Proceedings of the 1998 IEEE
International Conference on Robotics & Automation
Leuven, Belgium • May 1998

A Control Architecture to Achieve Manipulation Task Goals for a Humanoid Robot

Young-Jo Cho[*], Jung-Min Park[*], Jaehyun Park[**], Sang-Rok Oh[*], Chong Won Lee[*]

[*]Korea Institute of Science and Technology, Seoul, Korea

[**]Inha University, Inchun, Korea

ABSTRACT

Focusing on the manipulation tasks to be executed by humanoid robots, principal requirements which are to be satisfied by hardware/software of the control system are considered. In order to meet the requirements, a novel type of hardware structure and software architecture is proposed in this paper. Since the target humanoid robot consists of multiple subsystems such as a central controller for brain, a vision controller for eye, and five motion sub-controllers for two arms, two hands, one spine, the on-board hardware control system is designed to have a distributed control structure connected by pseudo real-time Ethernet interfaces. A goal-achieving software architecture is also proposed which meets the requirements of semi-autonomy, reactivity, expandability, and object-orientedness. Specifically, in order to achieve the reactivity, a coordination method is proposed to configure three kinds of executive modules, primitive module, flow-control module, and goal module, which have multiple exit states. The control architecture proposed has been implemented for performing toy-block assembly tasks on a humanoid robot as well as on the graphic simulator.

1. Introduction

Recently, as application areas of robots have been broadened, interests of most robotics researchers has been moving from conventional industrial robots to new types of robots such as service robots and humanoid robots. Several successful applications have been reported such as Sojourner / Rocky for Martian exploration[1][2], the six-legged Ambler for planetary exploration[3], TAROS for maintaining nuclear facilities[4], etc. Particularly, as innovative technologies in the areas supporting robotics have been developed in terms of mechanics, computer control engineering, artificial intelligence, etc., several research groups like the Waseda university in Japan[5] and the MIT AI lab.[6] have paid special attention to humanoid robots. The Korea Institute of Science and Technology(KIST) also has taken much interest in humanoid robot technologies. The humanoid robot we have been developing has a similar shape as Centaurius in ancient Greek myths, i.e., we combined two arms, two hands, four legs, and a head with two eyes to form a humanoid robot and called it Centaur. Our focus is not merely on constructing a human-like mechanism, but on developing key technologies by using the benchmark humanoid model. Specifically, concentrating on human-like manipulations, we design a goal-achieving control architecture for the upper body of the humanoid robot.

Robot control architectures that have been reported so far mainly concern navigation tasks performed by mobile or locomotive robots. However, humanoid robots are typically designed to have excellent capability of manipulating objects dexterously enough to achieve complicated and variable task goals as well as to have the locomotion capability. Thus, it seems to us that more emphasis has to be put on achieving manipulation tasks rather than navigation tasks in developing humanoid robots. Moreover, the environment that the humanoid robot copes with is not well structured and thus the conventional control architectures for robot manipulators that worked well in structured environments are not fit for humanoid robots any more. For this, a couple of attempts have been made to design control architectures enabling robots to achieve manipulation task goals in dynamic environments.

Specifically, based on the Brooks' subsumption architecture for mobile navigation[7], Connell implemented a behavior-based arm controller applied to a manipulation task collecting soda cans[8]. In his control scheme, 15 separate behavior modules were defined and arranged in 6 levels of competence. Each of the behaviors contains some grain of expertise manipulating objects with hand/arm and cooperates with each other to accomplish its goal. However, the control architecture is only suitable for use with an arm equipped with special sensors. As one of behavior-based approaches to the

0-7803-4300-x-5/98 $10.00 © 1998 IEEE

robotic manipulation, Wilson examined a practical method of using appropriate information to provide alternative control paths according to the possible events in real-time robotic assembly applications[9][10]. He introduced the concept of exit states that inform the control system of which internal control path was taken through a behavioral module, but the parallel execution of the behavioral modules, which might be important in achieving complex tasks was not supported as the constraint on the task-achieving process. Grupen *et. al.*,[11] proposed a bottom-up approach that composed behaviors on-line from a set of reusable feedback control laws called a control basis. In their architecture, carefully designed control bases reflect generic control strategies that are largely independent of the task geometry and robot kinematics, and the composition policy combining the control bases makes the overall controller to achieve the given tasks. In order to apply this concept to humanoid robots, however, the control bases incorporating lots of actuators and sensors have to be designed thoroughly, and then task-dependent composition policies reflecting various human-like behaviors must be specified well in a huge rule base.

These researches on the control architecture for manipulation are not originally aiming at the humanoid robots, but their design concepts are valuable for constructing basic schemes in the overall control architecture of our humanoid robot. In this paper, we first consider the functional requirements for the control architecture of humanoid robots in general sense, and then propose the hardware and software architecture to meet such requirements for our humanoid robot model.

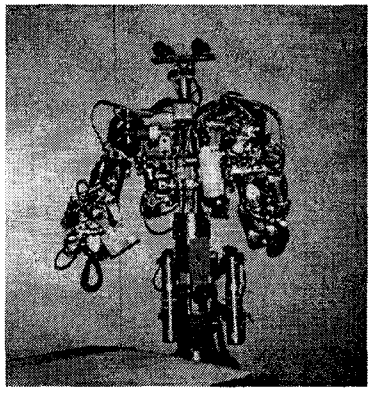

Fig. 1. A humanoid robot model constructed by KIST

2. Requirements for the Humanoid Robot Control Architecture.

In designing a control architecture for humanoid robots, the following three basic requirements are in general to be provided by the control hardware:

(1) *A Distributed Control Structure* : Humanoid robots for manipulation are supposed to have at least two arms, two hands, and an eye. Because quite many actuators and sensors have to be controlled by the overall control system for a humanoid robot, the distributed control structure is more efficient than the centralized one.

(2) *Versatile and Real-Time Communication Interfaces* : The communication interface between distributed control nodes is necessarily designed to have a real-time broadcasting network feature compatible with the off-line simulator as well as to follow the industry standards for expandability.

(3) *Embedded Real-time Operating Systems* : The control software executed in the distributed control system is highly timing-constrained. Fail to keep these timing constraints results in a serious malfunction of the entire robot system. Because the general-purpose operating systems cannot manage these timing constraints strictly, a real-time operating system should be able to be embedded in the control hardware.

In the software viewpoint, on the other hand, the following features are required in order for humanoid robots to achieve the manipulation task goals in human-like ways:

(1) *Autonomy* : Autonomy, which implies that robots can perform certain tasks by themselves without human assistance or intervention, is considered to be the ultimate goal of designing control architecture of humanoid robots. Specifically in dealing with manipulation tasks, however, full autonomy is not achievable due to the immense sensing and computing power required, and hence a semi-autonomous scheme to allow for human planning needs to be used.

(2) *Reactivity* : Due to the unstructured and dynamic nature of the real world, little assumptions could be made about the environments where humanoid robots work. Therefore, the control software should be able to handle sudden external events with time bounds so that the correct and efficient execution is assured.

(3) *Expandability* : Since much time and efforts are required to design, build and test the individual components of the control system independently, an expandable architecture is desirable to allow for developing the control software incrementally. Humanoid robots require a large number of different skills to cope with the diversity of the possible situations. The robot control system should have the capability of integrating new functions that do not interfere with the operation of the previous tested components.

(4) *Object-oriented programming* : Since humanoid robots are designed to handle various tasks including ones not predefined precisely. The object-oriented programming method rather than the traditional structured programming method is desirable in the sense that the method integrates data/procedures and treats the object that is a collection of data/procedures and functions operating on these variables.

(5) *Learning Capability* : It seems to us that the ultimate goal of the research on humanoid robots is to make the system like human-beings in the sense of not only its outer shape but also its internal functionalities. The human motor system may be characterized by the ability to learn, which is one of the properties not exhibited by traditional man-made machines. From this point of view, the learning capability should be considered to design software architecture of humanoid robots.

3. A Hierarchically Distributed Hardware Structure for the Humanoid Robot Control System

The on-board control system of the humanoid robot model is designed to have a distributed control structure which consists of a central controller (brain), a vision sub-controller (eye), and five motion sub-controllers (two arms, two hands, and one spine), as shown in Fig. 2. All of the embedded control components are programmed with the aid of VxWorks real-time operating system and thus a SUN SPARC station is needed for developing (compiling and debugging) the control programs. For high-level task planning, a host computer which supports some off-line functions such as human-assisted planning or graphic simulation is connected to the central controller through an Ethernet channel over which standard TCP/IP protocol is used.

The central control part, which is called C^3(Centaur Central Controller), is a real-time computer system that takes charge of coordination and dispatch of low level commands to the vision/motion sub-controllers. The vision sub-controller is tightly coupled with the C^3 to construct a human-like head and thus shares the same VME bus with C^3 for high data bandwidth. On the contrary, the communication channel between the C^3 and five motion sub-controllers need relatively less data bandwidth. Considering the requirement of versatile and real-time communication interfaces, we connect the C^3 to the motion sub-controllers through another Ethernet channel adopting an extended TCP/IP protocol for pseudo real-time network.

Even though Ethernet is recognized as improper to real-time applications, Ethernet TCP/IP has been adopted by most major real-time operating systems such as VxWorks or VRTX. The non-real-time characteristics of Ethernet are mainly caused by its 1-persistent binary back-off algorithm. By extending TCP/IP protocol in a way of limiting the communication traffic over Ethernet, this drawback is compensated.

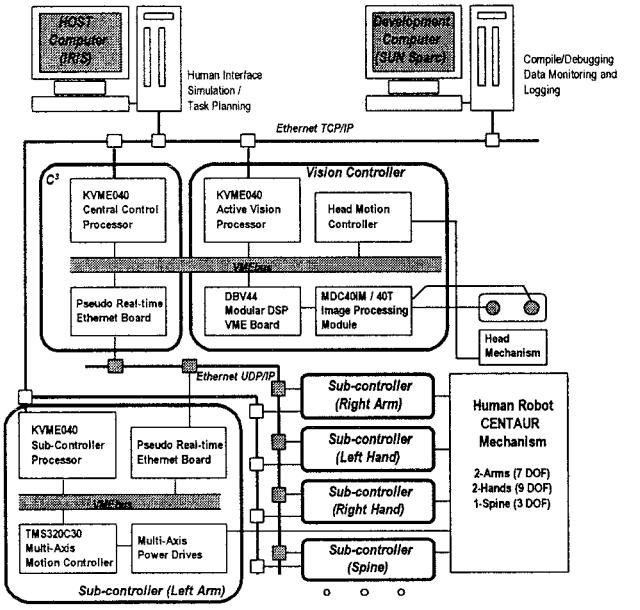

Fig. 2 A hardware structure for the humanoid robot control

4. A Goal-Achieving Software Architecture

4.1 The Overall Control Architecture

According to the definition by Saridis[12], intelligent machines perform anthropomorphic tasks, autonomously or interactively with a human operator in structured or unstructured, familiar or unfamiliar environments. He also defined the structure of intelligent machines to be the hierarchical control systems composed of three levels hierarchically ordered according to the principle of increasing precision with decreasing intelligence, i.e., the highest organization level, the middle coordination level, and the lowest execution level. With similar concept as his intelligent machine structure [12][13], we hierarchically decompose the overall control system of the humanoid robot into four levels, as shown in Fig. 3, namely:

(1) *The organization level* represents the highest level with functions to describe the organization of tasks. When a task is given, the task organizer in this level globally

specifies the task name, geometrical parameters of the objects to be used in the task, and the initial/final configuration of the objects. It also receives feedback from the lower dispatch level responding to the descriptive task commands issued.

(2) *The dispatch level* defines the interface between high and low levels of intelligence with functions to plan the task strategically, generate coordinator-oriented actions and dispatch the decomposed actions to the execution coordinator. The strategic task plan to arrange action commands in sequence/parallel for achieving the given task goal can be made autonomously or interactively with a human operator in structured or unstructured environments. The action dispatcher plans a batch of action commands to achieve a sequentially decomposed sub-task and then transmits it to the coordination level interactively.

(3) *The coordination level* represents the plan-guided reactive control with functions to plan the actions tactically and coordinate the execution level sub-controllers in real time reacting to the environmental change. Action commands for performing manipulation tasks describe the actions to be taken to handle the objects in the task executing environment. (They are typically represented by sensory-motor pairs that may be called 'behaviors') They cannot directly drive the vision/motion sub-controllers, since the sub-controllers do not have full knowledge of the world/object model in their database. Therefore, the execution coordinator once decomposes an action command into several interconnected executive modules off-line and executes the action plan by activating the decomposed executive modules on-line, referring to the world/object model. Primitive modules in a class of executive modules then directly trigger the sensory/motor operations of the vision/motion sub-controllers.

(4) *The execution level* is the lowest level with high requirement of precision with functions dominated by control theory to execute the specified control commands. While the execution coordinator is executing the action plans to achieve the goal, the decomposed primitive modules act as the commands to the corresponding sub-controllers for sensory/motor operations.

4.2 Reactive Control with Executive Modules

Reactivity is one of the most important requirements for designing a humanoid robot control architecture that deals with dynamic environments. Reactivity can be achieved by a set of behaviors triggered by specific events[7], but it should be programmable and controllable since a robot that is only driven by environment stimuli may never achieve its goal [14]. In order to achieve such reactivity, we decompose action commands into three classes of modular units : *primitive modules(PM)* that issue commands to the vision/motion sub-controllers, *flow-control modules(FM)* that control the sequential/ concurrent flow in executing the action plans, *goal modules(GM)* each of which is composed of other GM's, PM's, or FM's. Specifically, goal/primitive modules are designed to have multiple *exit states*, which reflect its internal structure with only one exit state returned for each of its calls, as Wilson's behavioral modules [9]. The concept of the exit states provides our control architecture with alternative execution paths as well as ideal one, reactively driven by the internal or environmental stimuli.

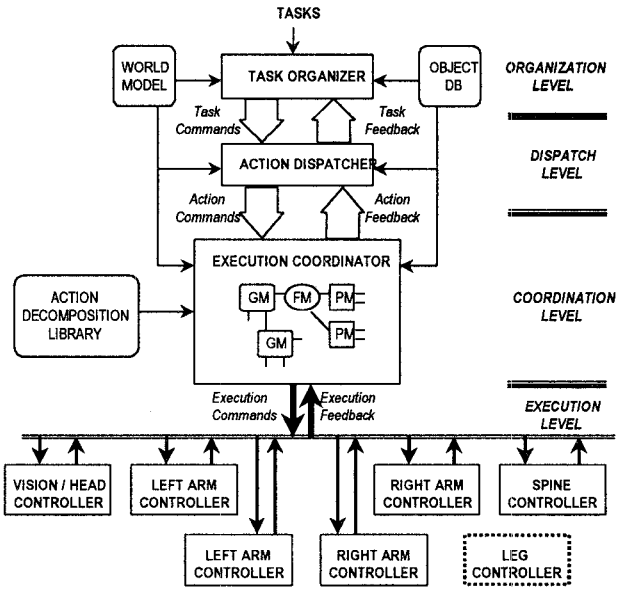

Fig. 3. A control software hierarchy for humanoid robots

(1) Primitive Modules (PM)

One action command typically uses multiple sensors and actuators, but the execution commands to activate the sensors and actuators are assigned to the specified sub-controllers one by one through real-time communication interfaces. Therefore, we define primitive modules to be the modular communication units to activate the corresponding sensory/motor sub-controllers for vision, left arm, left hand, right arm, right hand, or spine. Fig. 4 shows the procedure for executing the command functions designated to the primitive modules.

If a primitive module is activated by the module executor in the execution coordinator, the command is transmitted to the corresponding sub-controller and the sub-controller starts to run. When a motion sub-controller

is triggered, the sub-controller network interface periodically acquire the local sensory data such as the position/orientation of actuator links or force/torque sensory values and upload the data until a termination condition is reached. There are three termination conditions called exit states : success when the command is successfully completed, fail when the command is failed to execute by some local problems, and emergency stop when the execution monitor issues the command to the sub-controller on the emergent condition in C^3. A primitive module then returns one of three exit states: success, fail, and emergency. In the case when the vision sub-controller is triggered, it periodically updates the visual data such as the position/orientation of the object handled before one of the exit states aforementioned is reached. The followings describe representative primitive execution commands for the sub-controllers invoked by the primitive modules:

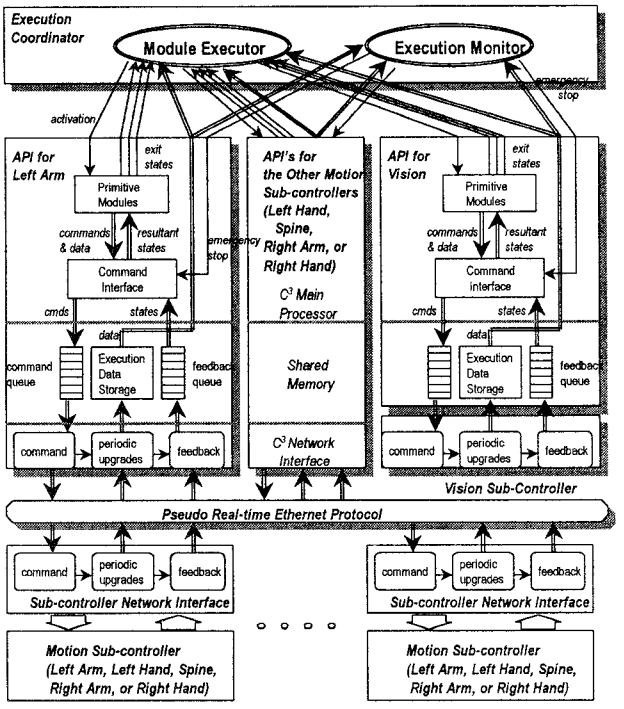

Fig. 4. The execution procedure for command executive modules

the left/right arm sub-controller

SetArmEffectFrame(int ArmID, int EffectID, struct frame Feffect); sets the position/orientation of an effective frame w.r.t. the wrist frame as the *EffectID* number.

MoveArmRelJoint(int ArmID, struct ajoint Jtarget); moves the arm to have the absolute joint values.

MoveArmAbsWorld(int ArmID, struct frame Ftarget); moves the arm to have the relative Cartesian position/orientation specified along with the local trajectory plan.

SetArmControlGain(int ArmID, int Gnum); specifies the arm control gains pointed by the *Gnum*.

the left/right hand sub-controller

InitHand(int HandID); initializes the configuration, where *HandID* specifies the identification number of the hand.

CloseHand(int HandID); closes the hand for power gripping.

PinchHand(int HandID, struct fdirect Dir[Nfinger], int Leng[Nfinger], int Thres[Nfinger]); moves the fingers to the direction until the threshold force/moment are detected or the fingers travel to the maximum length.

the vision sub-controller

MoveHead(int Roll, int Pitch, int Yaw); moves the head to the orientation.

FindObject(struct objectdata Obj); finds the position/orientation of the object center.

the spine sub-controller

InitSpine(); initializes the spine configuration.

MoveSpine(int Roll, int Pitch, int Yaw); moves the spine to the orientation.

(2) Flow-Control Modules (FM)

The primitive/goal modules are driven by the input activation signal and have only one of multiple exit states to be used as the activation signal for the other modules. However, the action commands are typically performed by coordinating multiple sub-controllers in parallel. Therefore, it is necessary to define several modules describing the control flow of executing primitive/goal modules. We define five flow-control modules, namely,

Fork that describes the parallel branches of the program flow,

Join that triggers the output signal if all of the input signals are activated,

Or that triggers the output signal if one of the input signals is activated at least,

Loop N that activates the output L(Loop) repeatedly by N-times or the output E(Exit) on the N-times,

Delay M that activates the output M-tick-time later,

as depicted in Fig. 5.

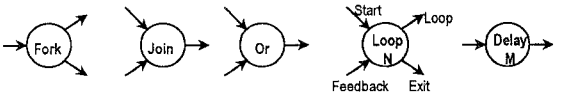

Fig. 5. Flow-Control Modules

(3) Goal Modules (GM)

Built in a hierarchical fashion, each goal module can be composed of any combination of other goal modules, primitive modules, and flow-control modules. The goal modules are designed to ensure that they are correctly adapted to the chosen environment. The variation and uncertainty present in the task to be performed are dealt with inside the module by using variation-reduced strategies devised by the designer and by correctly adapting sensors to the task. Each goal module is activated by signaling the activation input and returns one of the pre-defined exit states determined from the program flow through low-level executive modules. Then, the connections among the exit states, the activation inputs, and input/output of the flow-control modules determine the execution paths.

The ideal execution path and the alternative execution paths are defined to describe the program flows through the three classes of executive modules. The ideal execution path describes the best route, according to the system designer, through the executive modules performing tasks. However, the exit states for a particular goal/primitive module are determined from local data, from sensor data, and from exit states returned from lower level executive modules. One of the exit states may then belong to an alternative execution path designed to be reactive to the internal events or environmental stimuli. Fig.6 illustrates how to configure a goal module to achieve a "pick-up" action goal in a reactive manner. For instance, a goal of recognizing object can be achieved ideally by issuing a vision primitive, FindObj, but an alternative execution path with additional MoveHead primitives can be selected if the object is out of the visual range. In a goal module, Approach, a delayed motion can avoid expected collisions to obstacles or another arm detected by the execution monitor, and a flow-control module, Fork, enables the arm and the hand to move simultaneously.

5. System Integration and Experiments

We have constructed a task control architecture of a humanoid robot which satisfies the requirements of the control hardware/ software aforementioned. The top-down design concept has been implemented in the bottom-up way. Particularly, in the control hardware level, the most important issue is the design of versatile real-time communication interfaces between the central controller and the motion sub-controllers. For this purpose, we developed pseudo real-time Ethernet interfaces which support the extended TCP/IP protocol characterized by the following features: transmission interval limiting, pseudo window size, and periodic transmission mode. The Ethernet interface consists of Motorola 68EN360 CPU, Xlinks FPGA, and high-speed dual-port memory running on the VxWorks real-time operating system kernel.

The primitive module library which implies the application layer of the extended TCP/IP protocol in the central controller was implemented and the real-time performance was verified through several experiments. At the first stage, integrating all of the sub-controllers with the central controller, we successfully operated the humanoid robot by simply issuing the sub-controller execution commands in sequence/parallel. We have been implementing the control architecture for performing toy-block assembly tasks on the real humanoid robot, Centaur, as well as on a graphic simulator in SPARC-20 workstation. Since the TCP/IP protocol is basically used for the execution-level communication, the control architecture implemented on the central controller has been easily tested and modified by connecting with the graphic simulator before the real application to the humanoid robot. We have also implemented our region-based Q-learning method [15] to construct a goal module for visual servoing with learning capability.

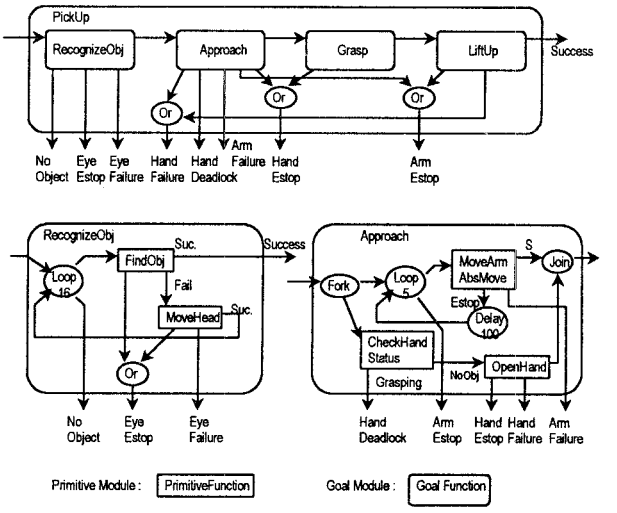

Fig. 6. An example : Construction of Goal Modules

6. Concluding Remarks

Robot control architectures have been putting more emphasis on executing navigation tasks performed by mobile or locomotive robots, but humanoid robots are typically designed to have excellent capability of manipulating objects dexterously rather than the legged

locomotion. Focusing on the manipulation tasks, we first examined the principal hardware/software requirements for the humanoid robot controller, and then designed the controller so as to meet the requirements. The on-board control hardware was designed and implemented to have a distributed control structure which consists of a central controller (brain), a vision controller (eye), and five motion sub-controllers (two arms, two hands, one spine). Pseudo real-time Ethernet interfaces was developed to meet the versatile real-time network requirement. A goal-achieving software architecture was proposed that met the requirements of semi-autonomy, reactivity, expandability, and object-orientation. Specifically, the reactivity was achieved by configuring three kinds of executive modules, primitive module, flow-control module, and goal module, which have multiple exit states. The control architecture proposed has been implemented on the humanoid robot Centaur as well as the graphic simulator.

Acknowledgements

The first author would like to thank Professor Roderic A. Grupen, Co-Director of Laboratory for Perceptual Robotics in the University of Massachusetts at Amherst, for admitting his short term visit for 3 months last summer and giving him several valuable comments on this work.

References

[1] R. Volpe, J. Balaram, T. Ohm, and R. Ivlev, "The Rocky 7 Mars Rover Prototype," *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Osaka, Japan, 1996.

[2] E. Gat, R. Desai, R. Ivlev, J. Loch, and D.P. Miller, "Behavior Control of Robotic Exploration of Planetary Surfaces," *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 4, pp. 490-503, 1994.

[3] R.G. Simmons, "Structured Control for Autonomous Robots," *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 1, pp. 34-43, 1994.

[4] H. Sato, *et. al.*, "Development of a Model-Based Remote Maintenance Robot System (I)(II)(III)," *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pp. 1225-1244, 1993.

[5] I. Kato, T. Matsuno, and A. Takanishi, "Development of an Anthropomorphic Head-Eye Robot with Two Eyes-Coordinated Head-Eye Motion and Pursuing in the Depth Direction," *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1997.

[6] Matthew M. Williamson, "Postural Primitives: Interactive Behavior for a Humanoid Robot Arm," *Proc. of Int'l Conf. on Simulation of Adaptive Behavior (SAB'96)*, Cape Cod, MA, USA, 1996.

[7] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23, 1986.

[8] Jonathan H. Connell, "A Behavior-Based Arm Controller," *IEEE Trans. on Robotics and Automation*, Vol.5, No. 6, pp. 784-791, 1989.

[9] Myra S. Wilson, "Behaviour-Based Robotic Assembly Systems: Reliability of Behavioural Modules," *Proc. of the 23-rd ISIR Conference*, Barcelona, Spain, 1992.

[10] A. Lush, J. Rowland and M. Wilson, "Characteristics of Robot Behaviour," *Advanced Robotics & Intelligent Machines, IEE Control Engineering Series 51,* Chapter 18, pp. 281-293, 1996.

[11] R.A. Grupen, M. Huber, J.A. Coelho, and K. Souccar, "Distributed Control Representation for Manipulation Tasks," *IEEE Expert*, pp. 9-16, Vol.10, No.2, 1995.

[12] G.N. Saridis and K. P. Valavanis, "Analytical Design of Intelligent Machines," *Automatica*, Vol. 24, No. 2, pp. 123-133, 1988.

[13] F.-Y. Wang and G.N. Saridis, "Task Translation and Integration Specification in Intelligent Machines," *IEEE Trans. on Robotics and Automation*, Vol. 9, No. 3, pp. 257-271, 1993.

[14] F.R. Noreils and R.G. Chatila, "Plan Execution Monitoring and Control Architecture for Mobile Robots," *IEEE Trans. on Robotics and Automation*, Vol. 11, No. 2, pp. 257-271, 1995.

[15] J.H. Kim, I.H. Suh, S.R. Oh, Y.J. Cho, and Y.K. Chung, "Region-Based Q-learning using Convex Clustering Approach," *Proc. of IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, Vol.2, pp. 601-607, 1997.