///VSP///   *Advanced Robotics 22 (2008) 1397–1420*   **ADVANCED ROBOTICS** www.brill.nl/ar

Full paper

Mobile Robot Control Architecture for Reflexive Avoidance of Moving Obstacles

Keum-Shik Hong [a,*], **Tua Agustinus Tamba** [a] **and Jae-Bok Song** [b]

[a] School of Mechanical Engineering, Pusan National University; 30 Jangjeon-dong, Gumjeong-gu, Busan 609-735, South Korea

[b] Department of Mechanical Engineering, Korea University, Anam-dong, Seongbuk-gu, Seoul 136-713, South Korea

Received 25 May 2007; revised 15 February 2008; accepted 7 April 2008

Abstract

In this paper, a three-layer (deliberative, sequencing, reflexive) architecture is adopted and the structure of the reflexive layer is discussed. The objective of this architecture is to extract the basic actions that require hard-real-time execution from non-real-time-allowed behaviors by separating them into the reflexive and sequencing layers, respectively. The reflexive layer consists of resources, actions, an action coordinator and a motion controller. To guarantee the hard-real-time execution, a set of simple actions and an action coordinator are designed using the functions provided in the RTAI (Real-Time Application Interface for Linux) environment. Also, an obstacle avoidance algorithm based upon data from a laser range scanner is developed. For the purpose of avoiding a moving obstacle, which is treated as a moving circle through segmentation and circularization processes, a Kalman filter is developed to estimate the distance and the heading of the center of the moving circle. The effectiveness and real-time characteristics of the proposed reflexive layer and the developed algorithms are examined through experiments using scattered stand-still obstacles as well as a moving human.

Keywords

Behavior based, control architecture, mobile robot, obstacle avoidance, real time control

1. Introduction

The necessity for productivity improvement has motivated the development of various industrial robots, whereas the desire of human beings for convenience and happiness has motivated the development of human-friendly robots. It can be said that the coexistence of humans and robots in a ubiquitous environment is not far

[*] To whom correspondence should be addressed. E-mail: kshong@pusan.ac.kr

away. As human expectations for robots grow, the hardware of robots becomes more complicated. Most of all, the system integration of various software components as well as diverse hardware also becomes more important, because software for navigation, manipulation, voice recognition, vision, obstacle avoidance and other tasks has to be integrated in one platform. For this purpose, a comprehensive architecture that assures an easy integration of various kinds of hardware and software is required. After an operating system (OS) is selected (which may depend on hardware requirements), a software framework should be designed according to the OS's characteristics. Such a framework that defines protocols, standards, expansions and requirements is called the control architecture. As will be discussed, it is difficult for the existing control architectures to fulfill the demanding performance requirements that have arisen lately and it is, therefore, necessary to introduce a new concept of control architecture to satisfy the various functions required.

In the early 1980s, the sense–plan–act structure of batch processing was the prevailing structure in machines and robots. In the batch processing method, the major concerns were how to design the planning part and how to model the environment surrounding the robot. The shortcomings of such a method are that a great deal of knowledge about the working environment is necessary in advance to design a stable planning part, and that it is difficult to cope effectively with uncertain and unpredictable actual environments [1, 2].

The control architectures in the 1980s were characterized by a layered architecture represented by Brooks' subsumption architecture [3, 4] and a behavior-based architecture represented by Arkin's motor schema [1, 2]. As a complicated task is arranged to be executed in the order of a prescribed plan, a layered control architecture has been widely applied in the automation of industrial robots. The subsumption architecture modeled after the reactive animal behaviors classifies the components of a robot system by ability, unlike the existing control methods that classify the components by function. The shortcomings of the subsumption architecture were that it failed to put forward mechanisms that could efficiently deal with a growing number of layers. In addition, it was difficult for the architecture to carry out highly sophisticated works because it failed to express the surrounding environment systematically and to diagnose and correct errors [4]. Arkin's behavior-based control architecture can be represented as a combination of behaviors that perceives specific sensor information as stimuli and react to them. This control architecture has been built to generate the movements of a robot.

Not only to complement the above shortcomings, but also to meet the varied performance requirements of complex hardware, studies have been conducted from the early 1990s on a hybrid control architecture that is characterized by (i) a layered structure, (ii) behavior-based controls and (iii) a scheme of combining competitive and cooperative coordinators [5–11]. A competitive coordinator drives a robot by selecting the most appropriate behavior from a number of candidate behaviors, whereas a cooperative coordinator drives a robot by combining necessary behaviors into a new one.

Examples of adopting a hybrid control architecture include 4D/RCS, developed by NIST, which was applied to unmanned military vehicles [8], and CLARAty, jointly developed by NASA, Caltech and Carnegie Mellon University that was applied to a Mars exploration robot [10]. These control architectures were intended to be used outdoors and, therefore, their design focused on securing stability, resulting in complicated structures and heavy dependence on a database. While the architectures of 4D/RCS and CLARAty were focused on navigation, the control architecture adopted for Care-O-bot [7], a robot developed by Frauhhofer IPA in Germany for the elderly and handicapped people, was intended for simultaneous manipulation of robot arms as well as navigation of a mobile platform. The typical three-layer architecture BERRA, developed by the Royal Technology Academy of Sweden for a museum guide robot, has been widely adopted and it influences the majority of traveling robots currently under development [6]. However, the roles of each layer in BERRA are ambiguous and they fail to fully realize the characteristics of behavior-based control. In addition, the control architecture developed for O2CA2, an undersea exploration robot, has emphasized the importance of a reactive layer in their object-oriented control architecture [12]. However, most of these architectures have the shortcoming that all situations should be constructed in a database, or predicted, in order for the robot to move autonomously under the control architecture [13, 14].

However, such conventional architectures, whose main focus lies in the realization of algorithms using various sensors, fail to characterize the imperativeness of matters and their inability to establish priorities causes them to perform poorly in real-time operations. Even though many previous works tried to construct software components using an object-oriented approach, the relationship and data flow between layers and components were not distinctive. In particular, actions that require real-time execution during navigation, manipulation or even during human–robot interaction were not identified at all. Also, the functions in the lowest layer were not designed to perform reflexive actions in addition to carrying out the orders from the upper layer. Most of all, as robot behaviors evolve, the separation (making independent) of the growing diversity of new behaviors from the basic motions of the robot is the objective of the proposed architecture.

In this paper, the roles of a reflexive (lowest) layer under the umbrella of a three-layer control architecture are characterized. The use of four components in the reflexive layer and their real-time implementation in the Linux RTAI (Real-Time Application Interface) kernel are proposed. As basic software components for the robot's movement, four actions are designed. For avoiding scattered motionless obstacles as well as moving ones, detection and avoidance algorithms are developed too. Since a two-dimensional (2-D) laser scanner is used as a sensor for detection, the moving obstacle is assumed to be a circle, featured by its center and radius, after segmentation and circularization processes. Then, a Kalman filter for estimating the distance and orientation to the center of the moving circle together with their time-rate of change is designed. To test the effectiveness of the proposed reflexive layer and designed components, the first version of the SILBO robot developed by KIST is used in experiments.

The novelty of this paper is the proposition of an architecture that separate the basic actions (*Move*, *Goto*, *Avoid*, *E-stop*, etc.) from the more complicated behaviors. As the navigation module in the sequencing layer can be improved, modified or replaced regardless of the change or variations of sensor update rate in the reflexive layer, the overall structure and individual algorithms in the deliberative and sequencing layers are independent of the sensor update rate and other uncertainties. The performance of the actions in the reflexive layer is hardware dependent in the RTAI environment, whereas the performance of the behaviors in the sequencing layer is not only hardware dependent but also software dependent in the Linux environment.

2. Robot Configuration and Architecture

2.1. Hardware Configuration

The autonomous robot is a robot that is capable of self-judgment and independent navigation in an unknown environment [15]. Figure 1 shows the first version of SILBO used for experiments in this study, which was developed by the 21st Century Frontier R&D Program, KIST, South Korea, in 2004. The robot has two driving wheels (left and right) and two auxiliary wheels (front and rear). Each driving wheel is equipped with a DC motor powered by two 12-V batteries, a 30:1 reduction gear and an encoder. The maximum speed of the robot is 0.5 m/s. Figure 2 depicts the overall hardware configuration of the robot, which includes a Pentium IV 2.2-GHz single-board computer with a PCI digital/analog interface card (Commell), a serial multi-8 cable (Kicom) and a DAQ board (NI).

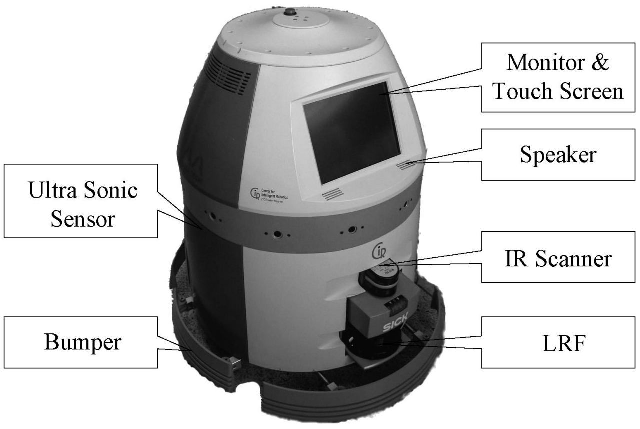


Figure 1. Mobile platform of SILBO developed by the 21st Century Frontier R&D Program, KIST, South Korea, in 2004.

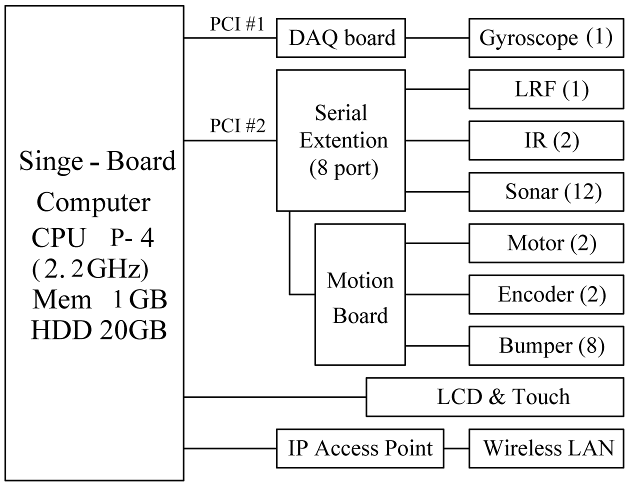


Figure 2. Hardware configuration of the mobile platform of SILBO in Fig. 1.

The sensors that are used are the following. One LM200-30106 (SICK) 2-D laser range finder (LRF) sensor is attached on the front side. The scanning ranges and resolutions are 180° (max)/100° (min) and 0.25/0.5/1°, respectively, and it can measure up to 30 m away with an accuracy of ±15 mm. Two PBS-03JN (HOKUYO) infrared ranger (IR) sensors (front and rear) are attached. The scanning range, resolution and distance are 180°, 1.8° and 3 m (max)/2 m (min), respectively. Twelve PS40S ultrasonic sensors (Nicera) are installed around the robot at intervals of 30°. With a PCI interface card, the distance between the robot and a wall or obstacle is detected by measuring the time interval from the transmission of ultrasonic waves to their returns at each sampling time. For dead-reckoning, an incremental-type encoder is built into each of the two DC motors. Using the encoder counter in the PCI interface card, the moving direction and velocity of the robot are updated. A 3Com wired and wireless IP router is used for linking out-world systems including the host computer. Using this router, control commands can be delivered to the robot or the internal status of the robot can be monitored. Also, remote modification and execution of control programs are possible.

2.2. Software Structure

It is important to construct software architecture in such a way that a part of the software can be replaced or extended. It is also required to have object-oriented characteristics for exchanging data among functions [16–18]. The tasks to be accomplished may have different cycle times, priorities and execution times, and some tasks may require real-time execution in relation to getting data from the sensors and driving the motors. Recently, the coordination of proper actions or behaviors among available candidates by assigning priority in real-time has drawn attention among researchers. Also, since not all programs can be executed in real-time, designing a software architecture that can distinguish the imperativeness of the events in an unknown environment becomes critical. Accordingly, real-time tasks like motor control, sensing and obstacle avoidance become more important for the improvement of autonomy in robotics [19–21]. In this respect, this study uses RedHat Linux v9.0 patched with RTAI v3.1 for Linux [22, 23].

The RTAI was initiated from the Dipartimento di Ingegneria Aeros-paziale del Politecnico di Milano to provide real-time functions to Linux in 1996. The advantage of RTAI lies in the fact that it provides real-time patches for a new kernel version more quickly than RT-Linux. Most of all, it is open to the public. In addition, it is possible to develop software that enables high-speed real-time task actions because the RTAI offers a real-time application program development function in the user domain. In addition, with the RTAI, multitasking is possible; all Linux application programs and functions can be used; and standard Linux drivers can be used as they are. The RTAI has two unique features: a real-time scheduler and an interrupt handler. For the inter-process communications offered by RTAI, the RT-FIFO offers real-time message communication handling functions between the user and kernel levels. The multiple-models approach assumes that a model can immediately capture the complex system behavior better than others.

2.3. Architecture Specification

Table 1 compares the specifications of the adopted architecture in this paper with others available in the literature: Saphira 6.2 [24], Teambots 2.0 [25] and ISR BERRA [6]. The main differences are: the real-time OS, RTAI 3.1, is adopted; the programming language in the kernel is C, but that in the user level is C++; a variety of sensors can be easily attached; various types of action coordinator can be applied; the sensor actuator latency in the kernel is 0.02 s, whereas that at the user level is 0.2 s.

3. Reflexive Layer

The control concept can be divided into two categories, i.e., deliberate control and reflexive control, depending on how the changes in the surrounding environment are incorporated in determining the behavior of the robot. As a process of ‘recognition and decision’, the deliberate control generates a series of tasks (plans) by using the knowledge and information obtained, and it enforces the consecutive fulfillment of the scheduled tasks. Hence, the deliberate control is effective in determining the tasks in the upper level, in that complex tasks can be optimized in some sense before their execution. The reflexive control, however, readily responds to the surrounding environment rather than thinking and getting the most optimal answer. In this respect, reflexive control is effective in determining the behaviors in the lower level.

Figure 3 shows the control architecture adopted in this paper, which has three layers. Briefly explaining individual layers, the deliberative layer plays the role of converting the robot’s perception of a human into computer language and *vice versa*, so that the human’s orders can be carried out by the robot. It also performs

Table 1.
Comparison of four architectures[a]

	Saphira 6.2	TeamBots 2.0	BERRA 2.0	SILBO v0.74
OS				
Linux	yes	yes	yes	yes (Redhat 9.0)
MS Windows	yes	yes	no	no
Real-time OS	no	no	no	yes (RTAI 3.1)
Layers	single	two	three	three
Main language	C	Java	C++	C/C++
Software tools requirement	gcc Motif	Java 1.2	gcc 2.95	gcc 3.2
Graphics				
GUI	yes	yes	yes	yes
graphics program	yes (limited)	yes	no	yes
HRI				
text	yes	yes	yes	yes
speech	no	no	yes	yes
GUI	yes	no	yes	yes
palm	no	no	yes	yes
Multi-agent support	no	yes	no	yes
Multi-host	no	no	yes	yes
Multi-process	threads	thread	multi-process	multi-process/thread
Data flow paradigm				
push	yes	no	yes	yes
pull	yes	yes	yes	yes
Platform portability				
hardware abstraction	good	very good	very good	very good
sensor extension capability	no	good	very good	very good
Sensor support				
sonar	yes	yes	yes	yes
LRF	yes	no	yes	yes
IR	no	no	yes	yes
camera	yes	yes	yes	yes
bumper	yes	yes	yes	yes
microphone	no	no	no	yes
gyro	no	no	no	yes
touch-screen	no	no	no	yes
Behavior coordinator	fuzzy logic	various	vector/histogram	action coordinator
Timing aspects				
real-time support	no	no	no	yes
sensor actuator latency	0.6 s	0.4 s	0.17 s	0.02/0.2 s
Bandwidth				
sensor to behavior	4 kB/s	OS limitation	OS limitation	OS limitation
Code size				
complete system	11 MB	45 MB	36 MB	5.82 MB

[a] The specifications of Saphira 6.2, TeamBots 2.0 and BERRA 2.0 are adopted from Oreback and Christensen [11].

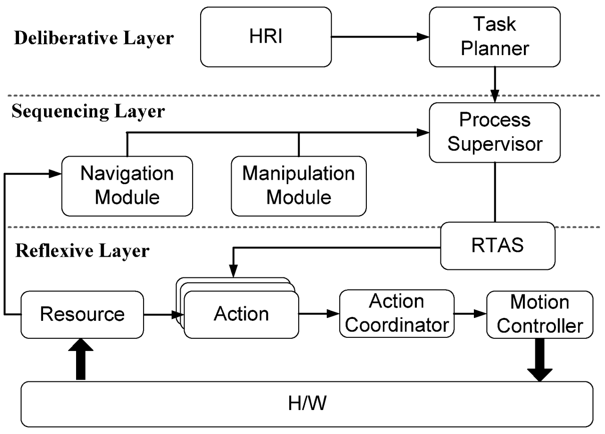


Figure 3. Adopted three-layer control architecture, SILBO v0.74.

the role of planning and dividing the arranged tasks into two categories: navigation and manipulation. The deliberative layer consists of a human–robot interface (HRI) and a planner.

The sequencing layer generates consecutive orders to perform the tasks generated in the deliberative layer by processing the acquired information from sensors. This layer consists of three modules: a navigation module, a manipulation module and a process supervisor module. The navigation module uses sensor data information, and runs various algorithms for self-position recognition, environmental map building, path generation and via-point calculation. The navigation module consists of three submodules: a map, a localizer and a path planner (Fig. 4). For map creation, the occupancy grid map method is used; for localization, the Monte Carlo algorithm is adopted; and for path planning, the A* search algorithm is employed [12, 26–28]. However, both deliberative and sequencing layers are not discussed due to space limitations.

The reflexive layer, which is the main focus of this paper, carries out reflexive actions as well as the orders from the upper layer, i.e., the orders from the process supervisor. This layer consists of resources, actions, an action-coordinator, motion controllers and the real-time action supervisor (RTAS). To guarantee the safety of humans and the robot, this layer repeatedly performs simple computations periodically or non-periodically to control the movements of the robot, and pursues obstacle avoidance in real-time in a varying environment. It sometimes combines a number of actions to generate smoother movements of the robot. As seen in Fig. 4, the programs/codes belonging to the deliberative and sequencing layers are run in Linux, but those belonging to the reflexive layer are run in the RTAI kernel.

3.1. Resource

A resource represents a shared memory in which the data acquired from a sensor is stored. Each resource has its own corresponding sensor, and the stored information

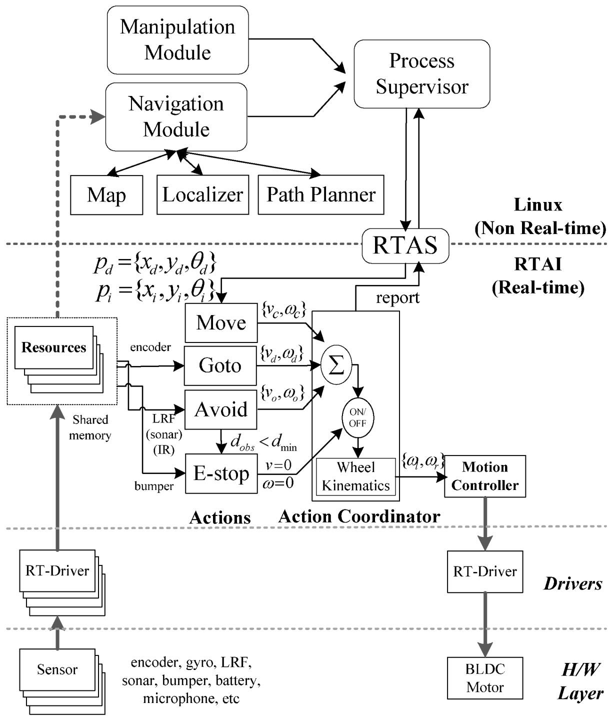


Figure 4. Structure of the reflexive layer.

is supplied to various actions in the reflexive layer and to the components in the upper-level sequencing and deliberative layers, if necessary. Resources can be classified into two types: a polling type and an interrupt type. To generate periodic data, the polling type uses a periodic thread provided in the RTAI. Using the functions provided in real-time device drivers, resources keep the data acquired from sensors in the shared memory. The polling type resources include the data from sonar, LRF, IR, encoder, gyro and the state of charge in the battery.

On the other hand, the interrupt type returns sensor data in response to interrupt signals from the motion board. When an outside interrupt is generated, the corresponding resource reads the sensor data from the real-time device driver and transmits the data to the shared memory. The RT-driver reads the data through the interrupt and offers a callback function as a way to inform the RT-thread of the data. Bumper signals are the interrupt type.

3.2. Action

An action is the smallest independent software component that realizes a specific function. Four actions (*Goto*, *Move*, *Avoid*, *E-stop*) are developed. Individual actions calculate the desired linear and angular velocities of the robot, based upon the current and target configurations of the robot and by using the available resources. Some action may fuse several pieces of information from various resources (experiments with the use of the LRF resource are performed in this paper). A series of actions should be designed in order for the robot to demonstrate a stable and smooth maneuver in a complex environment. The codes of the actions should be simple in order to minimize the computation time in the kernel of RTAI. Briefly outlining their roles, the *Goto* makes the robot advance to the next via-point in the robot's path in compliance with the command from the navigation module, the *Avoid* makes the robot react against something near to it, the *Move* lets the robot go at a constant linear/angular velocity and the *E-stop* makes the robot stop immediately.

3.2.1. Goto

Goto is an action that moves the robot from the current position to a destination position by receiving via-points in the route assigned from the process supervisor. The execution cycle of a *Goto* (20 ms) is faster than the update cycle in the sequencing layer (200 ms). Therefore, it updates the current position information using the encoder until it receives the correct position information from the localizer. Upon calculating the current position, the linear and angular velocities are calculated and delivered to the action coordinator in a consecutive manner. In this sense, the *Goto* performs as a local localizer, because it updates the current position every 20 ms until a correct global coordinate is reassigned from the sequencing layer. The robot's linear and angular velocities, v and ω, are generated as [29]:

$$v = (k_1 \cos\delta)\rho, \quad k_1 > 0 \tag{1}$$

$$\omega = k_4\delta + k_3 \frac{\cos\delta \sin\delta}{\delta}(\delta + k_2\theta_\mathrm{d}), \tag{2}$$

where ρ is the distance from the current position to the target position, δ is the angle between the robot principal axis and the distance vector ρ, θ_d is the desired orientation of the robot, and $k_i > 0, i = 1, \ldots, 4$, are the control gains.

3.2.2. Avoid

If the detected obstacle is motionless or if its moving region is far away from the current position, then it can be circumvented by planning a new path. Hence, the *Avoid* action considered in the reflexive layer should be a reflexive action. Prior to avoiding obstacles, it is necessary to identify the dynamic characteristics of the obstacles. To avoid stand-still obstacles, it is necessary to consider only the distance to the obstacle, but to avoid a moving obstacle, it is necessary to take into account the relative movement between the robot and obstacle. The designed *Avoid* action in this paper generates the linear and angular velocities of the robot as functions of the heading of the robot, as well as the distance to and the size of the detected obstacle. Detailed algorithms will be discussed in Sections 4 and 5.

3.2.3. Move

The *Move* is introduced to control the velocity of the robot, not to control its positions. In *Goto* or *Avoid*, the robot's velocity is not constant throughout the movement in general. However, in the cases of object tracking or wall following, it might be necessary for the robot to move at some constant speeds. In this case, controlling its velocity is easier than controlling its position. The *Move* delivers the velocity command from the RTAS to the action coordinator directly.

3.2.4. E-Stop

The *E-stop* means an emergency stop of the robot using mechanical brakes. Two kinds of *E-stop* can be considered: a hard *E-stop* and a soft *E-stop*. The hard *E-stop* is activated when a bumper of the robot collides with an object, whereas a soft *E-stop* is activated when the distance between the robot and an obstacle becomes smaller than a certain value. When an *E-stop* is activated, the robot generates an event that requests the navigation module in the upper sequencing layer to provide a new route to reach the target point.

3.3. Action Coordinator

The action coordinator fuses (mixes) the outputs from activated actions in an effective manner and delivers the fused outcome to the motion controller so that a variety of maneuvers can be achieved. The action coordinator should be able to carry out non-cyclical, as well as cyclical actions. This is because most actions are cyclical, but in some cases non-cyclical actions can also become involved in the input. Two roles of the action coordinator are (i) to deliver the output from an action to the motion controller either as it is or in the form of a weighted value after fusing several outputs, and (ii) to monitor the events occurring in the reflexive layer and report them to the process manager in the sequencing layer.

The following should be incorporated when designing an action coordinator:

(i) For given information from upper layers and resources, the action coordinator should be able to determine appropriate actions that are most suitable. For instance, in the case of *Goto*, the movement of the robot is made in accordance to via-points that require position control, but in the case of *Move*, the movement can be made through velocity control, not through position control.

(ii) The concepts of Arkin's competitive coordinator and Brooks' cooperative coordinator should be used properly in order to generate the robot's movements that can efficiently cope with all situations with a few simple actions.

In this paper, the simplest coordinating algorithm, i.e., an algorithm that switches in varying situations, is tested in Section 6. Figure 5 shows the data flow in the action coordinator.

3.4. Motion Controller

The motion controller is a component that actually delivers the control signals to the motion board. It receives the velocity commands of both wheels, converts them into proper signals to the actuator and delivers them to the two motor drivers. The motion controller also uses the functions provided by RTAI and is realized as a real-time thread.

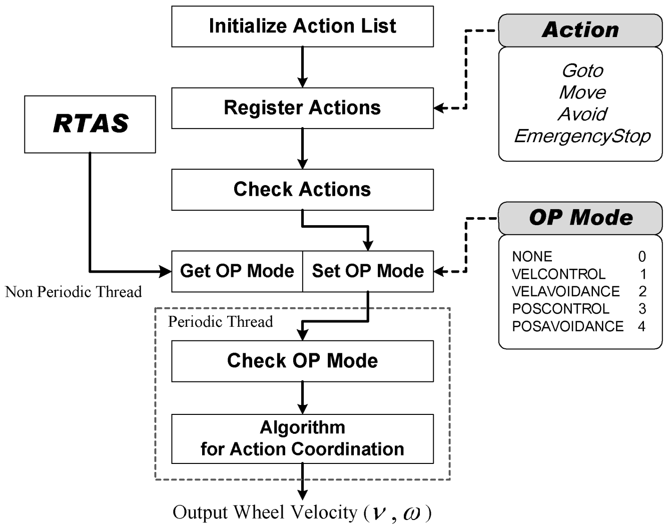


Figure 5. Flow chart in the action coordinator.

3.5. RTAS

The RTAS is a component in the reflexive layer that manages resources and actions. It offers a variety of communication methods among hardware and components in the reflexive layer. As a part that connects the user space to the kernel space, the RTAS also designates the operation modes of actions, and provides the position and velocity information of the robot to the actions. In addition, it communicates with other components in Linux and enables smooth integration of software components and, to this end, two RT-FIFOs are employed. One of two RT-FIFOs is used to transmit the orders in real-time and the other is used to convey the status in the reflexive layer (success or failure) to the sequencing layer. In this study, the RT-FIFOs are realized in the GUI environment and are used to evaluate the traveling control performance.

4. Obstacle Detection

Before activating an *Avoid* action, the robot needs to clearly identify the dynamic status of the obstacle. A stand-still obstacle can be recognized as a part of the environment. A moving obstacle can be avoided promptly only when the velocity and direction of the object are identified precisely. Also, even when the robot detects a stand-still obstacle, it still tries to avoid it if the obstacle is on the way to its destination. Accordingly, in this section, a method that identifies a moving obstacle using a LRF sensor in a cluttered environment is investigated.

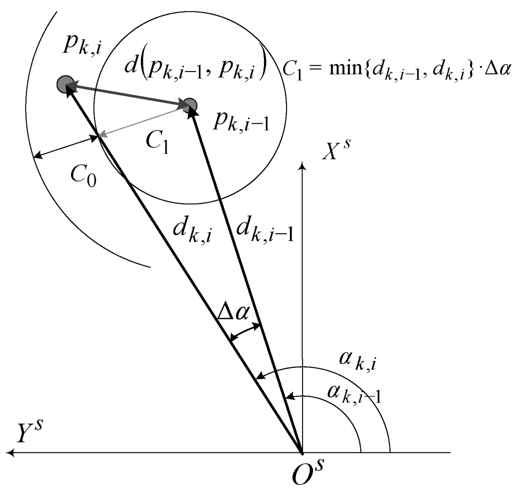


Figure 6. Segmentation scheme.

4.1. LRF Data Preprocessing

Here, the segmentation and circularization procedures for obtaining the center and the radius of a circle, which are regarded as the moving obstacle, are first discussed. Second, using the obtained center and radius information, a Kalman filter for estimating the distance and orientation to the obstacle from the LRF sensor (together with their time rate of change) is developed.

4.1.1. Segmentation

The information about the environment that surrounds the robot is captured with only a finite number of segmented data points. For instance, the LMS 200 LRF sensor that scans the forward 180° with interval of 0.5° in 20 ms generates 361 data points. Therefore, it is necessary to tell whether two neighboring points in the scanned data originated from the same object or not.

Consider the segmentation scheme depicted in Fig. 6. Let LRF_k represent the LRF data at time k as follows:

$$LRF_k = \{p_{k,i} = (d_{k,i}, \alpha_{k,i}) : \alpha_{k,i} = i \times \Delta\alpha, i = 0, 1, 2, \ldots, 360, \Delta\alpha = 0.5°\}, \tag{3}$$

where LRF_k represents data points in the polar coordinate, the subscript i represents the ith point, $\Delta\alpha$ is the scanning resolution and $d_{k,i}$ is the distance from the LRF origin to point $p_{k,i}$ at an angle $\alpha_{k,i}$ from the horizontal axis at time k. If no obstacle is detected, then a set-value, $d_{\max}$, is assigned (in this paper, $d_{\max} = 1000$ mm).

For two given neighboring points at time k, i.e., $p_{k,i-1} = (d_{k,i-1}, \alpha_{k,i-1})$ and $p_{k,i} = (d_{k,i}, \alpha_{k,i})$, it is necessary to tell whether or not the two points come from one object. Let $d(p_{k,i-1}, p_{k,i})$ be the distance between the two points. Let d_{seg} be the segmentation index defined as:

$$d_{\text{seg}} = C_1 + C_0 = \min\{d_{k,i-1}, d_{k,i}\} \times \Delta\alpha + C_0, \tag{4}$$

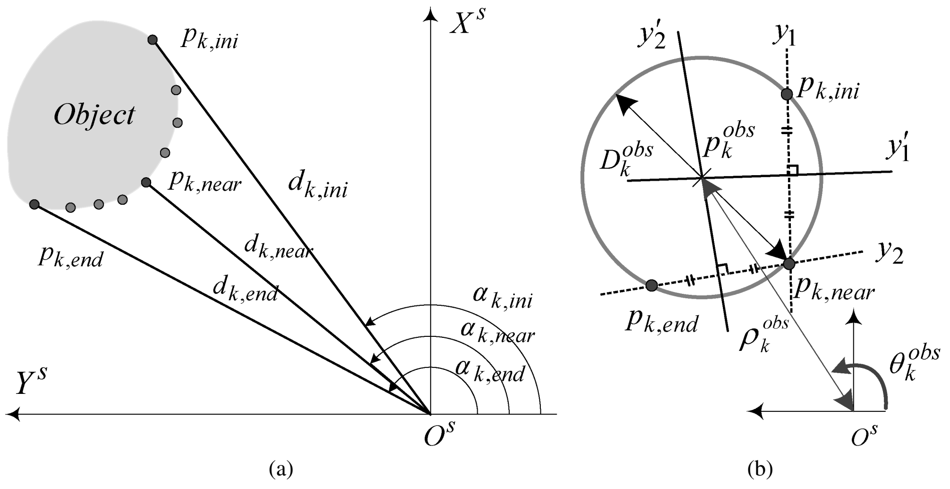


Figure 7. Detected obstacle assumed as a circle (center and radius). (a) Selection of three points from a set of data reflected from an object. (b) Circularization of the obstacle using three selected points.

where $C_1 = \min\{d_{k,i-1}, d_{k,i}\} \times \Delta\alpha$ and C_0 is the segmentation threshold, which is a design parameter ($C_0 = 20$ mm in this work). If $d(p_{k,i-1}, p_{k,i}) \le d_{\text{seg}}$, then we conclude that $p_{k,i-1}$ and $p_{k,i}$ belong to the same object. However, if $d(p_{k,i-1}, p_{k,i}) > d_{\text{seg}}$, then we conclude that $p_{k,i-1}$ and $p_{k,i}$ do not belong to the same object, which implies that there exists an open space between two points and therefore the robot can search for a possible passage to go through. If C_0 is too large, the segmentation becomes too conservative and the robot might think that there is no passage, even though there actually exists sufficient space for the robot to go through.

4.1.2. Circularization Using Three Selected Points

Once the segmentation process is over, we can extract a group of data originating from the same object. Figure 7a shows such a data set. From the set, three points (i.e., the right and left extreme points, and the nearest point to the robot) are extracted, which are used for the circularization purpose in obtaining the center and radius of a circle under the assumption that the obstacle is circular. The circularization process is necessary because a material point $P_{k,\text{near}}$ measured at time k does not, in general, correspond to the same material point $P_{k-1,\text{near}}$ measured at time $k - 1$. Therefore, a fictitious geometrical center of the obstacle, rather than a real material point, is considered for estimating the moving speed and direction of the obstacle.

Figure 7b shows the situation in which an obstacle is detected in the left-hand side of the robot's movement. Let $p_{k,\text{ini}}$, $p_{k,\text{end}}$ and $p_{k,\text{near}}$ be the two extreme points and the nearest point, respectively. Then, the x- and y-coordinates of the three points in the LRF sensor coordinate are:

$$p_{k,\text{ini}} = (x_{k,\text{ini}}, y_{k,\text{ini}}) = (d_{k,\text{ini}} \times \cos\alpha_{k,\text{ini}}, d_{k,\text{ini}} \times \sin\alpha_{k,\text{ini}}) \tag{5}$$

$$p_{k,\text{near}} = (x_{k,\text{near}}, y_{k,\text{near}}) = (d_{k,\text{near}} \times \cos\alpha_{k,\text{near}}, d_{k,\text{near}} \times \sin\alpha_{k,\text{near}}) \tag{6}$$

$$p_{k,\mathrm{end}} = (x_{k,\mathrm{end}}, y_{k,\mathrm{end}}) = (d_{k,\mathrm{end}} \times \cos\alpha_{k,\mathrm{end}}, d_{k,\mathrm{end}} \times \sin\alpha_{k,\mathrm{end}}), \tag{7}$$

respectively. Then, two straight lines passing through $\{p_{k,\mathrm{ini}}, p_{k,\mathrm{near}}\}$ and $\{p_{k,\mathrm{near}}, p_{k,\mathrm{end}}\}$ are as follows:

$$y_1 = m_1(x_1 - x_{k,\mathrm{ini}}) + y_{k,\mathrm{ini}} \tag{8}$$

$$y_2 = m_2(x_2 - x_{k,\mathrm{near}}) + y_{k,\mathrm{neari}}, \tag{9}$$

where $m_1 = (y_{k,\mathrm{near}} - y_{k,\mathrm{ini}})/(x_{k,\mathrm{near}} - x_{k,\mathrm{ini}})$ and $m_2 = (y_{k,\mathrm{end}} - y_{k,\mathrm{near}})/(x_{k,\mathrm{end}} - x_{k,\mathrm{near}})$ represent the slopes of individual lines, respectively. Also, the two vertical bisectors of (8) and (9) are

$$y_1' = -\frac{1}{m_1}\left(x_1' - \frac{x_{k,\mathrm{ini}} + x_{k,\mathrm{near}}}{2}\right) + \frac{y_{k,\mathrm{ini}} + y_{k,\mathrm{near}}}{2} \tag{10}$$

$$y_2' = -\frac{1}{m_2}\left(x_2' - \frac{x_{k,\mathrm{near}} + x_{k,\mathrm{end}}}{2}\right) + \frac{y_{k,\mathrm{near}} + y_{k,\mathrm{end}}}{2}. \tag{11}$$

Then, the coordinates of the intersection point of (10) and (11) become

$$x_k^{\mathrm{obs}} = \frac{m_1 m_2(y_{k,\mathrm{ini}} - y_{k,\mathrm{end}}) + m_2(x_{k,\mathrm{ini}} + x_{k,\mathrm{near}}) - m_1(x_{k,\mathrm{near}} + x_{k,\mathrm{end}})}{2(m_2 - m_1)} \tag{12}$$

$$y_{\mathrm{k}}^{\mathrm{obs}} = y_1'|_{x_k^{\mathrm{obs}}} \quad (\text{or } y_2'|_{x_k^{\mathrm{obs}}}). \tag{13}$$

Hence, the distance and orientation from the origin O^{s} of the LRF sensor to the center O_k^{obs} of the obstacle are:

$$\rho_k^{\mathrm{obs}} = \sqrt{(x_k^{\mathrm{obs}})^2 + (y_k^{\mathrm{obs}})^2} \tag{14}$$

$$\theta_k^{\mathrm{obs}} = \tan^{-1}(y_k^{\mathrm{obs}}/x_k^{\mathrm{obs}}). \tag{15}$$

Furthermore, if using two points, e.g., O_k^{obs} and $p_{k,\mathrm{near}}$, the diameter of the obstacle can be estimated as:

$$D_k^{\mathrm{obs}} = 2 \times \sqrt{(x_{k,\mathrm{near}} - x_k^{\mathrm{obs}})^2 + (y_{k,\mathrm{near}} - y_k^{\mathrm{obs}})^2}. \tag{16}$$

4.2. Estimated Distance and Orientation to the Obstacle and Their Rate of Change

Here, the Kalman filter approach is applied for estimating the distance and orientation to the moving obstacle from the sensor origin O^{s}. The purposes are (i) to filter out the noises included in (14) and (15), and (ii) to obtain their time rate of change.

Let $z_k^1 = [\,\rho_k^{\mathrm{obs}} \quad \dot{\rho}_k^{\mathrm{obs}}\,]^{\mathrm{T}}$ and $z_k^2 = [\,\theta_k^{\mathrm{obs}} \quad \dot{\theta}_k^{\mathrm{obs}}\,]^{\mathrm{T}}$. For each vector, the following discrete white noise acceleration model is introduced:

$$z_{k+1}^j = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} z_k^j + \begin{bmatrix} T^2/2 \\ T \end{bmatrix} w_k = F z_k^j + G w_k, \quad j = 1, 2 \tag{17}$$

$$y_k = [1 \quad 0] z_k^j + v_k = H z_k^j + v_k, \tag{18}$$

where T is the sampling time, w_k is the process noise, v_k is the measurement noise and the output y_k is either ρ_k^{obs} or θ_k^{obs}. The Kalman filter equations [30] are:

$$\begin{aligned}
&\hat{z}^j_{k+1|k} = F\hat{z}^j_{k|k} \\
&P_{k+1|k} = FP_{k|k}F^{\mathrm{T}} + Q \\
&\hat{z}^j_{k|k} = F\hat{z}^j_{k-1|k-1} + K_k[y_k - H\hat{z}^j_{k|k-1}] \\
&P_{k|k} = [I - K_kH]P_{k|k-1} \\
&K_k = P_{k|k-1}H^{\mathrm{T}}[HP_{k|k-1}H^{\mathrm{T}} + R]^{-1},
\end{aligned} \tag{19}$$

where $\hat{z}$ denotes the estimated value, K is the Kalman gain, P is the *a posteriori* estimate error covariance, and Q and R are the covariance of w_k and v_k, respectively.

5. Collision Avoidance Algorithm

In evaluating the safety and traveling capability of autonomous mobile robots, obstacle avoidance becomes the most basic and important criterion. Some of the motions of obstacles are predictable and some are not. For the obstacles with predictable motions, a path-planning approach from the navigation module in the sequencing layer may be pursued or a completely new path plan from the top deliberative layer can be arranged. However, for obstacles that may be encountered abruptly in an unexpected situation, a swift avoidance in real-time has to be accomplished. Here, such abrupt obstacle avoidance in the reflexive layer is developed. As far as the theory is concerned, there is not much difference between the algorithm executed in real-time and those that are not. However, in order to achieve the hard real-time, the algorithm needs to be run in the kernel of Linux RTAI.

5.1. Sensor Coordinate to Robot Coordinate

The command inputs for linear and rotational motions of the robot are given as values in the robot coordinate affixed to the origin of the robot, O^{R}. Also, as seen in Fig. 8, the values of ρ_k^{obs}, $\dot{\rho}_k^{\text{obs}}$, θ_k^{obs} and $\dot{\theta}_k^{\text{obs}}$ are obtained in the sensor coordinate. Therefore, it is necessary to convert the filtered values in (19) to the values in the robot coordinate. Then, the distance between the robot and obstacle (i.e., the gap between two bodies) can be calculated by subtracting the sum of the radius of the obstacle ($D_k^{\text{obs}}/2$) and the robot radius r from the distance between two center points, O_k^{obs} and O^{R}.

Let O_{k+1}^{obs} in Fig. 8 be the center of the obstacle at time $k+1$, whose polar coordinates are $(\rho_{k+1}^{\text{obs}}, \theta_{k+1}^{\text{obs}})$. Then, the distance between O^{R} and O_{k+1}^{obs} and the orientation of the obstacle in the robot coordinate become:

$$\rho_{k+1}^{\mathrm{R}} = \sqrt{(\rho_{k+1}^{\text{obs}})^2 + r^2 - 2\rho_{k+1}^{\text{obs}} r\cos(\pi/2 + \theta_{k+1}^{\text{obs}})} \tag{20}$$

$$\theta_{k+1}^{\mathrm{R}} = \cos^{-1}\left(\frac{\rho_{k+1}^{\text{obs}}}{\rho_{k+1}^{\mathrm{R}}}\cos\theta_{k+1}^{\text{obs}}\right), \tag{21}$$

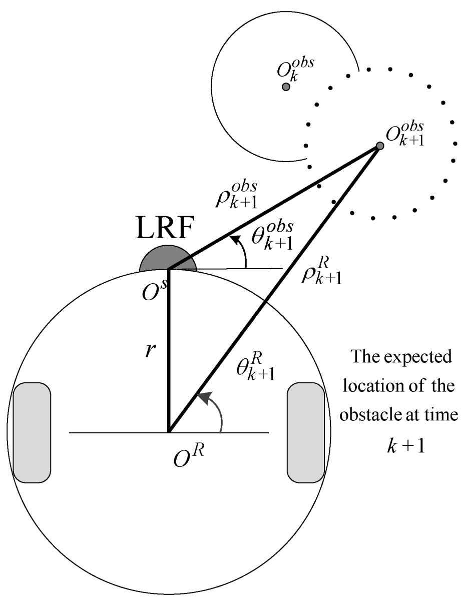


Figure 8. Transformation of the distance and orientation of the obstacle into the robot's coordinate.

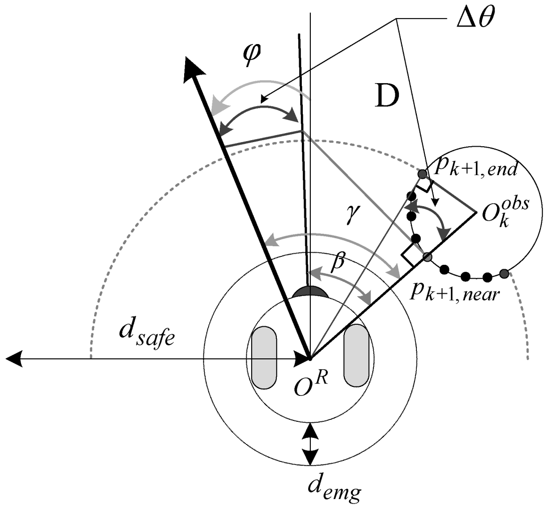


Figure 9. Geometry for determining the heading of the robot compared to a moving obstacle.

where the law of cosine was used in obtaining (20).

5.2. Moving Direction

The robot begins to avoid an obstacle if the distance between the robot and the obstacle, ρ_{k+1}^{R}, becomes smaller than some pre-designed value, d_{safe} (the safety distance, $d_{\mathrm{safe}} = 800$ mm is assumed) . As in Fig. 9, the heading of the robot is determined by using the geometry between the robot and the obstacle. Let d_{emg} be the emergency distance that evokes a soft *E-stop* of the robot ($d_{\text{emg}} = 50$ mm). Let D be the offset distance defined as follows:

$$D = 2(d_{\text{emg}} + r), \tag{22}$$

where r is the radius of the robot. Let $p_{k+1,\text{end}}$ and $p_{k+1,\text{near}}$ be two points as shown in Fig. 9. Then, the angle made by two line segments $\overline{p_{k+1,\text{near}}O^{\text{R}}}$ and $\overline{p_{k+1,\text{end}}O^{\text{R}}}$ is obtained as follows:

$$\Delta\theta = \sin^{-1}(D_k^{\text{obs}}/2\rho_{k+1}^{\text{R}}), \tag{23}$$

where the obstacle was assumed to be located in the right-hand side from the robot's heading (or the angle between $\overline{p_{k+1,\text{near}}O^{\text{R}}}$ and $\overline{p_{k+1,\text{ini}}O^{\text{R}}}$ can be calculated if the obstacle is located in the left-hand side).

Now, using the simple geometry in Fig. 9, two acute angles β and γ are obtained as:

$$\beta = \left|\tan^{-1}\left(\frac{D}{\rho_{k+1}^{\text{R}} - D_k^{\text{obs}}/2}\right)\right| \tag{24}$$

$$\gamma = \beta + \Delta\theta. \tag{25}$$

Finally, the heading angle ϕ from the x-axis of the robot coordinate is proposed as:

$$\phi = \begin{cases} \theta_{k+1}^{\text{R}} + \gamma - \dfrac{\pi}{2}, & 0 < \theta_{k+1}^{\text{R}} < \dfrac{\pi}{2} \\ \theta_{k+1}^{\text{R}} - \gamma - \dfrac{\pi}{2}, & \dfrac{\pi}{2} < \theta_{k+1}^{\text{R}} < \pi. \end{cases} \tag{26}$$

5.3. Linear and Angular Velocities of the Robot

For generating the robot's motion, a number of efficient algorithms were reported in the literature [21, 31–34]. In this paper, the method in Ref. [34] is adopted. Let $v_{\max}$ and $\omega_{\max}$ be the robot's maximum linear and angular velocities, respectively. Then, the following algorithm is given:

$$v_{\text{o}} = \frac{\rho_{k+1}^{\text{R}} - D_k^{\text{obs}}/2}{d_{\text{safe}}} \times \left(\frac{\pi/2 - |\phi|}{\pi/2}\right) \times v_{\max} \tag{27}$$

$$\omega_{\text{o}} = \frac{\phi}{\pi/2} \times \omega_{\max}, \tag{28}$$

where v_{o} represent the robot's linear velocity in the current direction and ω_{o} is the robot's angular velocity given as a function of distance ρ_{k+1}^{R}, obstacle size D_k^{obs}, safe distance d_{safe}, compass angle ϕ, $v_{\max}$ and $\omega_{\max}$. The subscript o represents 'obstacle avoidance'. As seen in (27) and (28), the robot's velocity decreases as (i) the obstacle gets near to the robot and (ii) the heading angle of the robot gets larger.

6. Experiments

The performance evaluation of the proposed control architecture was conducted in two ways: position control mode and velocity control mode. In the position control mode, three actions (*Goto*, *Avoid* and *E-stop*) were tested with stand-still obstacles and also in the presence of a moving human. Next, in the velocity control mode (*Move*), constant velocity rectilinear motions and curvilinear motions with increasing angular speed were evaluated. In this paper, however, due to space limitations, only *Avoid* tests are discussed.

The conducted *Avoid* experiments are to examine whether the transition from a *Goto* action to an *Avoid* action takes place smoothly when the robot comes across to obstacles on its way to the target position. Figure 10a shows an avoidance of stand-still obstacles: the large circles indicate the path of the robot marked at 0.4-s intervals and the small dots are stand-still obstacles. Figure 10b and 10c depicts the forward and rotational angular velocities of the robot, respectively, in that a *Goto* command from (1.5, 0) to (2.5, 6) was given (hence the velocity changed from 0 to 0.18 m/s) until it meets an obstacle, and then it keeps trying to avoid the obstacles ahead of it until it finally finds out an open space to the goal. The graphs show

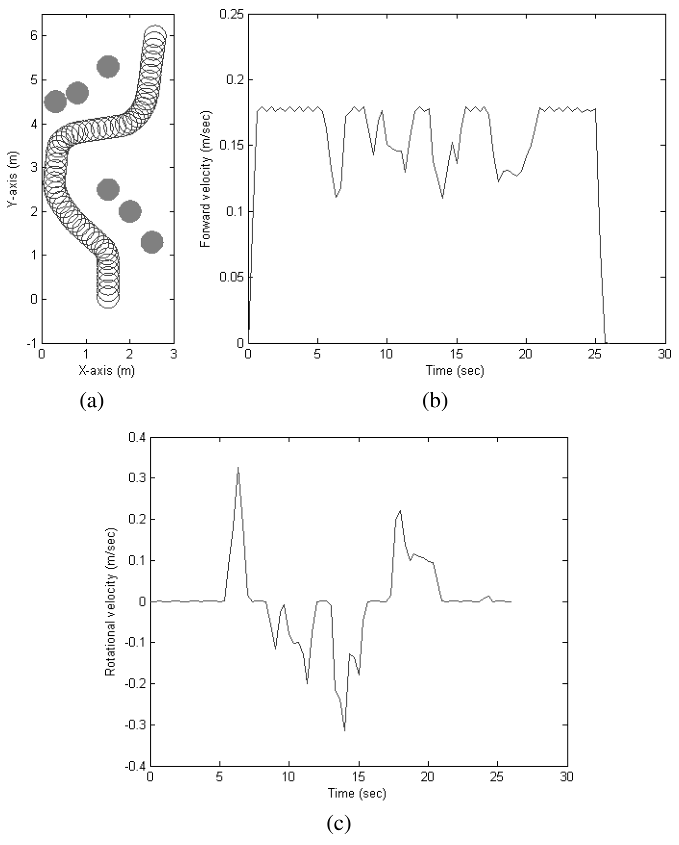


Figure 10. Experimental results of the avoidance of scattered stand-still obstacles: initial position (1.5, 0); goal position (2.5, 6); locations of six obstacles at (0.3, 4.5), (0.8, 4.7), (1.5, 5.3), (1.5, 2.5), (2, 2) and (2.5, 1.3). (a) Trajectory among scattered but stand-still obstacles (large circles: robot path, small dots: obstacles). (b) Forward linear velocity. (c) Rotational angular velocity.

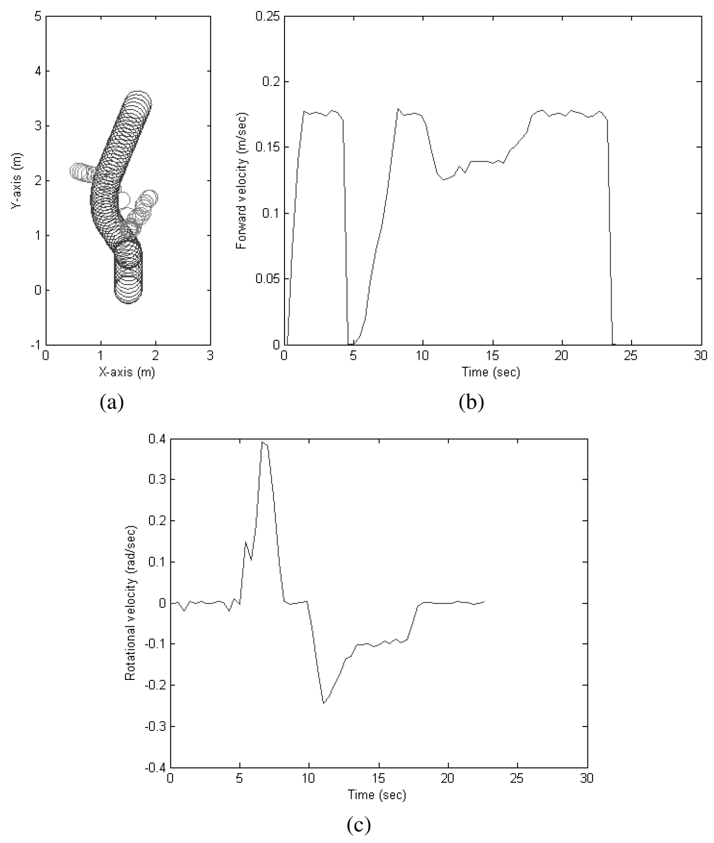


Figure 11. Experimental results: avoiding a moving human. (a) Trajectories of the robot (large circles: the robot's path, small circles: the human's path). (b) Forward linear velocity (a soft *E-stop* has occurred at about 5 s). (c) Rotational angular velocity.

a smooth transition from *Goto* to *Avoid* and then back to *Goto*. In this process, the velocity command was generated in accordance with the algorithm developed in Section 5.

On the other hand, Fig. 11 shows experimental results of avoiding a moving human. The small circles in Fig. 11a depict the movement of the human. The human blocked the robot path at about 5 s and therefore a soft *E-stop* was ordered at that moment. As seen in Fig. 11b, the velocity of the robot dropped to zero at that moment. The *E-stop* can be tested in two ways: by hitting a bumper (hard *E-stop*) or by blocking the LRF sensor (soft *E-stop*).

With a mobile robot with no fast dynamics, the effectiveness of the proposed architecture appears in the middle of face/voice recognitions (or other tasks in the Deliberate Layer in Fig. 3); under the assumption that only one computer is installed in the robot. As possible expansion of basic actions in the Reflexive Layer, avoidance of grasping a heavy (over-load) object, detection of a flying object toward the robot using a vision system, touching a fire and others can be added in the future.

7. Conclusions

In this paper, a novel control architecture of the reflexive layer for the autonomous navigation of a mobile platform was proposed. The prime purpose of the proposed architecture was to allow easy addition and evolution of innovative behaviors in the sequencing layer in a more systematic way without much concern about how the modified behaviors would be completed in real-time. This was accomplished by separating the ordinary intelligent and complicated behaviors from the actions that required hard-real-time execution. The designed reflexive layer had four components: resources, actions, action coordinator and motion controller. Resources are shared memories for sensor information. Actions are basic software components for realizing the movement of the robot. The designed four basic actions were *Goto*, *Avoid*, *Move* and *E-stop*. The role of the action coordinator was to fuse the outputs from activated actions in an effective way and to deliver the fused output to the motion controller. Among various sensors (sonar, IR, LRF), an avoidance algorithm using the LRF sensor was developed. A moving obstacle was identified as a moving circle, characterized by a center and a radius. A Kalman filter was designed to estimate the distance and orientation to the center of the moving obstacle. Experiments were conducted for testing *Goto*, *Move*, *Avoid* and *E-stop* procedures for a moving obstacle as well as scattered stand-still obstacles. Through experiments, it was verified that the proposed architecture of the reflexive layer and the moving obstacle avoidance algorithm are very effective for navigating in an unknown environment. Future work includes fusion techniques of multiple sensors and navigation schemes in the presence of multiple moving obstacles.

Acknowledgement

This work was supported by the Regional Research Universities Program (Research Center for Logistics Information, LIT) granted by the Ministery of Education, Science and Technology, Korea.

References

1. R. C. Arkin, *Behavior-Based Robotics*. MIT Press, Cambridge, MA (1998).
2. R. C. Arkin, Motor schema-based mobile robot navigation, *Int. J. Robotics Res.* **8**, 92–112 (1998).
3. R. A. Brooks, A robust layered control system for a mobile robot, *IEEE J. Robotics Automat.* **2**, 14–23 (1986).
4. R. A. Brooks, Behavior-based humanoid robotics, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Osaka, vol. 1, pp. 1–8 (1996).
5. J. H. Connell, SSS: a hybrid architecture applied to robot navigation, in: *Proc. IEEE Conf. on Robotics and Automation*, Nice, pp. 2719–2724 (1992).

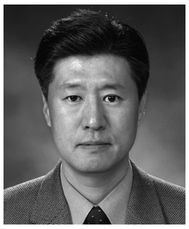
6. M. Lindstrom, A. Oreback and H. I. Christensen, BERRA: a research architecture for service robots, in: *Proc. IEEE Conf. on Robotics and Automation*, San Francisco, CA, USA, pp. 3278–3283 (2000).
7. M. Hans and W. Baum, Concept of a hybrid architecture for Care-O-bot, in: *Proc. ROMAN-2001*, Bordeaux, pp. 407–411 (2001).
8. J. S. Albus, 4D/RCS: a reference model architecture for intelligent unmanned ground vehicles, in: *Proc. SPIE Annu. Int. Symp. on Aerospace/Defence Sensing, Simulation and Controls*, Orlando, FL, pp. 303–310 (2002).
9. G. H. Kim, W. J. Chung, M. S. Kim and C. W. Lee, Control architecture design and integration of the autonomous service robot PSR, in: *Proc. Int. Conf. on Control, Automation, and Systems*, Muju, pp. 2379–2384 (2002).
10. I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin and W. S. Kim, CLARAty: an architercture for reusable robotic software, in: *Proc. SPIE Aerospace Conf.*, Orlando, FL, pp. 253–264 (2003).
11. A. Orebäck and H. I. Christensen, Evaluation of architecture for mobile robotics, *Autonomous Robots* **14**, 33–49 (2003).
12. P. Ridao, J. Batlle and M. Carreras, O2CA2; A new object oriented control architecture for autonomy: the reactive layer, *Control Eng. Practice* **10**, 857–873 (2002).
13. M. Aicardi, G. Casalino, A. Bicchi and A. Balestrino, Closed loop steering of unicycle-like vehicles *via* Lyapunov techniques, *IEEE Robotics Automat. Mag.* **2**, 27–35 (1995).
14. T. Taira and N. Yamasaki, Functionally distributed control architecture for antonomous mobile robots, *J. Robotics Mechatron.* **16**, 217–224 (2004).
15. R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, Cambridge, MA (2004).
16. K. S. Hong, K. H. Choi, J. G. Kim and S. Lee, A PC-based open robot control system: PC-ORC, *Robotics Comp.-Integr. Manufact.* **17**, 355–365 (2001).
17. C. S. Kim, K. S. Hong and H. Y.-S. Han, PC-based off-line programming in the shipbuilding industry: open architecture, *Adv. Robotics* **19**, 435–458 (2005).
18. H.-S. Choi, C. Lee and C. Chun, Development of a new 5 DOF mobile robot arm and its motion control system, *J. Mech. Sci. Technol.* **20**, 1159–1168 (2006).
19. Ş. Yıldırım and I. Eski, A QP artificial neural network inverse kinematic solution for accurate robot path control, *J. Mech. Sci. Technol.* **20**, 917–928 (2006).
20. Y. M. Cho, H. S. Kim, I. K. Kim, J. J. Woo and J. Kim, “Fast design of the QP-based optimal trajectory for a motion simulator,” *J. Mech. Sci. Technol.* **21**, 1973–1985 (2007).
21. B. S. Kim, J.-S. Park, C. Moon, G.-M. Jeong and H.-S. Ahn, A precision robot system with modular actuators and MEMS micro gripper for micro system assembly, *J. Mech. Sci. Technol.* **22**, 70–76 (2008).
22. L. Dozio and P. Mantegazza, Linux real time application interface (RTAI) in low cost high performance motion control, in: *Proc. Motion Control 2003 Conf. of ANIPLA*, Milan, pp. 27–28 (2003).
23. DIAPM RTAI, http://www.aero.polimi.it/∼rtai/
24. K. Konolidge and K. Myers, *The Saphira Architecture for Autonomous Mobile Robots*, SRI International, Menlo Park, CA (1996).
25. T. Balch, TeamBots, www.teambots.org (2000).
26. G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, Cambridge University Press, Cambridge (2000).
27. A. Elfes, Using occupancy grids for mobile robot perception and navigation, *IEEE Comp. Arch.* **22**, 46–57 (1989).

28. S.-J. Lee, J.-H. Lee and D.-W. Cho, Featured-based map building using sparse sonar data in a home-like environment, *J. Mech. Sci. Technol.* **1**, 74–82 (2007).
29. M. Aicardi, G. Casalino, A. Bicchi and A. Balestrino, Closed loop steering of unicycle-like vehicles *via* Lyapunov techniques, *IEEE Robotics Automat. Mag.* **2**, 27–35 (1995).
30. J. Tan and N. Kyriakopoulos, Implementation of a tracking Kalman filter on a digital signal processor, *IEEE Trans. Ind. Electron.* **35**, 126–134 (1988).
31. J. Borenstein and Y. Koren, Real-time obstacle avoidance for fast mobile robots, *IEEE Trans. Syst. Man Cybernet.* **19**, 1179–1187 (1989).
32. D. Fox, W. Burgard and S. Thrun, The dynamic window approach to collision avoidance, *IEEE Robot. Automat. Mag.* **4**, 23–33 (1997).
33. A. Ohya, A. Kosaka and A. Kak, Vision-based navigation by a mobile robot with obstacle avoidance using single camera vision and ultrasonic sensing, *IEEE Trans. Robotics Automat.* **14**, 969–978 (1998).
34. J. Minguez and L. Montano, Nearness diagram (ND) navigation: Collision avoidance in troublesome scenarios, *IEEE Trans. Robotics Automat.* **20**, 45–59 (2004).

About the Authors

Keum-Shik Hong received the BS degree in Mechanical Design and Production Engineering from Seoul National University, in 1979, the MS degree in Mechanical Engineering from Columbia University, New York, in 1987, and both the MS degree in Applied Mathematics and the PhD degree in Mechanical Engineering from the University of Illinois at Urbana-Champaign (UIUC), in 1991. From 1991 to 1992, he was a Postdoctoral Fellow at UIUC. He joined the School of Mechanical Engineering at Pusan National University, South Korea, in 1993; he is now Professor. During 1982–1985, he was with Daewoo Heavy Industries, Incheon, South Korea, where he worked on vibration, noise and emission problems of vehicles and engines. He serves as Editor-in-Chief of the *Journal of Mechanical Science and Technology* and serves as an Associate Editor on various IEEE and IFAC conferences editorial boards. He also served as an Associate Editor for the *Journal of Control, Automation, and Systems Engineering* and for *Automatica* (2000–2006), and as an Editor for the *International Journal of Control, Automation, and Systems* (2003–2005). His laboratory, the Integrated Dynamics and Control Engineering Laboratory, was designated a National Research Laboratory by the Ministry of Science and Technology of Korea, in 2003. He received the Fumio Harashima Mechatronics Award, in 2003, and the Korean Government Presidential Award, in 2007. He is a member of the ASME, IEEE, ICASE, KSME, KSPE, KIEE and KINPR. His current research interests include nonlinear systems theory, adaptive control, distributed parameter system control, robotics, vehicle control and innovative control applications to engineering problems.

Tua Agustinus Tamba received the BS degree in Engineering Physics from the Institute of Technology Bandung, Indonesia, in 2006. He is currently a Graduate student at the School of Mechanical Engineering, Pusan National University, Busan, South Korea. His research interests include the control of unmanned vehicles and path-planning technologies for autonomous robots.



Jae-Bok Song received the BS and MS degrees in Mechanical Engineering from Seoul National University, in 1983 and 1985, respectively, and the PhD degree in Mechanical Engineering from MIT, in 1992. He joined the faculty of the Department of Mechanical Engineering, Korea University, in 1993. Currently, he is a Director of the Intelligent Robotics Research Center at Korea University. He is also an Editor for the *International Journal of Control, Automation and Systems*. His current research interests include mobile robot navigation, design and control of safe manipulators, and haptics.