et al.*, 1983] F. Ozguner, S.J. Tsai, and R.B. McGhee. Rough terrain locomotion by a hexapod robot using a binocular system. In Michael Brady and Richard Paul, editors, *Robotics Research: The First International Symposium*, Brentton Woods, New Hampshire, 1983.
- [Paul *et al.*, 1985] Richard P. Paul, Hugh F. Durrant-Whyte, and Max Mintz. A robust, distributed sensor and actuation robot control system. In Oliver D. Faugeras and Georges Giralt, editors, *Robotics Research. The Third International Symposium*, pages 93–100, Gouvieux, France, 1985.
- [Quintero, 1991] Richard Quintero, editor. *Architecture for Real-Time Intelligent Control of Unmanned Vehicle Systems. Minutes of Workshop #2 on Technology Options and Architecture Approaches*, January 1991.
- [Raibert *et al.*, 1983] M.H. Raibert, Jr. H.B. Brown, and S.S. Murthy. 3D balance using 2D algorithms? In Michael Brady and Richard Paul, editors, *Robotics Research: The First International Symposium*, Brentton Woods, New Hampshire, 1983.

- [Raibert, 1990] Marc H. Raibert. Foreword. *International Journal of Robotics Research*, 9(2):2–3, April 1990.
- [Roth-Tabak and Jain, 1989] Yuval Roth-Tabak and Ramesh C. Jain. Building an environment model using depth information. *IEEE Computer*, 22(6):85–90, June 1989.
- [Schneider *et al.*, 1989] Jeff Schneider, Hansye Dulimarta, Lal Tummala, and George Stockman. Robot navigation using ultrasonic sensors and labmate. Technical report, Department of Computer Science, Michigan State University, East Lansing, Michigan 48824, December 1989.
- [Schwartz and Sharir, 1983a] J.T. Schwartz and M. Sharir. On the piano movers' problem I. *Comm. Pure and Applied Mathematics*, 36:345–398, 1983.
- [Schwartz and Sharir, 1983b] J.T. Schwartz and M. Sharir. On the piano movers' problem: II. *Advances in Applied Mathematics*, 4:298–351, 1983.
- [Schwartz and Sharir, 1984] J.T. Schwartz and M. Sharir. On the piano movers' problem: V. *Comm. Pure and Applied Mathematics*, 37:815–848, 1984.
- [Sharir and Ariel-Sheffi, 1984] M. Sharir and E. Ariel-Sheffi. On the piano movers' problem: IV. *Comm. Pure and Applied Math.*, 37:479–493, 1984.
- [Sharir, 1989] Micha Sharir. Algorithmic motion planning in robotics. *IEEE Computer*, 22(3):9–20, March 1989.
- [Simmons and Krotkov, 1991] R. Simmons and Eric Krotkov. An integrated walking system for the Ambler planetary rover. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2086–2091, Sacramento, California, 1991.
- [Simmons *et al.*, 1990] R. Simmons, L.J. Lin, and C. Fedor. Autonomous task control for mobile robots. In *Proceedings of 5th IEEE International Symposium on Intelligent Control*, pages 663–668, 1990.
- [Simmons *et al.*, 1992] Reid Simmons, Chris Fedor, and Jeff Basista. *Task Control Architecture*. Carnegie Mellon University, version 6.17 edition, November 1992.
- [Simmons, 1992] R.G. Simmons. Concurrent planning and execution for autonomous robots. *IEEE Control Systems Magazine*, 12(1):46–50, February 1992.
- [Stankovic, 1988] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, October 1988.
- [Steenstrup *et al.*, 1983] Martha Steenstrup, Michael A. Arbib, and Ernest G. Manes. Port automata and the algebra of concurrent process. *Journal of Computer and System Sciences*, 27:29–50, 1983.

- [Storjohann *et al.*, 1990] K. Storjohann, Th Zielke, H.A. Mallot, and W. von Seelen. Visual obstacle detection for automatically guided vehicles. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 761–766, Cincinnati, Ohio, 1990.
- [Sugihara and Suzuki, 1990] K. Sugihara and I. Suzuki. Distributed motion coordination of multiple mobile robots. In *Proceedings of 5th IEEE International Symposium on Intelligent Control*, pages 138–143, 1990.
- [Sutherland and Ullner, 1984] Ivan E. Sutherland and Michael K. Ullner. Footprints in the asphalt. *International Journal of Robotics Research*, 3(2):29–36, 1984.
- [Tanenbaum, 1992] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice-Hall, Inc., 1992.
- [Tang and Lee, 1990] Y.C. Tang and C.S.G. Lee. A geometric feature relation graph formulation for consistent sensor fusion. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 188–193, Los Angeles, California, 1990.
- [Thorpe *et al.*, 1991a] C. Thorpe, M. Herbert, T. Kanade, and S. Shafer. Toward autonomous driving: the CMU NavLab. I. Perception. *IEEE Expert*, 6(4), August 1991.
- [Thorpe *et al.*, 1991b] C. Thorpe, M. Herbert, T. Kanade, and S. Shafter. Toward autonomous driving: the CMU NavLab. II. Architecture and systems. *IEEE Expert*, 6(4), August 1991.
- [Thorpe, 1984] C.E. Thorpe. Path relaxation: Path planning for a mobile robot. Technical Report CMU-RI-TR-84-5, The Robotics Institute, Carnegie-Mellon University, April 1984.
- [Tou, 1988] Julius T. Tou. A knowledge-based system for redundant and multi sensing in intelligent robots. In Julius T. Tou and Jens G. Balchen, editors, *Highly Redundant Sensing in Robotics Systems*, pages 3–20. Springer-Verlag, 1988.
- [TRC, 1991] TRC. *LABMATE User Manual. Version 5.21L-e*. Transitions Research Corporation, 1991.
- [Walsh, 1992] Stephen J. Walsh. *Indoor Robot Navigation Using A Symbolic Landmark Map*. PhD thesis, Department of Computer Science, Michigan State University, East Lansing, Michigan, 1992.
- [Wise and Ciscen, 1992] J.D. Wise and Larry Ciscen. TelRIP distributed application environment operating manual. Version 1.6. Technical Report 9103, Universities Space Automation/Robotics Consortium, March 1992.
- [Wong and Payton, 1987] Vicent S. Wong and David W. Payton. Goal-oriented obstacle avoidance through behavior selection. In *SPIE Volume 852. Mobile Robots II*, pages 2–10, 1987.

- [Yamauchi and Nelson, 1991] Brian Yamauchi and Randal Nelson. A behavior-based architecture for robots using real-time vision. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1822–1827, Sacramento, California, 1991.