THESIS

60386831

This is to certify that the
dissertation entitled

DAV: A HUMANOID PLATFORM AND DEVELOPMENTAL LEARNING WITH CASE STUDIES

presented by

SHUQING ZENG

has been accepted towards fulfillment
of the requirements for the

PHD degree in COMPUTER SCIENCE

John Weng

Major Professor's Signature

8/20/04

Date

Doctoral Dissertation

MSU is an Affirmative Action/Equal Opportunity Institution

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

6/01 c:/CIRC/DateDue.p65-p.15

DAV: A HUMANOID PLATFORM AND DEVELOPMENTAL LEARNING WITH CASE STUDIES

By

Shuqing Zeng

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2004

Abstract

Dav: A humanoid platform and developmental learning with case studies

By

Shuqing Zeng

Motivated by the autonomous development process of humans, we are interested in building a robot that learns online in real-time by interacting with the environment through sensors and effectors, and develops a representation of the environment and tasks autonomously. We call such a robot a developmental robot, which should satisfy eight challenging requirements: environment openness, high-dimensional sensors, completeness in using sensory information, online processing and real-time speed, incremental processing, performing while learning, and handling increasingly more complex tasks.

To partially satisfy the above-mentioned challenging requirements for a developmental robot, this research contributes:

- Building the body of a humanoid robot, called Dav. Embodiment is the most overlooked aspect of human intelligence in traditional artificial intelligence. The morphology of the robot is preferably of a humanoid form. This human-like shape may allow humans to interact with the robot more naturally and provide

the similar physical constraints as humans during those interactions. Furthermore, directly situating the robot in the physical world relieves the needs for a symbolic intermediary representation. The actual physical world serves for this representation. Therefore, as a part of this research, we built the Dav robot, which is the only untethered mobile humanoid currently in the universities of the United States.

- Creating a developmental architecture and a modified version of incremental hierarchical discriminant analysis (IHDR) algorithm. This thesis also presents a theory of developmental mental architecture. Five architecture types, from the simplest Type-1 (observation-driven Markov decision process) to Type-5 (DOSASE MDP), are introduced. Further, we present the architecture design of the Dav robot. The framework of the Dav architecture is hand-designed, but the actual controller is developed, i.e., generated autonomously by the developmental program through real-time interactions with the real physical environment. We present the Dav architecture and the major components that implement the architecture.

- Based on the above developed robotic platform and the proposed architecture, we have designed and implemented a range-based navigation system.

ACKNOWLEDGMENTS

I am indebted to my advisor, Professor John J. Weng, who has been the guiding force of my professional development and an inexhaustible source of ideas throughout my Ph.D program at Michigan State University. Without his kind assistance and advice, this dissertation would not have been completed. I am grateful to Professors Anil K. Jain, George C. Stockman, Zhengfang Zhou and Ning Xi, for serving on my comittee and providing critical comments on my research work.

Dav has been an enormous scientific project which has benefited from the efforts and ideas of many people. This project simply could never have happened without Professor John J. Weng. Thank you, John, for offering me the opportunity to build a system from scratch. Thanks also to the members of the Dav team, past and present: Dr. Jianda Han, Jingliang Wang and David M. Cherba. Four of us have built extensively on the many previous iterations of hardware and software design. Particularly, David has been invaluable in making the torso and arms a usable resource, in helping me testing the embedded distributed system, mechanical system and software system.

Thanks also to the members of the EI Laboratory, past and present, at Michigan State University. I benefited from many discussions with my colleagues. Thanks

go to Wey-Shiuan Hwang, Yilu Zhang, Nan Zhang, Xiao Huang, Yi Chen, Ameet Joshi, Jason Massey, Raja Ganjikunta, Gil Abramovich, Christopher Osborn, Tham Kengyew, and David Bordoley.

I am greatly indebted to my parents for their continuous encouragement and support throughout my school years. Thanks also go to my wife, Yanhua Chen, for her patience and support during the years at Michigan State University.

TABLE OF CONTENTS

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Industrial robots have been widely and successfully used in controlled industrial settings for a variety of tasks. Great challenges exist for robots to work in general human environments. These are characterized by unknown, unpredictable and changing objects, and tasks that cannot be well specified algorithmically. We call them muddy tasks. Vision, audition and language are at the forefront among cognitive capabilities that are required for muddy tasks. Autonomous mental development seems to give a possible way for machines to handle these muddy tasks [117].

Not until the late 1980's did embodiment receive attention in the artificial intelligent (AI) community, when it was popularized by the behavior-based approach [7,18]. Action generations have been fueled by impressive advances in robot construction, especially in humanoid platforms.

The first humanoid robot was "Elektro the Motorman" at the 1939 New York World Fair. It was a legged humanoid robot that "talked," moved its limbs, and played some arithmetic tricks. With the arrival of computers, more impressive humanoid robots have been constructed to demonstrate their specially designed skills. Wabot-2 [83] (1980-1984) was designed and built solely to read musical notes and play piano.

WABIAN [61] was a humanoid robot capable of walking and dancing. An inspiring engineering achievement in humanoids is the P2 robot from Honda, which is capable of walking and climbing stairs [39]. More recent humanoid systems include H6 [69], ETL-Humanoid [68], and Robonaut [59]. These projects primarily focused on the challenges of mechatronic design and integration of anthropomorphic bodies and programming for generating action sequences.

1.1.1 Learning, Perception and World Models

Actions can be programmed in or learned through practice. However, an action is not very useful if the robot does not have a perception capability, not knowing in what environmental context to produce an action. Perception is well-known to be very challenging and typically a world model is predesigned. What are the limitation of such a hand-crafted model? Since these issues are closely related to robot construction, we discuss these issues in this section.

1.1.2 Actions through Action Practice

Instead of programing action into a robot, another category of efforts aims to train the robot to produce desired behaviors. With redundant degrees of freedom, it is challenging to learn behaviors, especially when training time is limited or when training must be conducted in real time. Such projects include LWPR on a SARCOS robot by Schaal et al. [97], the simulated humanoid of a 37 degrees of freedom by Mataric et al. [11], and the scheduling degrees of freedom by Grupen et al. [26] motivated the early child motor development.

1.1.3 Perception-guided Actions

In contrast with the above efforts that concentrate on behavior generation without requiring sophisticated perception, a series of efforts deals with perception and

perception-guided behaviors. The main challenging perceptual sensing modalities include vision, audition, and high-dimensional touch with many touch elements (including range sensing). No matter how complex a behavior (action sequence) is, the behavior is useful only when the robot can produce the behavior in the correct perceptual context and can suppress it in all other contexts.

Studies of perception driven behaviors have had a long history. A well-practiced approach is that a human programmer defines features (e.g., edges, colors, tones, etc) or environmental models for the system. For example, the recent works on H2 from Honda, Cog [19] and Kismet [14] produced impressive perception driven behaviors. Some speech features and visual features are used to establish audio-visual association by an active camera on a robot hand [78].

Programming perceptual capability using human defined features is a quick way to produce results in a controlled setting. A fundamental limitation is that the resulting robot cannot work well in unknown, partially unknown, or changing environments. Weng & Chen [110] explained the inefficiency of human defined features, the insufficiency of hand crafted models, and the difficulty in programming their applicability checker for an unknown environment.

1.1.4 World-model Free Approach

The world-model free approach has been studied somewhat independently in two research communities: robotics and computer vision. In the robotics community, they are called behavior-based methods [18] [7]. The emphasis is to concentrate on behavior generation directly from range sensor readings or directly from images [65] [43].F

The model-free approach in the computer vision community is called appearance-based methods. The appearance-based method started around 1990 [50] [94] and it appears to be the most popular method in the computer vision community now. Appearance-based methods use statistical tools directly on high-dimensional image vectors, normalized using the mean and variance of the pixels. Thus, an image with m

rows and n columns is considered as a d-dimensional vector $v \in R^d$, where $d = mn$. Since high-dimensional statistical tools are directly applied to the vectors in space R^d, these type of method can take into account not only correlations between nearby pixels, but also far-away pixels. The need to process a high-dimensional sensory vector brings out a sharp difference between behavioral modeling and perceptual modeling: the effectors of a robot are all known but the sensory space R^d is extremely complex and unknown and, therefore, very challenging.

The power of learning directly from entire images has been demonstrated in vision-guided autonomous navigation. They include ALVINN [74], which uses multilayer perceptron (MLP) networks; ROBIN [76], which uses radial basis function networks (RBFN); SHOSLIF [109] and state-based SHOSLIF [23], which use Fisher's multidimensional discriminant analysis in building a regression tree. As has been shown in the comparison study of SHOSLIF [23, 109], statistical regression methods perform significantly better than traditional non-statistics based networks such as multi-layer perception and radial base function networks.

Perception-learning-based action generation relieves the human programmer from the intractable task of programming perception. However, the above learning process is not fully autonomous in the following sense: (1) The human programmer designs a part of the task-specific representation, e.g., features and states. (2) The human programmer has direct control over internal modules during learning. This mode of manual development fundamentally limits the scalability of behavioral and perceptual capabilities.

1.1.5 Autonomous Mental Development

M. Sur et al. [55] find that if the optic nerve circuits carrying vision signals are re-wired into the auditory cortex of a young ferret; the auditory cortex, the brain zone usually assigned for sound processing, can develop the properties that are observed only in visual cortex. This reveals a point that a developed mammal brain is a prod-

uct of active, autonomous and extensive interactions with the environment around it. Motivated by this discovery and others, a new paradigm, autonomous mental development (AMD) proposed by [118]:

- A human designer designs a robot body according to the general ecological condition in which the robot will work (e.g., onland or underwater).
- A human programmer designs a task-nonspecific developmental program for the robot.
- The robot starts to run the developmental program and develops its mental skills through real-time, on-line interactions with the environment, which includes humans.

A robot built in this paradigm is called a developmental robot. Early examples of such developmental robots include SAIL (short for Self-organizing, Autonomous, Incremental Learner) robot [105,130,132] at Michigan State University and the Darwin V robot [5] at The Neurosciences Institute, San Diego. These robots share the characteristic that system behaviors are generated through online real time interactions with the environment.

Motivated by the above-mentioned research, this work follows a new engineering paradigm, autonomous mental development [117], which emphasizes embodiment, environmental openness, completeness in using sensory information, online processing, real-time speed, incremental processing, performing while learning, and scale-up to complex tasks. This dissertation is based on the research work of a mobile robot project at Michigan State University funded by DARPA (Defense Advanced Research Project Agency) DRS program. The purpose of the project is two-fold: to provide a general purpose, flexible, and dextrous robotic platform from engineering aspects and to understand human autonomous mental development from scientific aspects.

1.2 Outline of the Dissertation

The scope of this dissertation is about building the body of a humanoid robot and applying it to the applications of autonomous navigation.

Chapter 2 describes the design of the Dav robot. First, the mechanism and kinematic model of the robot are introduced. To facilitate the joint-level control, the robot's dynamic analysis is derived. Then, we analyze the robot's mobile base in detail since it is related to the later navigation task. The control system is also presented in this chapter.

Chapter 3 presents a theory of developmental mental architecture. Five architecture types from the simplest Type-1 (observation-driven Markov decision process) to Type-5 (DOSASE MDP) are introduced. The properties and limitation of the simpler one are discussed before the introduction of the next more complex ones. Furthermore, we present the architectural design of the Dav robot. The framework of the Dav architecture is hand-designed, but the actual controller is developed, i.e., generated autonomously by the developmental program through real-time interactions with the real physical environment. We present the Dav architecture and the major components that realize the architecture. The designed architecture for Dav is the next generation version from its extensively tested predecessor - the SAIL developmental robot.

Chapter 4 outlines a novel developmental perception-driven control architecture (DPDCA), which is adapted from the overall architecture presented in Chapter 3. Dav's range-based indoor navigation experiment is used to illustrate the power of DPDCA.

Chapter 5 introduces a learning-based approach to obstacle-avoidance behavior with a global planner. The robot operated in a partially known bounded 2-D environment populated by unknown static or moving obstacles (with slow speeds) of arbitrary shape. The sensory perception was based on a laser range finder. Local re-planning was implemented to account for unknown and dynamic parts of the en-

vironment. To greatly reduce the number of training samples needed, attention was used to learn the obstacle-avoidance behavior. An efficient, real-time implementation of the approach has been tested, demonstrating smooth obstacle-avoidance behaviors in a corridor with a crowd of moving students as well as static obstacles.

Chapter 6 summarizes the contributions of the research and concludes the dissertation with recommendations for the improvement of the Dav robot and for future research work might be conducted on the humanoid robot.

Chapter 2

The Body Design of the Dav Humanoid Robot

In this chapter, we first discuss the body requirements of a developmental robot. Section 2.2 presents the mechanism and kinematic analysis of the Dav robot whose overall appearance is shown in Fig. 2.1 and detailed specifications are listed in Table 2.1. We derive the dynamic analysis of Dav's serially driven manipulator in Section 2.3 and the mobile base in Section 2.5. The joint-level control for Dav is presented in Section 2.4. The design of the distributed control system is outlined at the end in Section 2.6.

2.1 The Body Requirement for a Developmental Robot

The essence of autonomous mental development by machines is the capability of learning directly, interactively, and incrementally from the environment using on-board sensors and effectors. A developmental robot should be a real robot that runs a developmental algorithm and is allowed to learn and practice autonomously in the real physical world.

The goal of the Dav robot project is to provide a next generation robot platform for

Table 2.1: The specification and installed sensors on the Dav robot.

Specification	Installed sensors
• Mobile humanoid, self-contained • 700mm(L) x 700mm(W) x 1800(cm) body dimensions • 242 Kg • 43 degrees-of-freedom (DOF). The mechanisms include an 8-DOF drive-base, a 2-DOF torso, a 3-DOF neck, a 5-DOF head, two 7-DOF arms, and two hands • Main computer includes a quadruple Pentium Xeon III desktop with 2 Giga memory and 100 Giga SCSI hard drives • 11 Motorola MPC555 embedded processors as low-level Actuator Control Unit (ACU). • All ACUs are connected to the main computer via CAN buses Wireless ethernet, Cisco Airo 340 series PCI card and access point • 2 Hauppauge WinTV PCI cards as image grabber • A Sound Blaster Live! 5.1 from Creative Labs • Four 110 AmpH lead acid batteries. 8-hour continuous operation without recharging	• Two color micro CCD cameras, QN42H from ELMO • Two Microphones with pre-amplifiers • Laser range scanner (covering 180 deg at 0.5 deg resolution), PLS, SICK company • Shaft encoder for each DC motor • Two acceleration Sensors (Analog ADXL202) mounted on the head to mimic vesicular organ • Current sensor or strain gage to sense the torque at each joint

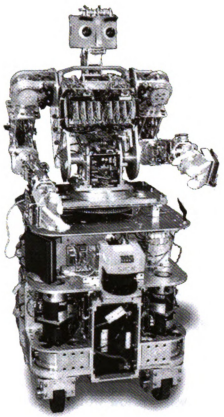


Figure 2.1: Dav: a mobile humanoid robot

research on robot mental development. Because Dav is meant for mental development, its design requirements are not the same as other humanoid robots in the following perspectives:

- *Mobility.* There have been a number of studies that show the importance of autonomous movements in developing the humans' and animals' senses of the world and their normal behaviors. "Kitten carousel" [38] especially shows that autonomous mobility is necessary for the kitten to develop the cliff avoidance behavior. Therefore, it appears that autonomous mobility is a necessary condition for mental development. The implication of mobility and autonomy means that the robot must be totally self-contained with no tethers. The amount of power and computational resources for an eight-hour period of actions require batteries of such capacity that a biped robot is not feasible for current

technology and a rolling base with high mobility was chosen.

- *Manipulators.* A developmental platform should be equipped with dexterous arms and hands. In other words, it should have sufficient degrees of freedom and range of motion. It is desirable for a developmental humanoid to take a human shape so that they can use a wide variety of tools designed for humans.

  However, the design faces conflicting conditions: power and size. We like to have a powerful limb for a larger payload. The higher the power, the larger the motors and other supporting devices. But the size is limited by a typical human arm size. The electronic boards of each limb must be contained in the same limb. The most challenging activity of all is to satisfy this self-containment requirement for Dav's hands. We expect that this problem will be better solved if the motors are custom designed.

- *Sensing.* Developmental robots grow world representations autonomously from a sensorimotor stream. Also the controller is perceptual driven architecture. Therefore in a general setting, major sensor modalities such as visual, auditory, touch, and somatosensory should be integrated.

- *Computational resource.* The developmental program must be able to update its memory for all the sensors within a fraction of a second, e.g., 10ms for sound and 100ms for vision. The control signals of effectors must also be updated at a rate of at least 10Hz.

  Currently, hardware implementation is not practical for an algorithm that is expected to change very often throughout the research project. Thus, the developmental program is to be implemented by software.

  The memory of a developmental robot is expected to be very large. Thanks to the logarithmic time complexity of IHDR [44] [114], it is expected that the refresh rate of the developmental program will not slow down too much when

the memory size increases. However, this is still an open question, since we need to test a robot with more extensive learning experiences.

- *Power supply.* Developmental robots need extensive training and practice. We should have enough battery capacity to allow the robot to operate without charging the batteries frequently. The battery capacity is still an unsolved problem, as with electrical cars. Currently, with a consideration of cost and capacity availability, we still think that a reasonable choice is sealed rechargeable batteries.

- *Wiring.* Wiring is a challenging issue for humanoids, especially for developmental ones. Due to a very large number of sensors and motors in the limbs, the number of wires required between each joint reaches a few hundred if data processing and computation are centralized at a single location.

  Such a large number of wires cause at least two major problems. First, the rigidity of the wire bundle interferes with dexterous manipulators. Second, the wire bundles cannot sustain repetitive bending during extended usage and will break.

  We have adopted a network scheme, namely, a distributed embedded control system is designed and implemented. Further mechanical parts (especially joints) are designed in such a way that the wires can go through the mechanical joints with minimal bending to increase the reliability of the system and reduce maintenance frequency.

- *Torque command.* In biology the muscles are controlled by the action potential in motor neurons. The larger the action potential, the more muscle fibers are recruited to contract. Therefore, the action potential can be regarded as a torque command. Torque commands can be easily implemented by using the DC motors except in the mobile base, which is a non-redundant dynamic system

with non-holonomic constraints. A decoupling controller is proposed to realize torque commands on the mobile base.

2.2 Mechanism and Kinematic Analysis

The Dav robot is designed as a wheel-based mobile humanoid with 43 degree-of-freedoms (DOF) to achieve the above-mentioned requirements. Figure 2.1 shows the body of the Dav robot. It is contained in a volume of 750(l) x 750 (w) x 1700(h) mm in default posture. The DOF distribution is shown in Fig. 2.2. The mechanical body contains over 300 types with a total of over 900 custom-made parts. All of the mechanical drawings of the custom-made parts were generated by *Solidworks*.

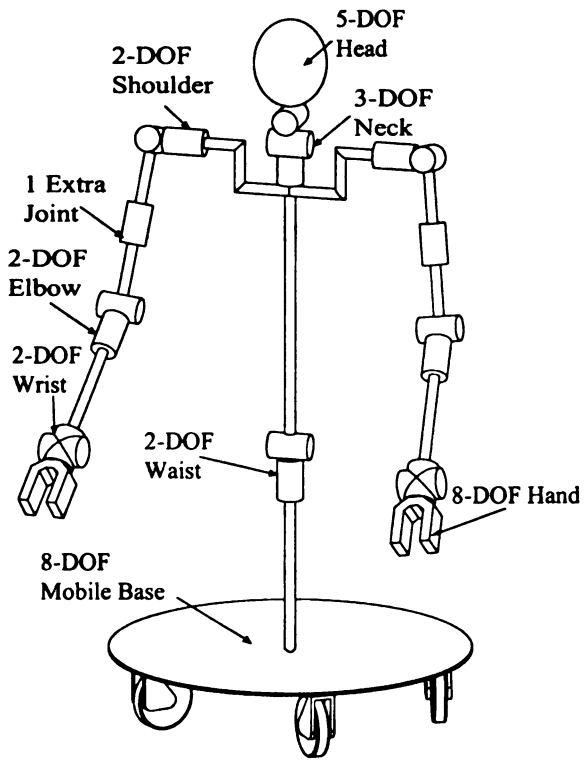


Figure 2.2: Location of degrees of freedom for Dav

Dav consists of a mobile base, a torso, two arms, a head, and a neck. Each module is designed in such a way that it is easy to assemble and maintain. In the following, we present the mechanism and kinematic analysis of these modules in detail.

2.2.1 Mobile Base

High mobility is essential for robots to enlarge their scope of sensing, which is essence for developmental robots having rich sensorimotor experiences. However, the car-like wheeled robots have nonholonomic constraints that reduce the mobility and make planning and learning difficult. Omni-directional vehicles are hence developed. Omni-directional (holonomic) vehicles can move in any direction immediately at robots' any configuration and the motion can be decoupled into three independent components (x, y, ϕ). This feature avoids the complicated manueuvering sequences to bring the vehicle to a side position and, thus, facilitates control and learning.

There are a variety of designs of omni-directional vehicles in the literature. These vehicles can be categorized into two classes based on their wheel design: a special wheel design and a conventional wheel design. Most special wheel design is based on a concept that the robot actively drives in one direction while allowing passive motion in the other [8, 120]. However, vehicles based on this holonomic wheel design have the following disadvantages under practical environments:

- Complicated mechanism
- Limited load capacity because of a reduced contact area
- Limited accuracy to estimate the pose of the vehicle by dead-reckoning because of the difficulty to decide the passive motion of wheels
- Difficulties in coping with uneven floor and other floor irregularities in cases of four-wheel driving vehicle without suspension
- Low clearance from the floor makes this design hard to deal with the floor's step features (e.g., an exposed thick wire on the carpet).

To overcome those difficulties, the other category (conventional wheel design) is proposed. For example, Wada & Mori [101] gives a design to achieve near omni-

directional mobility by using two active casters. Nomadic XR4000 is another example with four casters [40].

Because of its high load capacity, simple mechanical design, and tolerance to floor irregularities, conventional wheels with offset are chosen to realize omni-directional mobility. Shown in Fig. 2.3, Dav's mobile base has four wheels, and each one is driven by two DC motors for driving and steering separately.

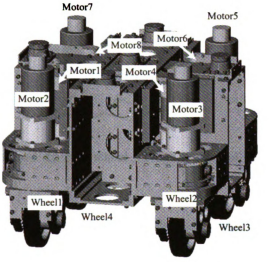


Figure 2.3: The *Solidwork*'s rendering of Dav's mobile base.

Fig. 2.4 shows an offset steered driving wheel (an actuated caster). At this configuration, the speed from the driving motor is along the wheel's orientation while the speed from the steering motor is perpendicular to the wheel's orientation. Thus it is possible to translate the steering pivot point to an arbitrary direction by controlling the two motors.

Supported by the steering pivot points, the vehicle frame can be considered as a plate controlled by four supporting sticks. The plate's translation and rotation movements are determined by the sticks' speeds. Since each stick can move in an arbitrary direction, the vehicle's movement is omni-directional. It is worthy noting that this is an over-actuated system because the four sticks' motion over-determines the vehicle plate's movement. A synchronizing control scheme is hence needed. In

the following, we derive the kinematic relation between the base's joint speeds and the vehicle speeds. Section 2.3 gives the dynamic analysis and control scheme for the base.

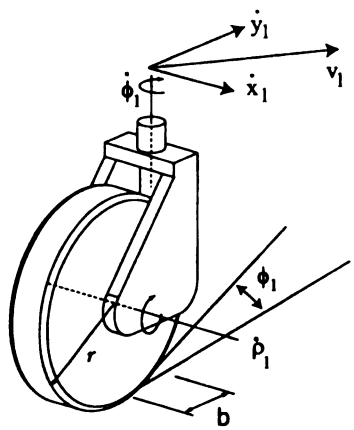


Figure 2.4: The offset steered driving wheel. The steering pivot's speed can be arbitrary **direction.**

For kinematic analysis, we introduce the following symbols that are illustrated in **Fig.** 2.5:

$\mathbf{v}$	The speed of the vehicle. The components are $[v_x, v_y, \omega]^T$
$\mathbf{q}$	The configuration of the vehicle. The components are $[\phi_1,\ \rho_1,\ ...,\ \phi_4,\ \rho_4]^T$
$\boldsymbol{\phi_i}$	Angle of the ith steering joint, $i = 1, ..., 4$
$\dot{\boldsymbol{\rho}}_i$	Speed rates of driving wheel
r, b	The radius and offset of the wheel

Consider the ith wheel, $i = 1, ..., 4$. Let ρ_i and ϕ_i be the wheel's driving and **steering** angular speeds respectively. Let $v_i = (\dot{x}_i, \dot{y}_i)$ denote the steering pivot's **speed.** From Fig. 2.4, assuming that the ideal rolling condition is satisfied, we have

$$\begin{bmatrix} \dot{\phi}_i \\ \dot{\rho}_i \end{bmatrix} = \begin{bmatrix} 1/b & 0 \\ 0 & 1/r \end{bmatrix} \begin{bmatrix} v_{\phi_i} \\ v_{\rho_i} \end{bmatrix}, \tag{2.1}$$

where v_{ϕ_i} and v_{ρ_i} are the wheel's translational speeds in the wheel's frame. The

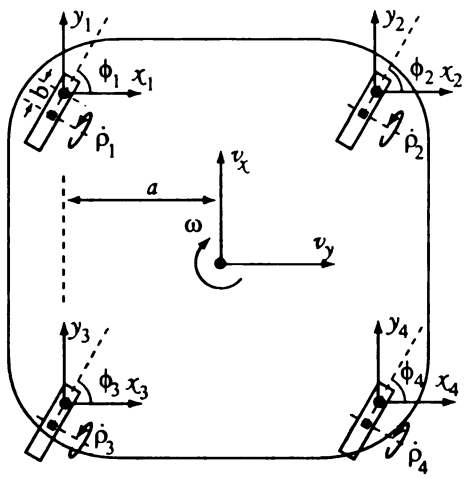


Figure 2.5: The top view of the mobile base. The black small dots are pivot axes along **which** the wheels can turn, and the small solid squares denote contact points between the **wheels** and the ground.

transformation matrix from the vehicle's frame is:

$$\begin{bmatrix} v_{\phi_i} \\ v_{\rho_i} \end{bmatrix} = \begin{bmatrix} \sin\phi_i & -\cos\phi_i \\ \cos\phi_i & \sin\phi_i \end{bmatrix} \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix}. \tag{2.2}$$

Plugging Eq. (2.2) into Eq. (2.1) and we obtain the relationship between the wheel's **joint** speed and its steering pivot's speed as:

$$\begin{bmatrix} \dot{\phi}_i \\ \dot{\rho}_i \end{bmatrix} = \begin{bmatrix} \sin\phi_i/b & -\cos\phi_i/b \\ \cos\phi_i/r & \sin\phi_i/r \end{bmatrix} \begin{bmatrix} \dot{x}_i \\ \dot{y}_i \end{bmatrix} \tag{2.3}$$

Referring Fig. 2.5, we obtain the following relation between the first wheel's steer**ing** **pi**vot's speed and the whole vehicle's speed:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \end{bmatrix} = \begin{bmatrix} v_x + a\omega \\ v_y + a\omega \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -a \end{bmatrix} \mathbf{v}. \tag{2.4}$$

The **sim**ilar procedure is applied to the other three wheels and we write the result in

matrix form as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{x}_3 \\ \dot{y}_3 \\ \dot{x}_4 \\ \dot{y}_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a \\ 1 & 0 & -a \\ 0 & 1 & -a \\ 1 & 0 & a \\ 0 & 1 & a \\ 1 & 0 & a \\ 0 & 1 & a \\ 1 & 0 & -a \end{bmatrix} \mathbf{v}, \tag{2.5}$$

where a denotes the offset between the center of the base and the wheels' pivot points.

Eq. (2.3) is equivalent to

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\phi}_1 \\ \dot{\rho}_1 \\ \dot{\phi}_2 \\ \dot{\rho}_2 \\ \dot{\phi}_3 \\ \dot{\rho}_3 \\ \dot{\phi}_4 \\ \dot{\rho}_4 \end{bmatrix} = \begin{bmatrix} T_1 & 0 & 0 & 0 \\ 0 & T_2 & 0 & 0 \\ 0 & 0 & T_3 & 0 \\ 0 & 0 & 0 & T_4 \end{bmatrix} \begin{bmatrix} \dot{x}_1 \\ \dot{y}_1 \\ \dot{x}_2 \\ \dot{y}_2 \\ \dot{x}_3 \\ \dot{y}_3 \\ \dot{x}_4 \\ \dot{y}_4 \end{bmatrix}, \tag{2.6}$$

where

$$T_i = \begin{bmatrix} \sin\phi_i/b & -\cos\phi_i/b \\ \cos\phi_i/r & \sin\phi_i/r \end{bmatrix},$$

for $i = 1, ..., 4$.

Applying Eq. (2.5) to Eq. (2.6), we can derive the wheel joints' speed $\dot{\mathbf{q}}$ from the **vehicle** cartesian speed, $\dot{\mathbf{x}}$:

$$\dot{\mathbf{q}} = B\mathbf{v}, \tag{2.7}$$

where

$$B = \begin{bmatrix} \sin\phi_1/b & -\cos\phi_1/b & -a\sin\phi_1/b + a\cos\phi_1/b \\ \cos\phi_1/r & \sin\phi_1/r & -a\cos\phi_1/r - a\sin\phi_1/r \\ \sin\phi_2/b & -\cos\phi_2/b & -a\sin\phi_2/b - a\cos\phi_2/b \\ \cos\phi_2/r & \sin\phi_2/r & -a\cos\phi_2/r + a\sin\phi_2/r \\ \sin\phi_3/b & -\cos\phi_3/b & a\sin\phi_3/b - a\cos\phi_3/b \\ \cos\phi_3/r & \sin\phi_3/r & a\cos\phi_3/r + a\sin\phi_3/r \\ \sin\phi_4/b & -\cos\phi_4/b & a\sin\phi_4/b + a\cos\phi_4/b \\ \cos\phi_4/r & \sin\phi_4/r & a\cos\phi_4/r - a\sin\phi_4/r \end{bmatrix}. \quad (2.8)$$

Therefore, Eq. (2.7) gives the kinematic equation we are looking for here. We can see Dav's mobile base is a redundant-actuated system with non-holonomic constraints. The system equation has 3DOF in velocity space while 8DOF in configuration space.

2.2.2 Torso

To imitate the human body, as well as to enable picking up objects from the floor, a 2DOF torso is placed on the top of the mobile base. The rolling joint, the one having an vertical axis, can be regulated with motion of the mobile base to keep the orientation of the torso.

Fig. 2.7 shows the D-H coordinate frame assignment on the torso mechanism. It is worthy noting that there are three end-effector frames: $ox2y2z2$, $ox3y3z3$ and $ox4y4z4$, corresponding to the left arm, right arm, and neck respectively. The four D-H geometric parameters associated with each link are listed in Table 2.2.2. Those four geometric parameters are defined as follows:

- θ_i is the joint angle from the x_{i-1} axis to the x_i about z_{i-1} axis (using the right handle rule).
- d_i is the distance from the origin of the $(i-1)$th coordinate frame to the intersection of the z_{i-1} axis with the x_i axis along the z_{i-1} axis.

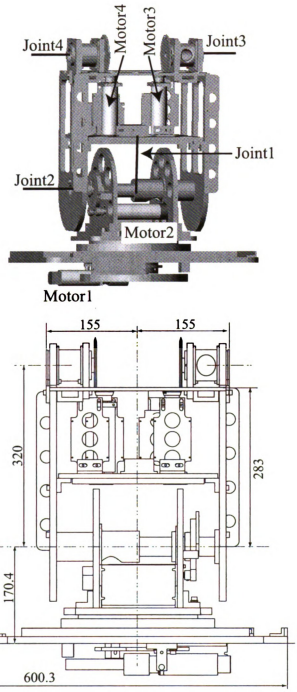


Figure 2.6: CAD drawing of Torso. Joint3 and Joint4 are the first joint of the left arm and **right arm** respectively. The unit is in mm.

- a_i is the offset distance from the intersection of the z_{i-1} axis with x_i axis to the origin of the ith frame along the x_i axis (or the shortest distance between the z_{i-1} and z_i axes).

- α_i is the offset angle from the z_{i-1} axis to the z_i axis about the x_i axis (using the right-hand rule).

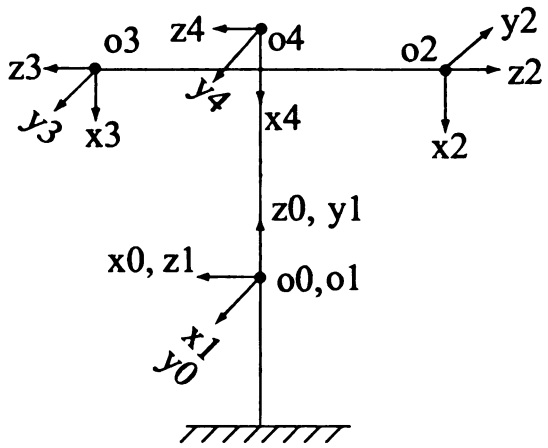


Figure 2.7: Coordinate frames of torso. Frames $ox4y4z4$, $ox4y4z4$ and $ox4y4z4$ are end-effector frames for mounting the left arm, right arm, and neck respectively.

2-2.3 Neck and Head

As a sensor platform, the neck and head (see Fig. 2.8) are designed to support active vision system. This 8DOF mechanism includes the 3DOF neck, whose rolling joint with the vertical axis is actuated by an anti-backlash geared motor. The brows and lips are computer-controlled for expressing emotion. The two cameras can independently pan and coupled tilt.

Fig. 2.9 shows the D-H coordinate frame assignment on the torso mechanism. There are two end-effector frames: $ox4y4z4$ and $ox4'y4'z4'$, corresponding to left and right eyeballs respectively. These joints' axes cross each other orthogonally: axes of joint 1 and joint 2; axes of joint 4 and joint 5; axes of joint 4′ and joint 5′. Their origins coincide, shown in Fig. 2.9.

Table 2.2: D-H geometric parameters. The joint variables are marked with *. The unit is in millimeter. (a) is the parameters for left arm, (b) is for the right arm, and (c) is for the neck. θ_1 and θ_2 denote the angles of joint 1 and joint 2 respectively.

Link	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1^*
2	320.0	180°	-155.0	θ_2^*

(a)

Link	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1^*
3	320.0	180°	155.0	θ_2^*

(b)

Link	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1^*
4	283.0	180°	0	θ_2^*

(c)

The four D-H geometric parameters associated with each link are listed in Table 2.2.3.

2.2.4 Arm

Dav has human symmetry with two arms. Each arm is a 7DOF anthropomorphic manipulator shown in Fig. 2.10. The extra joint is added to each arm for increasing the end effector's range of motion. Composed of shoulder, elbow, and wrist, the arm design has been influenced by the human skeleton structure. Most of joints of arm except Joint 1 which is driven by a set of bevel gears, are actuated by motors with gear-box either directly or through a simple pulley timing-belt mechanism. This design simplified the mechanical transmission system and saves a lot of room for installing servo units such as servo amplifiers, sensors, and control boards. All servo units are integrated inside the arm itself except those for the shoulder, which are located conveniently in the torso.

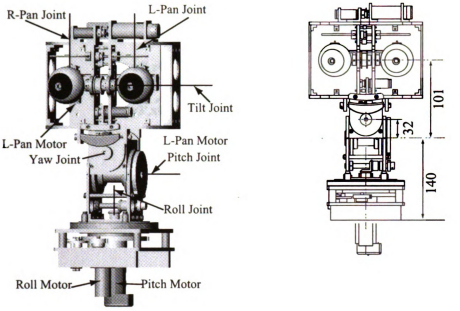


Figure 2.8: CAD drawing of the neck and head module. The unit is in mm.

Joints 1, 2, and 3 together realize a spherical shoulder configuration, shown in Fig. 2.11, in which the joint axes $z0$, $z1$, and $z2$ intersect at $o1$. Similarly, the wrist mechanism, consisting of Joints 5, 6, and 7, adopts the Stanford manipulator (spherical wrist) design. The angle of spherical joints are Euler angles, which simplified the kinematic analysis. D-H parameters are shown in Table 2.2.4.

Once the D-H coordinate system has been established for each robot link, the forward kinematics can be easily derived via homogeneous transformation. Let A_i be the matrix transforming the $i-1$th coordinate frame to the ith frame as follows:

$$A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i & a_i \sin\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

The matrix T_0^n which specifies the location of the ith coordinate frame with respect to the inertia frame (frame 0) is the chain product of successive transformation matrices

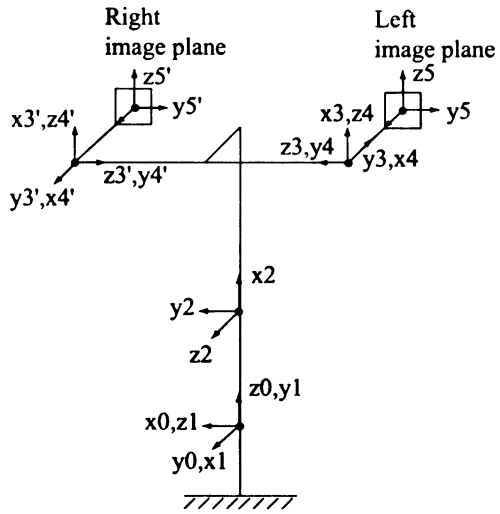


Figure 2.9: Coordinate frames of the neck and head. Frames $ox5y5z5$ and $ox5'y5'z5'$ are **end**-effector frames for the left and right cameras respectively.

of A_k, for $k = 0, ..., i$, and is written as

$$T_0^i = \prod_{k=0}^{i} A_k \tag{2.10}$$

From the Eq. (2.10), the robot kinematic model is thus expressed by the 4×4 **homo**geneous transformation matrix $T = T_0^n$, where n is the number of robot moving **link** under consideration. The T matrix has combined effects of rotation, translation, **and** scaling and can be decomposed into:

$$T = \begin{bmatrix} R & \mathbf{d} \\ \mathbf{0} & 1 \end{bmatrix}, \tag{2.11}$$

where R denotes the rotation matrix of the end-effector frame with respect to the **inertia** frame; $\mathbf{d}$ denotes the translation with respect to the inertia frame; $\mathbf{0} = [0, 0, 0]$.

Due to its redundant nature, Dav's inverse kinematic analysis is extremely complex **and does** have a unique solution. Usually, more constraints such as minimum power **consumption** and smoothing trajectory are needed, but this model-based inverse kine-

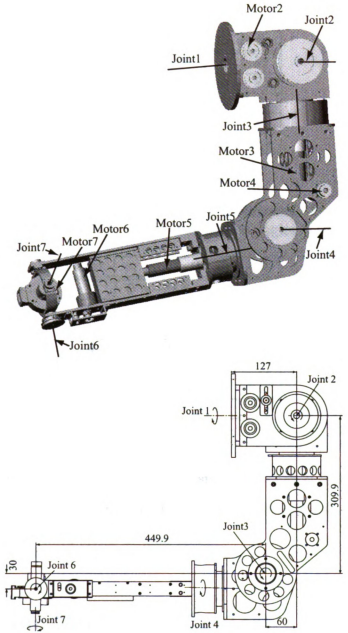


Figure 2.10: Dav's left arm. The unit is in mm. The actuator for Joint 1 is housed in the torso module.

Table 2.3: D-H geometric parameters of neck and head mechanism. The joint variables are marked with *. The unit is milli-meter. θ_1, θ_2, θ_3 denote the angles of the neck's Roll joint, Pitch joint, and Yaw joint respectively. Frames 4 and 4′ are the end-effector frames of the left and right eyeballs respectively. Note that we do not consider two pan joints in head.

Link	a_i	α_i	d_i	θ_i
1	0	90°	0	θ_1^*
2	32	−90°	0	θ_2^*
3	101	90°	77	θ_3^*
4	0	0°	-53	θ_4^*
4'	0	0°	53	$\theta_4'*$

Table 2.4: D-H geometric parameters of the arm mechanism. The joint variables are marked with *. The unit is milli-meter. θ_i denotes the angles of joint i, for $i = 1, ..., 7$.

Link	a_i	α_i	d_i	θ_i
1	0	−90°	0	θ_1^*
2	0	90°	0	θ_2^*
3	0	−90°	309.9	θ_3^*
4	60	0	499.9	θ_4^*
5	30	−90°	0	θ_5^*
6	0	90°	0	θ_6^*
7	0	−90°	80	θ_7^*

matics method is beyond the scope of this thesis. As studies in references [30, 32, 48], this work focuses on using learning-based methods to approximate the mapping from cartersian end-effector work space to joint space.

2.3 Dynamic Model and Control

In this section, we derive equations describing the time evolution of the robot's configures based on torques or forces under a known environment (e.g., repeated tasks on a manufacturing assembly line with fixed payload). It is worthy noting the known-environment assumption is not valid for some robotic applications. Those cases in-

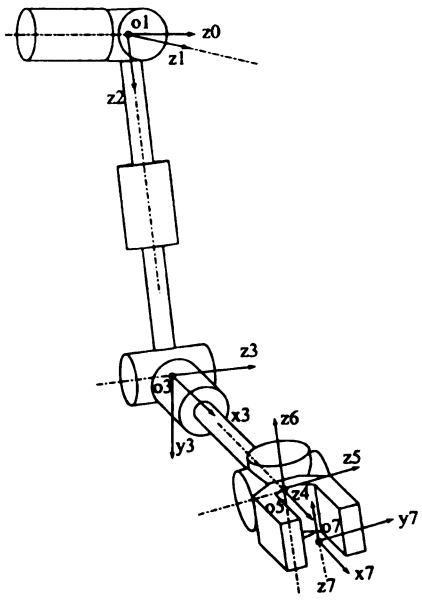


Figure 2.11: Coordinate frames of the left arm. Two spherical mechanisms (shoulder and arm) are designed.

clude: the dynamics of the manipulated object changes tremendously during a manipulation task; dynamic equations are not available due to the complexity of the system and lacking of domain knowledge. However, deriving such equations of motion is still useful for designing a suitable low-level control law for Dav's mechanism, and may serve as a survey of the model-based control method.

The dynamic model of a robot can be obtained from known physical laws such as the Newtonian mechanics and Lagrangian mechanics. The derivation of robotic dynamics based on the Lagrange-Euler (L-E) formulation is more systematic; we hence derive Dav's dynamic equations based on the L-E formulation. We do not present the whole robot, which is cumbersome. In the following, we first outline L-E analysis on a general setting. We then apply it to a general n-link robot manipulator. Mobile base's dynamic analysis and control are relegated to Section 2.5 because the base is a redundant-actuated system with non-holonomic constraints and, thus, needs special treatment.

2.3.1 Lagrange-Euler Formulation

In the L-E formulation, the system's dynamic behavior is described in terms of work and energy using generalized coordinates. All the constraint forces are eliminated in this method.

Assuming a robot has n DOFs, the L-E formulation is expressed in the following form

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{q}_i}) - \frac{\partial L}{\partial q_i} = F_i \tag{2.12}$$

for $i = 1, 2, ..., n$, where

- L: Lagrangian function and equals to kinetic energy K - potential energy P
- K: Total kinetic energy of the robot
- P: Total potential energy of the robot
- q_i: Generalized coordinates of the robot
- $\dot{q}_i$: Generalized velocity, or first derivative of q_i with respect to time
- F_i: Generalized actuating force (torque) applied to the robot at joint i

Now we are ready to derive the dynamic equation for a robot with n links. It is known the overall kinetic energy of a rigid body is given by

$$K = \frac{1}{2} m \mathbf{v}_c^T \mathbf{v}_c + \frac{1}{2} \omega^T I \omega, \tag{2.13}$$

where $\mathbf{v}_c$ and ω denote respectively, the mass center's translation velocity vector and angular velocity vector of the rigid body; m and I are the total mass and inertia matrix respectively.

2.3.2 Analysis of Manipulator with n-link

In the D-H representation, the joint variables can be served as generalized variables of L-E equation. According to the velocity kinematic analysis of [89, Chap. 5], the ith link's linear velocity of mass center and angular velocity are written as

$$\mathbf{v}_{c_i} = J_{v_{ci}}(\mathbf{q})\dot{\mathbf{q}}, \quad \omega_i = R_i^T(\mathbf{q})J_{\omega_{ci}}(\mathbf{q})\dot{\mathbf{q}} \tag{2.14}$$

respectively, where $J_{v_{ci}}$ and $J_{\omega_{ci}}$ are Jacobian matrix for link i; R_i denotes the rotation matrix of the ith frame with respect to the inertia frame (frame 0). Let $\mathbf{r}_{c_i}$ be the mass center of link i in the inertia frame $(ox_0y_0z_0)$. Since Dav has rotational joints only, $J_{v_{ci}}$ and $J_{\omega_{ci}}$ can be calculated by

$$\begin{bmatrix} J_{v_{ci}} \\ J_{\omega_{ci}} \end{bmatrix} = \begin{bmatrix} J_1 & J_2 & \dots & J_i & 0 & \dots & 0 \end{bmatrix}, \tag{2.15}$$

where $J_i = \begin{bmatrix} z_{i-1} \times (\mathbf{r}_{c_i} - \mathbf{o}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix}$ and $J_k = \begin{bmatrix} z_{k-1} \times (\mathbf{o}_k - \mathbf{o}_{k-1}) \\ \mathbf{z}_{k-1} \end{bmatrix}$ for $k < i$; o_i and z_i denote the origin and z-axis of the i frame of the D-H representation in the inertia frame respectively.

Therefore, the total kinetic energy can be expressed by

$$K = \frac{1}{2}\dot{\mathbf{q}}^T \sum_{i=1}^{n} \left[m_i J_{v_{ci}}^T J_{v_{ci}} + J_{\omega_{ci}}^T R_i I_i R_i^T J_{\omega_{ci}} \right] \dot{\mathbf{q}} = \frac{1}{2}\dot{\mathbf{q}}^T D(\mathbf{q})\dot{\mathbf{q}} \tag{2.16}$$

where I_i denotes the inertia matrix with respect to the frame i.

The Lagrangian function are then written as

$$L = K - V = \frac{1}{2}\dot{\mathbf{q}}^T D(\mathbf{q})\dot{\mathbf{q}} + V(\mathbf{q}). \tag{2.17}$$

Applying Eq. 2.17 into Eq. (2.12), we obtain the commonly used motion equation in

matrix form (see [89, Section 6.3] for the detailed derivation):

$$D(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) = \tau \tag{2.18}$$

where $C(\mathbf{q}, \dot{\mathbf{q}})$ denotes centrifugal and coriolis terms; $\mathbf{g}(\mathbf{q})$ denotes gravity related terms; τ denotes the vector of the output torque of the motors.

2.4 The Low-level Control Scheme

We use the proportional derivative (PD) control at the joint-level, where each joint of the robot is treated as a servo-mechanism. Notice that using joint PD control is ineffective because it does not take account of the coupling effects from the other robot links. However, the dynamic analysis of Section 2.3 provides a way to compensate the nonlinear factors. Hence, this control scheme can achieve better performance than that of conventional PD control.

2.4.1 Model of Motor

Each joint of the Dav robot is driven by a DC servo motor. This subsection outlines the DC motor's model by deriving the transfer function from which control algorithms will be obtained.

Illustrating in the schematic diagram of Fig. 2.12, we introduce the following notations:

- V = armature voltage
- L = armature inductance
- R = armature resistance
- V_b = back EMF voltage
- i_a = armature current

- θ_m = motor shaft's displacement (in radians)
- θ_L = joint shaft's displacement (in radians)
- τ_m = motor torque
- τ_L = load torque
- K_a = torque constant of the motor
- K_b = back EMF constant
- J_m = robot's moment of inertia
- B_m = viscous friction coefficient of the rotor
- n = gear train ratio

The following differential equations outline the behavior of a motor:

$$\begin{aligned} \tau_m(t) &= K_a i_a(t) \\ V_b(t) &= K_b \theta_m(t) \\ V(t) &= R i_a(t) + L\tfrac{di_a(t)}{dt} + V_b(t) \\ \tau_m(t) - n\tau_L(t) &= J_m \ddot{\theta}_m(t) + B_m \dot{\theta}_m(t) \end{aligned} \tag{2.19}$$

After applying Laplace transform to those equations, we obtain:

$$\begin{aligned} \tau_m(s) &= K_a I_a(s) \\ V_b(s) &= K_b \Theta_m(s) \\ V(s) &= R I_a(s) + sL I_a(s) + V_b(s) \\ \tau_m(s) - n\tau_L(s) &= s^2 J_m \Theta_m(s) + sB_m \Theta_m(s) \end{aligned} \tag{2.20}$$

Organizing terms in Eq. (2.20), we get the transfer function from the armature voltage V to the angular displacement of rotor Θ_m (with $\tau_L = 0$) as

$$\frac{\Theta_m(s)}{V(s)} = \frac{K_a}{s[(Ls+R)(J_m s + B_m) + K_a K_b)]} \tag{2.21}$$

And the transfer function from the load torque τ_L to Θ_m is given by (with $V = 0$)

$$\frac{\Theta(s)}{\tau_L(s)} = \frac{-n(Ls + R)}{s[(Ls + R)(J_m s + B_m) + K_b K_a]} \tag{2.22}$$

It is assumed that the "electrical time constant" $\frac{L}{R}$ is much smaller than the "mechanical time constant" $\frac{J_m}{B_m}$, the influence of L/R is negligible. Therefore Eqs. (2.21) and (2.22) are simplified to be

$$\frac{\Theta_m(s)}{V(s)} = \frac{K_a/R}{s(J_m s + B_m + K_a K_b/R)}, \tag{2.23}$$

and

$$\frac{\Theta(s)}{\tau_L(s)} = -\frac{n}{s(J_m s + B_m + K_b K_a/R)}, \tag{2.24}$$

respectively.

By superposition, Eqs. (2.23) and (2.24) represent a second order system

$$s^2 J_m \Theta_m(s) + s(B_m + K_b K_a/R)\Theta_m(s) = (K_a/R)V(s) - n\tau_L(s). \tag{2.25}$$

Usually gear trains are mounted between the motor and joint shafts to increase the motor load capacity. If the gear ratio $n \equiv \frac{\theta_L}{\theta_m}$, where θ_L denotes the angular displacement of the output shaft, then Eq. (2.25) becomes

$$s^2 \frac{1}{n^2} J_m \Theta_L(s) + s\frac{1}{n^2}(B_m + K_b K_a/R)\Theta_L(s) = \frac{K_a V(s)}{nR} - \tau_L(s), \tag{2.26}$$

whose block diagram is shown in Fig. 2.12. It is worthy noting that J_m in Eq. (2.26) represents the effective motor inertia, the sum of the rotor and gear inertias.

2.4.2 Tracking Algorithm

We first reformulate the nonlinear dynamic analysis by combining the dynamic effects of the motors. We then derive the PD feedback control law with acceleration and

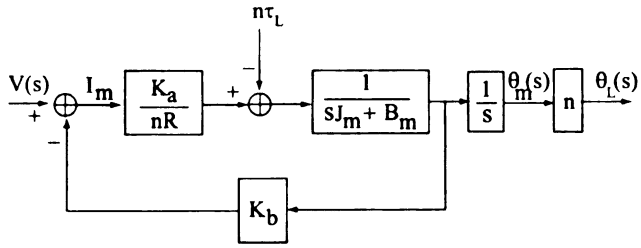


Figure 2.12: Block diagram for a DC motor.

gravity terms, which has been implemented on Dav's joint controllers.

Recalling the robot equations of motion Eq. (2.18) and the motor's equation of motion Eq. (2.26), we reinstate them as

$$\sum_{j=1}^{n} d_{jk}(\mathbf{q})\ddot{q}_j + \sum_{i,j=1}^{n} c_{ijk}(\mathbf{q})\dot{q}_i\dot{q}_j + g_k(\mathbf{q}) = \tau_k \tag{2.27}$$

and

$$\frac{1}{n_k^2} J_{m_k}\ddot{\theta}_{L_k} + \frac{1}{n_k^2}(B_{m_k} + K_{b_k}K_{a_k}/R)\dot{\theta}_{L_k} = \frac{K_{a_k}V_k}{n_k R_k} - \tau_{L_k}, \tag{2.28}$$

respectively. Comparing with Eq. (2.26), some notations of Eq. (2.28) are added a scription k for the kth motor and Eq. (2.28) represents ith motor's dynamics in time domain by differential equation.

We observe that the kth generalized variable q_k is actually represented by the joint shaft displacement θ_{L_k} and torque load $\tau_{L_k} = \tau_k$. Let $B_k = B_{m_k} + K_{b_k}K_{a_k}/R_k$. Plugging Eq. (2.28) into Eq. (2.27) yields

$$\frac{1}{n_k^2} J_{m_k}\ddot{q}_k + \sum_{j=1}^{n} d_{jk}\ddot{q}_j + \sum_{i,j=1}^{n} c_{ijk}\dot{q}_i\dot{q}_j + \frac{1}{n_k^2} B_k\dot{q}_k + g_k = \frac{K_m}{n_k R}V_k, \tag{2.29}$$

for $k = 1, ..., n$.

In a matrix form, Eq. (2.29) can be written as

$$(D(\mathbf{q}) + J)\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + B\dot{\mathbf{q}} + g(\mathbf{q}) = \mathbf{u}, \tag{2.30}$$

where $D(q)$ is the $n \times n$ inertia matrix of the mechanical links, and J is a diagonal matrix with diagonal elements $\frac{J_{m_k}}{n_k^2}$. The $C(\mathbf{q}, \dot{\mathbf{q}})$ and $\mathbf{g}(\mathbf{q})$ are defined as before and input control vector $\mathbf{u}$ is specified by

$$\mathbf{u} = \left[\frac{K_{a_1} V_1}{n_1 R_1} \quad \frac{K_{a_2} V_2}{n_2 R_2} \quad \cdots \quad \frac{K_{a_n} V_n}{n_n R_n} \right]^T. \tag{2.31}$$

The objective of joint-level tracking is to servo the motor such that the output shaft's displacement can track a reference path provided by a path planner. An independent joint PD control can be written in a vector form as

$$\mathbf{u} = -K_p(\mathbf{q} - \mathbf{q}^d) - K_d(\dot{\mathbf{q}} - \dot{\mathbf{q}}^d), \tag{2.32}$$

where K_p and K_d are diagonal matrices of proportional and derivative feedback gains respectively; $\mathbf{q}^d$ and $\dot{\mathbf{q}}^d$ are the desired displacement and speed of the joints respectively, which are given by a path planner in advance.

In [89, Page 217-218], Spong & Vidyasagar show that, in the absense of gravity, that is, if $\mathbf{g}$ is zero in Eq. (2.31) for all configurations, the PD control law specified in Eq. (2.32) achieves asymptotic tracking of the desired position $\mathbf{q}^d$ with $\dot{\mathbf{q}}^d = 0$. However, the presence of the gravitational term or nonzero $\dot{\mathbf{q}}$ does not guarantee the asymptotic convergence of Eq. (2.31). A "stable" tracking error from the desired trajectory is needed to balance the gravitational torque $\mathbf{g}$.

In the practical implementation, Dav moves at a slow speed: $\dot{\mathbf{q}}^d \approx 0$; the dynamic terms such as off-diagonal inertial elements d_{jk}, $j \neq k$, the centripetal and coriolis term $C(\mathbf{q}, \dot{\mathbf{q}})$, and damping from motor $B(\mathbf{q})$ are usually small. Notice that the computed torque τ_k is proportional to the gear ratio; thus those dynamic terms are reduced further for a small n_k, which adds to the validity of that the terms d_{jk}, $j \neq k$ are negligible for control. The gravity effect specified by $\mathbf{g}$ plays a dominant role in the robot dynamic model. In order to compensate for gravity, the following control

law is proposed for Dav's low level control:

$$\mathbf{u} = D_{\text{eff}}\ddot{\mathbf{q}} + \mathbf{g}(\mathbf{q}) - K_p(\mathbf{q} - \mathbf{q}^d) - K_d(\dot{\mathbf{q}} - \dot{\mathbf{q}}^d), \tag{2.33}$$

where D_{eff} is a diagonal matrix whose diagonal elements denote the effective inertia of each link of the robot.

Eq. (2.33) can be written in the component form as

$$u_k = d_{\text{eff}}\ddot{q}_k + g_k(\mathbf{q}) - k_p(q_k - q_k^d) - k_d(\dot{q}_k - \dot{q}_k^d), \tag{2.34}$$

where d_{eff}, k_p, and k_d are effective inertia, proportional gain, and derivative gain of the k joint respectively. Applying Eq. (2.34) into Eq. (2.29), by assuming $d_{\text{eff}} = d_{kk} + J_{m_k}/n_k^2$ and letting

$$c_k = \sum_{j \neq k} d_{jk}\ddot{q}_j + \sum_{i,j} c_{ijk}\dot{q}_i\dot{q}_j + \frac{1}{n_k^2}B_k\dot{q}_k$$

denote the disturbances, we obtain the following closed-loop system equation for the joint k:

$$d_{\text{eff}}\ddot{e}_k + c_k = -k_p e_k - k_d\dot{e}_k, \tag{2.35}$$

where $e_k = q_k - q_k^d$.

Let us remark that the control law specified in Eq. (2.34) is derived under the condition of low-speed with gear reduction. It should be note that, for high speed motion, or for motion without gear reduction at the joints, the coupling nonlinearities have a much larger effect on the performance of the system; treating them merely external disturbances can cause large tracking errors.

2.5 Dav's Mobile Base: Control of System with Nonholonomic Constraints

Like other wheeled mobile robots involving rolling contracts between two or more rigid bodies, Dav's mobile base is an example of a mechanical system with nonholonomic constraits. Moreover, Dav's mobile base is a redundantly actuated mechanism [40] whose surplus control inputs offer a solution to the holonomic vehicle that can move to arbitrary direction at any configuration.

In this section, we apply a theoretic framework similar to [80] to Dav's mobile base and derive a feedback linearization control scheme.

2.5.1 Nonholonomic Constraint

If a constraint equation is in form $h_i(\mathbf{q}) = 0$, or can be integrated into this form, it denotes a holonomic constrait. Otherwise, it denotes a nonholonomic constrait.

Holonomic and nonholonomic constraints affect mobility of a vehicle in a completely different way. For illustration, consider a single Pfaffian constrait[1] in an n-body system: $\mathbf{a}^T(\mathbf{q})\dot{\mathbf{q}} = 0$. If the constraint is holonomic, then it can be integrated as $h(\mathbf{q}) = c$ with $\frac{\partial h}{\partial \mathbf{q}} = \mathbf{a}^T(\mathbf{q})$. The motion of the system is hence confined to lie on a particular level surface of h, depending on the initial condition through $c = \mathbf{h}(\mathbf{q}_0)$. A common practice for L-E mechanics to cope with a holonomic constraint is to reduce a generalized variable since the total degree of freedom is $n - 1$. On the other hand, if the constraint is nonholonomic (i.e, such a h does not exist), then the system can reach any admissible configuration, although at each configuration the instantaneous motion (velocity) of the nonholonomic system is restricted to an $(n-1)$-dimensional space. More precisely, the velocities of the system are confined to lie on the null space of the constraint matrix $\mathbf{a}^T(\mathbf{q})$.

[1] Most wheeled robots have kinematic constraints of this form, which are linear in the velocities (e.g., Dav's kinematics in Eq. (2.7))

A well-known example to illustrate nonholonomy is a conventional wheel, shown in Fig. 2.13. The generalized coordinates $\mathbf{q} = (x, y, \theta)$ are defined in Fig. 2.13. The pure

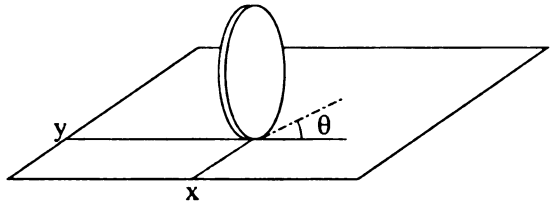


Figure 2.13: A rolling disk on a plane. The disk cannot move along the direction perpendicular to its face at each instant.

rolling constraint can be expressed as $\dot{x}\sin\theta - \dot{y}\cos\theta = 0$. The allowable velocities are confined in the null space of the constraint matrix $\mathbf{a}^T(\mathbf{q}) = (\sin\theta, -\cos\theta, 0)$, which is

$$\text{null}(\mathbf{a}^T(\mathbf{q})) = \text{span}\left\{ \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}.$$

However, any configuration $\mathbf{q}_f = (x_f, y_f, \theta_f)$ can be reached using the following strategy:

1. Roll the wheel until it aims the point (x_f, y_f)
2. Reach the point (x_f, y_f)
3. Rotate the wheel vertically until its orientation is θ_f

Now we consider a mechanical system with n degrees-of-freedom subject to m constraints of Pfaffian form:

$$\mathbf{a}_i^T(\mathbf{q})\dot{\mathbf{q}} = 0, \tag{2.36}$$

for $i = 1, 2, ..., m$. Eq. (2.36) can be written in the form

$$A(\mathbf{q})\dot{\mathbf{q}} = 0, \tag{2.37}$$

where $A(\mathbf{q})$ is an $m \times n$ full-rank matrix. Let $\mathbf{s}_1(\mathbf{q})$,...,$\mathbf{s}_{n-m}(\mathbf{q})$ be the basis vector fields of null(A), the null space of $A(\mathbf{q})$; i.e.,

$$A(\mathbf{q})\mathbf{s}_i(\mathbf{q}) = 0, \text{ for } i = 1, ..., n-m. \tag{2.38}$$

Let $S(\mathbf{q}) = [\mathbf{s}_1(\mathbf{q}), ..., \mathbf{s}_{n-m}(\mathbf{q})]$ and Eq. (2.38) can be written as

$$A(\mathbf{q})S(\mathbf{q}) = 0. \tag{2.39}$$

Let Δ be the space spanned by these vector fields

$$\Delta = \text{span}\{\mathbf{s}_1(\mathbf{q}), ..., \mathbf{s}_{n-m}(\mathbf{q})\}.$$

Δ may or may not be involutive[2]. If Δ is not involutive, we define Δ^* as the smallest involutive space containing Δ. Obviously, $\dim(\Delta) \leq \dim(\Delta^*)$. Campion et al. [20] observes three cases:

1. Δ is involutive. The m constraints are all holonomic

2. Δ is not involutive and $\dim(\Delta^*) = n$; i.e., Δ^* spans the entire space and all constraints are hence nonholonomic.

3. Δ is not involutive and $\dim(\Delta^*) < n$; both nonholonomic and holonomic constraints exist.

2.5.2 Dynamic Analysis of Dav's Mobile Base

Recall the mechanism of Dav robot's mobile base, as shown in Fig. 2.5. For dynamic analysis, we introduce the following symbols in addition to those defined in Section 2.2:

[2]For the concept of involutive space, reader may refer [89, Page 265].

- $\mathbf{x} = (x_c, y_c, \psi)$: The geometric center and orientation of the robot. Note that $\dot{\mathbf{x}} = \mathbf{v}$

- m: The mass of the whole platform

- I: The inertia momentum of the whole platform

- I_w: The inertia momentum of each wheel

- I_s: The inertia momentum of the moving part of steering mechanism along the joint axis in each wheel.

The configuration of the platform can be described by 11 generalized coordinates ($n = 11$). The first eight variables describe the angular displacements of the wheels $\mathbf{q}$; the remaining three are the platform configuration, represented by $\mathbf{x}$. Thus, $\mathbf{q}_v = [\mathbf{q}^T, \mathbf{x}^T] = [\phi_1, \rho_1, \phi_2, \rho_2, \phi_3, \rho_3, \phi_4, \rho_4, x_c, y_c, \psi]^T$.

We assume that the driving wheels roll and there is no slip along the direction perpendicular to the face of the wheel. Those kinematic constraints have been represented by Eq. (2.7), which gives eight equations ($m = 8$). Eq. (2.7) can be written in Pfaffian form as:

$$A(\mathbf{q})\dot{\mathbf{q}}_r = 0 \tag{2.40}$$

where A is an 11×8 matrix denoting

$$A(\mathbf{q}) = \begin{bmatrix} I_{8\times 8} & -B \end{bmatrix}. \tag{2.41}$$

and B is defined in Eq. (2.8).

It is straightforward to verify that $S(\mathbf{q})$, the null space of $A(\mathbf{q})$, can be selected as

$$S(\mathbf{q}) = \begin{bmatrix} B & I_{3\times 3} \end{bmatrix} \tag{2.42}$$

It is obvious, because

$$A(\mathbf{q})S(\mathbf{q}) = \begin{bmatrix} I & -B \end{bmatrix} \begin{bmatrix} B \\ I \end{bmatrix} = B - B = 0.$$

Since the system's velocities are confined in the space Δ spanned by $S(\mathbf{q})$'s column vectors, it is possible to define three velocities $\mathbf{v} = [v_1, v_2, v_3]^T$ such that

$$\dot{\mathbf{q}}_v = S(\mathbf{q})\mathbf{v}. \tag{2.43}$$

Note that $\mathbf{v}$ in Eq. (2.43) is actually the base's speed (v_x, v_y, ω). This is because

$$\dot{\mathbf{q}}_v = \begin{bmatrix} \dot{\mathbf{q}} \\ I \end{bmatrix} = S\mathbf{v} = \begin{bmatrix} B \\ I \end{bmatrix} \mathbf{v}.$$

Thus, $\dot{\mathbf{q}} = B\mathbf{v}$. Inspecting Eq. (2.7), we can see $\mathbf{v} = (v_x, v_y, \omega)$.

Now we are ready to derive dynamic equations for Dav's mobile base. Assuming Dav moves on a level floor, we neglect the effects of gravity. The Langrange function is equal to the total kinetic energy only, which is

$$L = T = \frac{1}{2}\sum_{i=1}^{4}(I_s\dot{\phi}^2 + I_w\dot{\rho}^2) + \frac{1}{2}(\dot{x}_c^2 + \dot{y}_c^2) + \frac{1}{2}I\dot{\psi}^2 \tag{2.44}$$

Applying Langrange-Euler to the above equation, we obtain

$$M\ddot{\mathbf{q}}_v = E\tau - A^T(\mathbf{q})\lambda, \tag{2.45}$$

where τ is an eight-dimensional vector representing motor's output torque and λ is the vector of constraint forces. E denotes input transformation matrix and M denotes

an 11×11 inertia matrix, which is defined

$$
\begin{aligned}
M &= \begin{bmatrix} J_0 & 0 \\ 0 & M_0 \end{bmatrix} \\
J_0 &= \mathrm{diag}([I_s, I_w, I_s, I_w, I_s, I_w, I_s, I_w]) \\
M_0 &= \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix}.
\end{aligned}
\tag{2.46}
$$

We notice that there is no actuator acting on variables (x_c, y_c, ψ) directly, thus E is designed to be $\begin{bmatrix} I_{8\times 8} \\ 0_{3\times 3} \end{bmatrix}$.

Differentiating Eq. (2.43) and plugging Eq. (2.45) at the left side, we obtain

$$
E\tau - A^T\lambda = M\dot{S}\mathbf{v} + MS\dot{\mathbf{v}}, \tag{2.47}
$$

where we omit the argument $\mathbf{q}$ of variables S and A, for simplicity.

Multiplying Eq. (2.47)'s both side by S^T, using the fact that $AS = 0$, we obtain

$$
S^T MS\dot{\mathbf{v}} = -S^T M\dot{S}\mathbf{v} + S^T E\tau. \tag{2.48}
$$

Combining the kinematic model Eq. (2.43), we obtain the reduced state-space model:

$$
\begin{cases}
\dot{\mathbf{q}}_v = S\mathbf{v} \\
S^T MS\dot{\mathbf{v}} = -S^T M\dot{S}\mathbf{v} + S^T E\tau
\end{cases}
\tag{2.49}
$$

Applying Eqs (2.42), (2.46) and E's definition to Eq. (2.49), we obtain the simplified dynamic equation for Dav's base, in the component form:

$$\begin{cases} \dot{\mathbf{q}} = B\mathbf{v} \\ \dot{\mathbf{x}} = \mathbf{v} \\ (B^T J_0 B + M_0)\dot{\mathbf{v}} + B^T J_0 \dot{B}\mathbf{v} = B^T \tau \end{cases} \tag{2.50}$$

We observed that

$$\dot{B}\mathbf{v} = \sum_{i=1}^{3}(v_i \frac{\partial \mathbf{b}_i}{\partial \mathbf{q}})\dot{\mathbf{q}} = \sum_{i=1}^{3}(v_i \frac{\partial \mathbf{b}_i}{\partial \mathbf{q}})B\mathbf{v}, \tag{2.51}$$

where $\mathbf{b}_i$ denotes the ith column vector of B. The Jacobian matrices $\frac{\partial \mathbf{b}_i}{\partial \mathbf{q}}$, $i = 1, 2, 3$ can be easily derived from Eq. (2.7), and are written as:

$$\frac{\partial \mathbf{b}_1}{\partial \mathbf{q}} = \begin{bmatrix} \cos\phi_1/b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\sin\phi_1/r & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos\phi_2/b & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\sin\phi_2/r & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\phi_3/b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\sin\phi_3/r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cos\phi_4/b & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\sin\phi_4/r & 0 \end{bmatrix} \tag{2.52}$$

$$\frac{\partial \mathbf{b_2}}{\partial \mathbf{q}} = \begin{bmatrix} \sin\phi_1/b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \cos\phi_1/r & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sin\phi_2/b & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \cos\phi_2/r & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sin\phi_3/b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\phi_3/r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sin\phi_4/b & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cos\phi_4/r & 0 \end{bmatrix} \tag{2.53}$$

$$\frac{\partial \mathbf{b_3}}{\partial \mathbf{q}} = \begin{bmatrix} -ac\phi_1/b - as\phi_1/b & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ as\phi_1/r - ac\phi_1/r & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -ac\phi_2/b + as\phi_2/b & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & as\phi_2/r + ac\phi_2/r & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & ac\phi_3/b + as\phi_3/b & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -as\phi_3/r + ac\phi_3/r & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & ac\phi_4/b - as\phi_4/b & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -as\phi_4/r - ac\phi_4/r & 0 \end{bmatrix} \tag{2.54}$$

In Eq. (2.54), s and c denote sin(.) and cos(.) respectively, for shortness.

2.5.3 Controller Design

Although mechanical system with nonholonomic constraints cannot be stabilized at a point by a smooth time-invariant feedback rule in general settings [17], we show experimentally that Dav's mobile base can track different curves with high performance.

We notice that $\text{rank}(B^T) = 3$ and $\dim(\tau) = 8$, thus the system is sufficient actuated, or redundantly actuated, to be more precisely. However, those surplus inputs offer a solution to implement the holonomity of the base.

For feedback linearization, we use following input transformation,

$$\tau = H((B^T J_0 B + M_0)\mathbf{u} + B^T J_0 \dot{B}\mathbf{v}), \tag{2.55}$$

where $\mathbf{u}$ denotes the input of the "new" system and H is an 8×3 matrix satisfying $B^T H = I$. A choice for H is the pseudoinverse of B^T; i.e., $H = (B^T)^\dagger$. Another choice of H is the solution of the following minimum norm problem[3]:

$$\min \tau^T \tau \text{ s.t. } F = B^T \tau \text{ where } F = (B^T J_0 B + M_0)\mathbf{u} + B^T J_0 \dot{B}\mathbf{v}. \tag{2.56}$$

Thus, by referring [81, Page 395], we can see the solution of Eq. (2.56) is $\tau = B(B^T B)^{-1} F$ and

$$H = B(B^T B)^{-1}. \tag{2.57}$$

It is worthy noting that the minimum norm solution of H has a nice property: it distributes the torques in a way that minimizes the actuators' output torque.

We notice that the minimum norm solution of H does not guarantee that the kinematic Eq. (2.7) be satisfied. In fact, due to external disturbance or unmodeled error, the $\dot{\mathbf{q}}$ might not confined the distribution Δ. We need a mechanism to entrap the velocities of the system within the distribution Δ. Motivated by sliding mode control [87, Chap. 7], we add an extra term to Eq. (2.57) which is servoing on the residue of Eq. (2.7). More formally, let $\dot{\mathbf{q}} = \dot{\mathbf{q}}_P + \dot{\mathbf{q}}_\perp$, where $\dot{\mathbf{q}}_P = BB^\dagger \dot{\mathbf{q}}$ and $\dot{\mathbf{q}}_\perp = (I - BB^\dagger)\dot{\mathbf{q}}$ (see Fig. 2.14). Thus, Eq. (2.57) becomes

$$H = B(B^T B)^{-1} + \Delta\tau \tag{2.58}$$

[3] F has a physical meaning. F denotes the desired equivalent force exerting on the vehicle platform. We treat the vehicle frame as an end-effector and τ as the joints' torque vector. The virtual displacement of the joint variable $\delta\mathbf{q} = B\delta\mathbf{x}$. The virtual work δw of the system is $\delta w = F^T \delta\mathbf{x} - \tau^T \delta\mathbf{q} = (F^T B - \tau)\delta\mathbf{x}$. $\delta w = 0$ if the platform is in equilibrium (the equivalent forces equal to the actual driving force), then $\tau = B^T F$.

where $\Delta\tau = -K_P\dot{\mathbf{q}}_\perp$ and K_P is the gain constant. We can easily verify that $B^T H = I$. Let us remark that, Eq. (2.58) will improve the transient behaviors, especially when the wheels are not synchronized initially. This is shown in Fig. 2.14. When $\dot{\mathbf{q}}$ does not lie on $\Delta(\mathbf{q})$, the $\Delta\tau$ will pull the $\dot{\mathbf{q}}$ back to the distribution Δ.

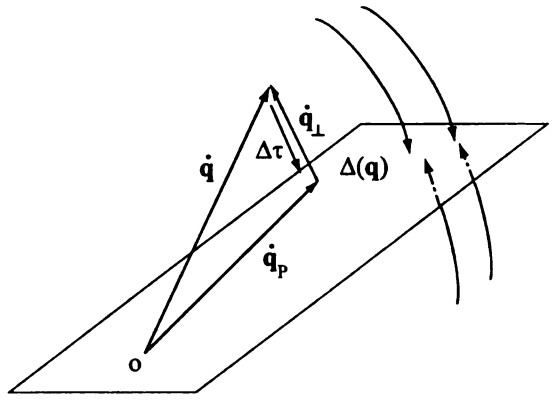


Figure 2.14: If $\dot{\mathbf{q}}$ is away from the distribution Δ, the extra term $\Delta\tau$ in Eq. (2.58) will pull it back.

Therefore, applying Eq. (2.55) to Eq. (2.50), we obtain the following linearized state-space equation

$$\begin{cases} \dot{\mathbf{q}} = B\mathbf{v} \\ \dot{\mathbf{x}} = \mathbf{v} \\ \dot{\mathbf{v}} = \mathbf{u} \end{cases} \tag{2.59}$$

The dynamic system described by Eq. (2.59) is known as the double integrator system since it represents 3 uncoupled double integrators with respect to $\mathbf{x} = (x_1, x_2, x_3)$ or coordinates of the base's center (x_c, y_c, ψ) in Fig. 2.5. This means that each input u_k, the kth component of $\mathbf{u}$, $k = 1, 2, 3$, is a function only of x_k. That is, u_1 and u_2 control the motion along x and y axes respectively, and u_3 controls the rotation of the base.

Since $\mathbf{u}$ can be designed to control the three simple linear second-order systems, the obvious choice is using PD control by setting

$$\mathbf{u} = -K_p(\mathbf{x} - \mathbf{x}^d) - K_d(\dot{\mathbf{x}} - \dot{\mathbf{x}}^d) + \ddot{\mathbf{x}}^d, \tag{2.60}$$

where $\mathbf{x}^d$, $\dot{\mathbf{x}}^d$ and $\ddot{\mathbf{x}}^d$ denote the desired trajectory of the base; K_p and K_d are the PD control's gain matrices with a common choice as

$$\begin{aligned} K_p &= \text{diag}[\omega_1^2, \omega_2^2, \omega_3^2] \\ K_d &= \text{diag}[2\omega_1, 2\omega_2, 2\omega_3] \end{aligned} \tag{2.61}$$

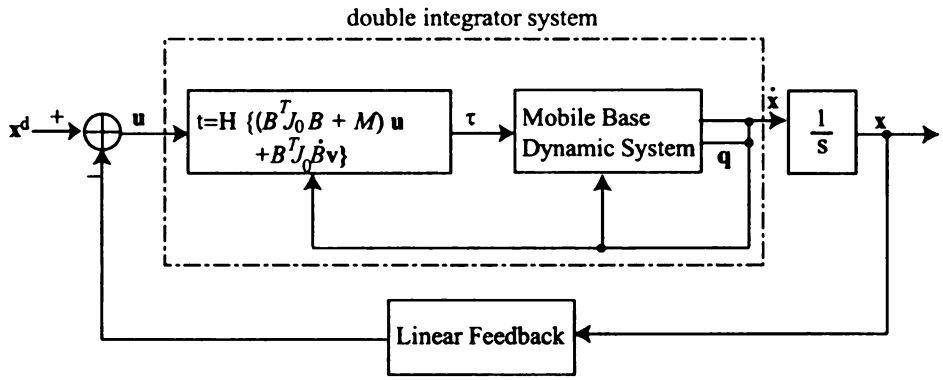


Figure 2.15: Schematic diagram of control algorithm of Dav's base.

Fig. 2.15 illustrates the control diagram of Dav's base, which is an inner-loop/outer-loop control. The computation of the nonlinear control Eq. (2.55) is performed in an inner loop, with $\mathbf{u}$, $\mathbf{q}$ and $\mathbf{v}$ as its inputs and τ as output. The outer loop in the system is the PD control calculating $\mathbf{u}$ based on the Eq. (2.60).

2.5.4 Experimental Evaluation

We developed a simulated base experiment to verify the validity of the dynamic model and the effectiveness of the control scheme presented in the previous section. The simulated base model is kinematically dynamically similar to the Dav's mobile base. The dimensions and the inertia parameters are specified in the following, according

to the previously introduced notations:

$$\begin{aligned} r &= 0.0889\text{m} \\ a &= 0.223\text{m} \\ b &= 0.0508\text{m} \\ m &= 250\text{kg} \\ I &= 174.5\text{kg-m}^2 \\ I_s &= 0.0313\text{kg-m}^2 \\ I_w &= 0.005\text{kg-m}^2 \end{aligned}$$

The goal of the experiment is to control the base such that the base's geometric center coincides on a predefined trajectory. As [86], we formulate the control problem as a path-following problem instead of trajectory tracking.

Consider a circular trajectory. Let (x_0, y_0) and R be the center and radius of the circular path respectively. v denotes the desired forward translation and $v = \sqrt{v_x^2 + v_y^2}$. We assume the base moves along the path while keeping the orientation unchanged, i.e., $\dot{\psi} = 0$. This path-following problem can be formulated as controlling the following system such that the output $\mathbf{z}$ follows a desired trajectory $\mathbf{z}^d$:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{u} \\ \mathbf{z} &= \begin{bmatrix} \mathbf{h}_1(\mathbf{x}) \\ \mathbf{h}_2(\mathbf{v}) \end{bmatrix}, \end{aligned} \tag{2.62}$$

where $\mathbf{h}_1(\mathbf{x}) = (x_c - x_0)^2 + (y_c - x_0)^2$ and $\mathbf{h}_2(\mathbf{v}) = \begin{bmatrix} v_x^2 + v_y^2 \\ \omega \end{bmatrix}$. The Jacobian matrix

of the output equation of the system specified in Eq. (2.62) can be written as

$$
\begin{aligned}
H_1 &= \frac{\partial \mathbf{h}_1}{\partial \mathbf{x}} = \begin{bmatrix} 2(x_c - x_0) & 2(y_c - y_0) & 0 \end{bmatrix} \\
H_2 &= \frac{\partial \mathbf{h}_2}{\partial \mathbf{v}} = \begin{bmatrix} 2v_x & 2v_y & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{2.63}
$$

Therefore, if we regard the output $\mathbf{z}$ as the state variables, by using $\dot{z}_1 = H_1\dot{\mathbf{x}}$ and $\begin{bmatrix} \dot{z}_2 \\ \dot{z}_3 \end{bmatrix} = H_2\dot{\mathbf{v}}$ to the first two equations of Eq. (2.62), then we obtain:

$$
\begin{aligned}
\ddot{z}_1 &= 2(x_c - x_0)u_1 + 2(y_c - y_0)u_2 + 2v_x^2 + 2v_y^2 \\
\dot{z}_2 &= 2v_x u_1 + 2v_y u_2 \\
\dot{z}_3 &= u_3
\end{aligned}
\tag{2.64}
$$

where a feedback linear control law might be applied.

The simulated sensors were the encoders placed on each motor. At each sampling instant (every 10 milli-seconds), the base received the joint angular displacement vector $\mathbf{q}$ and the corresponding derivative $\dot{\mathbf{q}}$. From Eq. (2.7), the vehicle's speed $\mathbf{v} = B\dot{\mathbf{q}}$. The base's pose (x_c, y_c, ψ) were estimated via deadreckoning.

Let the base's initial position is $(x_c, y_c, \psi) = (0.0, 0.8, 90°)$; initial velocity was zero; $R = 5$; $v = 1$. Therefore, $\mathbf{z}^d$ can be chosen to be $(25, 0.25, 0)^T$. Fig. 2.16 shows the result. Subplot (a) shows the actual and desired path, which are illustrated by, respectively, solid and dot-dash lines. In order to demonstrate the robustness of our model respect to the dynamic parameters, we increase the the mass and inertia momentum (m, I, I_s, I_w) by 100 percent. The result is shown in (b) of Fig. 2.16. We can see the overshot and small stable deviation from the desired path; but the overall performance is impressive.

Finally, let us remark that, although the system shows the robustness for the

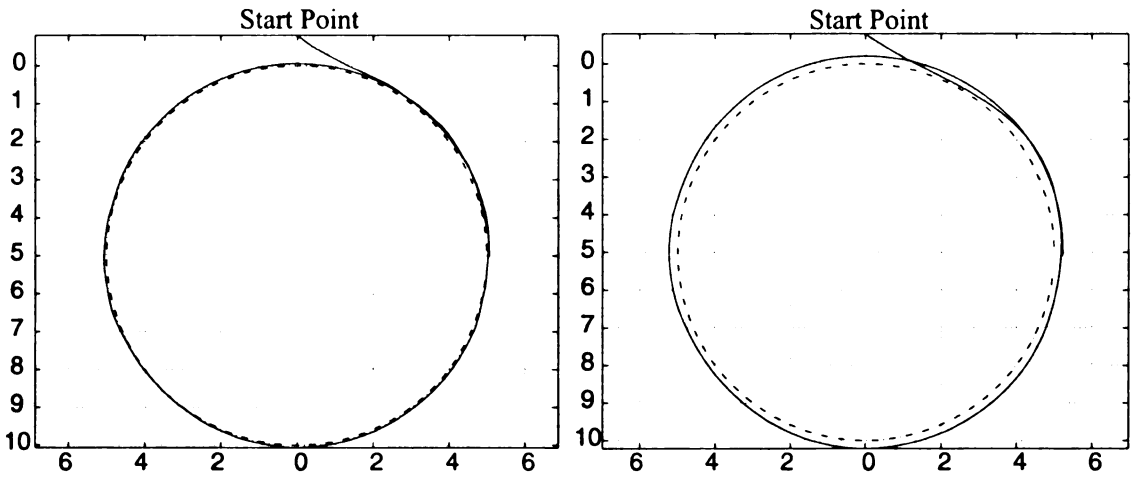


Figure 2.16: The result of the simulated base experiment.

changing dynamic parameters, it is sensitive to the accuracy of kinematic model and deadreckoning. If the kinematic model is wrong, the base cannot decouple the wheel motion correctly and, in an extreme case, the wheel might fight each other. In addition, it is known that deadreckoning performance corrupts quickly with increasing traveling distance. Feedback from the other sensor modalities are needed; for example, laser ranger and GPS might be used for indoor and outdoor navigation respectively.

2.6 The Distributed Control System Design

As a self-contained mobile humanoid robotic system, Dav carries its own power source, sensors, control system, learning system, and associated hardware.

2.6.1 Hardware Design

The hardware architecture of Dav has two levels: a main computer system and low-level servo control modules, as shown in Fig. 2.17.

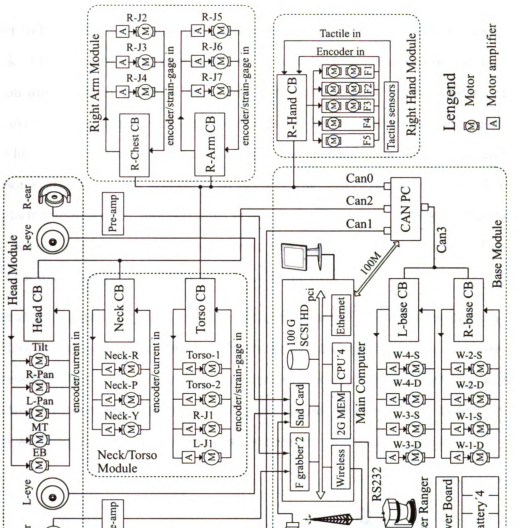


Figure 2.17: The hardware architecture for Dav has two levels: a main computer system and low-level servo control modules.

Main Computer System

In order to build the main computer system, it is essential to know the requirement of the task. The developmental program must process high-dimensional sensor data such as vision and auditory input in real time. The computation power is hence critical to achieve this real-time processing requirement. Various solutions were considered for the high-level computer system to achieve the requirements. The single and dual processor systems are not sufficient to achieve the required throughput. A choice for a quadruple-processor motherboard is required. Therefore, desktop motherboards (ATX) with 4 Pentium III Xeon processors and 6 PCI slots were chosen. Fig. 2.18 shows this main computer system with a LCD monitor.

The main computer system situates in the middle of Fig. 2.17, including an 100G SCSI RAID0 disk array, two frame-grabbers for left and right cameras, a sound card, a wireless Ethernet card, and an Ethernet card connecting CANPC (see Fig. 2.20) for interaction with low-level modules. Being the "brain" of Dav, this computer system handles all "cognitive" computation during learning or testing sessions.

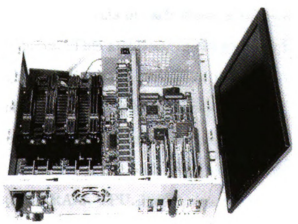


Figure 2.18: The "Brain" of Dav, a quadruple processor desktop PC. The SCSI RAID0 disk array is not included.

Sensors

Dav has visual, auditory, tactile, range, encoder, strain gage, and current sensors. The visual system consists of two micro color CCD camera heads with diameters of 7mm and focal lengths of 2.2mm. They are installed on the eye platform in the head. Each camera has a separate control box that is located in the torso, providing automatic gain control and automatic electronic shutter control. Video signal sampling is performed by two frame-grabber cards, which communicate with processors via a PCI interface bus. Two microphones are mounted on each side of the head. The auditory signals go through pre-amplifiers before they are sampled by a sound card in the main computer. The touch and torque sensors are important for hand-in-hand teaching, i.e., supervised learning. A laser range scanner is mounted in the front of the robot to realize range sensing.

Embedded Controller

In the low-servo control level, a distributing modular scheme was adopted. The low-level actuator control units (ACU) were placed as close to actuators or sensors as possible, to reduce the wiring. Eleven ACU modules have been designed and fabricated to satisfy the requirements of each motor groups, and were networked to CANPC by four CAN buses. This architecture reduces hundreds of signal wires to only two wires, through the entire body, plus other three wires for power supply (See Fig. 2.19). Consequently, the system reliability increases.

Each of microprocessor-based servo modules is composed of two connected boards: servo interface board and an embedded Motorola PowerPC MPC555 board with 448k internal flash, 24K internal RAM, TPU, PWM, and ADC, as shown in Fig. 2.19 (b). The signal connector between the two boards serves two purposes: to attach modules physically together as well as electrically. Those ACU modules, with capability of interfacing up to four motor/joints, communicate over a CAN bus with a rate of 1M bps.

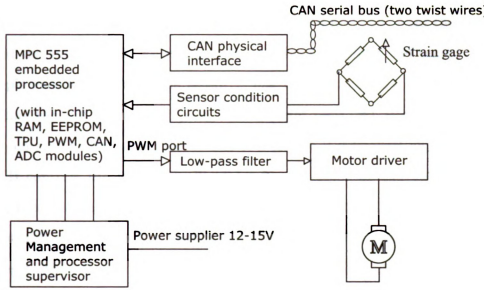
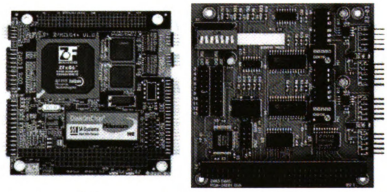


Figure 2.19: The block diagram of PowerPC based controller.



(a) (b)

Figure 2.20: CANPC: a PC104 embedded system running RealTime Linux operating system with four CAN interface (a) PC104 computer board (MZ104+). (b) PCM3680 Dual-CAN interface board.

Distributed Architecture

Fig. 2.17 illustrates the overview of the hardware design. There are head, left arm, neck-torso, right arm, left hand, base, and right hand modules, from the top to bottom and the left to right. Each motor has a driving amplifier and sensors such as a quadruple encoder and a torque sensor[4]. The head module has an ACU: HeadCB, controlling the Tilt, L-Pan (left pan), R-Pan (right pan), MT (mouth), and EB (eye brow) joints. The left arm module has two ACUs: L-ChestCB and L-ArmCB. L-ChestCB controls the left arm's J2 (Joint 2), J3 (Joint 3) and J4 (joint 4) while L-ArmCB controls the left arm's J5 (Joint 5), J6 (joint 6) and J7 (joint 7). Similarly, the right arm module has R-ChestCB and R-ArmCB ACUs, which control the corresponding to the left arm. The torso module has two ACUs: NeckCB and TorsoCB. NeckCB controls the neck's three joints: Neck-R (roll), Neck-P (pitch), Neck-Y (yaw) while TorsoCB controls the torso's two joints (torso's Joints 1 and 2) and the first joint of the two arms (left arm's Joint 1 and right arm's Joint 1).

The base module is the hub of the whole system where the main computer system is housed. The base module has two ACUs: L-BaseCB and R-BaseCB. L-BaseCB controls W-1-D (Wheel 1's driving motor), W-1-S (Wheel 2's steering motor), W-2-D (Wheel 2's driving motor), and W-2-s (Wheel 2's steering motor) joints. R-BaseCB controls W-3-D (Wheel 3's driving motor), W-3-S (Wheel 3's steering motor), W-4-D (Wheel 4's driving motor), and W-4-s (Wheel 2's steering motor) joints. In addition, the fabricated power board and 4 sealed lead acid batteries (with capacity of 110AH @12V for each one) are placed in the base module. The range finder (PLS, SICK Co.), connected to main computer by a RS232 interface (port 1), was mounted at the front side of the base for realizing the range sensing.

We notice that those ACUs are not directly connected to the main computer system. They are connected via CANPC instead. CANPC has four CAN buses: Can0,

[4]Armature current sensors were placed on motors of all modules except the two arms.

Can1, Can2, and Can3. As shown in Fig. 2.17, Can0 links TorsoCB, R-ChestCB, R-ArmCB and R-HandCB; Can1 links L-ChestCB, L-ArmCB and L-HandCB; Can2 links Neck-CB and Head-CB; Can3 links L-BaseCB and R-BaseCB.

At the top of Fig 2.17. Two microphones with pre-amplifier are linked to a sound card plugged into main PC's PCI slot. Two micro camera heads, fixed in the inside of the head mechanism, are linked to two digital control boxes housed in the base. The control boxes' outputting RGB signals are sent to the frame-grabbers in the main computer.

2.6.2 Software Development

Dav's control system involves the development of the large-scale control system that must operate with a significant and high bandwidth. A network of embedded processors adds to the complexity of software development. Many issues such as modularity, scalability, reusability, internal communication transparent to location, flexibility, plug-and-play capabilities and open architecture, are needed to be considered.

There are substantial amounts of work addressing those issues and a lot of solutions are proposed. For example, Ayllu [53] and COLBERT/Saphira [52] are used to control the ActivMedia Pioneer robots. However, those proposed solutions are designed for a particular control philosophy. They do not intend to encapsulate and to hide the low-level device details, in the meanwhile offer enough flexibility for high-level program if a different control strategy is needed. Player [96], DCA [60, 70] and OROCOS [1] are hence proposed. The latter provides standard software packages (e.g., abstract device model) to facilitate high-level robot software development.

As in Player [96], Dav's software development was guided by our desire to concurrently support many heterogeneous devices and many heterogeneous clients. Each device operates at some inherent frequency, with wide variation among devices. For example, the popular SICK PLS laser ranger returns a full scan at approximately 8Hz, while the robotic joints can give encoder feedback at almost 1000Hz. If we were

to take the naive approach and poll each device in turn at the rate of the slowest device, we would be discounting the full capabilities of the available resources. In most cases, we want to obtain data from and send commands to each device at the highest rate possible in order to fully exploit the hardware and maximize the responsiveness of the system. Similarly, each client operates at some inherent frequency; while a simple client written in C++ may be capable of consuming new data at 100Hz, a graphically intensive client written in X-Window might operate at less than 1Hz. It should be possible for these two clients to be connected to Dav simultaneously and to execute as fast as they want without interfering with each other.

Moreover, in order to design a software architecture on a highly distributed sensing and control hardware system as Dav, we need handle data flows between sensors, processors and actuators effectively in groups and across different CAN buses. Dav's designed software architecture, shown in Fig. 2.21, provides a client/server model for realizing the robot interface. Acting as a client, the high-level control program is separate from the server, which executes low-level device operations either on ACU boards or local main computer system, via a unified UDP socket interface. We have three major considerations for choosing such a UDP socket based robot interface.

- Distribution: A client (high-level program) can access to sensors and actuators from any workstation on the Ethernet network. For example, a clients may connect to multiple ACUs while a ACU accepts connections from multiple clients if no conflict exists.

- Convenience: Clients can be written in any language as long as they support socket programming. For example, the user can choose C/C++ for high speed application, Java for trans-platform capability, and MATLAB for quick prototyping.

- Light-weight device server: We feel the commonly accepted CORBA interface is too sophisticated to fit robotic application where real-time constraint is pri-

mary important, even though a lot of effectors have spent to ensure CORBA's performance (e.g., latency, throughput) is reasonable. ACE (Adaptive Communication Environment) and RPC (Remote Procedure Call) are two communication packages provides what we required, but they are not suit for embedded application because of their big footprint. We use a simple and direct approach, UDP socket interface shown in Fig. 2.21. Acting as the device server for low-level ACUs, CANPC receives UDP packets and forwards CAN packets to ACUs, and vice versa. Each UDP packet may contain several CAN packets (up to 64 packets) to improve the throughput.

Dav is implemented in C++ and makes extensive use of the POSIX-compliant pthread interface [5]. Since clients and physical devices may operate at different time-scales, we assign an asynchronous daemon thread for each physical device. A main thread listens for new client connections on a well-known UDP port, spawning client-stub threads on demand to service clients and the devices they request. The overall system structure of Dav is shown in Fig. 2.21. The center portion of the figure is Dav's software architecture; on the left are the physical devices and on the right are the clients. Each client has a UDP socket connection to Dav. If the client is executing on the same host as Dav, then this socket is simply a loopback connection; otherwise, there is a physical network in between the two. At the other end, Dav connects to each device by whatever method is appropriate for that device (e.g., RS-232 for the ranger finder and CAN for ACUs). Within Dav's architecture, the threads communicate through a shared global address space. As indicated in Fig. 2.21, each device has associated with it a command buffer and a data buffer. These buffers provide an asynchronous communication channel between the device threads and the client threads. For example, when a client reader thread receives a new command for a device, it writes the command into the command buffer for that device. Later,

[5]Although the current implementation of Dav's software architecture is on LINUX system, it is straightforward to port on Window system

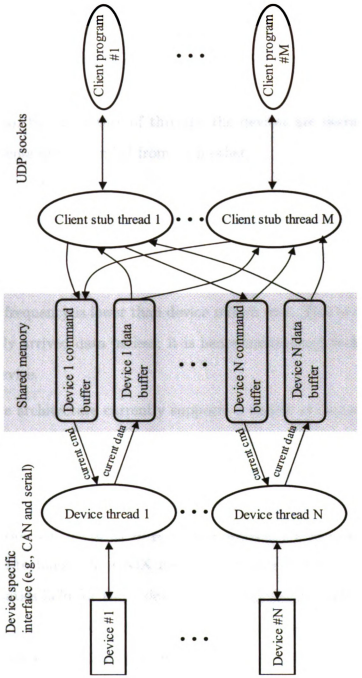


Figure 2.21: The software architecture of Dav's control system.

when the device thread is ready for a new command, it will read the command from its command buffer and send it on to the device. Similarly, when a device thread receives new data from its device, it writes the data into its data buffer. Later, when a client thread is ready to send new data from that device, it reads the data from the data buffer and passes it on to its client. In this way, the client program threads are decoupled from the device daemon threads (and thus the clients are decoupled from the devices). Also, by the nature of threads, the devices are decoupled from each other, and the clients are decoupled from each other.

Since the bandwidth and latency are primarily important for real-time system, we do not apply synchronization (e.g., semaphores) on the shared memory in the middle of Fig. 2.21. Thus, old data may be read when the client thread's reading frequency is higher than device thread's refresh rate. On the other hand, a client may miss data when its reading frequency is lower than device refresh rate. This is reasobable, since we treat the lately arrived data useless; it is hence unnecessary to keep a long data buffer for each device.

Dav's software architecture currently supports a variety of devices, including the popular peripherals (e.g., ACUs, GPS, camera and frame-grabber combination, microphone and sound card, range finder). Devices with serial (RS-232), CAN or PCI interface can be easily added to the system by developing a device driver and modifying a configuration file. In order to provide a uniform abstraction for a variety of devices, we chose to follow the UNIX model of treating devices as files. Thus the familiar file semantics hold for Dav's devices. For example, to begin receiving sensor readings, the client opens the appropriate device with read access; likewise, before controlling an actuator, the client must open the appropriate device with write access.

By default, client programs receive data at 10Hz, which is roughly the signal latency via neural circuit from the brain to peripheral neurons. Namely, a client can expect to receive a data packet containing the current data from all the subscribed devices in every 100ms. Certainly, by sending all the data at once, Dav's device

server might repeatedly send old data from a device that operates at less than 10Hz. We designed Dav's architecture in this way for one reason: simplicity. By always transmitting the current state for all subscribed devices, regardless of the time-scale of the device, we facilitate the writing of client programs. As a result, clients are able to use a simple read loop to receive data from Dav's device server. It is worthy to note client programs can issue command to Dav at any time.

2.7 Experiment

All Dav's joint and the control system have been successfully implemented and tested. Experiments were conducted to evaluate the performance of the robot.

Fig. 2.22 shows the movements of the neck and head modules. Fig. 2.23 shows the movements of mobile base. Fig. 2.24 shows the arms' movement.

2.8 Summary

This chapter presented the mechanical structure of the robot. The kinematic model that describes the relation between the joint variables and the position of end-effectors was derived. For the purpose of the lower-level controllers, the dynamic models of Dav's mechanism were outlined. The distributed embedded control system and the main computer system for mental development were sketched.

It is known the kinematic and dynamic models of a humanoid robot are extremely complex [30]. For example, due to the redundancy DOFs of a humanoid, the inverse kinematics, the mapping from end-effector coordinates to joint variables can not be easily written a program to simulate. The learning seems to be a promising alternative solution. It is known that a robotics model that does not encompass learning will always require a human programmer for building the movement plans [47], which is not acceptable for a autonomous robot.

In the next chapter, we will introduce a new learning framework for robotics: a theory of developmental mental architecture and Dav's implementation of this architecture.

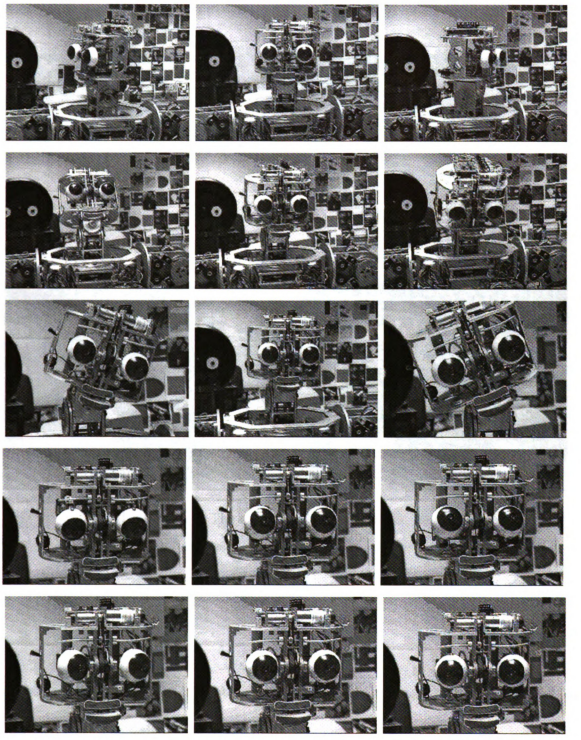


Figure 2.22: The movement of Dav's neck and head modules. The first, second and third rows show the rotations of the neck's Roll, Pitch, and Yaw joints respectively; the fourth and fifth rows show the tilt and pan motions of the eyeballs respectively.

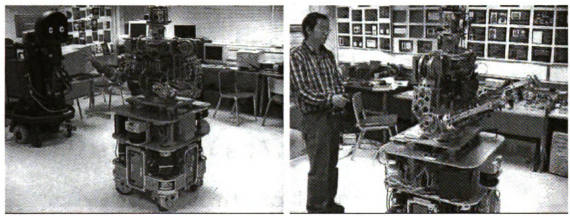


Figure 2.23: The movement of Dav's mobile base.

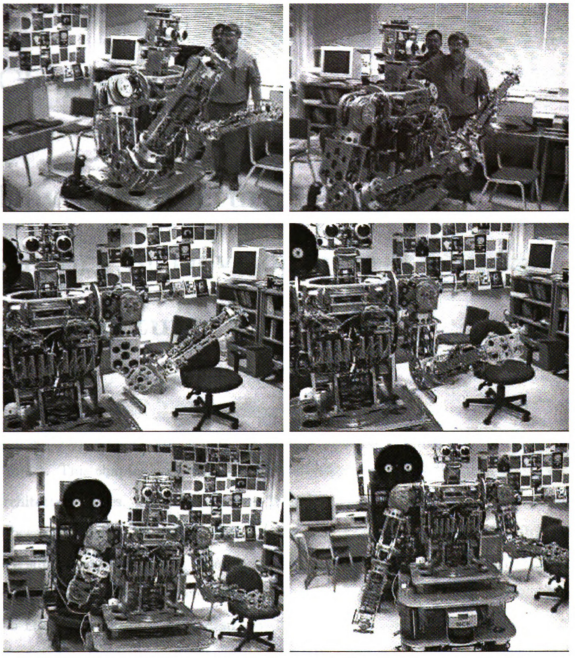


Figure 2.24: The movement of the arm's Joint 1, Joint 3 and Joint 4, from the top row to the bottom.

Chapter 3

Dav Architecture Design: A Theory of Developmental Mental Architecture

The software architecture of a developmental robot is a challenging new research subject. This chapter presents a theory of developmental mental architecture. Five architecture types, from the simplest Type-1 (observation-driven Markov decision process) to Type-5 (DOSASE MDP), are introduced. The properties and limitation of a simpler one are discussed before the introduction of the next more complex one. Further, we present the architecture design of the Dav robot. The framework of the Dav architecture is hand-designed, but the actual controller is developed, i.e., generated autonomously by the developmental program through real-time, online interactions with the real physical environment. We present the Dav architecture and the major components that realize the architecture. The designed architecture for Dav is the next generation version from its extensively tested predecessor - the SAIL developmental robot.

3.1 Introduction

Many different architectures have been proposed in the intelligent robot community. Robot perception and perception-based behavior generation have been proved very difficult, especially in unknown or partially unknown environments. Some work on robot learning was motivated by human learning and development: from simple to complex. BAIRN (a Scottish word for "child") is a symbolic self-modifying information processing system used as an implementation for a theory of cognitive development [102]. Drescher [29] utilized the schema, a symbolic tripartite structure in the form of a "context-action-result," to model the ideas of child sensory-motor learning in Piaget's constructivist theory of cognitive development. Soar [56] and ACT-R [6] are two well-known symbolic systems with an aim to model cognitive processes, although cognitive development was not the goal of these efforts. The model of Albus [4] takes sensors as input and produces control signals to effectors as output, thus, allowing a finer numeric representation of sensory features when a specific task is given. The Finite State Machine (FSM) or its probability based variant, the Markov Decision Process (MDP), are two general frameworks that have been used for conducting autonomous learning with a given goal in a symbolic world (e.g., Shen [82]) or reinforcement learning (e.g., Kaelbling et al. [49]). The explanation-based neural network, called "lifelong learning," was used by Thrun [92].

Designing a program and its representation in a task-specific way using a traditional approach is typically very complex, ad hoc, and labor intensive. The resulting system tends to be brittle especially in unknown and uncontrolled environments. Recently an important direction of research aims at reducing or avoiding the human imposed limitations (e.g., features, models) of the world for better environment adaptation in learning. Cresceptron [108] is a system that allows a human teacher to interactively segment natural objects from complex images through which it incrementally grows a network that performs both recognition and segmentation. SHOSLIF [43], SARCOS [97], Cog [19], and Kismet [14] are motivated by simulating infant skills via

learning through interactions with the environment.

The efforts discussed above were motivated by human learning and cognitive development to various degrees. However, these proposed architectures are fundamentally different from human learning and have not reached autonomous mental development (AMD) [118]. The SAIL robot [106] and the Darwin V robot [5] are two prototypes of developmental robot. Darwin V was designed to provide a concrete example of how the computational weights of neural circuits are determined in a controlled environment by the behavioral and environmental interactions of an autonomous device. The SAIL developmental robot was designed for developing perceptual and behavioral skills through interactions in *uncontrolled*, complex human environments (see the eight challenges for AMD in Section 2). There has been a lack of systematic theory for developmental mental architecture.

This chapter introduces a theory of mental architecture suited for autonomous development. As an architecture design example, it presents a new architecture for the Dav robot [37], as shown in Fig. 2.1 the next generation developmental robot after SAIL. The major differences between SAIL and Dav fall into two categories, the body and the "mind." Dav's body is substantially more sophisticated than SAIL's body which only has a single arm. The software architecture of Dav takes advantage of the rich sensors, especially interaction sensors, available on the Dav body. The Dav architecture, presented here, is significantly more complete and integrated than SAIL (e.g., the SASE concept) and has not been published before. However, it takes into account the extensive experience that has been gained from its predecessor SAIL.

Although the overall design of the Dav software architecture has not been tested yet, most of its key components have been successfully tested on SAIL. Architecture design for an intelligent robot is one of the most difficult research topics in intelligent robot research, especially for humanoids. As far as we know, no such highly extensive and complete developmental architecture has been published.

Section 3.2 outlines the AMD paradigm. A theory of mental architecture is pre-

sented in Section 3.3. Sections 3.4 and 3.5 presents the software architecture and its major components of the Dav robot. Section 3.6 provides concluding remarks.

3.2 AMD Paradigm

Shown in Fig. 3.1, the AMD paradigm has two phases. In the first phase, construction and programming, tasks that the robot will end up learning are unknown to the programmer. A task-nonspecific program, called the developmental program, is written during this stage. The most fundamental difference between a developmental program and a traditional program is that the tasks that the machine will execute are unknown to the programmer during the time of programming.

In the second phase, autonomous development, the robot is turned on at time $t = 0$, the robot starts to interact with the physical environment in real-time through continuously sensing and acting. Humans teach the robot to learn tasks from simple to complex via teacher "arranged experience" called scaffolding[1].

Unlike a traditional non-developmental robot, the following eight challenging requirements are necessary for practical AMD:

1. *Environmental openness*: Due to task non-specificity, the developmental robot must deal with unknown, uncontrolled complex environments.

2. *High-dimensional sensors*: The developmental robot must directly handle continuous raw signals from high-dimensional sensors, e.g., vision, audition and tactile sensors.

[1] *Scaffolding* is the process of using developed simple capabilities to further develop more complex capabilities, through further experience (with or without a teacher), without the need of manual modification of the developmental program. Lev Vygotsky [100] proposed the concept of "zone of proximal development" (PZD) which is a latent learning gap between what a child can do on his or her own and what can be done with the help of a teacher. Wood, Burner & Ross [121] used the term "scaffolding" to describe such an instructional support through which the child can extend or construct current skills to higher levels of competence. Through this process, the scaffolding (arranged experience) is slowly removed.

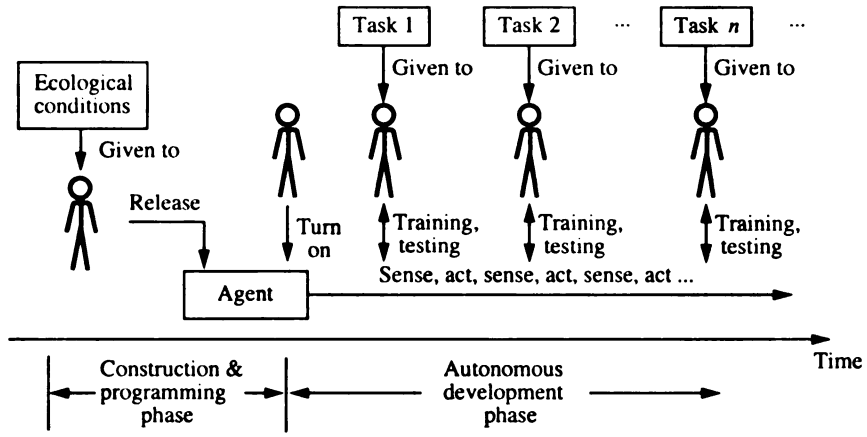


Figure 3.1: The autonomous development paradigm

3. *Completeness in using sensory information*: Due to environment openness and task non-specificity, it is not desirable for a developmental program to discard, at the design phase, sensory information that may be useful for future, unknown tasks. Certainly it selects related information useful for a particular task, after its task-specific representation is autonomously derived after birth.

4. *Online processing*: At each time instant, what the machine will sense next is affected by what the machine does now.

5. *Real-time speed*: The sensory/memory refresh rate must be fast enough so that a physical event (e.g., motion, voice, and vision) can be sampled properly and processed in real-time (e.g., at about 10Hz). It must handle one-instance learning: learning from one instance of experience. Time consuming iterations should be avoided.

6. *Incremental processing*: Acquired skills must be used to assist in the acquisition of new skills as a form of "scaffolding". Each new observation data must be used to update the current representation and discarded after updating.

7. *Performing while learning*: Conventional robots perform after they are built, however, a developmental robot must perform while it "builds" itself (mentally).

8. *Scale-up to complex tasks*: A developmental robot must be able to perform increasingly more complex tasks while gaining more experience.

After solving a series of technical problems, SAIL realized all of the above necessary capabilities. However, a realization of the above capabilities has a level. The Dav developmental architecture aims to significantly advance such a level plus following new concepts and components:

1. The realization of the SASE agent model, which enables the robot to perceive and autonomously control the voluntary parts of internal "brain" activities.
2. Integration of multiple sensorimotor subsystems for multimodal learning.
3. Integration of the architecture with a value system.

This chapter concentrates on these new concepts and new components in the Dav software architecture design.

3.3 Observation-driven Markov Processes

The architecture of intelligent agents is a complex issue. In this section, we introduce a series of architectures, from simple to complex, and the associated properties.

We first define the concept of internal and external environments of an agent.

Definition 3.3.1 (Environment) *The* internal environment *of an agent is the brain (or "the central nervous system") of the agent. The* external environment *consists of all the remaining parts of the world, including the agent's own body (excluding the brain).*

Having defined the external and internal environments, we define the sensors and effectors associated with these concepts.

Definition 3.3.2 (Internal sensors and effectors) *An external sensor S_e is a sensor that senses the external environment. An internal sensor S_i is a sensor that*

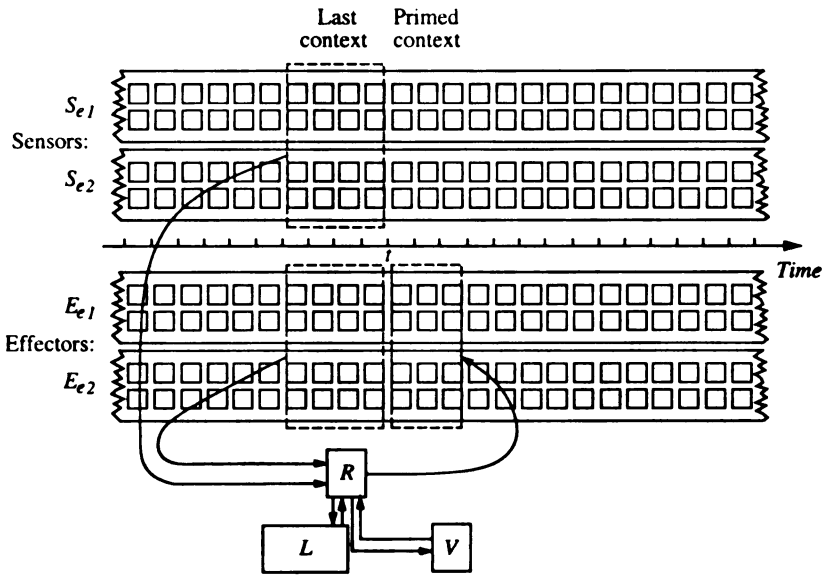


Figure 3.2: The Type-1 architecture of a multi-sensor multi-effector agent: Observation-driven Markov decision process. Each square in the temporal streams denotes a smallest admissible mask. The Type-1 architecture takes the entire image frame without applying any mask. The block marked with L is a set of context states (prototypes), which are clusters of all observed context vectors $l(t)$.

senses the internal environment. An external effector E_e is an effector that acts on the external environment. An internal effector E_i is an effector that acts on the internal environment.

3.3.1 Type-1: Observation-driven MDP

An agent has a number of sensors and effectors. Fig. 3.2 illustrates a multi-sensor multi-effector model of an agent. The agent $A(t)$ operates at equally spaced discrete time instances $t = 0, 1, ...$ We assume that an image is produced at each time instance by the sensor, independent of the sensing modality, visual, auditory, touch, etc. Without loss of generality, we assume that the agent has two external sensors and two external effectors. Each external sensor S_{ei}, $i = 1, 2$, senses a random multi-dimensional sensory frame $x_e(t) = (x_{e1}(t), x_{e2}(t))$ at each time instance t and the sensed signal is fed into the agent. Each external effector E_{ei}, $i = 1, 2$, receives from the agent an effector frame $a_e(t) = (a_{e1}(t), a_{e2}(t))$ at each time instance t.

Let $x_t \in \mathcal{X}$ and $p_t \in \mathcal{P}$ be the observations and outcome covariates (i.e., random vectors) at time t, respectively. Note that we change a variable of a vector to its subscript (e.g., change $x(t)$ to x_t) when it is more convenient to consider the variable as a discrete index number. Let H_t be the random vector of the history: $H_t = \{x_t, x_{t-1}, ..., x_0, p_{t-1}, ..., p_0\}$. At time t, the agent $A(t)$ needs to estimate the distribution of $P(p_t \mid H_t = h)$.

If t is large, H_t is too large to be practical and it contains much information that is not very related to the outcome H_t.

Definition 3.3.3 (Type-1) *The* Type-1 mental architecture *is a k-th order Observation-driven Markov Decision Process (MDP) [27, 122] that reduces the H_t to contain only the last k observations:*

$$l_t = \{x_t, x_{t-1}, ..., x_{t-k}, p_{t-1}, ..., p_{t-k}\}$$

as shown in Fig. 3.2. The random observations in l_t across time $t = 0, 1, ..., t$ are the source from which the agent automatically generates states in the form of clusters $l \in \mathcal{L}$, where $\mathcal{L}$ consists of all possible observations of the last contexts $\mathcal{L} = \{H_t \mid 0 \leq t\}$. The predicted consequence p_t consists of predicted action a_t and the predicted value v_t, $p_t = (a_t, v_t)$.

A major difference between a regular MDP [49, 91] (or HMM [75]) and an observation-driven MDP are that the states s_t with a regular MDP are hand-designed by a human programmer but the states with an observation-driven MDP can be automatically generated (developed). With a regular MDP, the programmer must provide initial estimates for the prior probability distribution $P(s_0)$, the state transitional probability $P(s_t \mid x_t, s_{t-1})$ and the state observation probability $P(x_t \mid s_t)$. It in turn, requires that the human programmer establishes a correspondence between the meanings of the physical events being modeled and the states. Due to the fact that physical events are not known at the time of programing for the developmental robots,

the regular MDP is not suited for the developmental program. In contrast, the observation driven MDP requires no prior probability and all the probability distribution $P(p_t \mid l_t = l)$ can be estimated incrementally on-the-fly.

Another important difference between the states in the traditional MDP and the observation driven MDP is the very different nature of the representation. The states in the former correspond to some objects or events in the world by human hand-design. The entire set of states is a monolithic representation of the modeled part of the world. That results in the high brittleness of the agent in dealing with unexpected events. In contrast, the states in the latter are clusters of observational vectors. They are not monolithic in the sense that an object in the world can correspond to many state clusters. Further, each cluster may correspond to an observation of several objects in the world. Therefore, there is no strict one-to-one correspondence between a state in the observation-driven MDP and an object of the world. It is the behavior of the agent (e.g., the same picking apple behavior from different views of the apples) that shows the discrimination and generalization power of the agent.

In practice, we implement the regressor R using the Incremental Hierarchical Discriminant Regression (IHDR) [112, 114]. Given any observed (last) context $l(t)$, the regressor R produces multiple consequences (primed context) $p_1(t), ..., p_k(t)$ with a high probability:

$$\{p_1(t), ..., p_k(t)\} = R(l(t)). \tag{3.1}$$

Thus, the regressor R is a mapping from the space of the last context $\mathcal{L}$ to the power set of $\mathcal{P}$:

$$R : \mathcal{L} \mapsto 2^{\mathcal{P}}. \tag{3.2}$$

R is developed incrementally through the real-time experience. For any $t > 0$ (after the birth), it is a total function since it is defined for all elements in $\mathcal{L}$, but it does not do well for most elements in $\mathcal{L}$ that it has not experienced. It is not an onto function since its range covers only a very small part of $2^{\mathcal{P}}$.

Therefore, we need a value system that selects desirable contexts from multiple primed ones. The value system $V(t)$ takes a set of (e.g., k) contexts from the regressor R and selects a single context:

$$V(R(l(t))) = V(\{p_1(t), p_2(t), ..., p_k(t)\}) = p_i(t) \tag{3.3}$$

where $1 \leq i \leq k$ and k varies according to experience. The value function selects the best consequence $p_i(t)$ that has the best value $v_i(t)$ in $p_i(t) = (a_i(t), v_i(t))$. For example, $V(\{p_1(t), p_2(t), ..., p_k(t)\}) = p_i(t)$ if $i = \arg\max\{v_1(t), v_2(t), ..., v_k(t)\}$.

The real-time Q-learning algorithm [104] can be used to estimate the value of each consequence $p_i(t)$, $i = 1, 2, ..., k$, and the agent selects the one (action) with the highest value.

Therefore, the value system V is a mapping from the power set of $\mathcal{P}$ to the space of $\mathcal{P}$:

$$V : 2^{\mathcal{P}} \mapsto \mathcal{P}. \tag{3.4}$$

3.3.2 Type-2: Observation-driven Selective MDP

The Type-1 mental architecture is sensor nonselective in the sense that it is not able to actively select a subpart of relevant information from the sensory frame (intra-modal attention) or to attend a particular modality but not the others (inter-modal attention).

Given a d-dimensional input vector x, the attention can be modeled by an attention mask m, where m is a d-dimensional vector whose elements are either 0 or 1. Suppose that the input vector is $x = (x_1, x_2)$ and the mask is $m = (m_1, m_2)$. Then the corresponding attended input vector is $x' = x \otimes m = (x_1 m_1, x_2 m_2)$, where $\otimes$ denotes vector pointwise product. Not all the masks are admissible. For example, the set of admissible masks consists of circles with different radii ρ at different center positions (r_0, c_0) of the image frame. Then, the attention selection effector has three

control parameters: (r_0, c_0, ρ).

Without loss of generality in our theoretical discussion, we assume, as shown in Fig. 3.2, that there are only four addmissible masks for each image frame at time t, denoted by the upper attention square, the lower attention square, and two trivial masks: no square is selected and both squares are selected, respectively. Suppose $x = (x_1, x_2)$ is an input image frame, where x_1 and x_2 denote the subimage under the upper and lower attention square, respectively. With these four admissible masks, the attended image $x' = x \otimes m$ has only four cases, (x_1, x_2), $(x_1, 0)$, $(0, x_2)$, and $(0, 0)$.

For the clarity of the discussion, we will use the notation in the theory of formal languages and automata. Assume that the subimage in each attention square can represent a string $x \in \Sigma^*$, where Σ is the alphabet and Σ^* represents the set of all strings of a finite length formed from the letters of the alphabet. Each attention square can contain a long string as long as the image that it models has a sufficient number of pixels. Therefore, we can write $x_1(t)$ = Karl's body, $x_2(t)$ = trees, etc. A similar notation is applicable to effectors. For example, we can write $a_1(t)$ = move-forward, $a_2(t)$ = move-backward, where a_1 and a_2 are two independently controllable motors of the effector E_{ei}, $i = 1, 2$.

Definition 3.3.4 (Type-2) *The* Type-2 mental architecture *is a Type-1 mental architecture, with an addition of an attention selector*

$$T : \mathcal{Y} \times \mathcal{A}_i \mapsto \mathcal{L},$$

as shown in Fig. 3.3, where $\mathcal{Y}$ is the space of all possible pre-attention contexts $\mathcal{Y} = \{l(t) \mid 0 \leq t\}$, $\mathcal{A}_i$ is the space of all possible attention selections for T and $\mathcal{L}$ is the space of attention-masked last contexts.

In order to show the properties of different architectures, we define a concept called *higher*.

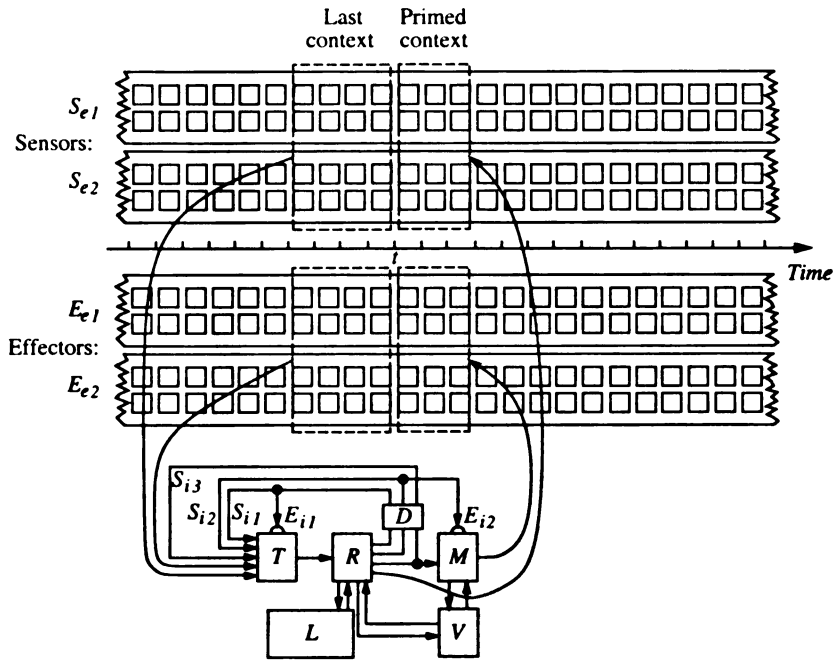


Figure 3.3: Progressive addition of architecture components for Type-2 to Type-5. Type-2: adding T and E_{i1}. Type-3: Adding M and E_{i2}. Type-4: Adding S_i and primed sensation. The block marked with D is a delay module, which introduces a unit-time delay. Type-5: Developmental T, R, M and V.

Definition 3.3.5 (Higher) *Given a set D of tasks, we say that a developmental architecture A_2 is* higher *than another developmental architecture A_1, if given the same teaching environment E, the architecture A_2 requires fewer statistically expected teaching examples than A_1, over the environment E and over the tasks in D.*

Here the term "higher" is motivated by the concept "higher" animals.

As a convention, we regard environment as a part of a task. A task is more challenging if the environment is uncontrolled. In the above definition, since the architectures are developmental, the developmental program that guides the development of the architecture must be task-nonspecific. Otherwise, one can always define an architecture specialized for a particular type of task and this architecture requires fewer teaching examples than a more general developmental architecture. However, the former is not able to deal with other tasks that require more general cognitive capabilities.

Theorem 3.3.1 (Existence of higher architecture) *There is at least one class D of tasks in which only a proper subcomponent of sensory input is related to the tasks and the associated teaching environment E in which the Type-2 architecture is higher than the Type-1 architecture.*

Proof: Assume a set D of tasks whose goal is to classify sensory information in set C (e.g., human body). Without loss of generality, assume that at any time, only one of the attention squares of sensor S_{e1} contains an element (e.g., human body) in C and the other window does not (e.g., natural background that is free of human bodies). For Type-2 architecture, the teacher creates such a teaching environment. First, he teaches the attention selection control of T (e.g., through reinforcement learning), so that T will open two attention squares one after another through a loop (to simulate saccades). If the attention-masked input belongs to C, the regressor R produces the class label as the action output to an external effector (e.g., using supervised learning). Otherwise, R does not produce any output to the external effector. Suppose that the attention window does not contain any element in C but instead contains uncontrolled changing natural settings that never repeat. The Type-2 architecture performs reasonably well, since at least one of the two fixations enables it to detect an element in C while disregarding the other square which always presents a new natural background. In contrast, the Type-1 architecture can only learn from the monolithic input $x(t) = (x_1(t), x_2(t))$. Every vector $x(t)$ is very different from all others because the natural settings are always new in the environment E, the Type-1 architecture never observes a learned input $x(t)$ and, thus, almost never performs correctly. Therefore, for this class of tasks D and this type of environments E, Type-2 is higher than Type-1. □

It is not true that a higher architecture can learn faster than a lower architecture in any setting for any tasks. If the environment is such that the entire input vector $x(t)$ contains only elements in C and nothing of the natural background (which is very rare in reality), then the attention selection mechanism enabled by the Type-2

architecture does not help to reduce the number of training examples.

3.3.3 Type-3: Observation-driven Selective Rehearsed MDP

The Type-2 architecture does not have a motor mapping M. Therefore, it cannot rehearse an action sequence to evaluate its consequences without actually carrying out the action sequence.

Definition 3.3.6 (Type-3) *The* Type-3 mental architecture *is a Type-2 mental architecture, with an addition of an action releaser M:*

$$M : \mathcal{P} \times \mathcal{A}_i \mapsto \mathcal{P},$$

as shown in Fig. 3.3, where $\mathcal{P}$ is the space of all possible predicted consequences, $\mathcal{A}_i$ is the space of all possible attention selection for M.

The action releaser M is a special case of a more general motor mapping which also generates representation for frequently practiced action sequences (e.g., using the principal component analysis PCA) so that smooth action sequences can be generated.

The Type-3 architecture enables the agent to hold an action before it is released to be executed by the environment. It makes it possible for the agent to generate several actions consecutively but to hold each action without actual execution.

With a traditional MDP with hand-designed states, it is possible to compute all the possible next states and perform planning. The Q-learning method uses the estimated action value $Q(s, a)$ of action a at state s to select the best action $a^* = \max_{a \in A} Q(s, a)$, from the set A of all the possible actions. This best next action a^* maximizes the expected rewards in the future. This kind of approach has two fundamental problems. First, the agent has a rigid value system. No matter what value model is used, finite horizon, time discount model with a small or a large time discount factor γ, the agent cannot change the way the value is determined [49, 91].

For example, if a time discount model is used, the agent is short-sighted. It always prefers multiple small rewards in the near furture to a far away but important reward. Second, the agent is not able to learn to "think" (predict) using the events that it learned from experience. For example, fed well and sleep well can be a reasonable goal for a human infant, but the same value system is not suited for a human adult.

The observation-drive MDP does not suffer from these limitations. However, as long as the predicted action is released, the effect that it causes to the external world will result. Such an effect cannot be undone. For example, suppose that the robot predicts (primes) an action to drop a cup from a hand. As long as it executes this action, the cup will drop to the concrete floor and will be broken. Can we design an architecture that enables the robot to "think" and "plan" a significant amount of time ahead before it releases the action?

3.3.4 Type-4: Observation-driven SASE MDP

Defined in Weng [107] a Self-Aware and Self-Effecting agent not only has external sensors S_e and external effectors E_e that sense and act upon the external environment (including the robot body), but also has internal sensors S_i and internal effectors E_i that sense and act upon the internal representation of the brain (not including its body).

In neuroscience, there is no concept of internal sensors and effectors. The brain does not need a receptor to sense the signals in the brain, since the brain signals are already in the desirable form. The concept of internal sensors and effectors is introduced to facilitate computational understanding of the SASE agent.

Definition 3.3.7 (Awareness) *If an agent A senses the states (presence, absence, different forms) of object b through its sensors, we say that the agent A* senses *the object b. If the agent is able to recall the association between the sensed various states of b and their resulting effects in a task domain D sensed by the agent, we say that the agent is* aware *of object b in the task domain D.*

Some explanation is in order here. By definition, an agent must use its sensors, the entry point of its sensory architecture (the input of T), to sense an object. For example, if the state of an object is fed into the architecture, e.g., through the middle of the regressor R, the motor mapping M or the value system V, the agent does not sense the object by the definition because the agent cannot use such information properly as it does with its sensors. In the above definition for awareness, we consider a task domain D because any awareness has a scope. A person who is aware of the boiling temperature of water in a domain (e.g., in a normal environment) may not necessarily be aware of the boiling temperature of water in another domain (e.g., lower in a low pressure environment).

In the definition, we require that awareness must associate various states of b and the corresponding resulting effects. For example, if a human is not aware of the correspondence between various states of gravity (presence, absence, strong gravity) and the resulting effects (e.g., to a falling object), he is not aware of the object.

With the above definition, we are able to deal with the issue of self-awareness and self-effecting.

Theorem 3.3.2 (Necessary conditions of awareness) *Suppose an agent is aware of its mental activities in a domain. Then the following points must be true: (1) It senses such activities using its sensors. (2) It feeds the sensed signal into its perceptual entry point just like that for external sensors. (3) It recalls the association between the different status of the activities and the resulting effects to the environment.*

Proof: Point (1) is true because, according to Definition 7 for the awareness of an object, the agent must sense the object using its sensors. Point (2) is true because the status of the object must be sensed and fed into the entry point for sensors for proper perception and effect recall. Point (3) is true because the definition of awareness requires the recall of such an association. □

Based on the previous theorem, let us examine the issue of self-awareness. If an agent runs a Q-learning algorithm (or any algorithm for that matter) but it does not sense the algorithm using its sensors which are linked to its entry point for sensors, the agent is not aware of its own algorithm. For the same reason, humans do not sense the way their primary cortex works and, therefore, normally they are not aware of their own earlier visual processing. This early processing is subconscious, in the sense that it does not require a conscious decision. However, the voluntary part of the mental decision process does require a conscious, willful decision. Therefore, in the architecture design, the parts that require voluntary decisions must be sensed by the agent, and the sensed signals must enter through the entry point of the sensors.

Definition 3.3.8 (Type-4) *The* Type-4 mental architecture *is a Type-3 mental architecture, but additionally, the internal voluntary decision is sensed by the internal sensors S_i and the sensed signals are fed into the entry point of sensors, i.e., the entry point of the attention selector T. In order to recall the effects of the voluntary actions, not only is the expected reward value estimated by the value system, but also the primed context, which includes not only the primed action, but also the primed sensation.*

The architecture illustrated in Fig. 3.3 is a Type-4 architecture. Two voluntary internal actions are modeled by E_{i1} for attention selection, and by E_{i2} for action release. Both internal actions are sensed by the internal (virtual) sensors S_{i1} and S_{i2}, respectively. The rehersed external action (not released) is sensed by the virtual internal sensor S_{i3}. Note that when the action is released, it is sensed as external action from E_{e1} and E_{e2}. The primed sensation (which predicts the sensation of S_{e1} and S_{e2}) is used by the value system in selecting the best action according to, e.g., the novelty (suprise) or the nature of the reward (e.g., sweet or bitter).

The regressor R maps each attended context $l \in \mathcal{L}$ to a set of multiple primed contexts, from which the value system selects a single primed context $p \in \mathcal{P}$. In other

words, the composite function of R followed by V gives a mapping: $V \circ R : \mathcal{L} \mapsto \mathcal{P}$. With a SASE agent, both external context (sensed by S_e) and internal context (sensed by S_i) are available in l.

With a consecutive time series $t = 1, 2, ..., k$, the composite function $V \circ R$ performs a series of regressions, represented by a regression sequence:

$$s = ((l_1, p_1), (l_2, p_2), ..., (l_k, p_k))$$

where each regression pair (l_i, p_i) is an input-output pair of the composite function $V \circ R$, $p_i = V \circ R(l_i)$, $i = 1, 2, ..., k$. The link between two consecutive regression pairs can be realized by two paths, the external path and the internal path, denoted by e and i respectively. If a part of the primed context p_i is executed by an external effector and the effect of the action on the external world is sensed by an external sensor S_e, the path corresponds to an external path. For example, if a mobile robot moves ahead, the objects look closer after the motion. If a part of p_i is sensed by one or multiple internal sensors, the path corresponds to an internal path. For example, if the primed action is to jump, the following input to the regressor R will contain a representation of this action through a (virtual) S_i, no matter if the action is executed or not. Symbolically, the reasoning process can be represented by the following composite reasoning sequence:

$$s = \left((l_1, p_1), \begin{bmatrix} e_1 \\ i_1 \end{bmatrix}, (l_2, p_2), \begin{bmatrix} e_2 \\ i_2 \end{bmatrix}, ..., \begin{bmatrix} e_{k-1} \\ i_{k-1} \end{bmatrix}, (l_k, p_k), \begin{bmatrix} e_k \\ i_k \end{bmatrix} \right) \tag{3.5}$$

where

$$\begin{bmatrix} e_i \\ i_i \end{bmatrix}, i = 1, 2, ..., k$$

represents the external and internal paths. Whether the result of external and internal paths are taken into account by the regressor in the reasoning process depends on the

attention selection in T.

Definition 3.3.9 (External and external reasoning process) *There are three types of reasoning processes, external, internal, and mixed, corresponding to the attention in which the attention module T attends to external, internal or both, respectively.*

We can readily arrive at the following conclusions:

- Type-1 through Type-3 architectures allow the agent to perform external reasoning processes (i.e., practice grounded in the physical world).
- Type-1 through Type-3 architectures do not allow the agent to perform internal reasoning as defined. For example, Type-3 is not aware of the primed context because it does not have internal sensors.
- A Type-4 architecture is able to execute external, internal, and mixed reasoning processes.

One might conclude at this point that the internal process looks like "thinking." However, if the internal representation is hand-designed for a given specific task, the internal process is still far from what is called thinking by higher animals and humans.

3.3.5 Type-5: Developmental observation-driven SASE MDP

In the architecture discussed above, no requirements have been placed in terms of how the architecture is created. Specifically, how should the mappings T, R (and L), M, and V be created?

Definition 3.3.10 *(DOSASE MDP) The developmental observation-drive SASE MDP (DOSASE MDP) has an architecture Type-4 or higher, that satisfies the following requirements:*

1. *During the programming time, the tasks that the agent will learn are unknown to the programmer.*

2. *The agent $A(t)$ starts to run at $t = 0$ under the guidance of its developmental program P_d. After the birth, the brain of the agent is not accessible to humans.*

3. *Human teachers can only affect the agent $A(t)$ as a part of its environment through its sensors and effectors recursively: At any time $t = 0, t = 1, ...$, its observation vector at time t is the last context $l(t)$. The output from $A(t)$ at time t is its selected primed context $p(t) \in \mathcal{P}$. $A(t-1)$ is updated to $A(t)$, including T, R (and L), M, and V.*

In contrast with the traditional MDP, the DOSASE MDP $(A(t), P_d)$ is developmental in the sense that the developing program P_d does not require a given estimate of the *a priori* probability distribution $P(l)$ for all $l \in \mathcal{L}$, nor even a given set of states. Consequently, P_d does not require a given estimate for the state observation probability $P(l_t \mid l_{t-1})$ nor that for the state observation probability $P(x_t \mid l_t)$.

When the number of states is very large, it is sufficient practically to keep track of the states that have a high probability, instead of estimating probability of all the states, which is too computationally expensive to reach real-time speed.

3.3.6 Type-6: Multi-level DOSASE MDP

The Type-5 architecture has only one sensorimotor level, although each mapping T, R, and M have multiple levels in their own internal structure. We call it a sensorimotor level because the pathway from T through R up to M corresponds to a pathway from sensory input to motor output. The primed context of such a level can be fed into another sensorimotor level for the following reasons:

1. Abstraction. While a low level is tightly linked to fine time steps, the higher levels become more "abstract," in the sense that the higher level clusters of

context states are coarser in temporal granularity and grouped more according to actively attended events. That is self-directed abstraction according to active attention. For example, when the agent visually tracks a walking person, the attended part of the person's images are grouped.

2. Self-generated context: Allow voluntarily generated motor actions to serve as context input to the higher level. Thus the agent is able to "talk to itself." This capability also helps abstraction because the variation of self-generated motor actions has a smaller within-class variation and a lower dimension that the typical sensory inputs from the external environments.

3. Enabling a higher degree of sensory integration: It is not practical to integrate all the receptors in a human body by a single attention selection module T, otherwise, the attention is too complex. Sensors that are related more closely (e.g., touch sensors of the same finger) are integrated first by a low level sensorimotor system and the output of these sensorimotor systems are fed into the next level which integrates a larger scale of sensors (e.g., touch sensors of different fingers).

With the above considerations, we introduce the Type-6 architecture.

Definition 3.3.11 *The Type-6 mental architecture is a Type-5 mental architecture with several levels of sensorimotor systems. Each Type-5 architecture is considered a sensorimotor system. The primed contexts from the lower-level sensorimotor systems are fed into the sensory input of the higher-level sensorimotor systems.*

Fig. 3.4 illustrates the Type-6 architecture. The input to the attention selector $T^{(2)}$ at level 2 includes the primed context $p(t) = (x_p(t), a_p(t))$ from level 1, where $x_p(t)$ and $a_p(t)$ are primed sensation and primed action, respectively. One or multiple sensorimotor systems can feed their primed contexts into a single sensorimotor system at the next higher level for sensory integration.

It is possible to prove that there is at least one task set and the associate teaching environment where the Type-6 architecture is higher than the Type-5 architecture.

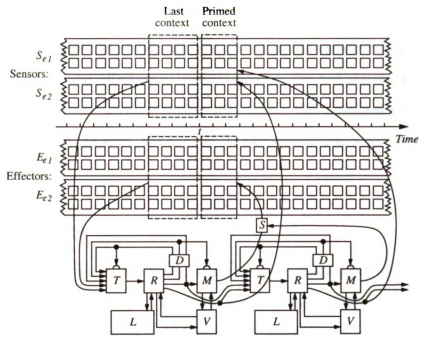


Figure 3.4: The Type-6 architecture.

We have systematically introduced six types of architectures. Although the order in which new capabilities are added is more a design choice than a necessity, the order used here is motivated by a relatively large payoff in capability with a minimal addition of the architecture complexity.

3.4 Dav Architecture

The Dav architecture design was guided by the theory of architecture discussed in the previous section. It's architecture belongs to Type-6. The basic modules T, R, and M in the last section have more functions with the Dav architecture. In the Dav architecture, the attention selection module T discussed above corresponds to the sensory mapping, the regressor R corresponds to the cognitive mapping and the action releaser M corresponds to the motor mapping. In the sensorimotor system of the Dav architecture, two cognitive mappings are used, implemented by the same engine IHDR. The reality mapping R predicts contexts of near (immediate) future, while the priming mapping F predicts contexts of far future.

As shown in Fig. 3.5, the Dav architecture consists of parallel layers of the sensorimotor levels. The lower the level, the less spatial and temporal extent it deals with. The fewer receptors and effectors a level integrates, the less spatial extent it deals with. Also the lower the level, typically the shorter the temporal context it copes with. Thus, the lower the level, the simpler its perceptual and behavioral capabilities are.

3.4.1 Open view

Fig. 3.6 gives an open view of a sensorimotor system. A sensorimotor system can be regarded as an observation-driven MDP. However, as we discussed in the last section, it is not the same as the traditional MDP with hand-designed states.

Each circle in Fig. 3.6 indicates a local context state, corresponding to a single time frame. At each time frame, the system grabs the current sensory input x, which is used for updating the last context L. Two mapping engines: R and F are used to map from the last context L to a set of primed contexts P. In Fig. 3.6, a priming transition from one context to the next is indicated by a dashed line.

Fig. 3.7 illustrates how the temporal trajectory is generated from the visited context states. What happens in the physical world (including the action of the agent) and the internal world (including the internal action of the agent) jointly determine the last context, which in turn determines which context state is visited. The next context visited provides information for updating the primed contexts from the current context state. In Fig. 3.7, such updating is simply denoted as an addition to the set of primed contexts P.

3.4.2 Recursive view

The above open model of a sensorimotor system is straight forward to visualize the concept of context state, but it is not suited for showing how the architecture realizes the system. The recursive model of a sensorimotor system, shown in Fig. 3.8, can

better illustrate the computational architecture of a sensorimotor system. As shown in the figure, both the reality mapping R and priming mapping F are accomplished by a mapping engine. Each context state is represented by a vector in the input space of the mapping engine.

With the recursive model, we can see that the effect of a value system V can be modeled as a mapping that takes multiple primed contexts from the reality mapping R or the priming mapping F, and selects a preferred context. Its action part is sent to all the corresponding effectors. In the past, supervised learning is separated from reinforcement learning. Reinforcement learning only works on an externally given scalar reward value. We treat a value system in a more general way. Its innate part is driven by physical pleasure (e.g., a sweet taste) or pain (e.g., an electric shock). Its learned part is responsible for learning values of events from the external world (e.g., working hard is good). Thus, the value system uses not only a reward value, but the primed context as well (including sensations and actions).

3.4.3 Architecture view

A more detailed block diagram of the architecture of a sensorimotor system is shown in Fig. 3.9. Each internal and external action output feeds back, through a delay unit, into the next sensory input. This is required by the SASE agent model: the agent must sense and perceive what it does, internally and externally. The input to a sensorimotor system, indicated by the two left-most arrows in Fig. 3.9, is its target for perception and cognition.

A sensory mapping [127] is needed wherever an attention selection effector (local analysis) or dimension reduction is needed. As shown in Fig. 3.9, a sensory mapping is used to enable attention selection from the current sensory input vector and another sensory mapping enables the attention selection of the current input, the previous context, or both. As mentioned earlier, each sensorimotor system has two cognitive mappings, the reality mapping R and the priming mapping F. A near future context

from R is useful for carrying out skilled procedures. A far future context from F is needed to predict the future consequence (e.g., 10 frames or more into the future).

The source of input to the priming mapping has two alternatives: the same input as the reality mapping (as shown in the figure) or the primed context output from the reality mapping. The former allows more accurate control of the timing in the primed context while the latter may enable more efficient "abstraction" of the context in the priming mapping, since the primed context from the reality mapping has already taken the output action into account (e.g., the discriminant analysis by the IHDR tree in section 4.4).

The priming mapping, implemented by a cognitive mapping engine, IDHR needs a *prototype updating queue* whose function is to predict far future context using the Q-learning algorithm [41] in a recursive way. The reality mapping does not need such a queue because it only predicts the next near future.

The motor mapping of a sensorimotor system generates concise representation for stereotyped actions, in addition to the function of action release.

3.5 Major components

The architecture in Fig. 3.9 divides the information processing into three major mappings: the sensory, cognitive and motor mappings. The sensory mapping takes sensory inputs. It may be followed by another sensory mapping to increase the extent of coverage of space and time. The sensory mappings are followed by a cognitive mapping.

The basic difference between the sensory mapping and the cognitive mappings is that the sensory mapping does not use the information of motor output for its feature space development, but the cognitive mapping does.

The motor mapping has two functions: (1) generating a higher motor representation in space and time for motor output space and (2) receiving signals from cognitive

mappings in terms of a higher motor representation and recovering the detailed control signal for every motor that it handles.

In this section, we first discuss these three major mappings, next introduce an innate value system, and finally an algorithm is given to summarize the sensorimotor architecture.

3.5.1 Sensory mapping

The sensory mapping provides representation for all possible receptive fields in space and time and allows attention selection. In the past the role of representation has been well recognized but the purpose of attention selection has not been adequately studied.

Fig. 3.10 shows the hierarchical spatiotemporal organization of sensory mapping. The major functions of the sensory mapping include:

1. Grouping different sources of sensory input (e.g., different pixels in an image or visual and auditory inputs) in space and time. The term "sensory" should be understood as including both raw sensory inputs and processed internal signals.

2. For the grouped set of input, maintaining a complete representation for a hierarchy of all possible (sampled) receptive fields, for the purpose of attention. By sampled receptive fields, we mean a large but finite number of receptive fields at different positions and sizes in space and time.

3. Automatically deriving features in the combined space as new representation. It cannot assume a predetermined representation and therefore, it does not use a symbolic representation.

4. Reducing the dimensionality of the new representation, while minimizing the loss of necessary information.

5. Execution of attention selection as an attention effector, controlled by a signal from other parts of the brain (top down control).

In [127], a simplified sensory mapping, Staggered Hierarchical Mapping (SHM), is implemented. SHM uses the Incremental Principal Component Analysis (CCIPCA) method to automatically develop orientation sensitive and other needed filters. In addition, the internal representation generated by SHM for receptive fields at different locations and sizes is nearly complete in the sense that it does not lose important information.

3.5.2 Complementary Candid Incremental Principal Component Analysis

Complementary Candid Incremental Principal Component Analysis (CCIPCA) is a stochastic approximation algorithm to estimate eigenvalue and eigenvector iteratively. It is the major tool to implement the sensory mapping.

PCA is a well-known technique in data compression and feature extraction. It gives a linear transform that converts a set of d-dimensional data into a lower-dimensional space by minimizing the error in the least mean square (LMS) sense. Therefore, PCA is used to generate new representations from sensory inputs.

A well-known approach to PCA is to solve an eigensystem problem. Given A as the sample covariance matrix of the data set, one can find its eigenvectors and eigenvalues, sort the eigenvalues in descending order, and construct a $k \times d$ matrix T with the rows being the eigenvectors corresponding to the largest k eigenvalues. The matrix T is the sought transform [35]. This is the basic idea behind most techniques for PCA, such as the QR method [36]. However, since this approach requires an estimate of the covariance matrix, the data set usually needs to be completely available at the computation time. This is not appropriate for a developmental robot because of two reasons. First, a developmental robot is an online agent, which senses and responds

to the environment continuously. It should not wait until all data is accumulated before doing the processing. Second, when the dimension of the data is high, both the computation and storage complexity grow dramatically. For example, in the eigenface method [50] [94], one of the promising face recognition methods that involves PCA, a moderate grey level image is of 88 rows and 64 columns, which results in a 5632-dimensional vector. Since the sample covariance matrix of a data set of d-dimensional random vectors contains $d(d+1)/2$ independent numbers, this amounts to 15,862,528 numbers!

[119] proposes an algorithm, called complementary candid incremental PCA (CCIPCA), to incrementally compute the principal components of sequentially arrived samples without estimating the covariance matrix. Suppose that sample vectors are acquired sequentially, $u(0), u(1), \ldots$. Each $u(n)$, $n = 0, 1, \ldots$, is a d-dimensional vector and d can be as large as 5000 or even beyond. The $d \times d$ covariance matrix is neither known nor affordable to be estimated and stored as an intermediate result. Without loss of generality, [129] assumes that $u(n)$ has a zero mean. The algorithm of CCIPCA is as follows,

$$v_i(n) = \frac{n-1}{n} v_i(n-1) + \frac{1}{n} u_i(n) u_i^T(n) \frac{v_i(n-1)}{||v_i(n-1)||}, \tag{3.6}$$

$$u_{i+1}(n) = u_i(n) - u_i^T(n) \frac{v_i(n)}{||v_i(n)||} \frac{v_i(n)}{||v_i(n)||}, \tag{3.7}$$

where, $u_1(n) = u(n)$, and $v_i(n)$ is the i-th dominant eigenvectors of the samples' covariance matrix estimated after collecting the n-th sample.

The idea underlying CCIPCA comes from the concept of statistical efficiency [46]. Basically, this definition says that the most efficient estimate is one that has the least variance from the real parameter and the variance is bounded by the Cramér-Rao inequality. For example, the sample mean, $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$, is the most efficient estimate for the mean of a Gaussian distribution with known standard deviation σ [33].

For CCIPCA, by substituting itself recursively, Eq (3.6) can be written as,

$$v_i(n) = \frac{1}{n}\sum_{j=1}^{n} u_i(j)u_i^T(j)\frac{v_i(j-1)}{||v_i(j-1)||}.$$

Thus, $v_i(n)$ can be viewed as the mean of "samples" $w_i(j)$,

$$v_i(n) = \frac{1}{n}\sum_{j=1}^{n} w_i(j), \tag{3.8}$$

where $w_i(j) = u_i(j)u_i^T(j)\frac{v_i(j-1)}{||v_i(j-1)||}$. Although $w_i(j)$ is not necessarily drawn from a Gaussian distribution independently, the estimate $v_i(n)$ seems to have a close relation with the estimate of a sample mean which is the most efficient estimate. [129] have proved that $v_i(n) \to \pm\lambda_i e_i$ with probability one when $n \to \infty$, where λ_i is the i-th largest eigenvalue of the covariance matrix of $\{u(n)\}$ and e_i is the corresponding eigenvector.

The above analysis prompts a further improvement to CCIPCA. As shown in Eq. (3.8), all the "samples" ($\{w_i(j)\}$) are weighted uniformly when $v_i(n)$ is estimated. Since $v_i(j)$ is far away from its real value at early estimation stage, the so-generated $w_i(j)$ can be regarded as a "sample" with large "noise" when j is small. To help the convergence of the estimation, it is preferable to give smaller weight on these early "samples". A way to implement this idea is to use an amnesic average by changing Eq. (3.6) into,

$$v_i(n) = \frac{n-1-l}{n}v_i(n-1) + \frac{1+l}{n}u_i(n)u_i^T(n)\frac{v_i(n-1)}{||v_i(n-1)||}, \tag{3.9}$$

where l is called *amnesic parameter*, typically, ranging from 2 to 4. With l, Eq. (3.9) automatically adjusts the weights of old estimates and new samples.

In summary, the CCIPCA algorithm is as shown in Fig. 3.11. CCIPCA has been shown empirically to be a very efficient estimation algorithm compared with SGA and GHA for high-dimensional data [128].

3.5.3 The cognitive mapping: IHDR tree

The R and F mappings in Fig. 3.9 are implemented by two Incremental Hierarchical Discriminant Regression (IHDR) trees [44]. IHDR is not new for this thesis research. We first survey other related regression methods, then outline the improvements over the old versions [111, 113]. An experimental evaluation is provided at the end.

Learning a function $f : \mathcal{X} \mapsto \mathcal{Y}$ in real time for high-dimensional data remains a nontrivial problem, particularly when the complexity of the function to be estimated are unknown. By surveying the literature of regression in statistical learning, one can identify two classes of methods: global model fitting and local model fitting methods.

Global model fitting methods, including Multiple Layer Perceptron (MLP) [12], Radial Basis Function (RBF) [22], Lasso regression shrinkage [93] and Support Vector Machine based Kernel Regression methods (SVMKR) [88], are characterized by adjusting a predefined model using a pre-collected training data set via optimizing global criteria. They are not suited for the real-time learning in high-dimensional spaces despite their theoretic background. First, they require a priori task-specified knowledge to select right topological structure and parameters. Their convergent properties are sensitive to the initialization biases. Second, those methods are designed primarily for batch learning and are not easy to adapt for incrementally arrived data. Third, their fixed topological structures eliminate the possibility to learn increasingly complicated scenes.

Local model fitting methods (e.g., IHDR) use temporal-spatially localized (thus computationally efficient) models, in the meanwhile, grow the complexity automatically (i.e., the number and organization of local models) to account for the nonlinearity and the complexity of the problem. They are more suited for incremental real-time learning, especially for the situation where there is limited knowledge about the scene and the robot, itself, needs to develop the representation of the scene in a generative and data driven fashion.

IHDR generates local models to sample the high-dimensional space $\mathcal{X} \times \mathcal{Y}$ sparsely

based on the presence of data points in a vector quantization (VQ) [51] manner. IHDR enjoys two nice properties: First, IHDR derives automatically discriminating feature subspaces in a coarse-to-fine manner from input space $\mathcal{X}$. Discriminating features are automatically derived discriminating features at the internal nodes of the tree. The features are most discriminative in the sense that they maximize the trace (or the determinant) of between-class scatter. In this way input components that are irrelevant to the mapping's output are disregarded to achieve better discrimination and generalization. Second, IHDR organizes its local models in a hierarchical way, as shown in Fig. 3.14. IHDR's tree structure recursively excludes many far-away local models from consideration (e.g., an input face does not search among nonfaces), thus, the time to retrieve and update the tree for each newly arrived data point $\mathbf{x}$ is $O(\log(n))$, where n is the size of the tree or the number of local models. This extremely low time complexity is essential for real-time learning with a very large memory.

Two kinds of nodes exist in IHDR: internal nodes (e.g., the root node) and leaf nodes. Each internal node has q children, and each child is associated with a discriminating function:

$$l_i(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{c}_i)^T W_i^{-1}(\mathbf{x} - \mathbf{c}_i) + \frac{1}{2}\ln(|W_i|), \tag{3.10}$$

where W_i and $\mathbf{c}_i$ denote the distance metric matrix and the x-center of the child node i, respectively, for $i = 1, 2, ..., q$. Meanwhile, a leaf node, say c, only keeps a set of prototypes $\mathcal{P}_c = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_{n_c}, \mathbf{y}_{n_c})\}$. The decision boundaries for internal nodes are fixed and have quadratic form. A leaf node may develop into an internal node by spawning (see Step 8 of Procedure 1, Appendix B) once enough prototypes are received.

The needs of learning the matrices W_i, $i = 1, 2, ..., q$ in (3.10) and inverting them make it impossible to define W_i (for $i = 1, 2, ..., q$) directly on the high dimensional

space $\mathcal{X}$ [2]. Given the empirical observation that the true intrinsic dimensionality of high dimensional data is often very low [77]. As shown in Figs. 3.12 and 3.13, it is possible to develop most discriminating feature (MDF) subspace $\mathcal{D}$ to avoid the degeneracy and redundancy caused by irrelevant dimensions of the input data ("curse of dimensionality" [10]).

As shown in Fig. 3.14, IHDR partitions the input space $\mathcal{X}$ into regions: $\mathcal{R}_1$, ..., $\mathcal{R}_N$ and each of them corresponding to a leaf node. In each of those region, a linear regression model is used to represent the data samples received. Let the $\mathbf{x}$ be the input querying point. Approximating function f is then accomplished by computing the local linear regression models and by forming a final prediction from the weighted average of the individual predictions

$$\hat{\mathbf{y}} = \sum_{k=1}^{N} \mathcal{I}_{\mathcal{R}_k}(\mathbf{x}) \hat{\mathbf{y}}_k,$$

where $\hat{\mathbf{y}}_k$ denotes the predicted output for the linear regression model covering the region $\mathcal{R}_k$ (see Eqs (A.5) and (A.6) in Appendix B). The weight $\mathcal{I}_{\mathcal{R}_k}(\mathbf{x})$ represents an indicator function, which denotes the likelihood of the input $\mathbf{x}$ belonging to the region $\mathcal{R}_k$ and is defined as

$$\mathcal{I}_{\mathcal{R}_k}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{R}_k \\ 0 & \text{otherwise.} \end{cases}$$

Due to IHDR's hierarchical organization of those regions, the computational complexity $\mathcal{I}_{\mathcal{R}_k}(\mathbf{x})$ is $O(\log N)$ where N is the number of leaf nodes. Therefore the speed of computing the weight is fast even the tree has grown to be a huge one.

In summary, the following improvements were suggested by this thesis:

1. Amnesic average is proposed to relieve the problem of initializing bias. At the

[2]The complexity of inverting an $d \times d$ matrix is $O(d^3)$. When d is big (e.g., 5000), it is infeasible to compute online in real-time.

beginning of the incremental procedure, because the number of samples is small, the clusters are usually ill-generated. So are the boundaries of the cluster, which will influence later partitions of the tree. We use the amnesic average technique to gradually get rid of the effects of earlier data.

Suppose $\{x_i, i = 1, 2, \ldots, n-1\}$ are the data entering the system sequentially. Written in an incremental style, their sample mean $\bar{x}^{(n)}$ and scatter matrix $\Gamma_x^{(n)}$, after receiving the nth sample, are given by,

$$\bar{x}^{(n)} = \frac{n-1-\mu(n)}{n}\bar{x}^{(n-1)} + \frac{1+\mu(n)}{n}x_n, \tag{3.11}$$

$$\Gamma_x^{(n)} = \frac{n-1-\mu(n)}{n}\Gamma_x^{(n-1)} + \frac{1+\mu(n)}{n}(x_n - \bar{x})(x_n - \bar{x})^T. \tag{3.12}$$

$\mu(n)$ is the amnesic parameter which controls the update rate, depending on n in the following way:

$$\mu(n) = \begin{cases} 0 & \text{if } n \leq n_1 \\ b(n-n_1)/(n_2-n_1) & \text{if } n_1 < n \leq n_2 \\ b + (n-n_2)/d & \text{if } n_2 < n \end{cases}, \tag{3.13}$$

where b, n_1, n_2 and d are parameters. Let us remark that $\mu(n)$ is a piece-wisely linear non-decreasing function. $\mu(n) \to 1/d$ when n becomes big. This means that new samples still have effects and old samples are "forgotten" gradually when $n \to \infty$.

2. We use the augmented space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ for clustering, while [111,113] use $\mathcal{Y}$ for labeling. This is desirable since it encourages more elaborated representation using the information of input space. For example, in supervised learning, there are samples whose $\mathbf{x}$ might be dramatically different but their labels y are the same. In this case, using $\mathcal{Y}$ information for clustering will cause those samples indistinguishable and, thus, the derived discriminating subspace $\mathcal{D}$ is singular

and the yielding tree is unbalanced. Furthermore, adding $\mathcal{X}$ information may increase the locality of the sgenerated clusters, which facilitates linear regression model on a local region.

We use the following distance metric for clustering:

$$d_i = w_x \frac{\|\mathbf{x} - \mathbf{c}_i\|}{\sigma_x} + w_y \frac{\|\mathbf{y} - \bar{\mathbf{y}}_i\|}{\sigma_y}, \tag{3.14}$$

where w_x and w_y are two positive weights that sum to 1: $w_x + w_y = 1$; σ_x and σ_y denote incrementally the estimated average lengths of $\mathbf{x}$ and $\mathbf{y}$ vectors, respectively; and $\mathbf{c}_i$ and $\bar{\mathbf{y}}_i$ denote, respectively, the x-center and y-center of the ith cluster of the node c. It is worth noting that the parameters σ_x and σ_y should take account of the dimensionalities of $\mathcal{X}$ and $\mathcal{Y}$. In most of IHDR's application, $\dim(\mathcal{X}) \gg \dim(\mathcal{Y})$ and, hence, the x-part dominates the distance in Eq. (3.14) if we do not weight them properly.

3. In [111,113], the nearest-neighbor model[3] is used in the leaf node. Although the nearest-neighbor's error rate is bounded above by twice the Bayes rate (which is the optimal error rate for a classification problem) [31], Atkeson et al. [9] point out the nearest-neighbor models might overfit linear areas, as shown in Fig. 3.15 (a).

   We propose to use the linear regression in the leaf node instead of the nearest-neighbor model. Let $\mathbf{x}$ denote the query point. Assume IHDR finds the leaf node c whose prototype set $\mathcal{P}_c$ is:

$$\mathcal{P}_c = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_{n_c}, y_{n_c})\}$$

   where the output labels are scalar[4].

[3]The nearest-neighbor regression simply chooses the closest point and uses its output value.
[4]For notation simplicity, we only consider the scalar output.

As in locally weighted learning (LWR) [9], $d(.,.)$ denotes the Euclidean distance and $K(d)$ denotes a distance kernel function (e.g., $K(d) = \exp(-\frac{d^2}{h})$). We compute the following diagnal weight matrix:

$$W = \text{diag}[K(d(\mathbf{x}_1, \mathbf{x}), K(d(\mathbf{x}_2, \mathbf{x}), ..., K(d(\mathbf{x}_{n_c}, \mathbf{x})]$$

with each one corresponding to a prototype in $\mathcal{P}_c$. Let $Y = \begin{bmatrix} y_1 & ... & y_{n_c} \end{bmatrix}^T$ and $X = \begin{bmatrix} \mathbf{x}_1^T & ... & \mathbf{x}_{n_c}^T \end{bmatrix}^T$. We formulate the problem as finding the parameter β such that the regression output is

$$\hat{y} = \beta^T \mathbf{x}$$

and the following cost function is minimized

$$C = (Y - X\beta)^T W (Y - X\beta).$$

The solution of this weighted least square problem [81, Page 386] is

$$\beta = (X^T W X)^{-1} X^T W Y. \tag{3.15}$$

It is important to note that $X^T W X$ is a $d \times d$ matrix, where d denotes the dimension of the vector $\mathbf{x}$. If $\mathcal{X}$ is a high dimensional space, then inverting such a matrix is impossible for a real-time application. However, we can conduct such regression on the discriminating subspace $\mathcal{D}$ (see Appendix A and [123]).

In the following, we experimentally show the augmented IHDR's feature extraction and real-time learning capabilities on an artificial data set. As a first test (2d-cross), we ran IHDR on the noisy training data drawn from a two dimensional function

$$y = \max\{\exp(-10x_1^2), \exp(-50x_2^2), 1.25\exp(-5(x_1^2 + x_2^2)\} + N(0, 0.1^2),$$

as shown in Fig. 3.16 (a). This function is a combination of nonlinear and linear areas and an interesting test of learning and generalization capabilities of a learning algorithm [99]: learning system with simple model find it hard to capture the nonlinearity well, while more complex models easily overfit the linear areas. A second test (100d-cross) added 98 irrelevant noise features to the input, each having a density $N(0, 0.025^2)$. We thus obtain a 100-dimension input space. A third test (200d-cross) added another 100 irrelevant noise features with the density $N(0, 0.05)$ to the input space of the second test. The learning curves with those data set are illustrated in Fig. 3.17 (a). In all three cases, the normalized mean square errors (nMSE) are reported on an independent test set (1681 points on a 41×41 grid in the unit-square in the input space). As the number of training number increasing, all nMSEs converged to the nice function approximation results, namely, $nMSE < 0.03$ after 100,000 training data points. Fig. 3.16 (c) shows the learned surface of the third test after 100,000 training samples presented. For comparison, at the same condition of (c), Fig. 3.16 (b) shows the learned surface using the nearest-neighbor model in leaf node. Fig. 3.17 (b) illustrates the execution time of both training and test processes with respect the number of training samples. It is interesting to note that the execution time increases linearly w.r.t. the number of training samples. The reason is that IHDR's updating and retrieval procedures have the logarithmic complexity and, thus, the average time of adding a training sample and retrieving a testing sample does not change much even though the size of tree has grow tremendously. As considering the third case (200d-cross), execution time on a 200-dimension data set takes only about 500 seconds for 100,000 sample, in other words, the average time for a sample is about 5 milliseconds. Therefore, IHDR is fast, which is extremely important for later real-time robot collision avoidance experiment.

To sum up, the power of feature extraction is due to IHDR's deriving discriminating features automatically. The real-time learning performance is achieved by the logarithmic complexity of IHDR.

3.5.4 Motor mapping

A major goal of motor mapping is to generate a concise representation for stereotypical motor trajectories so that skilled actions that involve many motors can be represented by high-level motor primitives in a lower-dimensional space. As shown in Fig. 3.18, the architecture of the motor mapping is very similar to the spatiotemporal sensory mapping but it works backwards. Motor mapping receives lower-dimensional "feature" space signals (motor primitives) to synthesize higher-dimensional raw motor signals. Motor mapping has many issues similar to those of sensory mapping.

Instead of having attention selection in sensory mapping, motor mapping has a gating system. The gating system plays two roles. The first is the role of gating by which the motor mapping evaluates whether the intended action has sufficient action thrust to pass the gate threshold. The second is the role of subsumption [18] in subsuming the lower-level action by the integrated action, as shown in Fig. 3.19.

The primed action from the cognitive mapping includes the desired control signal for each effector as well as action thrust. The action thrust indicates how consistently the action is issued. The higher the action thrust, the more consistent the action generator is. Therefore, enough thrust must be generated to pass the gate before an action can be issued to the effector.

Subsumption is needed where inconsistent actions for a single effector are issued from multiple sources, and each source has a different priority. One typical use is that the action derived from a higher level (more sensory integration) has a higher priority over the action derived from a lower level (less sensory integration). The subsumption belongs to innate internal behaviors. When the robot becomes more mature, the selection of behaviors from low levels can be overidden by learned (voluntary) internal behaviors.

If a single motor is considered, motor mapping includes only a gating system for each of the single motor as well as the subsumption mechanism for integration from other sensorimotor systems as shown in Fig. 3.19. Through developmental experience,

the motors that are highly correlated enable the growth of a new part of motor mapping, denoted as an attached (top right) block to the basic motor mapping in Fig. 3.9. The new part of the motor mapping plays the corresponding role of the gating system, but it is for correlated multi-motor actions. Further, it not only performs the gating function, but also the reconstruction of higher-dimensional raw motor signals from a lower-dimensional "feature" representation (higher motor primitives).

In [131], an action gating system is implemented on the SAIL robot to decide whether an action is actually triggered.

3.5.5 Innate value system

Given the last context $l(t)$, the IHDR tree finds the best matched context P' which is associated with a set of primed contexts $\{p_1, ..., p_k\}$. How does the controller select the best primed context? A Q value with each primed context is needed. This is represented by a list $Q = \{q_1, q_2, ..., q_k\}$, in parallel with the primed contexts. Thus, each primed context consists of three components, $(x_{pi}(t), a_{pi}(t), q_i)$, $i = 1...k$.

Definition 3.5.1 *The innate value system chooses the action that has the best value* q.

The value system receives a list of primed contexts at each time instant t and occasional environment reward signal $r(t)$, which is from the biased sensors of the robot (e.g., the aversive sensors such as a "bad button" or appetitive sensors such as a "good button"). Each element in the Q vector starts from an initial value, e.g., a zero value. The updating rule that we summarize here is adapted from Q-learning [103], tested in [41]:

$$q(l', n) = \frac{n - 1 - \mu(n)}{n} q(l', n - 1) + \frac{1 + \mu(n)}{n} (r + \gamma V(p')),$$

and

$$V(p') = \max_{1 \le i \le k} q_i,$$

where $q(l', n)$ denotes the matched element in Q and has been updated n times, and l' is the best matched prototype for the current last context $l(t)$. γ is a positive number between 0 and 1, and is used for discounting in time. p' is the primed context found in the next time frame. To keep the temporal order of the latest retrieved primed contexts for real-time local propagation, a primed context update queue (see Fig. 3.9) is needed. The future primed context prototype (including reward) propagates back to the previous prototypes recursively for each time frame in the context update queue. Other prototypes (not in the queue) are not affected and thus are not updated.

3.5.6 Sensorimotor algorithm

The following is a summary of a sensorimotor system with an innate value system:

1. Initialize the mental cycle count to 0.
2. Grab the current input from the sensory mapping to form the last context $l(t) = (x(t), a(t))$.
3. Go through the sensory mapping to reduce the dimensionality of $l(t)$ while applying the internal actions to the attention selection effector of the sensory mapping.
4. Retrieve the reality (cognitive) mapping R using $l(t)$ to find the best matched prototype l' in the matched leaf node of the cognitive mapping. Its output part is a list of primed near contexts $(A_r(t), X_r(t))$.
5. The innate value system selects the near primed context $p_r(t)$ from the lists $(A_r(t), X_r(t))$.
6. Retrieve the priming (cognitive) mapping F from $l(t)$ to produce the primed far contexts $(A_f(t), X_f(t))$.
7. The innate value system selects the far primed context $p_f(t)$ from the lists $(A_f(t), X_f(t))$.

8. Feed the actions in $p_r(t)$ and $p_f(t)$ through the motor mapping. The current internal action determines whether primed near context or primed far context receives attention in the motor mapping. The attended primed context is $p(t)$.

9. Update both the reality and priming mappings using the primed context $p(t)$. If there is an imposed action from the environment (supervised learning), replace the corresponding part of $a_p(t)$ in $p(t) = (a_p(t), x_p(t))$.

10. Spatially update the primed context lists $(A_r(t), X_r(t))$ and $(A_f(t), X_f(t))$ using the Incremental Vector Quantization technique.

11. Temporally update primed contexts in the prototype update queue for the priming mapping F.

12. Push the current primed context $p(t)$ into the prototype update queue from the tail and pop out the oldest primed context.

13. The motor mapping produces internal and external actions.

14. If the mental cycle time has not been used up, sleep for the remaining time so that the exact time actually spent by this cycle is equal to the fixed pre-specified mental cycle.

15. Increase mental cycle count by 1 and go to Step 2.

The SAIL robot has successfully tested those major components of the proposed architecture. For example, a Type-6 architecture (shown in Fig. 3.20) was used in action-chain learning [131]. The result of the experiment is promising and we plan to implement the whole architecture on the Dav robot. Other experiments on SAIL include: (1) vision-guided navigation [115], (2) grounded speech learning [116], (3) communicative learning [131], (4) novelty and reinforcement learning in the value system [41], and (5) sensory mapping development [127].

3.6 Summary

This chapter has outlined the major differences between a non-AMD robot and an AMD robot: An AMD robot's developmental program is task-nonspecific. Further, this chapter introduces eight challenging operational requirements of an AMD robot, which pose a series of challenges for the design of the architecture. The series of architecture types introduced here demonstrates several architecture limitations of the current hand-designed MDP and a possible route toward higher mental architectures. The proposed architectures leave much freedom for different implementations of its major components, e.g., sensory, cognitive, and motor mappings. Although the Dav developmental program that is currently being implemented could be explained in further detail, such details are beyond the scope of this chapter. Dav's developmental architecture has yet to be implemented and tested, but its predecessor, SAIL, has successfully tested the major components of the designed architecture. The presented architecture will be tested in future studies and the performance will be reported.

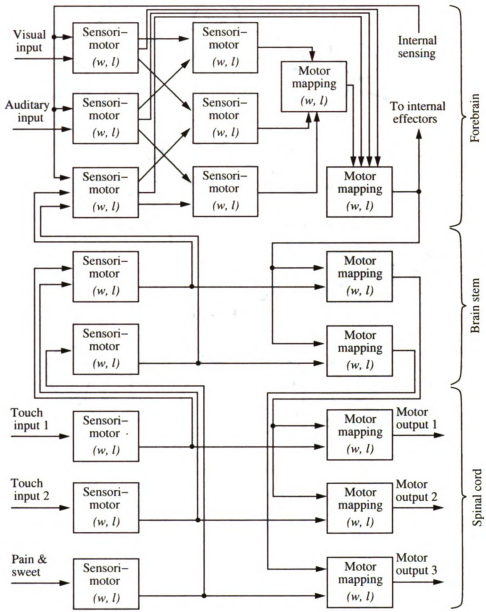


Figure 3.5: An outline view of the Dav architecture. The closed-loop control is realized by the sensors that sense each effector, as well as a set of rich sensors that sense the internal and external environments. w denotes the working memory and l is the long-term memory.

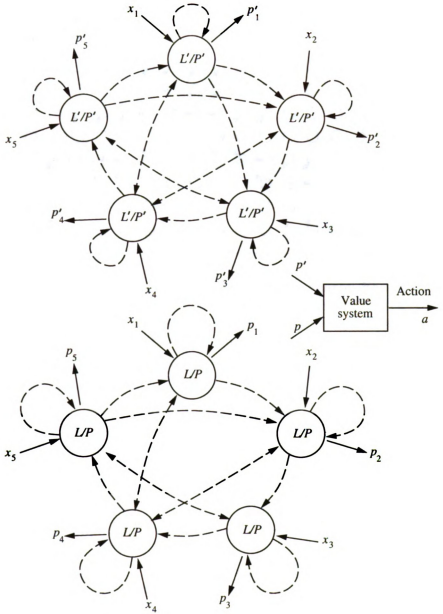


Figure 3.6: An open view of a sensorimotor system. A circle designates a local context state determined by the last context L. An inward solid arrow indicates a sensory input at that time frame (including internal and external sensors) at each time frame. An outward solid arrow indicates an output at that state. A dashed arrow indicates an experienced transition between states. The lower part corresponds to the reality mapping R and the upper part corresponds to the priming mapping F. The two generally do not have the same number of states. The priming mapping is used to predict farther future contexts.

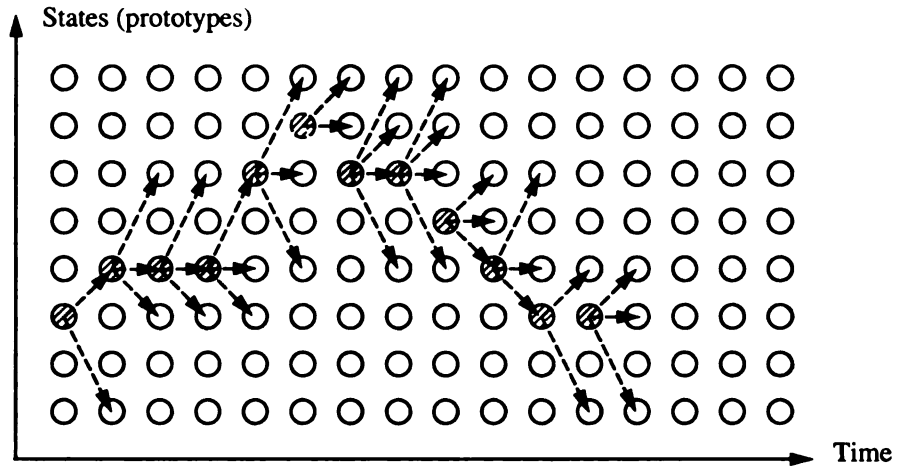


Figure 3.7: A temporal trajectory view of a sensorimotor system. Each cycle represents a context state at a particular time. A shaded cycle corresponds to a visited state. A dashed arrow indicates a priming of a context that corresponds to the cycle that the arrow points to. The priming arrows from a state are updated according to the real experience. An actually visited state is not necessarily primed.

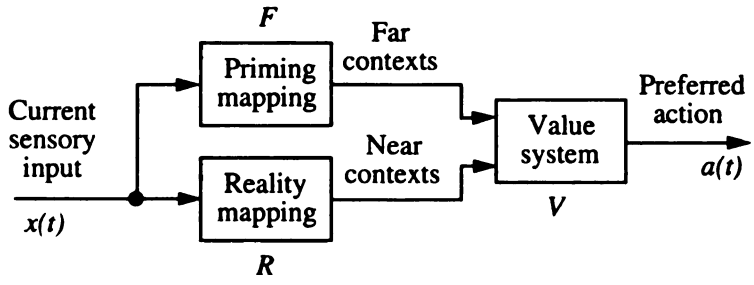


Figure 3.8: A recursive view of a simplified sensorimotor system. Not all components are shown.

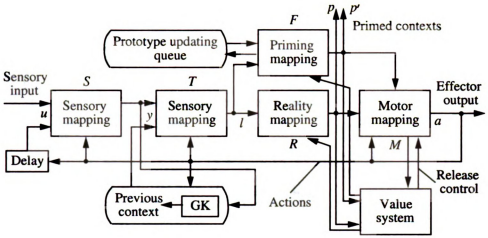


Figure 3.9: A block diagram of the architecture of a sensorimotor system. The lower cognitive mapping realizes the reality mapping and the upper one realizes the priming mapping. GK: gate keeper, an internal effector to actively control the update of the last context.

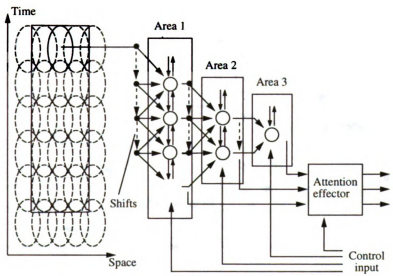


Figure 3.10: The spatiotemporal organization of areas in the sensory mapping. The ellipses represent receptive fields covering both space and time. Areas are organized in a hierarchical way, and the output of an earlier (low order) neural area is used as input to the later (higher order) neural area. Along the pathway of information processing, a neuron in a later area has a larger receptive field than one in an early area. In order to make the correct signal available at the right time for the right input line, we use a time shifting technique. Acting as a time delay unit, the shifter moves each signal to the next line at each time instant.

For $n = 1, 2, \ldots$, do,

1. $u_1(n) = u(n)$.
2. For $i = 1, 2, \ldots, k$, do,
   2.1. If $i = n$, initialize the ith eigenvector,
   $v_i(n-1) = u_i(n)$.
   2.2. If $i <= n$,
   $v_i(n) = \frac{n-1-l}{n} v_i(n-1) + \frac{1+l}{n} u_i(n) u_i^T(n) \frac{v_i(n-1)}{\|v_i(n-1)\|}$,
   $u_{i+1}(n) = u_i(n) - u_i^T(n) \frac{v_i(n)}{\|v_i(n)\|} \frac{v_i(n)}{\|v_i(n)\|}$.
   2.3. If $i > n$, initialize the ith eigenvector,
   $v_i(n) = 0$.

Figure 3.11: CCIPCA algorithm [119]: compute first k dominant **eigenvectors**, $v_1(n), v_2(n), \ldots, v_k(n)$, directly from $u(n)$, where $n = 1, 2, \ldots$.

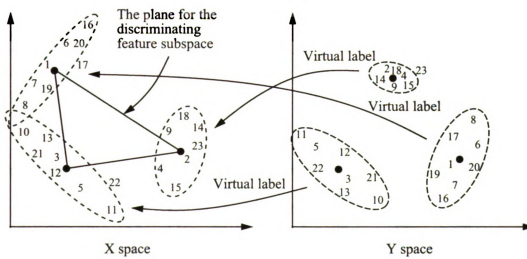


Figure 3.12: Y-clusters in space $\mathcal{Y}$ and the corresponding X-clusters in space $\mathcal{X}$. Each sample is indicated by a number which denotes the order of arrival. The first and the second order statistic are updated for each cluster. The first statistic gives the position of the cluster, while the second statistics gives the size, shape and orientation of each cluster.

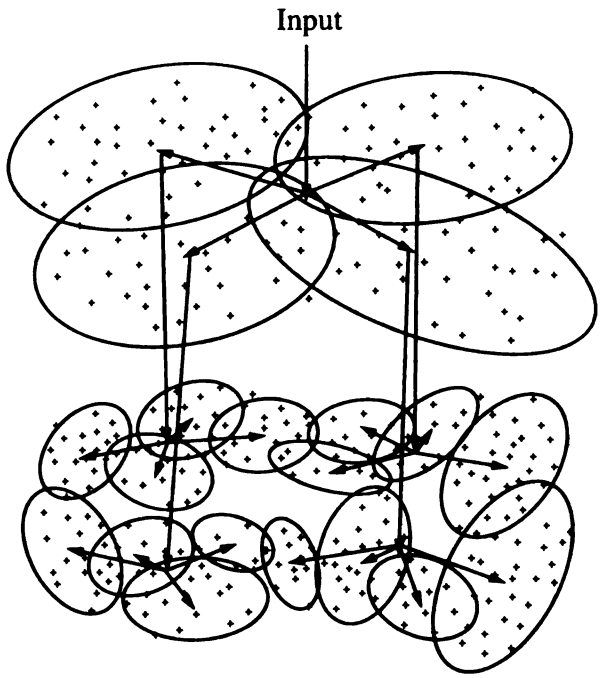


Figure 3.13: The autonomously developed IHDR tree [114,123]. Each node corresponds to a local model and covers a certain region of the input space. The higher level node covers a larger region, and may partition into several smaller regions. Each node has its own Most Discriminating Feature (MDF) subspace $\mathcal{D}$ which decides its child models' activation level.

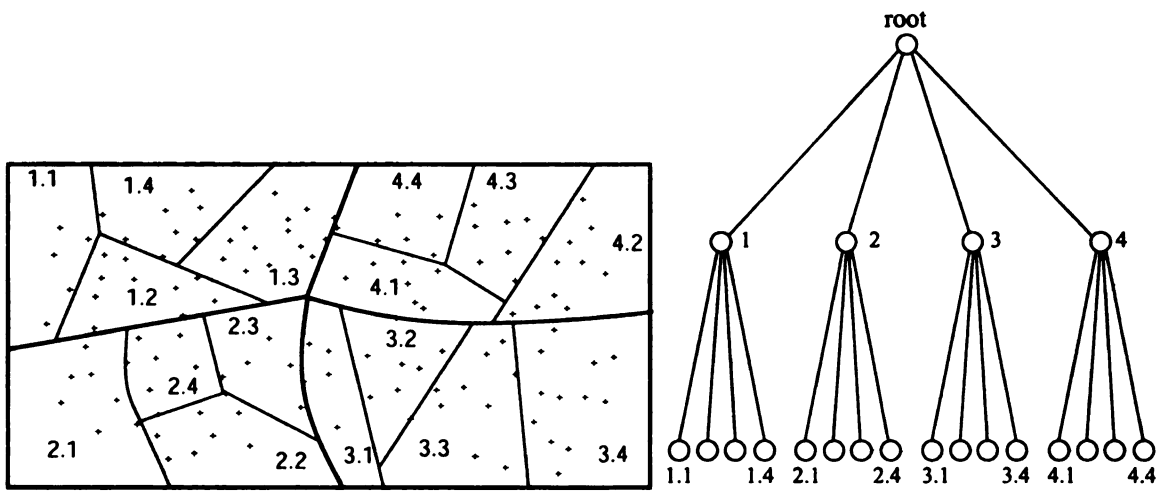


Figure 3.14: Regions in the input space marked $i.j$ where i is the index of the parent region while j is the index of child region. The leaves of the tree represent the finest partition of the space. The decision boundary of the region is of quadratic form.

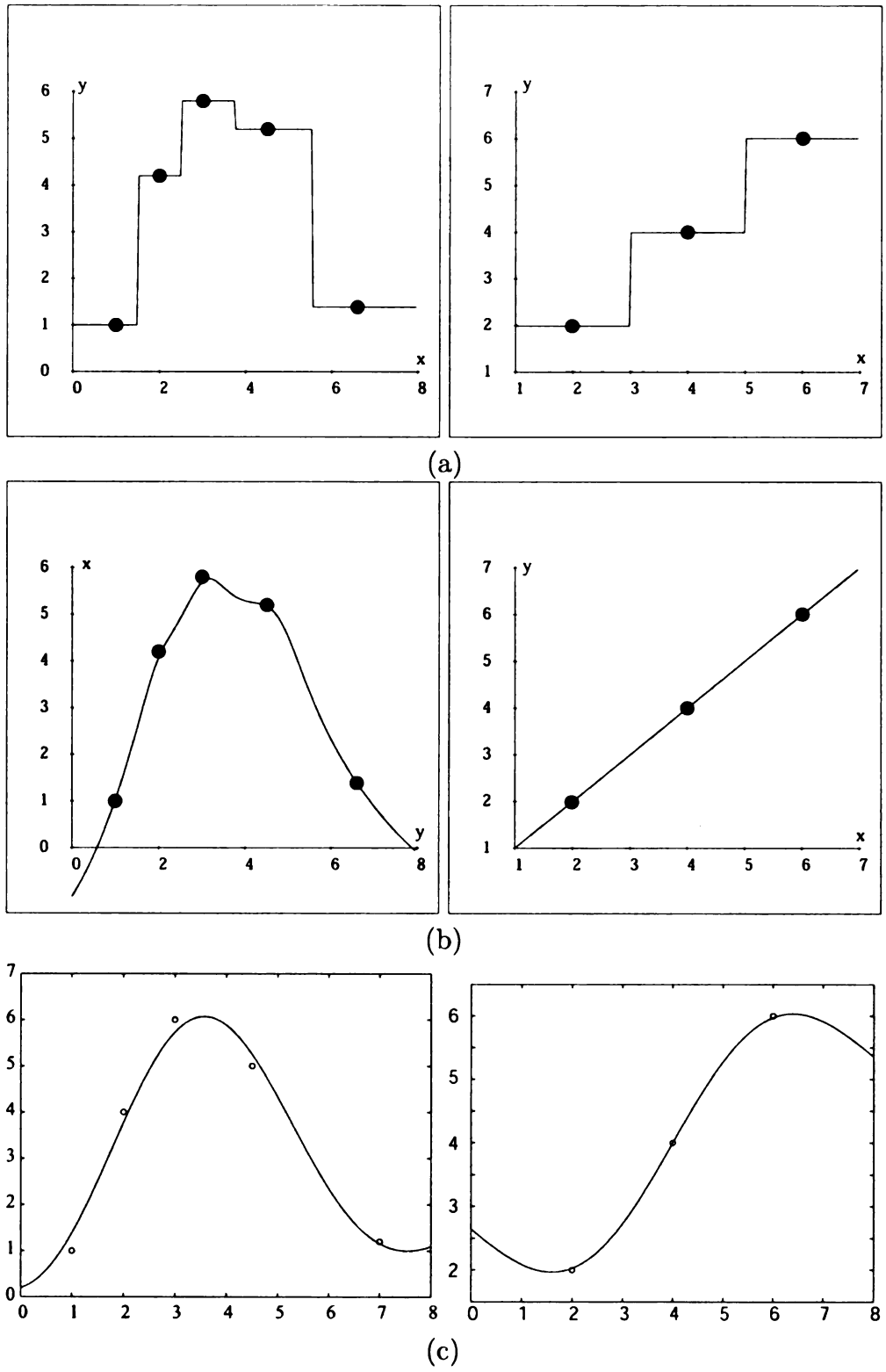
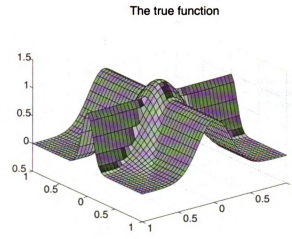
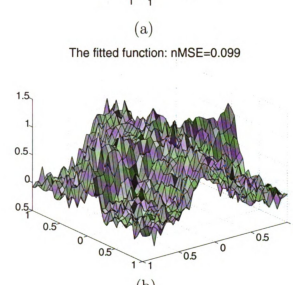


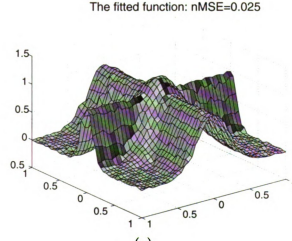
Figure 3.15: (a) Fits using the nearest-neighbor rule. (b) Fits using locally weighted regression. (c) Fits using kernel regression. (a) and (b) are adapted from Atkeson et al. [9].



(a)

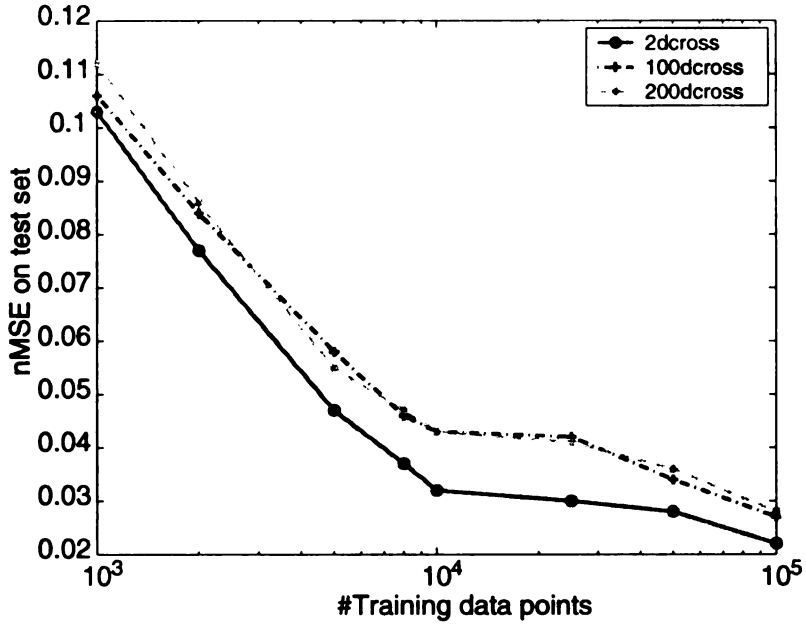


(b)

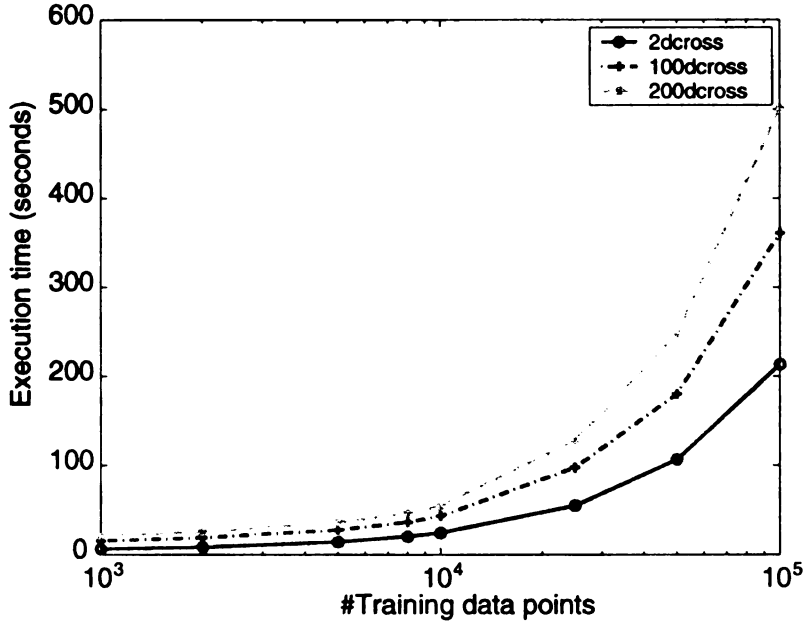


(c)

Figure 3.16: The result of IHDR on an artificial data set.



(a)



(b)

Figure 3.17: (a) The learning curves of the artificial data set. (b) The execution time

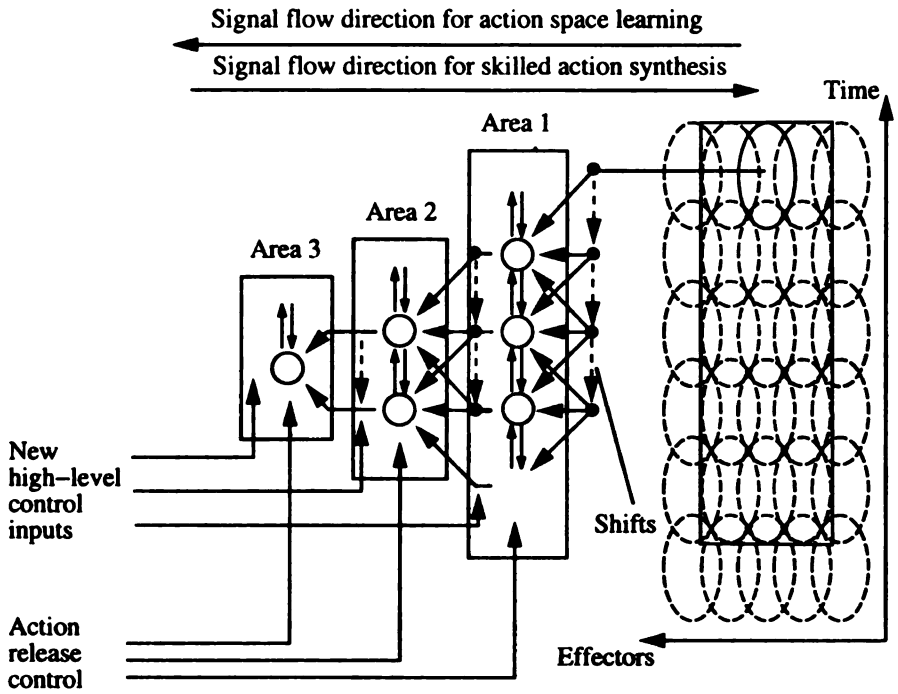


Figure 3.18: The motor mapping as a reverse application of the sensory mapping, but with signal reconstruction.

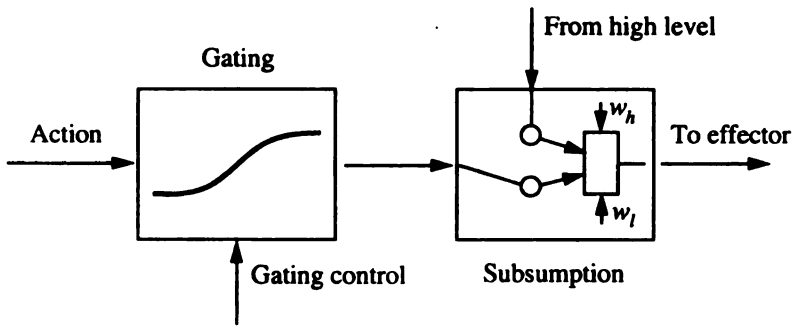


Figure 3.19: An illustration of a simple motor mapping for a single effector. The left is a gating mechanism and the right is a subsumption mechanism.

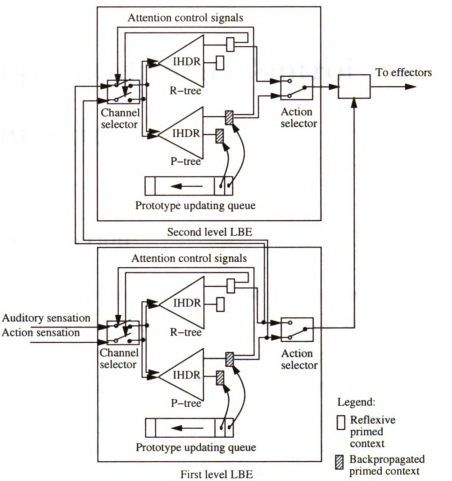


Figure 3.20: A hierarchical developmental architecture (Type-6) for SAIL procedure learning. Each Level Building Element (LBE) corresponds to a sensorimotor system.

Chapter 4

Perception-driven Control Architecture

4.1 Introduction

From Chapter 2, we can see that conventional robotics relies on human supplied equations of dynamics and kinematics to plan trajectories. This approach has been proven to be very useful when those equations and environment representation are available. However, it is shown that this approach has difficulties when the system is extremely complicated and insufficient analytical knowledge is available. For example, humanoid robots have so many degrees of freedom that even for simple actions there are nearly infinite set of joint position sequences that accomplish a single action [98]. Some researchers hence propose to learn actions through practice. Related work in the area includes LWPR on a SARCOS robot by Schaal et al. [97], the simulated model of a 37 degrees of freedom by Mataric et al. [11], and the scheduling degrees of freedom by Grupen et al. [26] motivated the early child motor development.

On the other hand, the traditional approaches usually require a human supplied representation of the environment. Robots designed with those approaches have difficulty in adapting to unknown and changing environments, for example, in sensing

and controlling a robot's arm to pick apples in an orchard. Further, the goal can change at any time (e.g., a larger apple becomes the target before the current one is picked). Some efforts have been made to program behavior-based humanoid robots (e.g., Kismet [14], Cog [3,65]). Recently, progress has been made in realizing robots that can develop their mental skills autonomously through what is called Autonomous Mental Development (AMD), which we have discussed in Chapter 3.

In this chapter, we present a novel developmental perception-driven control architecture (DPDCA), which is adapted from the overall architecture presented in Chapter 3. The novelty of DPDCA includes: (1) No task (goal) is given at programming time. (2) The task-specific representation is generated autonomously from sensor-motor space, not in a 3-D world space. (3) The robot learns while performing. (4) The robot is "alive" while humans interact with it.

In the following, we first outline the proposed framework. we then illustrate the power of DPDCA via Dav's range-based indoor navigation experiments.

4.2 Control Architecture

Machine perception has been proven difficult. Programming perceptual capability using human defined features (e.g., edges, colors, tones) provides a way to produce results in a controlled setting. A resulting fundamental limitation is that the robot does not work well in unknown, partially unknown, or changing complex environments. In this section, a Developmental Perception Driven Control Architecture (DPDCA) will be presented to take up these issues.

Definition 4.2.1 *A Sensory vector is a vector that contains raw data from all sensors. A sensory vector is a stochastic process indexed by discrete time t, denoted by* $x(t)$.

The sensing modalities of a developmental robot typically include vision, audition, 3-D range sensors, joint encoders, tactile sensors and strain gages, plus "brain" internal

sensors such as sensors that sense attention.

Definition 4.2.2 *An action vector at time t is a vector denoted by $a(t)$, which consists of the control signals sent to all effectors at time t.*

Definition 4.2.3 *A context at time t is the sensory information that is needed by the controller to generate a proper action at that time. With the current sensation denoted by a sensory vector (including sensing of action) $g(t) = (x(t), a(t))$, the last context is defined as $l(t) = (g(t), g(t-1), ..., g(t-m+1))$ at time t. Here m denotes the number of time frames in the context.*

The context so-defined includes only the latest m sensory vectors. It is desirable to use a small subset in order to reduce the complexity of learning.

Definition 4.2.4 *The control law of the DPDCA is a decision process R_t: $g'(t) = R_t(l(t))$, illustrated in Figs. 4.1 and 4.2. The primed (recalled) context vector $g'(t)$ consist of two parts: the primed sensory vector $x_p(t)$ and the primed action vector $a_p(t)$. The action vector $a_p(t)$ is the output signal vector sent to the controllers.*

Rather than programmed manually, the decision process (a function) R_t is developed through time. The function R_t depends on the robot's sensorimotor experience, and can be regarded as an internal representation of the environment and its own body.

Definition 4.2.5 *An internal representation R_t is the agent's representation of its own body as well as that of the external environment. It is grown from the agent's continuous interaction with the external environment.*

The representation R_t is an accumulation of the experience up to time t. The R_t is generated based on what we called developmental algorithm, from its own experience indicated by $\{g(\tau) | \tau = 0, ..., t\}$. R_t does not necessarily describe the world accurately, however, it may be improved by new experiences.

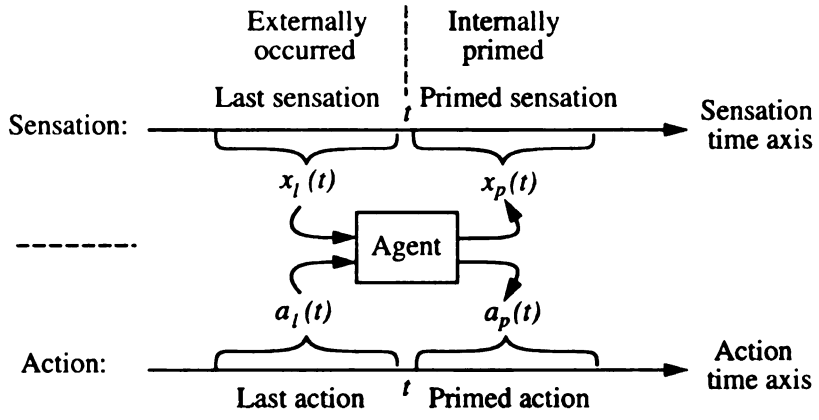


Figure 4.1: The decision process of an agent. The agent tries to use the last context $l(t) = (x_l(t), a_l(t))$ to predict the future contexts: primed sensation $x_p(t)$ and primed action $a_p(t)$.

R_t is not a monolithic representation typical in human designed representation. Instead, it is high dimensional, numeric, and hierarchical. Each experience $g(t)$ must be used to update the representation R_t which is then used for computing the primed action $a_p(t)$ and the primed sensation $x_p(t)$. Afterward, $g(t)$ is discarded. Therefore, the entire experience series from time 0 to $t-1$ is not available at time t. The robot simply does not have space to store all of them. The representation R_t is crucial in order to keep all the necessary information about what the agent has learned so far. The following mappings are needed to learn incrementally: (1) Representation updating from current context $l(t)$ input and the current representation $R_{t-1}(t)$:

$$R_t = \mathcal{R}(R_{t-1}, l(t)), \tag{4.1}$$

(2) Control law:

$$P = R_t(l(t)), \tag{4.2}$$

where the set P is a list of primed contexts $\{g'_i(t) \mid i = 1, ..., k\}$ as an approximation of distribution of primed context. Each element of P consists of three elements, $(x_{p_i}(t), a_{p_i}(t), q_i)$. The action $a_{p_i}(t)$ of the primed context with highest q_i value is

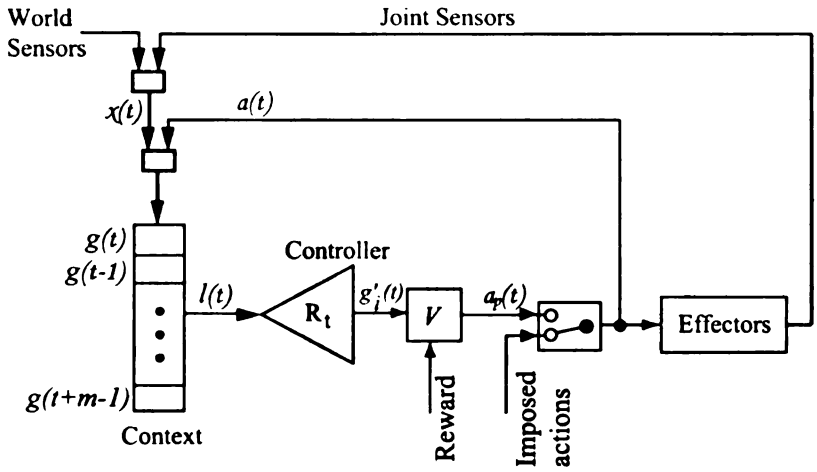


Figure 4.2: The architecture of the robot controller. Note that symbols $a(t)$, $x(t)$ and $l(t)$ denote the action, sensation, and context at time t, respectively. The internal representation R_t is realized by IHDR. The world sensors include range finder and cameras. Block V denotes the innate value system.

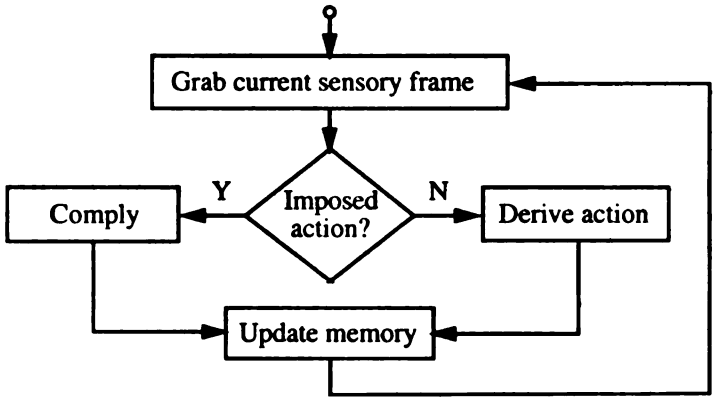


Figure 4.3: Flow chart of learning procedure: the system learning while performing.

chosen as the next output control signal $a(t)$, that is

$$a(t) = a_{p_c}(t), \quad c = \arg \min_{i=1,\ldots,k} q_i. \tag{4.3}$$

The "innate" value system V updates q values using an augmented Q-learning algorithm [42], based on handle external rewards $r(t)$.

A detailed illustration of DPDCA architecture is in Figure 4.2. R_t is implemented by an IHDR tree and $\mathcal{R}$ is its updating function. IHDR learns incrementally by building a hierarchy of automatically derived discriminating feature subspaces. Given each input, it retrieves output in logarithmic time.

As shown in Figure 3.1 the robot trainers are an active part of the environment, they do "robot sitting", training the robot step-by-step, and imposing reward or punishment according the robot's current behavior.

In each learning episode, if the trainer imposes a desired action on an effector, the robot will comply using its force sensor depending on whether the imposed action is consistent with what the robot intend to do (see Fig. 4.3). If everything is fine, proceed with the action and the robot may receive a reward for it. On the other hand, if the robot is not doing what the trainer wants, it will modify its learned internal representation R_t so it follows the trainer's instruction (the imposed action). This is a unified learning procedure, that combines both supervised and reinforcement learnings.

Major differences exist between the DPDCA and a traditional controller. The DPDCA has following characteristics: It develops a controller (software) by learning tasks on the fly, from simple tasks (e.g., moving around according to what it "sees") to more and more complex tasks (e.g., learning sensing based object manipulation). The same developmental program runs continuously for learning all the tasks. Different environments and commands invoke different task execution. Other differences are summarized in Table 4.1.

To conclude, let us remark that, comparing with control diagram in Fig. 2.15 as well as Eq. (2.33), the proposed architecture does not need a desired trajectory to be specified. In fact, the physical environment itself shapes the robot's behavior based on the current task context (if multiple tasks are considered).

4.3 Dav's Range-based Navigation Experiment

4.3.1 Wall Following Behaviors

Our mobile humanoid robot, Dav, as shown in Fig. 2.1, was used to test our proposed the DPDCA framework. In this experiment, we trained the robot to navigate along

Table 4.1: Differences between the DPDCA controller and the traditional controller.

DPDCA controller	Traditional controller
• No task (goal) given at programming time. • No need of 3-D Euclidean space (end-effector frame) representation. • When human interacts with a robot, the robot is "alive" in a sense of learning to associate the current sensory context and the imposed action. • Robots learn their dynamic behavior online. • "Sensing and learning". Generate task-specific representation autonomously "on the fly".	• Tasks (goals) are given (e.g., trajectories). • Inverse kinematic transform is needed since planned trajectories are represented in the end-effector frame. • Dynamic model of the robotic body is needed for tracking the trajectories. • When human overrides machine is dead (in a manual operation mode). • Human programmer supplies the representation.

the loop shown in Fig. 4.4, which is the second floor of Engineering Building at Michigan State University. The floor plane covers an area of approximately 136×116 square meters.

We have three types corners in this test site: 'L', 'T', '+', and 'Z'. Some training range images around those corners and intersections are shown in Fig. 4.5. Those corner types are typical for indoor navigation. The training session was conducted under human control via the graphical user interface (GUI), shown in Fig. 4.6. The imposed action can be calculated from the mouse pointer's position (refer Eq. (5.14) and Fig. 5.7). At each sampling instant, an input range image, previous robot's velocities and an imposed action vector (the desired forward translational speed and angular speed) were fed into the IHDR tree for training if an action was imposed by the trainer. On the other hand, the laser input and the robot's velocities were used to retrieve the action sent to low-level controller if no action was imposed. During

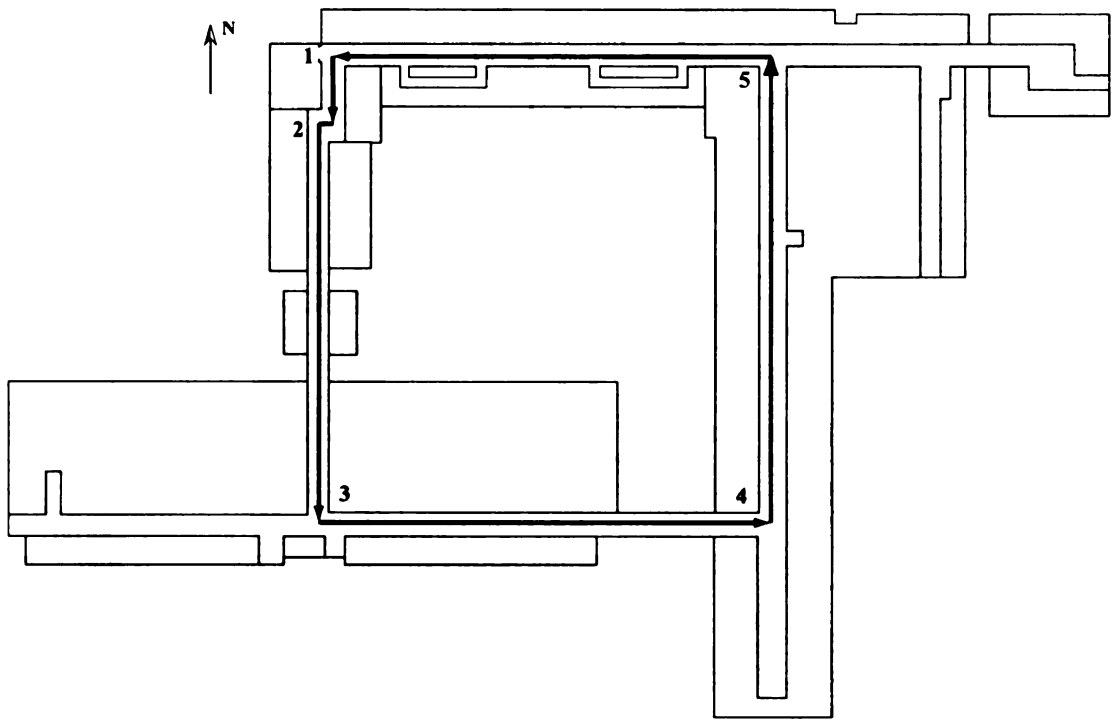


Figure 4.4: A map of the training site at the second floor of the Engineering Building at Michigan State University. The test loop (desired trajectory) is indicated by the thick solid lines. 5 different corners are denoted by bold-faced arabic numbers.

impose action cycle, it is worth noting that if the retrieved range image was similar to the current range image (small Euclidean distance), the old stored action value was replaced with the new imposed one.

First we took Dav for a training run. During the training session, we usually let the robot run on its own (the current version of controller or the IHDR tree). If the robot's action was wrong, we overrode it either with a correct one or gave it a negative feedback (reward); in the mean while, Dav associated the current laser map with the imposed action or attributed the negative reward to the current action. This training procedure resembles how babies learn to walk. When we satisfy with the performance (e.g., small cross-validation error), Dav can be set free.

Dav learned very quickly during the straight section of the corridor, several incidences of imposing action were sufficient for a reasonable performance. More training samples were needed for Dav to turn correctly at the corners. After we trained the robot with 2,215 samples online, Dav successfully finished the desired trajectory (the

test loop in Fig. 4.4). The distribution of the 2215 samples is the following: 215 samples is from the straight sections; the other 2000 samples are from corners, with each 400 samples from a different corners.

Secondly, to illustrate the performance of the navigator, we partitioned the 2215 samples into six bins, with first five bins corresponding to the five corners in Fig.4.4 respectively and the sixth bin corresponding to the samples from straight sections. We performed five tests. In each of these tests, a bin from a corner was left out for testing. In Fig. 4.7, we show the average error histogram of the five tests. The x-axis denotes the Euclidean distance between the true and the predicted outputs, while the y-axis denotes the number of samples. The range of output (v, ω) is $v \in [0, 1]$ and $w \in [-\pi/2, \pi/2]$. The average normalized mean square error is 11.7%.

In the third experiment, we used the data set from the second floor as the training samples. In the test, the data set with 4747 samples (manually labeled) from the third floor was used. Fig. 4.8 shows the error histogram, in which the x-axis denotes the Euclidean distance between the true and the predicted outputs. The normalized mean square error is 8.6%. By inspecting the testing sample with large prediction error, we found that in most cases, the labels of the testing samples were not consistent with the training samples.

In order to provide a sense about how stable the navigating behavior is at a different environment, we designed the fourth experiment. We trained the robot at the second floor while testing it at the third floor. Fig. 4.9 shows how the robot navigates at the third floor. The thick line in Fig. 4.9 denotes the trajectory of the robot. One pass of autonomous navigation in the tested site took about 37 minutes. The robot was observed to continuously run for 3.5 hours before the onboard batteries were low. In several such tests conducted so far, the robot all performed well without hitting the wall and objects on the floor. During these navigation tests, passers-by were allowed to pass naturally. The mobile robot was not confused by these passers-by largerly because the IDHR tree uses automatically derived discriminating features,

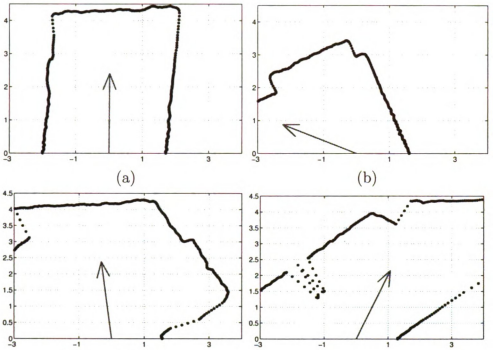


Figure 4.5: A subset of training samples. The red arrows denote the labeled heading for the robot.

which covers the entire scene. In addition, we measured the minimal clearance from the obstacles in those test sessions. In the worse case the clearance is of about 50cm, which happens when the robot was in a narrow corridor.

4.4 Summary and Future Work

The framework presented here does not restrict the scene type and, thus, is potentially applicable to other similar indoor environments. Instead of relying on particular scene features, the proposed controller uses the raw range image and automatically derived features (due to IHDR's feature extracting capability). Since no manually extracted features (e.g., lines and corners) are used in the system, the controller is less restrictive and more robust to the noise of sensors and effectors than the manually designed ones. Finally, let us remark that, although the range-based indoor navigation system is less challenging than the vision-based one, the result of Dav's indoor navigation is still

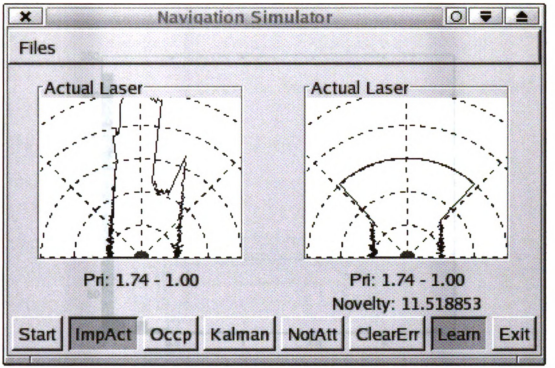


Figure 4.6: The graphical user interface to train the robot online. The left window shows the current laser map in the robot's local frame. The trainer may impose an action via clicking the mouse button in the window. The red arcs illustrate the possible circular path that the robot may have. The right window shows the primed range image (retrieved from the IHDR tree).

very impressive. It shows the power of DPDCA framework.

The future work includes applying this framework to the robot's eye-hand coordination task, which is more challenging because more degrees-of-freedom are used and the high-dimensional vision is involved.

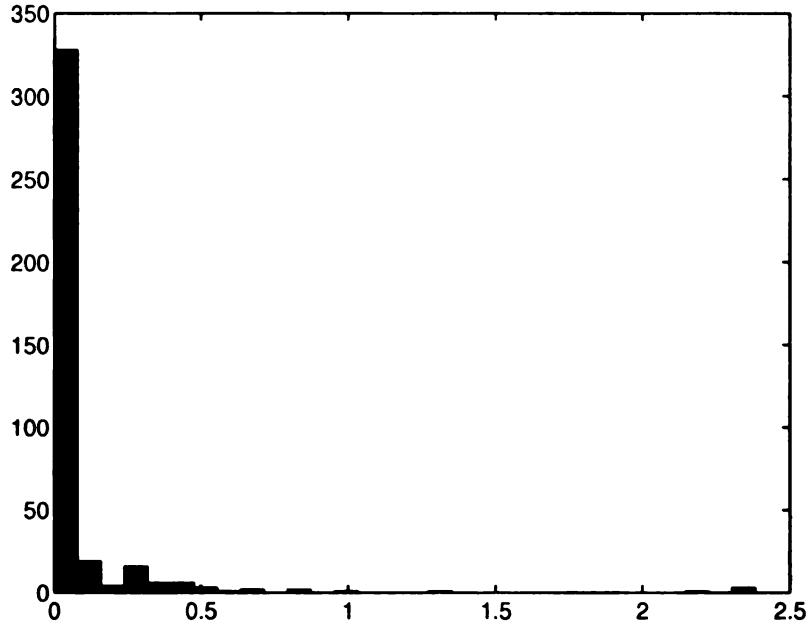


Figure 4.7: The error histogram of the leave-one-out test.

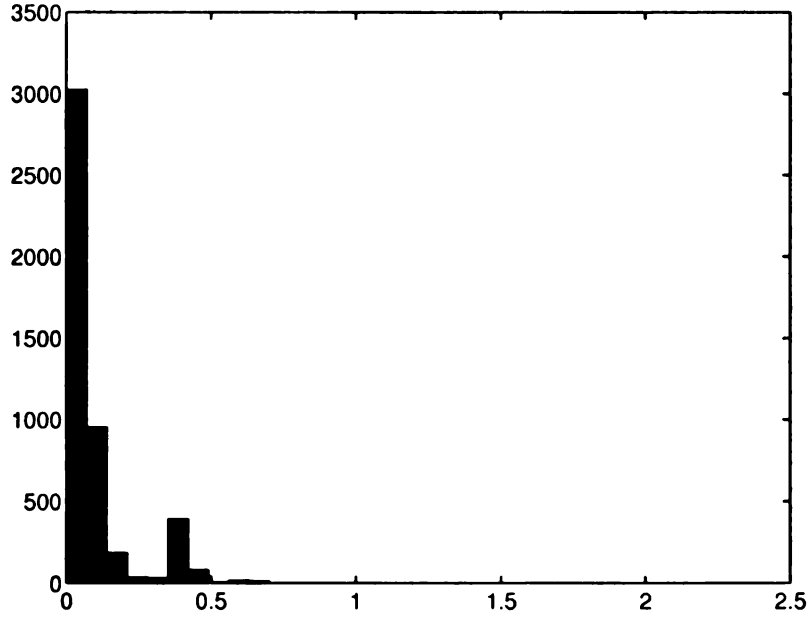


Figure 4.8: The error histogram of the testing data set collected from the third floor.

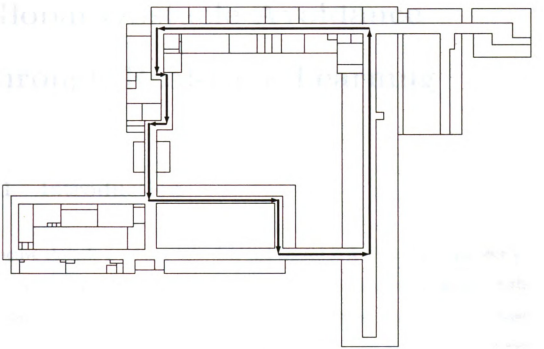


Figure 4.9: A map of the test site at the third floor of the Engineering Building at Michigan State University. The thick solid lines denotes the trajectory along which the robot traveled.

Chapter 5

Global Obstacle Avoidance through Real-time Learning

5.1 Introduction

Reactive obstacle-avoidance behaviors and limited look-ahead searching offer a solution for a mobile robot to navigate in unknown and dynamic environments. Although substantial amount of work on robot's obstacle-avoidance behavior exists, many approaches are still restrictive. A common restriction is that the mapping between the sensory context and desired behaviors is too complex to write a program to simulate accurately.

It seems that real-time learning provides a promising alternative. Rather than using hand-designed behaviors in an off-line fashion, the learning allows robots to explore the environments themselves while improving their performance through practice. During the learning, the robot associates the desired behavior with the current sensory context and plans ahead within a limited step in real time. This architecture design endows robots with great flexibility to cope with unknown or changing environments, which are not easily handled by programmed-in behaviors.

5.2 Related work

The problem of range-based obstacle avoidance has been studied by many researchers. Various reported methods fall into two categories: path planning approaches and local reactive approaches. Path planning approaches are conducted off-line in a known environment. It is shown that path planning for a robot with bounded velocity and arbitrary many obstacles, is intractable (NP-hard) [21]. In Hwang & Ahuja [25,45], an artificial potential field is used to find a nearly optimal collision-free path in a space. Such methods can handle long-term path planning but are computationally expensive for real-time obstacle avoidance in dynamic environment (moving obstacles).

The local reactive approaches are efficient in unknown or partially unknown dynamic environments since they reduce the problem's complexity by computing short-term actions based on current local sensing. In the vector field histogram method (VFH) [13], a 1-D polar histogram representation is constructed to model the environment. The curvature-velocity [84] and dynamic window methods (DW) [34] formulate obstacle avoidance as a constrained optimization problem in a 2-D velocity space. Obstacles and the robot's dynamics are considered by restricting the search space to a set of admissible velocities. Other local approaches include: Nearness diagram (ND) [66] uses high-level description of the environment to generate the motion signals via five laws. Ego-Kinematic space method [67] suggests a nonlinear transform to relieve the problem of the non-holonomic constraint of two-wheel vehicles. In order to deal with local minima, limited look-ahead search strategies are integrated. For example, the enhanced VFH algorithm (VFH* [95]) uses A* [79] and the global DW approach [16] uses NF1 [57].

A major challenge of scene-based behavior generation is the complexity of the scene. Human expert's knowledge has been used to design rules that produce behaviors using pre-specified features. In Lee & Wu [58], a fuzzy logic control approaches is used to incorporate human expert knowledge for realizing obstacle avoidance behaviors. One difficulty of a pure fuzzy approach is to obtain the fuzzy rules and

membership functions. The neuro-fuzzy approach [2, 62, 126] is introduced with the purpose of generating the fuzzy rules and the membership functions automatically. Their training processes are usually conducted using a human supplied training data set (e.g., trajectories) in an off-line fashion. The dimensionality of the input variables (features) is usually less than 10 to be manageable.

In contrast with the above efforts that concentrate on behavior generation without requiring sophisticated perception, a series of research deals with perception-guided behaviors. Studies for perception-guided behaviors have had a long history. Usually human programmers define features (e.g., edges, colors, tones, etc.) or environmental models [15, 43]. An important direction of research, the appearance-based method [23,74], aims at reducing or avoiding those human-defined features for better adaptation of unknown scenes. The need to process high dimensional sensory vector inputs in appearance-based methods brings out a sharp difference between the behavioral modeling and perceptual modeling: the effectors of a robot are known with the former, but the sensory space is extremely complex and unknown with the latter and, therefore, very challenging.

In this chapter, we present an approach of developing an obstacle avoidance behavior by a mobile humanoid through online real-time incremental learning. This learned behavior also integrated the result of look-ahead search seamlessly to solve the "nearsightness" of purely reactive approaches. By limiting look-ahead search to performed within real time, the path planner of this work is of the same spirit of the learnng real-time A* (LRTA*) [54]. The major distinctions of the work include: First, we used the appearance-based approach for range-map learning, rather than an environment-dependent algorithm (e.g., obstacle segmentation and classification) for obstacle avoidance. Second, a novel data fusion technique was used to integrate difference modalities, e.g., laser map, robot's speeds, and planned heading. Third, we extended the search scope of LRTA* to the range of the robot's sensing and Dijkstra algorithm was used. The new appearance-based learning method was able to

distinguish small range map differences that are critical in altering the navigation behavior (e.g., passable and not passable sections). In principle, the appearance-based method is complete in the sense that it is able to learn any complex function that maps from the range-map space to the behavior space. This also implies that the number of training samples that are required to approximate the complex function is very large. To reduce the number of training samples required, we introduced the attentional selective mechanism [124] which dynamically selected regions in near approximity for analysis and treated other regions as negligible for the purpose of local object-avoidance. Further, online training was used so that the trainer could dynamically choose the training scenarios according to the system's current strengths and weakness, further reducing the time and samples of training.

The remainder of this chapter is organized as follows. Section 5.3 presents the proposed approach. The robotic platform and the real-time online learning procedure are described in Section 5.4. The results of simulation and real robot experiments are reported in Section 5.5. Discussions and concluding remarks are given in Section 5.6.

5.3 Approach

The sensitivity to local minima of the reactive approach leads us to divide the problem into two components: obstacle avoidance and path planner. This combined framework can generate motions for mobile robots that accomplish their tasks, while securely navigating in an unknown and dynamic environment (with low speeds).

5.3.1 Obstacle avoidance

We consider the obstacle avoidance behavior as a decision process, which converts the current sensing to action, including the range image and robot's velocities, and the desired heading. At this level, the robot does not sense and store scene configuration (e.g., global map of the environment) nor the global robot position. That is, we

assume that the current sensing and heading contain all of the information that is sufficient for robots to derive the next motor control signal. In fact, it uses the real world as its major representation. The goal of this component is to move safely according to the scene, meanwhile moving toward the goal.

The range scanner observes $\mathbf{r}(t) \in \mathcal{R} \subset R^n$ at time t, where $\mathcal{R}$ denotes the space of all possible range images in a specific environment. $\mathbf{r}(t)$ is a vector of distance, whose ith component r_i denotes the distance to the nearest obstacle at a specific angle. The vector $\mathbf{v}(t)$ gives the current velocities of the vehicle at time t, which are measured by the vehicle's encoders. Let $\theta(t)$ denote the desired heading given by the path planner. The variables $\mathbf{r}(t)$, $\mathbf{v}(t)$, and $\theta(t)$ are given by difference sensors or action whose dimension and scale are different. We thus need a normalization procedure when fusing them together as an integrated context vector.

Definition 5.3.1 *The vector $\boldsymbol{x}(t)$ denotes the system's context of the environment at time t, defined as:*

$$\boldsymbol{x}(t) = (\frac{\boldsymbol{r}(t) - \bar{\boldsymbol{r}}}{w_r}, \frac{\boldsymbol{v}(t) - \bar{\boldsymbol{v}}}{w_v}, \frac{\theta(t) - \bar{\theta}}{w_\theta}) \in \mathcal{X}, \tag{5.1}$$

where w_r, w_v, and w_θ are positive numbers denoting the scatter measurements[1] of the variates $\boldsymbol{r}$, $\boldsymbol{v}$, and θ, respectively, while $\bar{\boldsymbol{r}}$, $\bar{\boldsymbol{v}}$, and $\bar{\theta}$ are their corresponding means.

The action vector $\mathbf{y}(t) \in \mathcal{Y}$ consists of control signals sent to all of the effectors at time t, where $\mathcal{Y}$ denotes the sample space of action vectors.

The obstacle avoidance behavior can be formulated as a mapping $f : \mathcal{X} \mapsto \mathcal{Y}$, i.e., the primed (predicted) action $\mathbf{y}(t+1)$ (the signal sent to the motors) is a function of $\mathbf{x}(t)$:

$$\mathbf{y}(t+1) = f(\mathbf{x}(t)). \tag{5.2}$$

[1] *For simplicity, we assume the covariance matrix (Σ_u) of a variate $\boldsymbol{u}$ is equal to $\sigma^2 I$, where I denotes an identical matrix. Thus, its corresponding scatter is $w_u = \sqrt{tr\Sigma_u} = \sqrt{n}\sigma$, where n denotes the dimension of the variate $\boldsymbol{u}$ (see Marida [64] page 13).*

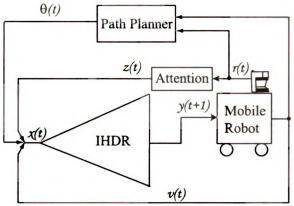


Figure 5.1: The overall architecture of the range-based navigation. "Attention" denotes an attentional module.

Fig. 5.1 shows the coarse architecture of the navigator. An IHDR tree, which we discussed in Section 3.5.3, is generated to estimate the control signal $\mathbf{y}$ from $\mathbf{x}$. The current range image $\mathbf{r}(t)$, the vehicle's velocities $\mathbf{v}(t)$ and the heading $\theta(t)$ from the path planner are used for deriving the next control signal $\mathbf{y}(t+1)$. An attentional module is added to extract partial views from a whole view of a scan.

5.3.2 Attentional mechanism

Direct use of an image as a long vector for statistical feature derivation and learning is called the appearance-based approach in the computer vision community. Usually the appearance-based approach uses monolithic views where the entire range data (or visual image) frame is treated as a single entity. However, the importance of signal components is not uniform. There are cases where appearances of two scenes are quite similar globally, but different actions are required. Further, similar actions are needed where the appearance of two scenes look quite different globally. Both cases indicate that there are critical areas where differences critically determine the action needed. This necessitates an attentional mechanism to select such critical areas.

For example, in Fig. 5.2, (a1), (b1), and (c1) show three scenarios along the robot's path. Range images (a2) and (b2) are quite similar globally judged from the entire image (except the critical area on the left side). In the context of an appearance-based

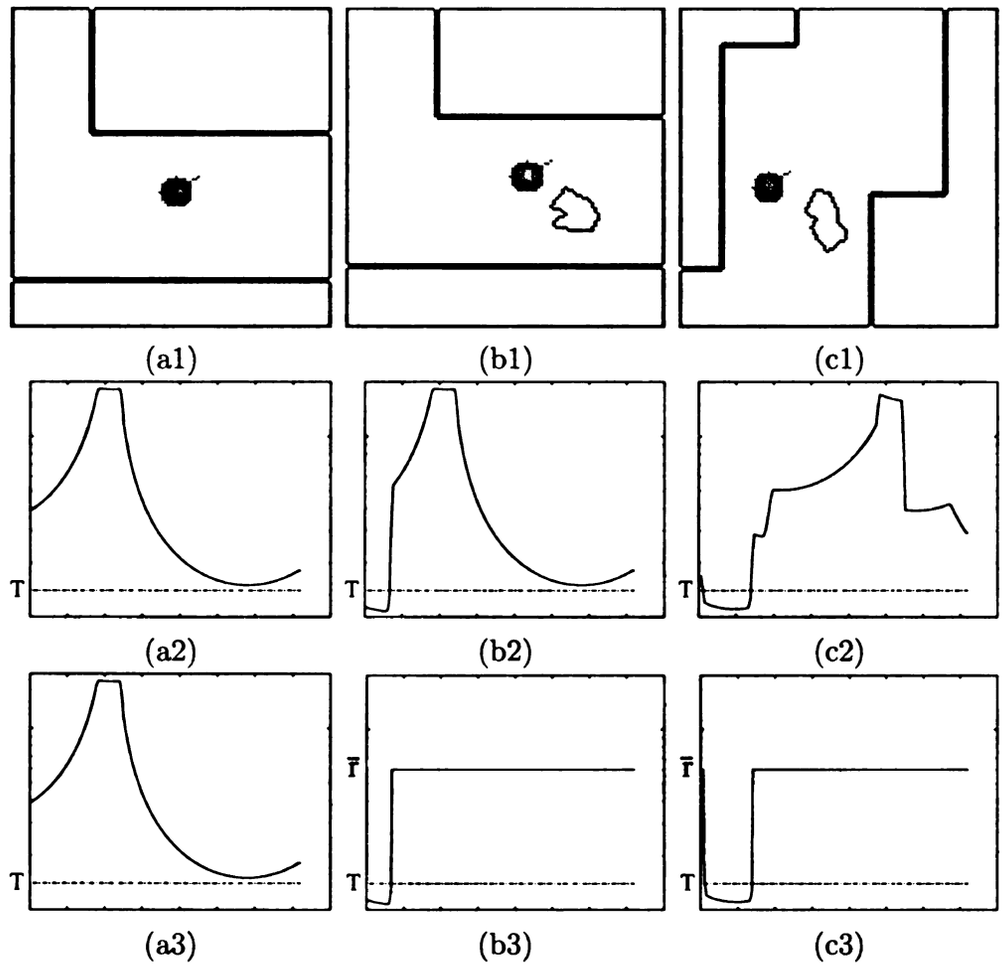


Figure 5.2: Why attention is needed? The solid small circles denote the robot with a short line segment at the front indicating orientation. Thick lines mark the walls along the corridor. T and $\bar{r}$ denote the threshold and the mean, respectively. (a2), (b2) and (c2) are range images taken by the robot at the three scenarios, respectively. (a3), (b3) and (c3) are the corresponding images after passing the attentional module. The diagrams in lower two rows use logarithmic scales for the Y-axis. We can see that the distance between (a3) and (b3) becomes larger while the distance between (b3) and (c3) becomes smaller.

method, this means that the distance (e.g., Euclidean one) between the two is small. They require different actions: turning right for (a1) and going straight or turning left for (b1). In another case, range images (b2) and (c2) are very different globally, but their desired actions are similar: going straight or turning left. Thus, it is difficult to discriminate the three cases correctly by using a distance metric defined on the entire image. But, if we look at the left regions in (a2), (b2), and (c2) of Fig. 5.2, we can see that the similarities and differences are clear. Without a capability to attend to this critical region, the learning system requires significantly more training samples

when complex scenarios are considered.

In the above three cases, the critical area is the input component where range readings are very small. This is true, in general, because near obstacles determine heading more than, and often take precedence over, far-away objects. As we shall see later, this can be accomplished by an attentional module.

We first define the scalar attentional effector.

Definition 5.3.2 *The operation of the attentional effector $a(t)$ for input $r(t)$ and output $z(t)$ is defined by:*

$$z(t) = g(r(t), a(t)) = \begin{cases} r(t) & a(t) = 1, \\ \bar{r} & a(t) = 0, \end{cases} \tag{5.3}$$

where $\bar{r}$ denotes the sample mean of the raw signal $r(t)$.

For intended application, we would like to have $a(t)$ to behave in the following way. First, when all input components have large values, the attentional selection is in its default mode, turning on all components. Second, when there are nearby objects, the attentional selection activates only nearby objects which are critical for object avoidance while far-away objects are replaced by their mean readings. This attentional action can be realized by two programmed functions g and f:

$$z_i(t) = g(r_i(t), a_i(t)) \tag{5.4}$$

and

$$a_i(t) = f(\mathbf{r}(t)) = \begin{cases} 1 & \text{if } r_i < T \text{ or } \forall j \;\; r_j(t) \geq T, \\ 0 & \text{otherwise,} \end{cases} \tag{5.5}$$

where T is a threshold, $i = 1, 2, ..., n$, and $\mathbf{r}(t) = (r_1(t), r_2(t), ..., r_n(t))$ denotes the input vector. We write $\mathbf{z}(t) = (z_1(t), z_2(t), ..., z_n(t))$ as the output of the attentional action and $\mathbf{a}(t) = (a_1(t), a_2(t), ..., a_n(t))$ as the attention vector. The above function f will suppress some far-away components ($a_i(t) = 0$) if there are objects closer than

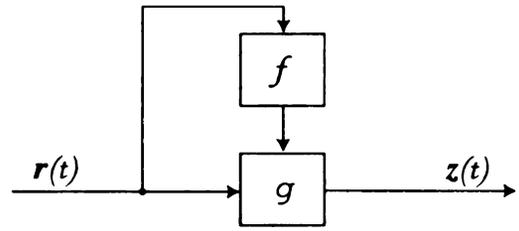


Figure 5.3: The attentional signal generator f and attentional executor g. $\mathbf{r}(t)$ and $\mathbf{z}(t)$ denote the input and output, respectively.

T. If all readings are far-away, we do not want to turn the attention off completely and, therefore, we leave all attentional effectors on ($\forall j,\ a_j(t) = 1$). This operation is illustrated in Fig. 5.3.

In practice, this raw attentional vector $\mathbf{a}(t)$ is smoothed by convoluting with a flat window, as

$$a'_i(t) = [\frac{1}{11} \sum_{j=i-5}^{i+5} a_j(t)],$$

where $[\cdot]$ denotes rounding to the nearest integer. This smoothing serves to eliminate point-wise noise and to provide a neighborhood influence to the output attentional vector.

In Fig. 5.2, the readings of the left part of diagrams (b2) and (c2) are smaller than T. Thus, only the left part passes through the attentional module without change whereas other parts are suppressed by being set to the mean, as shown in (b3) and (c3) of Fig. 5.2. This is needed for the robot to pass through tight areas, where a small change in the width of a gap determines whether the robot can pass. The attentional mechanism enables the robot to focus on critical areas (i.e., parts with close range) and, thus, the learned behaviors sensitively depend on the attended parts of the range map. All range readings are attended when there is no nearby object as shown by (a2) and (a3) of Fig. 5.2.

In Fig. 5.1, the learner IHDR is a hierarchically organized high-dimensional regression algorithm. In order to develop stable collision-avoidance behaviors, the robot needs sufficient training samples. Here, we show that the attentional mechanism

greatly reduces the number of necessary training samples when there are objects close to the robot. To quantitatively analyze the attentional mechanism proposed, we make the following definition.

Definition 5.3.3 *Consider a scanner operating in an environment, which can be approximated by piecewise 3-D planes for simplicity. Each range map* $\mathbf{r}$ *can be approximated by a polygon with* h *segments (as in Fig. 5.4 (a)).* $P = \{p_1, p_2, ..., p_h\}$ *is the map, where the* i*th end point* p_i *is denoted by its polar coordinate* (r_i, α_i)*. Without lost of generality, the angle coordinate are sorted:* $\alpha_1 < \alpha_2 < \cdots < \alpha_h$*.* $P' = \{p'_1, p'_2, ..., p'_h\}$ *is the post-attentional map, where* $p'_i = (z_i, \alpha_i)$ *whose range* z_i *has been defined earlier.*

Remark. The larger h is, the closer the approximation of $\mathbf{r}$ in general. In a particular case, the polygon representation becomes a regular grid when $h = n$.

We write the post-attentional approximation P' as the function of P, i.e.,

$$P' = g^*(P). \tag{5.6}$$

The attentional mechanism, defined in Eq. (5.6), is not a one-to-one mapping, as shown in Fig. 5.4. The post-attentional map P' is the representative for a set of pre-attentional maps besides P if condition C: $\exists p_j \; p_j \in P \wedge l_j < T$ is satisfied. We denote this set by $\mathcal{R}(P')$, i.e. $\mathcal{R}(P') \equiv \{P | g^*(P) = P'\}$. The following theorem gives a lower bound of the average size of $\mathcal{R}(P')$ when there are objects within the distance T from the robot.

Theorem 1 *Let* Δ *and* r_m *denote, respectively, the range resolution and maximum distance of each radial line. If the* i*th end point* p_i*'s radial length* r_i *is a random variable with a uniform distribution in the sample space* $\{0, \Delta, 2\Delta, ..., r_m\Delta\}$*. Then the average size of the set* $\mathcal{R}(P')$ *conditioned on* C *is:*

$$E_{P'}\{\mathcal{R}(P')|C\} > \frac{(1-p)hq^{h-1}}{1-p^h}, \tag{5.7}$$

where $q = (r_m - T)/\Delta$, $p = (r_m - T)/r_m$, *and* $E_{P'}\{\cdot|C\}$ *denotes the expectation on condition* C.

Proof Consider the cases where there are k $(1 \leq k \leq h)$ end points located within the half-circle T (see Fig. 5.4). The number of possible configurations for the $h - k$ end points outside the circle, denoted by s_k, is:

$$s_k = q^{h-k}. \tag{5.8}$$

Because the radial distance of the $h - k$ end points have the freedom to choose values from the interval $[T, r_m]$, which has q discrete values. By definition:

$$E_{P'}\{\mathcal{R}(P')|C\} = \sum_{k=1}^{h} s_k P(k|C),$$

where $P(k|C)$ denotes the conditional probability when k end points are located within the half-circle T. We can see

$$P(k|C) = C_h^k (1-p)^k/(1-p^h).$$

Therefore,

$$E_{P'}\{\mathcal{R}(P')\} = \sum_{k=1}^{h} q^{h-k} \frac{C_h^k (1-p)^k}{1-p^h}$$

$$= \frac{\sum_{k=0}^{h} C_h^k q^{h-k}(1-p)^k - q^h}{1-p^h}$$

$$= \frac{(q+(1-p))^h - q^h}{1-p^h} > \frac{(1-p)hq^{h-1}}{1-p^h}.$$

In the last step, the inequality, $(x+\delta)^n - x^n > nx^{n-1}\delta$ if $0 < \delta << x$, is used. □

Table 5.3.2 shows the lower bound of the size due to attention at two typical parametric settings. We see that the reduction is large. Of course the size of remaining space to learn is also large. The ratio of space reduced over original space is roughly p.

Table 5.1: The lower bound of the reduction ratio at several parametric setting

Parameters	Reduced size
$r_m = 50\text{m}$, $\Delta = 0.2\text{m}$, $T = 2\text{m}$, and $h = 10$	9.51×10^{20}
$r_m = 10\text{m}$, $\Delta = 0.2\text{m}$, $T = 2\text{m}$, and $h = 15$	7.52×10^{22}

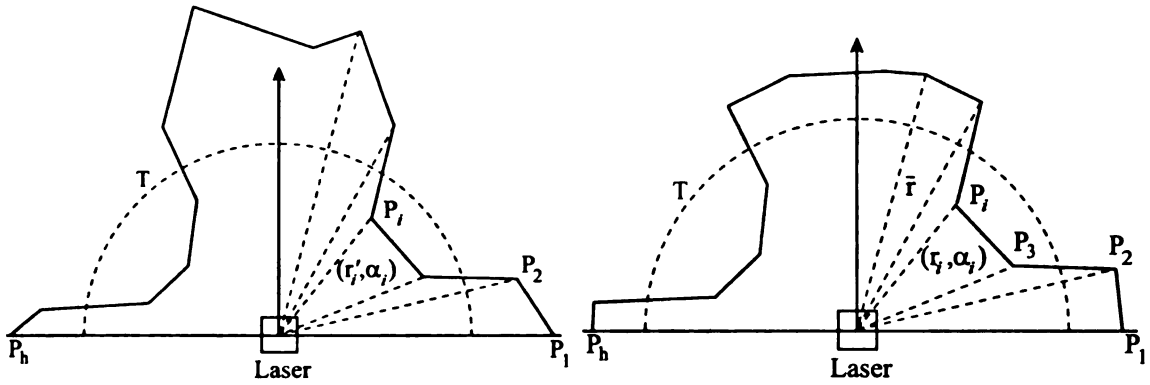


Figure 5.4: (a) shows that the pre-attentional map $\mathbf{r}$ is approximated by a polygon P. P is specified by h end points: $p_1, p_2, ..., p_h$. The kth end point is denoted by (l_k, α_k). (b) shows the post-attentional approximation P', after the attentional function g^*. Those end-points whose distances are larger than T are set to $\bar{r}$. The half-circle with symbol T shows the area the robot pays attention to. We can see numerous pre-attentional approximations map to a single post-attentional approximation P'. It is clear that the points outside the half-circle of (a) have the freedom to change positions without affecting the shape of (b).

5.3.3 Path planner

The path planner gives the desired heading θ required by obstacle-avoidance, based on the robot's estimated pose and the occupancy model of the environment. We divide the path planning into two steps: First, a global search algorithm is applied on the occupancy grid model of the terrain. Second, to account for unknown and dynamic parts of the environment, a partial search algorithm is used within the scope of the robot's current sensing.

How to estimate the robot's pose is not the scope of this chapter, we assume the robot is given a method for calculating its coordinates in the occupancy grid via its sensing. For example, GPS receiver gives the robot's location within a digital map as the robot navigating outdoor; for indoor environment, Kalman filter (KF) estimates

the robot pose based on the positions of the observed beacons and odometry.

More formally, let $G = (V, C)$ be a weighted grid graph. V is a set of cells while C is a cost function defined on the cells. In the grid graph, each cell s, except at the boundary, has eight direct neighbors, denoted by Neighbor(s). The cost of an cell $x \in V$ is given by $c(x)$, representing the likelihood of the cell being occupied by obstacles. For simplicity's sake we assume $0 \leq c(x) \leq 1$ (e.g., zero means the cell is not occupied while one means the cell is occupied). We define a collection of paths through G, from the source s to the destination d, as

$$\pi(G; s, d) = \{(s_0, s_1, ..., s_N) \mid s_{n+1} \in \text{Neighbor}(s_n), s_0 = s, s_N = d, n = 0, ..., N-1\}.$$

The cost of a path is then defined by

$$c(s_0, s_1, ..., s_N) = \sum_{n=0}^{N} c(s_n) \tag{5.9}$$

and we seek the minimal cost path through the grid graph. It is well known the path with minimal cost can be found via dynamic programming (DP) by recursively calculating $k(x)$ and $b(x)$, where $k(x)$ denotes the minimum-cost path from the x to d cell and $b(x)$ the next cell from r to the goal cell along the minimal cost path. $k(x)$ and $b(x)$ are defined respectively as:

$$k(x) = \begin{cases} 0 & x = d \\ \min_{y \in \text{Neighbor}(x)} \{k(y) + c(x)\} & \text{otherwise,} \end{cases}$$

and

$$b(x) = \text{argmin}_{y \in \text{Neighbor}(x)} \{k(y) + c(x)\}.$$

The path with minimal cost is then $(s_0^*, s_1^*, ..., s_N^*)$, where $s_0^* = s$ and $s_n^* = b(s_{n-1}^*)$,

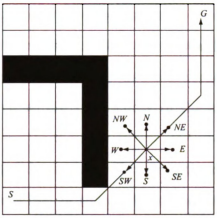


Figure 5.5: The occupancy grid of an environment. Black cells denote to occupied cells, while others are free ones. Each cell has eight directly connected neighbors: N, NE, E, SE, S, SW, W and NW, each of which corresponds to an angle: heading θ. The optimal path from S to G can be encoded by the heading sequence: (E, E, E, NE, NE, NE, NE, N, N, N).

for $n = 1, 2, ..., N$. The minimal cost from the source to destination is

$$c(s, d) = c(s_0^*, s_1^*, ..., s_N^*) = k(s).$$

Since the cost function $c(x)$ is positive, Dijkstra algorithm [28] can be used to search the optimal path. If the priority queue is used, the complexity of the algorithm is $O(|V| \log |V|)$ [24].

Now we can define the planned heading θ based on the function $b(x)$, as illustrated in Fig. 5.5. The heading $\theta(x)$ of the cell x has eight possible directions: N (90°), NE (45°), E (0°), SE (−45°), S (−90°), SW (−135°), W (180°), and NW (135°). We define the head $\theta(x)$ as the direction toward the next cell along the optimal path, e.g., the heading of the cell x in Fig. 5.5 is NE.

The above-mentioned DP procedure computes the heading θ for the entire occupancy grid. This is motivated by the fact that the same heading can be reused for every location of the robot if the environment does not change. To account for the discrepancy of the occupancy grid from the actual environment, re-planning is needed. For example, when the robot senses an obstacle blocking the planned path, the path

is replanned from the robot's current pose. It is not necessary and not desirable to recompute the heading for the entire grid again. Instead, the heading is computed within the scope of robot's current sensing. In other words, this local-search strategy limits the look-ahead search within the vicinity of the area enveloping the robot's current laser map.

More specifically, let $L(x)$ denote the cells within the range of the robot's sensing from the pose x, which is a subset of the whole occupancy grid S. $B(x)$ denotes the boundary of $L(x)$ but not belonging to $L(x)$, i.e.,

$$B(x) = \{z \mid z \in S - L(x), \exists y \ (z \in \text{Neighbor(y)})\}.$$

Now we are ready to specify the local search scheme in Algorithm 1. Providing the cost of a cell is positive, Eqs. (5.10) and (5.11) can be calculated by Dijkstra algorithm and the complexity is $O(|L| \log(|L|)$. By carefully choosing the size of L, Algorithm 1 can be realized on a real-time mobile robot.

Algorithm 1 Local search algorithm within L

1: **if** the planned path is obstructed by an obstacle **then**
2: Let x_{cur} be the robot's current pose.
3: For each cell $x \in L(x_{\text{cur}})$, set $k(x)$ to its initial value $k_0(x)$ (e.g., non-informative zero).
4: **for** all each cell $x \in L(x_{\text{cur}})$ **do**
5: Look ahead and value update: Update the heuristic estimate $k(x)$ by:

$$k(x) = \max\{k(x), \min_{y \in B(x_{\text{cur}})}[c(x,y) + k(y)]\}, \tag{5.10}$$

where $c(x,y)$ denotes the minimal cost of path enveloped within L from the cell x to y.
6: **end for**
7: Choose a path to a boundary cell $y \in B(x_{\text{cur}})$ such that

$$y = \text{argmin}_{y \in B(x_{\text{cur}})}[c(x_{\text{cur}}, y) + k(y)] \tag{5.11}$$

8: Return the head θ along the found path from x_{cur} to y.
9: **end if**

Not being an exhaustive search strategy as NF1 [57], oscillation re-plannings occur

rarely in badly conditioned obstacle configuration but never endanger the accomplishment of the task. More specifically, like LRTA* [54], the proposed re-planning scheme never fails to reach a goal under assumption the the k-values of all cells outside space L do not overestimate the true values. Our scheme is better than the LRTA* in the oscillation issue by employing look-ahead within the envelope of current laser map, while LRTA* has an one-step look-ahead strategy. We obtained reasonable suboptimal solutions for almost all cases faced by mobile robots, for example, this strategy can deal with all kinds of U-shaped obstacles as long as they are detectable within the robot's local sensing scope.

5.3.4 Cost function

The cost function of the paths $\pi(G; s, d)$ in Eq. (5.9) can easily modified to incorporate the length and curvature of path. It is desirable to obtain a short and less wiggly path. The modified cost function is then defined as:

$$c'(s_0, ..., s_N) = \alpha \sum_{n=0}^{N-1} c(s_n) + \beta \sum_{n=0}^{N-1} \mathrm{dist}(s_n, s_{n+1}) + \gamma \sum_{n=1}^{N-1} \mid \theta_{n-1} + \theta_{n+1} - 2\theta_n \mid, \quad (5.12)$$

where $\mathrm{dist}(x, y)$ denotes the Euclidean distance from the cell x to y; α, β, and γ are weighting parameters controlling the influence of each factors. The cost defined in Eq. 5.12 increases with the length and wiggly nature of a path. This makes the path search algorithm preferring short and straight (less wiggly) paths.

In addition, as in NF1, the cost function Eq. 5.12 produces trajectories grazing obstacles (e.g., see (b) and (c) of Fig. 5.6). The reason is that there is no mechanism enforcing the result path maintaining a minimum clearance from the obstacle. We hence add another term $\sum_{n=0}^{N-1} \mathrm{V}(s_n)$ into Eq. 5.12, where

$$\mathrm{V}(x) = \sum_{\mathrm{dist}(y,x) \leq T} c(y).$$

The term V(x) is designed in the same fashion as repulsive field. It sums up all x's occupied neighbor cells within the distance T. The more occupied cell nearby, the larger $V(x)$ is. For example, if x has a clearance distance larger than T for all occupied cells, then V(x) = 0.

Thus, the new cost function is:

$$c''(s_0, ..., s_N) = c'(s_0, ..., s_N) + \delta \sum_{n=0}^{N-1} \mathrm{V}(s_n) \tag{5.13}$$

Fig. 5.6 shows the result of an evaluation for our proposed planning scheme in a simulated environment. The size of the grid is 30m by 25m; the resolution is 10cm and the robot's sensing scope is not more than 15m. In Fig. 5.6, (a), (b), (c), and (d) illustrate the planned paths under different cost functions. (a) uses the cost defined in Eq. (5.9) without regularization terms. Because the cost of the free cells is zero, the total cost of path is zero despite the wiggly nature of the path. (b) shows the planned path after adding the Euclidean term (the second term in Eq. (5.12)). The result demonstrates the straightness of path is improved comparing that of (a), but there is a noticeable portion of path grazing the wall and some undesirable turns near the goal. Adding the curvature term (the third term in Eq. (5.12)), both the clearance and straightness of the path are improved, as shown in (c). However, the clearance of (c) is not satisfiable since some portions of path still touch the occupied cells. (d) shows the yielding path by introducing the repulsive term defined in Eq. (5.13). The repulsive force pushes the result path away from the occupied cells, hence the path in (d) is significantly better than those of the other three.

The third row of Fig. 5.6 shows a situation where sensory information indicates that the original planned path is obstructed by an unmodeled U-shaped obstacle. The heading is needed to recomputed within the shaded area, indicated in Fig. 5.6 (e). This shaded area is the envelope of the laser range map in the occupancy grid. After the new path is planned, the robot moves along the new path while checking

whether new obstacles block the planned path. (f) illustrates the final actual path of the robot and the updated occupancy grid.

5.4 The robotic system and online training procedure

The tests were performed in a humanoid robot, called Dav [37, 125], as shown in Fig. 2.1. This humanoid is built in the Embodied Intelligence Laboratory at Michigan State University.

Mounted on the front of Dav, the laser scanner (SICK PLS) has 180° of view and 0.5° of resolution. The local vehicle coordinate system and control variables are depicted in Fig. 5.7. During training, the control variables (r, d) are given interactively by the position of the mouse pointer P through a graphical user interface. Once the trainer clicks the mouse button, the following equations are used to compute the imposed (taught) action $\mathbf{y} = (v_y, \omega)$:

$$\begin{aligned} v_y &= Kd \\ \omega &= v/r \end{aligned} \tag{5.14}$$

where K is a positive constant and r denotes the radius of the arc through the center of robot and the mouse pointer. The mouse pointer hence controls the translation and angular speeds simultaneously, which facilitates the online training procedure.

Dav's drive-base has four wheels, each driven by two DC motors. Let $\dot{\mathbf{q}}$ denote the velocity readings of the encoders of four wheels. Suppose v_x and v_y denote the base's translation velocities, and ω denotes the angular velocity of the base. By assuming that the wheels do not slip, the kinematics of the base is: [125]:

$$\dot{\mathbf{q}} = B(v_x, v_y, \omega)^T, \tag{5.15}$$

where B is an 8×3 matrix, decided by the wheels' configuration (known). The base velocities $(v_x, v_y, \omega)^T$ is not directly available to learning. It can be estimated from the wheels' speed vector $\dot{\mathbf{q}}$ in a least-square-error sense:

$$\mathbf{v} = (v_x, v_y, \omega)^T = (B^T B)^{-1} B^T \dot{\mathbf{q}}. \tag{5.16}$$

We use two velocities, (v_y, ω), as the control vector $\mathbf{y}$. Thus, the IHDR tree learns the following mapping incrementally:

$$\mathbf{y}(t+1) = f(\mathbf{z}(t), \mathbf{v}(t)).$$

During the interactive learning, $\mathbf{y}$ is given. Whenever $\mathbf{y}$ is not given, IHDR approximates $\mathbf{y}$ while it performs (testing). At the low level, the controller servoes $\dot{\mathbf{q}}$ based on $\mathbf{y}$.

5.4.1 Online incremental training

The learning algorithm is a simplified version of sensorimotor algorithm presented in Section 3.5.6 and is outlined as follows:

1. At time frame t, grab a new laser map $\mathbf{r}(t)$, the wheels' velocity $\dot{\mathbf{q}}(t)$ and heading $\theta(t)$. Use Eq. (5.16) to calculate the base's velocity $\mathbf{v}(t)$.

2. Computer $\mathbf{a}(t)$ based on $\mathbf{r}(t)$ using Eq. (5.5). Apply attention $\mathbf{a}(t)$ to given $\mathbf{z}(t)$ using Eq. (5.4). Merge $\mathbf{z}(t)$, $\mathbf{v}(t)$, and $\theta(t)$ into a single vector $\mathbf{x}(t)$ using Eq. (5.1).

3. If the mouse button is clicked (training), Eq. (5.14) is used to calculate the imposed action $\mathbf{y}(t)$, then go to step 4. Otherwise go to step 6.

4. Use input-output pair $(\mathbf{x}(t), \mathbf{y}(t))$ to train the IHDR tree by calling Procedure *AddPattern* in Appendix B as one incremental step.

5. Send the action $\mathbf{y}(t)$ to the controller which gives $\dot{\mathbf{q}}(t+1)$. Increment t by 1 and go to step 1.

6. Query the IHDR tree by calling the Procedure *Retrieval* in Appendix B and get the primed action $\mathbf{y}(t+1)$. Send $\mathbf{y}(t+1)$ to the controller which gives $\dot{\mathbf{q}}(t+1)$. Increment t by 1 and go to step 1.

The online incremental training process does not explicitly have separate training and testing phases. Whenever $\mathbf{y}$ is not given, the robot performs.

5.5 Experiments and results

Two kinds of experiments were designed and conducted. The first kind was purely object-avoidance behavior without modeling the environment. The second one incorporated the proposed path planner for the goal-directed navigation. It is interesting to note the complete knowledge of the environment is not necessary even for the second kind. The robot automatically built and updated the occupancy grid based on sensing.

5.5.1 Attention mechanism

Example 1. In this simulation experiment, it will show the importance of the attentional mechanism for generalization. Two IHDR trees were trained simultaneously: one used attention and the other used the raw range image directly without attention. We interactively trained the simulated robot in 16 scenarios[2] as shown in Fig. 5.8. During this training process, 1971 samples were acquired and each of the sample is associated with a label given by the trainer online.

[2]There are several training episodes for each scenario. For simplicity, we only show the robot's trace of a single episode in Fig 5.8. We do not show the goal pose, which is at right of each scene, except (2) whose goal pose is at the top.

Table 5.2: The results of tests with random starting positions.

	Range	Mean error (with attention)	Mean error (without attention)
θ	$[0, \pi]$	0.05	0.09
v	$[0, 1.0]$	0.005	0.007

Table 5.3: The results of tests with random starting positions.

	With attention	Without attention
Rate of success	0.91	0.63

In order to test the generalization capability of the learning system, we performed a leave-one-out test for both IHDR trees. The 1971 training samples were divided into 10 bins. We chose 9 bins for training and left one bin for testing. The mean square error (MSE) of the cross-validation test of two IHDR trees are shown in Table 5.5.1. Comparing the results, we can see that the mean error was decreased about 40 percent by introducing attention, which indicates that generalization capability was improved by adding attention.

The tests were performed in an environment different from the training scenarios. In Fig. 5.9 (a), with attention, the simulated robot performed successfully a continuous 10-minute run. The robot's trajectory is shown by small trailing dot-lines. Remember that no environmental map was stored across the laser maps and the robot had no global position sensors. Fig. 5.9 (b) shows that, without attention, the robot failed twice in a 3-minute test run.

Secondly, we ran a test in which the two learned IHDR trees were performed on the environment shown in Fig. 5.9 for 100 times. Each time we randomly chose a start position in the free space. In Table 5.5.1, we report the rate of success for the two trained IHDR trees. We treated a run to be successful when the robot can run continually for three minutes without hitting an obstacle. From the Table 3, we can see that the rate of success increased greatly by introducing attention.

5.5.2 Obstacle-avoidance on the Dav robot

Example 2. In this experiment, the robot's only goal is to move safely according to the scene. Such a navigation system is useful for applications where a human guides global motion but local motion is autonomous.

The Dav robot has been trained at different object configurations. IHDR was used to learn the mapping from sensory context (laser map and the robot's current speed) to desired command (i.e., translation v_y and angular speeds ω) for robot. The desired command was supplied by online interactively via a graphical user interface (see Fig. 5.7). Totally 4,655 samples were used for the reliable collision avoidance behavior.

The Dav robot has been repeatedly tested for the learned range-based collision avoidance and performance has been very satisfactory. For example, during a visit by high school students, as shown in Fig. 5.10, Dav successfully navigated in this dynamic changing environment without collisions with moving students. It is worth noting the testing scenarios were not the same as the training scenarios.

5.5.3 Integrate with the path planning layer

Example 3. In this simulated example, we show the results of integrating the path planning component with the learned obstacle-avoidance behavior on a bounded environment (43 meters by 30 meters). The resolution of the grid is of 10 cm and the robot's sensing scope is of 15 meters (for re-planning). Illustrated in Fig. 5.11, all the obstacles (shaded polygons) were completed unknown in advance to the simulated robot. Like the attention experiment (Example 1), we trained the robot on the 16 scenarios, shown in Fig. 5.8, but with specified goal headings θ. About 2000 samples were labeled online and trained for the low-level obstacle-avoidance behavior.

The performance of the proposed navigator with the path planner was verified from the same inital configuration to different goals in this environment. Fig. 5.11 depicts

some difficult cases where the goals are hidden by a U-shaped object or the goal are located very near the obstacles. The navigator was able to navigate successfully to the goals in all cases.

Example 4. The integrated navigating system has been implemented and tested on the Dav robot (see Fig. 2.1). The size of occupancy grid is 37.3 meters by 24.0 meters with the resolution 0.05 meter. The look-ahead sensing scope is not more than 10 meters. The heading has coarse resolution (i.e., eight angles); we hence convoluted it with a flat window for smoothing, that is:

$$\theta'(t) = \sum_{i=1}^{10} \theta(t-i).$$

Since the way of localizing the robot is not the topic of here, we only mention methods employed in this experiment here. Before navigation, the robot knew the occupancy grid of the environment and the sensor-detectable landmarks (beacons), such as the corner points and lines. Assuming the robot knows its initial pose in the map, the Kalman filter was used to estimate the robot's pose by fusing the odometry and matched landmark information.

Fig. 5.12 shows an execution of the proposed navigator on a corridor corner on the second floor of the Engineering Building at Michigan State University. Obstacles are shown in black in the occupancy grid. Except the walls, all other obstacles were unknown before navigation. As the robot starts moving, observed new obstacles were added to the grid. The local search algorithm was employed to recompute the new heading within the robot's sensing scope if the obstacle obstructed the planned path. Fig. 5.12 (a) corresponds to the moment after the robot starts for about a minute; obstacles were only those directly observed from its current pose. The red curve (labeled with "A") depicts the current planned path. The blue curve (labeled with "B") shows the robot's trajectory. Fig. 5.12 (b) illustrates the final result of Dav's navigation task, including the planned path and the robot's trajectory. Dav

successfully navigated on a number of obstacle configurations, and there was no case in which the robot ran into obstacles or the goal was not achieved.

5.6 Discussion and Conclusion

The system failed when moving obstacles were outside the field-of-view of the laser scanner. Since the laser scanner has to be installed at the front of the robot, nearby objects on the side are not "visible." This means that the trainer needs to "look ahead" when providing desired control signals so that the objects are not too close to the "blind spots." In addition, the trainer may label the samples such that the translation speed is proportional to the side clearance. The robot learns traveling at high speed through a wide corridor with sparse objects while slowing down through a narrow corridor or areas with dense objects. This shows the online real-time learning offers more flexibility than that of manually designed behaviors.

Our local search scheme may not be globally optimal if the k-values of the cells outside the sensing scope (L) are not equal to their true values. However, as argued by Simon [85], it is relatively rare that real-time applications need strict optimal solutions and near-optimal ones are often acceptable. The planner's $O(|L|\log(|L|))$ complexity allows us to search on a fairly large space for such a near-optimal solution. Further, the planner's termination at the goal is ensured if the k-values of cells outside L are not overestimated the true values.

This chapter described a range-based obstacle-avoidance learning system with a path planner implemented on a mobile robot. The attention selection mechanism reduces the importance of far-away objects when nearby objects are present. The power of the learning-based method is to enable the robot to learn very complex functions between the sensing context and the desired behavior, such a function is typically so complex that it is not possible to write a program to simulate it accurately. Indeed, the complex range-perception based human action learned by $\mathbf{y} = f(\mathbf{z}, \mathbf{v}, \theta)$

is too complex to write a program without learning. The success of the learning for high dimensional input $(\mathbf{z}, \mathbf{v}, \theta)$ is mainly due to the discriminating power of IHDR, and the real-time speed is due to the logarithmic time complexity of IHDR. The optimal subspace-based Bayesian generalization enables quasi-optimal interpolation of behaviors from matched learned samples. The online incremental learning is useful for the trainer to dynamically select scenarios according to the robot's weakness (i.e., problem areas) in performance. It is true that training needs extra efforts, but it enables the behaviors to change according to a wide variety of changes in the scenes.

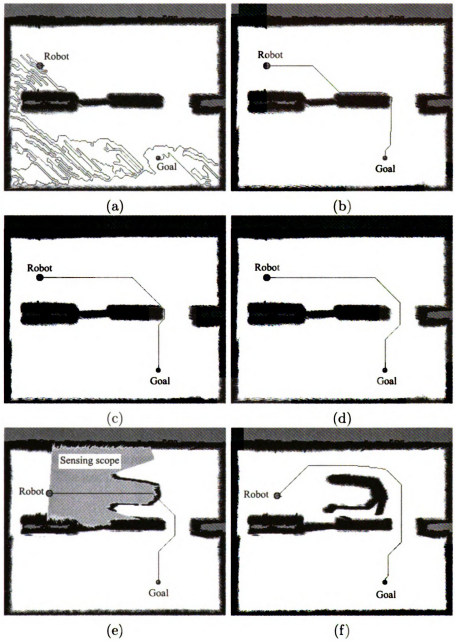


Figure 5.6: The result of tests on a simulated environment

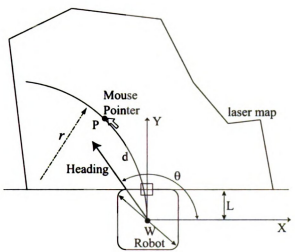


Figure 5.7: The local coordinate system and control variables. Once the mouse button is clicked, the position of the mouse pointer P provides an imposed action. Let r denote the radius of the arc. The arc's length d controls the translation speed while the radius controls the angular speed.

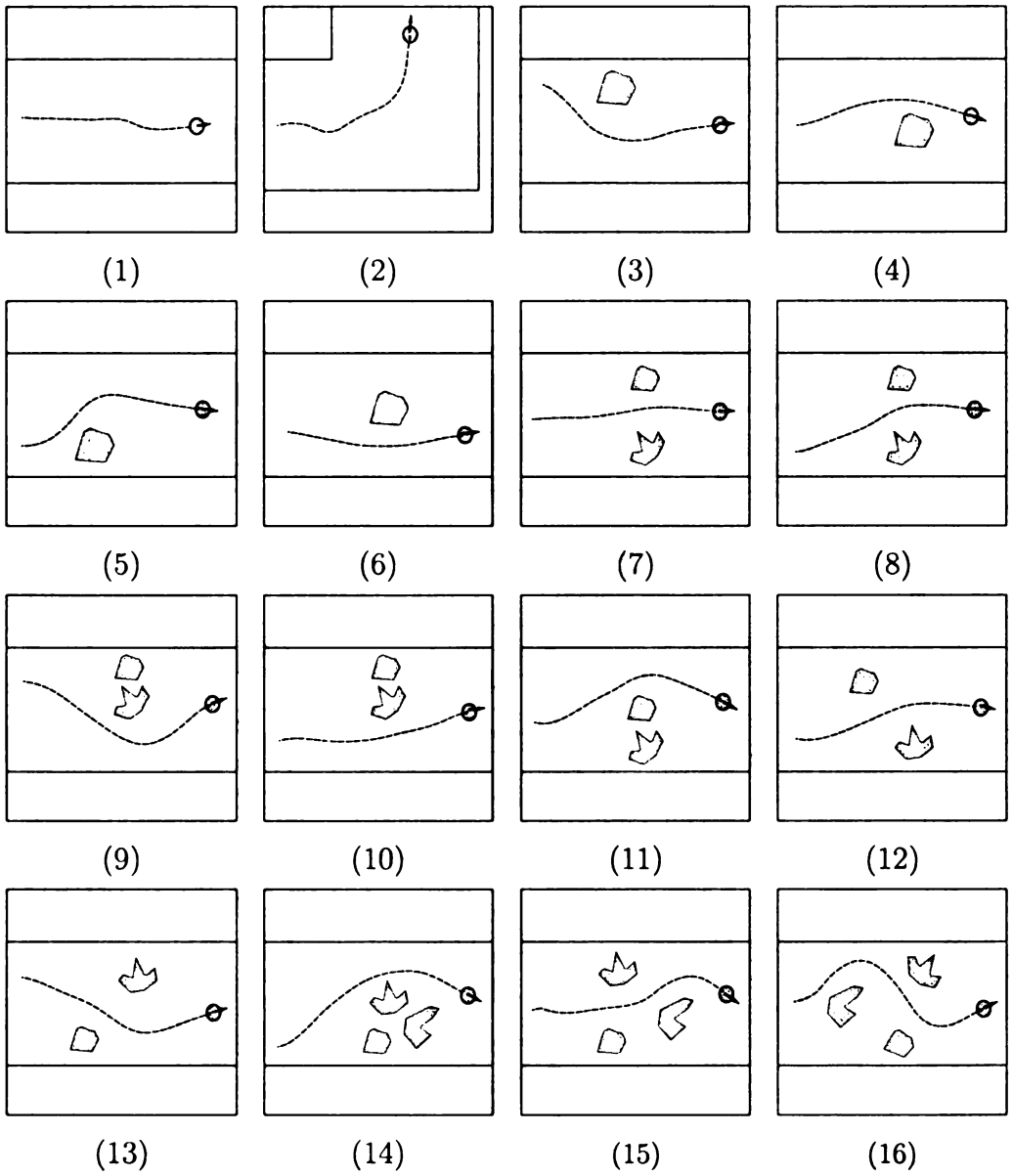
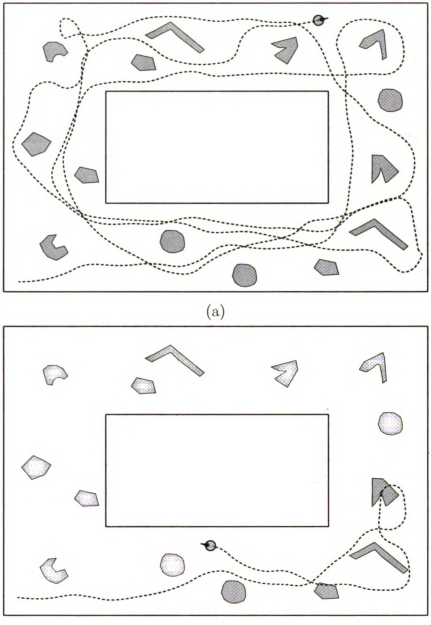


Figure 5.8: The 16 scenarios are used to train the simulated robot. The small solid circles denote the robot, and solid dark line represents the walls. The dot line recorded the online training trajectories.



(a)

(b)

Figure 5.9: The solid dark lines denote walls and the dot lines show the trajectory. Obstacles of irregular shape are scattered about the corridor. (a) A 10-minute run by the simulated robot with the attentional module. (b) The result of the test without attentional selection. Two collisions indicated by arrows are occurred during the first two minutes simulation.

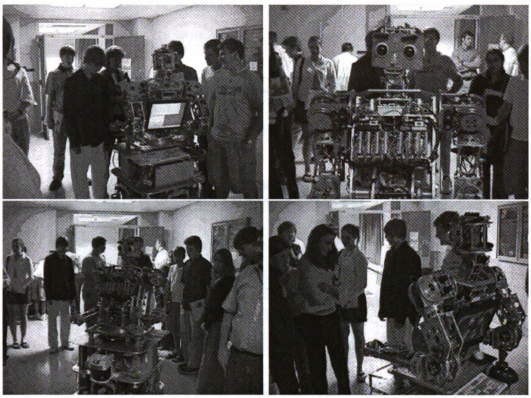


Figure 5.10: Dav moved autonomously in a corridor crowded with people.

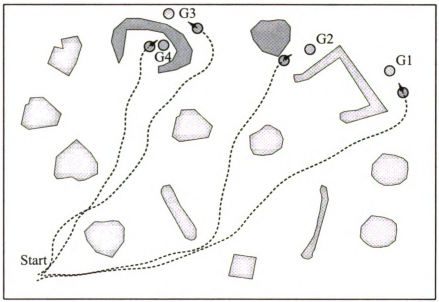


Figure 5.11: Navigating in an simulated environment with unknown obstacles. The initial configuration is chosen to be about (4.8m, 3.0m). The goal configurations G1, G2, G3 and G4 are chosen to be (37.1m, 25.0m), (29.6m, 24.3m), (18.6m, 27.6m) and (16.2m, 25.6m), respectively. For clearness, we do not draw the portion of trajectory very close to the goals.

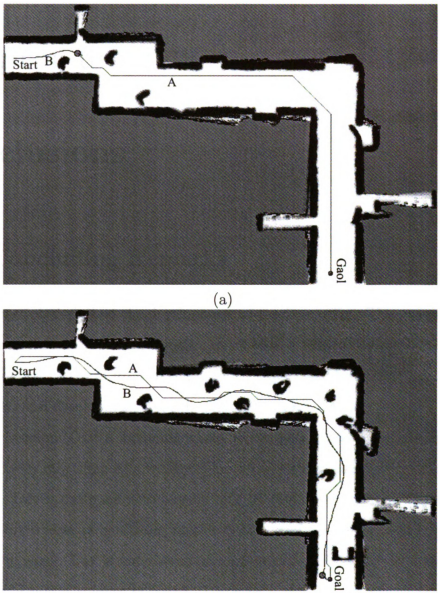


Figure 5.12: Dav run in the northwest corner of the corridor on the second floor of Engineering Building at Michigan State University. The red curve (or labeled with "A" depicts the actual planned path from the path planning layer, while the blue one (or label with "B") shows the actual trajectory in the occupancy grid.

Chapter 6

Conclusions

6.1 Concluding Remarks

This dissertation described the development of the Dav robot. This general-purpose anthropomorphic robot has two goals: to provide a general purpose, flexible, and dextrous robotic platform from the engineering aspect and to understand human autonomous mental development from the scientific aspect. Dav consists of a total 43 of degrees-of-freedom (DOF), including wheel-driven base, torso, arms, hands, neck and head. The body may support a variety of locomotive and manipulative behaviors. For perception, Dav is equipped with sensors such as vision (two cameras), auditory (two microphones), a laser range finder, tactile sensors, and somatic sensors (e.g., encoder and strain gauges). Dav is untethered and self-contained with all the computational resources and power sources onboard. The base's redundantly actuated mechanism provides holonomic locomotive capability.

The mechanical structure and kinematics of the robot were analyzed. The D-H coordination representation of the joints was derived. To facilitate the low-level joint control, Dav's dynamic model was derived based upon the Lagrange-Euler formulation. The dynamic model describes the interference between the robotic links and relation between the input joint torques and the output motion. Based upon the dy-

namic analysis, we find that the gravitational force dominates when the robot moves at low speeds. Therefore, the implemented proportional derivative feedback law with gravitational compensation is asymptotic stable. The mobile base's dynamic analysis and control were described. The overall embedded control system and the underlying software architecture were introduced.

A general-purposed and task-unspecific architecture was proposed for the Dav robot. The proposed architecture leaves much freedom for different implementations of its major components, e.g., sensory, cognitive, and motor mappings. Although the Dav developmental program that is currently being implemented could be explained in further detail, such details are beyond the scope of this dissertation.

We also presented a range-based obstacle-avoidance system with a path planner. The attention selection mechanism reduces the importance of far-away objects when nearby objects are present. The power of the learning-based method is to enable the robot to learn a very complex function between the sensing context and the desired behavior, such a function is typically so complex that it is not possible to write a program to simulate it accurately. The integration with the path planner makes sure the navigation system is free of local minima.

Finally, we need to remark that the work reported in this dissertation has not reached the level of a full-fledged developmental robot but can serve as a starting point of the on-going research on the Dav robot.

6.2 Future Work

In this section, we first recommend some improvements for the current implementation of the Dav robot; then, we list a few possible future perspectives along the line of research of the developmental robot.

As a prototype robot, the Dav robot has some mechanical limitations and hence needs improvement.

- Drive base. Although the redundantly actuated four-wheel mechnanism provides increased contact area, robustness in coping with irregularities of floor, and high clearance to tolerate the step features of the floor; it requires a very complex control scheme to synchronize the wheels. Turning wheels vertically also consumes lots of power to overcome the friction on a rough surface. In addition, the current base module occupies too much space since eight motors are needed. To simplify the base module, the author suggests use of a specially designed wheel mechanism, which has been available in the market (e.g., the omni-wheel from *Kornylak Corporation*). We can see the wheel actuated on a direction while rolling passively along the perpendicular direction; hence, no nonholonomic constraint exists. Three of those wheels are sufficient to realize the holonomic motion; hence the size of base module is reduced.

- Arm. The current version of arm mechanism is a serially actuated arm with rigid links. Also, in order to simplify the driving mechanism, DC motors are directly mounted at a place close to the joints. As a consequence, the motors of the low arm are a heavy load on the motors of the upper arm. In order to solve the problem, the author suggests using a more sophisticate 4DOF spherical shoulder mechanism [90]. This design moves all the shoulder motors into the torso.

- Hand. Dav's hand design has a similar shortcoming as the arm. Placing the motor close to the finger joints causes the hand to become intolerably heavy. A more complex mechanism is needed to move the finger motors to the wrist mechanism. The author suggests using a mechanism similar to NASA's Robonaut hand [63].

- Embedded system. Dav's distributed embedded system currently uses the prototype board (PB555) from *Axman Corporation*. The size of the board is a little big. A customized PowerPC board (MPC555, *Motorola Corporation*)

with more flash memory and static RAM memory is needed. For example, phyCORE-MPC555 from *Phytec America, LLC* may be a better choice than the old one.

Research in humanoid robotics has uncovered a variety of new problems and a few solutions to classical problems in robotics and control theory. There is now a wide consensus in the community that a humanoid that can duplicate humans' abilities will be a "grand challenge" for this millennium.

Autonomous mental development by a robot is an interdisciplinary area. It has drawn upon work in artificial intelligence, developmental psychology, ethology, systems theory, philosophy, and linguistics. This dissertation represents one of the early stage work in this line of research. This work has raised a few future perspectives:

- Dav's developmental architecture has yet to be implemented and tested, but its predecessor, SAIL, has successfully tested some major components of the designed architecture. The presented architecture will be tested in future studies and the performance will be reported.

- Attention mechanism. The flood of information (e.g., video stream) that enters a system can easily overwhelm the system's limited resources. To solve the problem, attention, a selective processing must be included to the system to let a part of the sensory input (e.g, goal related information) passing through. In our collision avoidance system, a simple programming-in attention effector is implemented. However, there is not much work done to develop (or learning autonomously) an attention system. This attention mechanism is primarily important in a robot's outdoor navigation experiment, where the lighting condition changes dramatically. The robot should pay attention to a small part of the scene (e.g., the road boundaries) in which the contrast is relatively stable comparing the change of pixel values of the overall image. It is known that

attention can help a system to work in untrained scenes, however, it is not clear how to develop an attention mechanism in an unstructured and highly inconsistent environment.

- Value system. The attention mechanism is an internal effector which is not observable by outsiders. Hence, supervised learning is not suitable for learning such an attention system. We believe a sophisticated value system is indispensable for development of a useful attention system. The problem becomes how to design the architecture and to find the computational model for the value system. In Chapter 3, we presented an architecture for the value system (Type-5 DOSASE MDP). However, more detailed work is needed to implement it.

- How to integrate high-level symbols? We observed that very young children (birth to 2 years) [71–73] cannot manipulate symbols. When the children get older, they develop symbolic representation based on their early sensorimotor experiences. It is unknown how those sensorimotor grounded symbols are expressed in the brains. However, from engineering perspectives, we can shortcut the development process, by plugging some human chosen symbols into the "brain" of a developmental robot at early stage. This can be justified since the human brain is not totally blank at birth. The biological brain is biased with some innate behaviors (e.g., rooting and sucking). In addition, as Chen and Weng [23] have shown, the performance of the indoor navigation system is improved by introducing environment states, which can be regarded as symbols. In Chapter 5, an occupancy grid was used to resolve the local minimum issue of reactive behaviors. How to design a representation of the environment but leaving sufficient freedom for later learning is still a challenging problem.

Appendix A

IHDR algorithm

Procedure 1 ***AddPattern.*** Given a labeled sample $(\mathbf{x}, \mathbf{y})$, update the IHDR tree T.

1: Find the best matched leaf node c by calling Procedure 3.

2: Add the training sample $(\mathbf{x}, \mathbf{y})$ to c's prototype set $\mathcal{P}_c$.

3: Find the closest cluster to the $\mathbf{x}$ and $\mathbf{y}$ vectors by computing:

$$m = \arg \min_{1 \le i \le q} (w_x \frac{\|\mathbf{x} - \mathbf{c}_i\|}{\sigma_x} + w_y \frac{\|\mathbf{y} - \bar{\mathbf{y}}_i\|}{\sigma_y}), \tag{A.1}$$

where w_x and w_y are two positive weights that sum to 1: $w_x + w_y = 1$; σ_x and σ_y denote incrementally estimated average lengths of $\mathbf{x}$ and $\mathbf{y}$ vectors, respectively; and $\mathbf{c}_i$ and $\bar{\mathbf{y}}_i$ denote, respectively, x-center and y-center of the ith cluster of the node c.

4: Let $\mu(n)$ denote the amnesic function that controls the updating rate, depending on n, in such a way as:

$$\mu(n) = \begin{cases} 0 & \text{if } n \le n_1 \ , \\ b(n - n_1)/(n_2 - n_1) & \text{if } n_1 < n \le n_2 \ , \\ b + (n - n_2)/d & \text{if } n_2 < n, \end{cases} \tag{A.2}$$

where n denotes the number of visits to the cluster the closest m (see Eq. A.1) in the node c, and b, n_1, n_2 and d are parameters. We update the x-center $\mathbf{c}_m$ and y-center $\bar{\mathbf{y}}_m$ of the cluster m in the node c, but leave other clusters unchanged:

$$\begin{aligned} \mathbf{c}_m(n) &= \tfrac{n-1-\mu(n)}{n}\mathbf{c}_m(n-1) + \tfrac{1+\mu(n)}{n}\mathbf{x}, \\ \bar{\mathbf{y}}_m(n) &= \tfrac{n-1-\mu(n)}{n}\bar{\mathbf{y}}_m(n-1) + \tfrac{1+\mu(n)}{n}\mathbf{y}, \end{aligned} \tag{A.3}$$

where $\mathbf{c}_m(n-1)$ and $\bar{\mathbf{y}}_m(n-1)$ are the old estimations (before update) of x-center and y-center of the mth cluster in the node c, respectively, while $\mathbf{c}_m(n)$ and $\bar{\mathbf{y}}_m(n)$ are the new estimations (after update). Readers should note that the weights w_x and w_y control the influence of $\mathbf{x}$ and $\mathbf{y}$ vectors on the clustering algorithm. For example, when $w_x = 0$, Eq. (A.3) is equivalent to the y label clustering algorithm used in Hwang & Weng [44].

5: Compute the sample mean, say $\mathbf{c}$, of all the q clusters in the node c. Let the current x-centers of the q clusters by $\{\mathbf{c}_i | \mathbf{c}_i \in \mathcal{X}, i = 1, 2, ..., q\}$ and the number of samples in the cluster i be n_i. Then,

$$\mathbf{c} = \sum_{i=1}^{q} n_i \mathbf{c}_i / \sum_{i=1}^{q} n_i.$$

6: Call the Gram-Schmidt Orthogonalization (GSO) procedure (see Hwang & Weng [44, Appendix A]) using $\{\mathbf{c}_i - \mathbf{c} | i = 1, 2, ..., q\}$ as input. Then calculate the projection matrix M of subspace $\mathcal{D}$ as:

$$M = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{q-1}], \tag{A.4}$$

where $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_{q-1}$ denote the orthnormal basis vectors derived by the GSO procedure.

7: Update W_m, the distance matrix of the mth cluster in the node c, as:

$$
\begin{aligned}
b_e &= \min\{n-1, n_s\} \\
b_m &= \min\{\max\{2(n-q)/q, 0\}, n_s\} \\
b_g &= 2(n-q)/q^2 \\
w_e &= b_e/b, \qquad w_m = b_m/b, \quad w_g = b_g/b \\
b &= b_e + b_m + b_g \\
A &= M'^T M \\
B &= M^T(\mathbf{x} - \mathbf{c}_m)(\mathbf{x} - \mathbf{c}_m)^T M \\
\Gamma_m(n) &= \tfrac{n-1-\mu(n)}{n} A^T \Gamma_m(n-1) A + \tfrac{1+\mu(n)}{n} B \\
S &= \textstyle\sum_{i=1}^{q} n_i \Gamma_i(n) / \sum_{i=1}^{q} n_i \\
W_m &= w_e \rho^2 I + w_m S + w_g \Gamma_m(n)
\end{aligned}
$$

where $\mu(n)$ is the amnesic function defined in Eq. (A.2), ρ and n_s are empirical defined parameters, M' is the old estimation of the projection matrix (before update using Eq. (A.4)), and $\Gamma_m(n-1)$ and $\Gamma_m(n)$ denote, respectively, the old and new estimations of the covariance matrix of the cluster m.

8: **if** the size $\mathcal{P}_c$ is larger than n_f, a predefined parameter, **then**

9: Mark c as an internal node. Create q nodes as c's children, and reassign each prototype $\mathbf{x}_i$ in $\mathcal{P}_c$ to the child k, based on discriminating functions defined in Eq. (3.10). This is to compute:

$$k = \arg\min_{1 \le i \le q}(l_i(\mathbf{x}_i)).$$

10: **end if**

Procedure 2 ***Retrieval.*** Given an IHDR tree T and an input vector $\mathbf{x}$, return the corresponding estimated output $\hat{\mathbf{y}}$.

1: By calling Procedure 3, we obtain the the best matched leaf node c.

2: $p = \text{parent}(c)$, i.e., p denotes the parent node of c.

3: Let the M be the projection matrix of subspace $\mathcal{D}$ in the node p. Compute the projection of the prototype set $\mathcal{P}_c$ on subspace $\mathcal{D}$:

$$\mathcal{P}'_c = \{(\mathbf{u}_1, \mathbf{y}_1), (\mathbf{u}_2, \mathbf{y}_2), ..., (\mathbf{u}_{n_c}, \mathbf{y}_{n_c})\}$$

where $\mathbf{u}_i = M^T(\mathbf{x}_i - \mathbf{c})$ for $1 \leq i \leq n_c$ and $\mathbf{c}$ denotes the x-center of the node p.

4: The projection of query input vector $\mathbf{x}$:

$$\mathbf{u} = M^T(\mathbf{x} - \mathbf{c})$$

5: **if** $|\mathcal{P}'_c| = N_q$, i.e., the number of prototype equals to predefine constant N_q **then**

6: Compute $\hat{\mathbf{y}}$ by using the nearest-neighbor decision rule in the set:

$$\mathcal{P}_c = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), ..., (\mathbf{x}_{n_c}, \mathbf{y}_{n_c})\}$$

, such that

$$\hat{\mathbf{y}} = \mathbf{y}_m, \tag{A.5}$$

where

$$m = \arg \min_{1 \leq i \leq n_c} \| \mathbf{x}_i - \mathbf{x} \| .$$

7: **else**

Compute $\hat{\mathbf{y}}$ by using locally weighted regression (LWR) [9] on dataset $\mathcal{P}'_c$. Let

$$W = \text{diag}(\sqrt{K(d(\mathbf{u}_1, \mathbf{u}))}, \sqrt{K(d(\mathbf{u}_2, \mathbf{u}))}, ..., \sqrt{K(d(\mathbf{u}_{n_c}, \mathbf{u}))})$$
$$Y = (\mathbf{y}_1^T, \mathbf{y}_2^T, ..., \mathbf{y}_{n_c}^T)^T$$
$$X = (\mathbf{u}_1^T, \mathbf{u}_2^T, ..., \mathbf{u}_{n_c}^T)^T$$
$$Z = WX$$
$$V = WY$$
$$\beta = (Z^T Z)^{-1} Z^T V$$

where $d(x, y)$ denotes the Euclidean distance between vectors x and y; $K(.)$ is a kernel function:

$$K(d) = \exp(-\frac{|\, d \,|}{h_T})$$

and h_T is a predefined parameter. Thus, the predicted output for the querying point $\mathbf{x}$ is:

$$\hat{\mathbf{y}}^T = \mathbf{u}^T \beta \tag{A.6}$$

8: **end if**

9: Return $\hat{\mathbf{y}}$.

Procedure 3 ***SelectLeaf***. Given an IHDR tree T and a sample $(\mathbf{x}, \mathbf{y})$, where $\mathbf{y}$ is either given or not given. Output: the best matched leaf node c.

1: $c \leftarrow$ the root node of T.

2: **for** c is an internal node **do**

3: $c \leftarrow$ the mth child of the node c, where $m = \arg\min_{1 \le i \le q}(l_i(\mathbf{x}))$ and $l_i(\mathbf{x})$, $i = 1, 2, ..., q$ are defined in Eq. (3.10).

4: **end for**

5: Return node c.

Bibliography

[1] Orocos@kth. In *http://cogvis.nada.kth.se/orocos/.*

[2] L. Acosta, G. N. Marichal, L. Moreno, J. A. Méndez, and J. J. Rodrigo. Obstacle avoidance using the human operator experience for a mobile robot. *Journal of Intelligent and Robotic Systems*, 27:305–319, 2000.

[3] B. Adams, C. Breazeal, R. A. Brooks, and B. Scassellati. Humanoid robots: A new kind of tool. *IEEE Intelligent System*, 15(4):25–31, July/August, 2000.

[4] J. S. Albus. Outline for a theory of intelligence. *IEEE Trans. Systems, Man and Cybernetics*, 21(3):473–509, May/June 1991.

[5] N. Almassy, G. M. Edelman, and O. Sporns. Behavioral constraints in the development of neural properties: A cortical model embedded in a real-world device. *Cerebral Cortex*, 8(4):346–361, 1998.

[6] J. R. Anderson. *Rules of the Mind.* Lawrence Erlbaum, Mahwah, New Jersey, 1993.

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

[8] H. Asama, M. Sato, L. Bogoni, H. Kaetu, A. Matsumoto, and I. Endo. Development of an omni-directional mobile robot with 3 dof decoupling drive mechanism. In *Proc. International Conference on Robotics and Automation*, pages 1925 – 1930, May 1995.

[9] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[10] R. E. Bellman. *Adaptive Control Processes.* Princeton University Press, Princeton, NJ, 1961.

[11] A. Billard and M. Mataric. Learning human arm movements by imitation: Evaluation of a biologically-inspired connectionist architecture. In *Proc. IEEE International Conference on Humanoid Robots*, Cambridge, MA, September 7-8, 2000.

[12] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[13] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Transactions Robotics and Automation*, 7(3):341–346, 1991.

[14] C. Breazeal. Social constraints on animate vision. In *Proc. IEEE International Conference on Humanoid Robots*, Cambridge, MA, June 20-25, 2000.

[15] C. Breazeal, A. Edsinger, P. Fitzpatrick, B. Scassellati, and P. Varchavskaia. Social constraints on animate vision. *IEEE Intelligent Systems and Their Applications*, 15(4):32–37, 2000.

[16] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *International Conference on Robotics and Automation (ICRA99)*, pages 341–346, Detroit, MI, June 1999.

[17] R. Brockett. Asymptotic stability and feedback stabilization. In R. Brockett, R. Millman, and H. Sussmann, editors, *Differential Geometric Control Theory*, pages 181–191. Birkhauser, Boston, 1983.

[18] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[19] R. A. Brooks, C. Breazeal, M. Marjanovic, B. Scassellati, and M. M. William. The cog project: Building a humanoid robot. In C. L. Nehaniv, editor, *Computation for Metaphors, Analogy and Agents, vol. 1562 of Springer Lecture Notes in Artificial Intelligence.* Springer-Verlag, New York, NY, 1999.

[20] G. Campion, B. d'Andrea Novel, and G. Bastin. Controllability and state feedback stabilization of nonholonomic mechanical systems. In C. Canudas de Wit, editor, *Lecture Notes in Control and Information Science.* Springer-Verlag, New York, 1991.

[21] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28th IEEE Symposium on Foundation of Comptuer Science*, pages 49–60, Los Angeles, California, Oct. 12-14, 1987.

[22] S. Chen, C. F. N. Cowan, and P. M. Grant. Orthogonal least squares learning for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):321–355, 1991.

[23] S. Chen and J. Weng. State-based SHOSLIF for indoor visual navigation. *IEEE Trans. Neural Networks*, 11(6):1300–1314, November 2000.

[24] Boris V. Cherkassky, Andrew V. Goldberg, and Tomasz Radzik. Shortest paths algorithms: theory and experimental evaluation. *Math. Program.*, 73(2):129–174, 1996.

[25] J. H. Chuang. Potential-based modeling of tree dimensional workspace for obstacle avoidance. *IEEE Transactions on Robotics and Automation*, 14(5):778–785, 1998.

[26] J. Coelho, J. Piater, and R. Grupen. Developing haptic and visual perceptual categories for reading and grasping with a humanoid robot. In *Proc. 1st IEEE International Conference on Humanoid Robots*, Cambridge, MA, September 7-8, 2000.

[27] D. R. Cox. Statistical analysis of time series: Some recent developments. *Scand. J. Statist.*, 8(2):93–115, 1981.

[28] E. W. Dijkstra. A note on two problems in connection with graphs. *Number. Math.*, 1:269–271, 1959.

[29] G. L. Drescher. *Made-Up Minds.* MIT Press, Cambridge MA, 1991.

[30] A. D'Souza, S. Vijayakumar, and S. Schaal. Learning inverse kinematics. In *International Conference on Intelligence in Robotics and Autonomous Systesm (IROS 2001)*, volume v.1, pages 298–303, Maui, Hawaii, October 2001.

[31] R. Duda, P. Hart, and David G. Stork. *Pattern Classification and Scene Analysis.* John Weily & Sons, New York, NY, second edition, 2000.

[32] E. Eoyama, A. Agah, K. F. MacDorman, T. Maeda, and S. Tachi. A modulator neural network architecture for inverse kinematics model learning. *Neurocomputing*, 38-40:797–805, June 2001.

[33] M. Fisz. *Probability Theory and Mathematical Statistics.* John Wiley & Sons Inc., New York, third edition, 1963.

[34] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[35] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, NY, second edition, 1990.

[36] G. H. Golub and C. F. van Loan. *Matrix Computations.* The Johns Hopkins University Press, Baltimore, MD, 1989.

[37] J.D. Han, S.Q. Zeng, K.Y. Tham, M. Badgero, and J.Y. Weng. Dav: A humanoid robot platform for autonomous mental development. In *Proc. IEEE 2nd International Conference on Development and Learning (ICDL 2002)*, pages 73–81, MIT, Cambridge, MA, June 12-15, 2002.

[38] R. Held and A. Hein. Movement-produced stimulation and the development of visually guided behaviors. *Journal of Comparative and Physiological Psychology*, 56:872–876, 1963.

[39] K. Hirai, M. Hirose, Y. Haikaw, and T. akenak. The development of honda humanoid robot. *Proc. of IEEE International Conference on Robotics and Automation*, pages 1321–1326, 1998.

[40] R. Holmberg and O. Khatib. Development and control of a holonomic mobile robot for mobile manipulation tasks. *International Journal of Robotics Research*, 19(11):1066 – 1074, 2000.

[41] X. Huang and J. Weng. Novelty and reinforcement learning in the value system of developmental robots. In *Proc. Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems (EPIROB'02)*, pages 47–55, Edinburgh, Scotland, August 10 - 11, 2002.

[42] X. Huang and J. Weng. Novelty and reinforcement learning in the value system of developmental robots. In *Proc. Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems (EPIROB 2002)*, Edinburgh, Scottland, 2002.

[43] W. Hwang and J. Weng. Vision-guided robot manipulator control as learning and recall using SHOSLIF. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2862–2867, Albuquerque, NM, April 20-25, 1997.

[44] W. S. Hwang and J. Weng. Hierarchical discriminant regression. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(11):1277–1293, 11 2000.

[45] Y. K. Hwang and N. Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

[46] Shun ichi Amari. Learning and statistical inference. In Michael A. Arbib, editor, *The handbook of brain theory and neural networks.* The MIT Press, Cambridge, Massechusetts, second edition, 2003.

[47] R. Ivry. Representation issues in motor learning: phenomena and theory. In H. Heuer and S. W. Keele, editors, *Handbook of perception and action, volumn two, motor skill.* Academic Press, London, San Diego, New York, Boston, Sydney, Tokyo, Toronto, 1996.

[48] M. I. Jordan and D. E. Rumelhart. Forward models: supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[49] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[50] M. Kirby and L. Sirovich. Application of the karhunen-loève procedure for the characterization of human faces. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(1):103–108, January 1990.

[51] T. Kohonen. *Self-Organizing Maps.* Springer-Verlag, Berlin, second edition, 1997.

[52] K. Konolige. Colbert: A language for reactive control in saphira. In *Proceedings of the German Conf. on Artificial Intellgence*, Freiburg, Germany, 1997.

[53] K. Konolige. Ayllu: Distributed port-arbitrated behavior-based control. In *Proceedings of the Intl. Symp. on Distributed Autonomous Robotic Systems (DARS)*, Knoxville, Tennessee, October 2000.

[54] R.E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[55] S. L. Pallas L. von Melchner and M. Sur. Visual behavior mediated by retinal projections directed to the auditory pathway. *Nature*, 404:871–876, 2000.

[56] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[57] J. Latombe. *Robot motion planning.* Kluwer Academic Publishers, Boston, MA, 1991.

[58] T. Lee and C. Wu. Fuzzy motion planning of mobile robots in unknown environments. *Journal of Intelligent and Robotic Systems*, 37:177–191, 2003.

[59] L. Li, B. Cox, M. Diftler, S. Shelton, and B. Rogers. Development of a telepresence controlled ambidextrous robot for space applications. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 58–63, 1996.

[60] W. Li, H. Christensen, and A. Oreback. An architecture for indoor navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, New Orleans, LA, April. 26-30, 2004.

[61] H. Lim, A. Ishii, and A. Takanishi. Motion pattern generation for emotion expression. In *Proc. 2nd International Conference on Humanoid Robots*, pages 36–61, Tokyo, Japan, October 8-9, 1999.

[62] Chin-Teng Lin and C.S. George Lee. *Neural Fuzzy Systems: A Neuro-Fuzzy Synergism to Intelligent Systems.* Prentice-Hall PTR, 1996.

[63] C. Lovchik, H. Aldridge, and M. Diftler. Design of the nasa robonaut hand. In *Proc. of the ASME Dynamic Systems and Control Division*, volume DSC-Vol. 67, pages 823–830, 1999.

[64] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis.* Academic Press, London, New York, 1979.

[65] M. J. Marjanovic, B. Scassellati, and M. M. Williamson. Self-taught visually-guided pointing for a humanoid robot. In *From Animals to Animats: Proc. of 1996 Society of Adaptive Behavior*, pages 35–44, Cape Cod, MA, 1996.

[66] J. Minguez and L. Montano. Nearness diagram navigation (nd): A new real time collision avoidance approach. *Proc. of IEEE/RSJ International Conference On Intelligent Robots and Systems*, pages 2094–2100, 2000.

[67] J. Minguez, L. Montano, and J. Santos-Victor. Reactive navigation for non-holonomic robots using the ego-kinematic space. In *International Conference on Robotics and Automation (ICRA2002)*, pages 3074–3080, Washington D. C., May 11-15, 2002.

[68] A. Nagakubo, Y. Kuniyoshi, and G. Cheng. Etl-humanoid - a high-performance full body system for versatile action. *Proc. of IEEE/RSJ International Conference On Intelligent Robots and Systems*, pages 1087–1092, 2000.

[69] K. Nishiwaki, T. Sugihara, S. Kagami, F. Kanehiro, M. Inaba, and H. Enoue. Design and development of research platform for perception-action integration in humanoid robot : H6. *Proc. of IEEE/RSJ International Conference On Intelligent Robots and Systems*, pages 1559–1564, 2000.

[70] L. Petersson, D. Austin, and H. Christensen. Dca: A distributed control architecture for robotics. In *International Conference on Intelligence in Robotics and Autonomous Systesm (IROS 2001)*, volume v.1, pages 2361–2368, Maui, Hawaii, October 2001.

[71] J. Piaget. *The origins of intelligence in children.* International Universities Press, Madison, NY, 1952.

[72] J. Piaget. *The construction of reality in the child.* Basic Books, New York, NY, 1954.

[73] J. Piaget. *The Psychology of Intelligence.* Littlefield & Adams, Totowa, NJ, 1973.

[74] D. A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[75] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2):257–286, 1989.

[76] M. Rosenblum and Larry S. Davis. An improved radial basis function network for visual autonomous road following. *IEEE Trans. on Neural Networks*, 7(5):1111–1120, 1996.

[77] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, December 22, 2000.

[78] D. Roy, B. Schiele, and A. Pentland. Learning audio-visual associations using mutual information. In *Proc. International Conference on Computer Vision, Workshop on Integrating Speech and Image Understanding*, Corfu, Greece, 1999.

[79] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Upper Saddle River, NJ, second edition, 2003.

[80] N. Sarkar, X. Yun, and V. Kumar. Control of mechanical systems with rolling constraints: Application to dynamic control of mobile robots. *The International Journal of Robotics Research*, 13(1):55–69, February 1994.

[81] L. L. Scharf. *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1991.

[82] Wei-Min Shen. *Autonomous Learning from the Environment.* Computer Science Press, New York, 1994.

[83] S. Shigeki and K. Ichiro. Wabot-2: Autonomous robot with dexterous finger-arm coordination control in keyboard performance. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 90–97, 1987.

[84] R. Simmons. The curvature-velocity method for local obstacle avoidance. In *IEEE International Conference on Robotics and Automation ICRA'96*, pages 2275–2282, Minneapolis, April 1996.

[85] H.A. Simon. *The sciences of the Artificial.* MIT Press, Cambridge, MA, third edition, 1996.

[86] S. Singh, D. Feng, P. Keller, G. Shaffer, W. Shi, D. Shin, J. West, and B. Wu. A system for fast navigation of autonomous vehicles. Technical Report CMU-RI-TR-91-20, The RoboticsCarnegie Institute, Mellon University, Pittsburgh, PA, 1991.

[87] J. E. Slotine and W. Li. *Applied Nonlinear Control.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[88] A. J. Smola and B. Scholkopf. A tutorial on support vector regression. Technical Report NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998.

[89] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control.* John Wiley & Sons, New York, first edition, 1989.

[90] M. Stanišić, J. Wiitala, and J. Feix. A dexterous humanoid shoulder mechanism. *Journal of Robotic Systems*, 18(12):737–745, 2001.

[91] R. S. Sutton and A. Barto. *Reinforcement Learning.* The MIT Press, Cambridge, MA, 1998.

[92] S. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach.* Kluwer Academic, Boston, MA, 1996.

[93] R. Tibishirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[94] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[95] Iwan Ulrich and Johann Borenstein. Vfh*: Local obstacle avoidance with look-ahead verification. In *International Conference on Robotics and Automation (ICRA2000)*, pages 2505–2511, San Francisco, CA, April 2000.

[96] Richard T. Vaughan, Brian Gerkey, and Andrew Howard. On device abstractions for portable, reusable robot code. In *International Conference on Intelligence in Robotics and Autonomous Systesm (IROS 2003)*, pages 2121–2427, Las Vegas, Arizona, October 2003.

[97] S. Vijayahumar and S. Schaal. Real time learning in humanoids: A challenge for scalability of online algorithms. In *Proc. IEEE International Conference on Humanoid Robots*, Cambridge, MA, September 7-8, 2000.

[98] S. Vijayakumar, A. D'Souza, T. Shibata, J. Conradt, and S. Schaal. Statistical learning for humanoid robots. *Autonomous Robot*, 12(1):55–69, 2002.

[99] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An o(n) algorithm for incremental real time learning in high dimensional space. In *Proc. of Seventeenth International Conference on Machine Learning (ICML2000)*, pages 1079–1086, Stanford, CA, June 19-July 2, 2000.

[100] L. S. Vygotsky. *Thought and language.* MIT Press, Cambridge, Massachussetts, 1962. trans. E. Hanfmann & G. Vakar.

[101] M. Wada and S. Mori. Holonomic and omnidirectional vehicle with conventional tires. In *Proc. International Conference on Robotics and Automation*, pages 3671 - 3676, April 1996.

[102] I. Wallance, D. Klahr, and K. Bluff. A self-modifying production system of cognitive development. In D. Klahr, P. Langley, and R. Neches, editors, *Production System Models of Learning and Development*, pages 359–435. MIT Press, Cambridge, MA, 1987.

[103] C. Watkins. Learning from delayed rewards. Technical report, PhD thesis, King's College, Cambridge, England, 1989.

[104] C. Watkins. Q-learning. *Artificial Intelligence*, 8:55–67, 1992.

[105] J. Weng. The living machine initiative. Technical Report CPS 96-60, Department of Computer Science, Michigan State University, East Lansing, MI, December 1996. A revised version appeared in C. W. Chen and Y. Q. Zhang. *Visual Communication and Image Processing*, Marcel Dekker, NY, 431 - 487, 1998.

[106] J. Weng. Learning in image analysis and beyond: Development. In C. W. Chen and Y. Q. Zhang, editors, *Visual Communication and Image Processing*, pages 431 – 487. Marcel Dekker, New York, NY, 1998. A revised version from "Living Machine Initiative," MSU CPS Tech. Report CPS-96-60, 1996.

[107] J. Weng. A theory for mentally developing robots. In *Proc. IEEE 2nd International Conference on Development and Learning (ICDL 2002)*, pages 131–140, MIT, Cambridge, MA, June 12-15, 2002.

[108] J. Weng, N. Ahuja, and T. S. Huang. Learning recognition using the Cresceptron. *International Journal of Computer Vision*, 25(2):109–143, November 1997.

[109] J. Weng and S. Chen. Vision-guided navigation using SHOSLIF. *Neural Networks*, 11:1511–1529, 1998.

[110] J. Weng and S. Chen. Visual learning with navigation as an example. *IEEE Intelligent Systems*, 15:63–71, September/October 2000.

[111] J. Weng and W. Hwang. An incremental learning algorithm with automatically derived discriminating features. In *Proc. Asian Conference on Computer Vision*, Taipei, Taiwan, Jan. 8-9, 2000.

[112] J. Weng and W. Hwang. Online image classification using IHDR. *International Journal on Document Analysis and Recognition*, 5(2-3):118–125, 2002.

[113] J. Weng and W. Hwang. Online image classification using ihdr. *International Journal on Document Analysis and Recognition*, 5(2-3):118–125, 2003.

[114] J. Weng and W. S. Hwang. An incremental learning algorithm with automatically derived discriminating features. In *Proc. Asian Conference on Computer Vision*, pages 426 – 431, Taipei, Taiwan, January 8-9, 2000.

[115] J. Weng, W. S. Hwang, Y. Zhang, and C. Evans. Developmental robots: Theory, method and experimental results. In *Proc. 2nd International Conference on Humanoid Robots*, pages 57–64, Tokyo, Japan, October 8-9, 1999. IEEE Press.

[116] J. Weng, W. S. Hwang, Y. Zhang, C. Yang, and R. Smith. Developmental humanoids: Humanoids that develop skills automatically. In *Proc. First IEEE Conf. on Humanoid Robots*, MIT, Cambridge, MA, September 7-8, 2000. IEEE Press.

[117] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, JAN. 2000.

[118] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291:599–600, January 26, 2001.

[119] J. Weng, Y. Zhang, and W. Hwang. Candid covariance-free incremental principal component analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(8), 2003.

[120] M. West and H. Asada. Design of ball wheel mechanisms for omnidirectional vehicles with full mobility and invariant kinematics. *Journal of Mechanical Design*, 119:153–161, 1997.

[121] D. J. Wood, J. S. Bruner, and G. Ross. The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry*, pages 89–100, 1976.

[122] S. L. Zeger and B. Qaqish. Markov regression models for time series: A quasi-likelihood approach. *Biometrics*, 44:1019–1031, 1988.

[123] S. Zeng and J. Weng. Obstacle avoidance through incremental learning with attention selection. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume v.1, New Orleans, LA, April. 26-30, 2004.

[124] S. Zeng and J. Weng. Online-learning and attention-based obstacle avoidance using a range finder. In *Proceedings of the 17th International FLAIRS Conference*, Miami Beach, FL, May 17-19, 2004.

[125] S.Q. Zeng, D. M. Cherba, and J. Weng. Dav developmental humanoid: The control architecture and body. In *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 974–980, Kobe, Japan, July 2003.

[126] S.Q. Zeng and Y. B. He. Learning and tuning fuzzy logic controllers through genetic algorithm. In *IEEE International Conference on Neural Networks*, volume 3, pages 1632–1637, Orlando, Florida, June 1994.

[127] N. Zhang and J. Weng. A developing sensory mapping for robots. In *Proc. IEEE 2nd International Conference on Development and Learning (ICDL 2002)*, pages 13–20, MIT, Cambridge, MA, June 12-15, 2002.

[128] Y. Zhang and J. Weng. Complementary candid incremental principal component analysis. Technical Report MSU-CSE-01-24, Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, August 2001.

[129] Y. Zhang and J. Weng. Convergence analysis of complementary candid incremental principal component analysis. Technical Report MSU-CSE-01-23, Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, August 2001.

[130] Y. Zhang and J. Weng. Grounded auditory development by a developmental robot. In *Proc. INNS-IEEE International Joint Conference on Neural Networks*, pages 1059–1064, Washington, DC, July 14-19, 2001.

[131] Y. Zhang and J. Weng. Action chaining by a developmental robot with a value system. In *Proc. IEEE 2nd International Conference on Development and Learning (ICDL 2002)*, pages 53–60, MIT, Cambridge, MA, June 12-15, 2002.

[132] Y. Zhang and J. Weng. Chained action learning through real-time interactions. In *Proc. IEEE Int'l Conf. on Neural Networks*, Honolulu, USA, May 2002.


MICHIGAN STATE UNIVERSITY LIBRARIES
3 1293 02504 2874