A Self-Adaptive Intelligent Control System with Hierarchical Architecture

CHENG-HSIUNG CHIANG1,* AND LIANG-HSUAN CHEN2
1Department of Information Management
Hsuan Chuang University
Hsinchu, 300 Taiwan
E-mail: chchiang@hcu.edu.tw
2Department of Industrial and Information Management
National Cheng Kung University
Tainan, 701 Taiwan
E-mail: lhchen@mail.ncku.edu.tw

This paper presents an Intelligent Control System with Hierarchical Architecture, namely ICSHA. The advantages of three-layered ICSHA are able to carry out multiple tasks and to adjust its control rules automatically to adapt environments. The first layer, Planning Layer, we propose the ASGO (an extension version of Ant System with Genetic Operators) to determine the executing order of subtasks. The visiting schedule is then supervised by second layer, Executive Layer. The third layer, Behavior Layer, is to execute each subtask by using the proposed intelligent control module. While the control module cannot adapt the environment, the proposed iQGA (an improved Quantum Genetic Algorithm) is activated to explore better control actions for producing adaptable control rules. An application of robotic trash collection task is constructed to demonstrate the proposed methods. The simulation results showed that the performances of ASGO and iQGA are satisfied. Simulation result also reveals the adaptability of ICSHA.

Keywords: ant system, genetic algorithm, intelligent control, path planning, quantum genetic algorithm

1. INTRODUCTION

Machine intelligence that is an important challenge of artificial intelligence research is to mimic human intelligence. Albus [1] presented a description of intelligence:

“At a minimum, intelligence requires the ability to sense the environment, to make decisions, and to control actions. Higher levels of intelligence may include the ability to recognize the objects and events, to represent knowledge in a world model, and to reason about and plan for the future.”

We can investigate machine intelligence from different point of views, such as computing with words [28], self-adaptive systems [18], intelligent agents [20], and intelligent control [17]. An intelligent control system (ICS) is not only able to emerge partially intelligence but also able to be applied to practical applications, such as industrial and manufacturing process, biological and medical engineering, aeronautical engineering,
oceanographic engineering, robotics, automobile, etc. The three basic functions of an ICS are perception, decision-making, and action [5].

We may develop an ICS to imitate human intelligence based on artificial intelligence technologies, especially using artificial neural networks, fuzzy control, and evolutionary computation (EC). To more approximate human intelligence, an ICS shall have the self-adaptive ability that can automatically adjust actions to adapt various environments. In this paper, the self-adaptive behaviors are regarded as continuous learning or life-long learning. Several ICSs with a continuous learning mechanism have been developed, such as SEICS (Self-Exploration process based Intelligent Control system) [5], mSEICS (multi-objective Self-Exploration process based Intelligent Control System) [6], the fuzzy-genetic system [15], the Interactive Smart House Control System [3], SyICS (symbolic controller based Intelligent Control System) [7], qSyICS (a symbolic controller based intelligent control system with quantum particle swarm optimization based hybrid genetic algorithm) [8], and QRICS (a rule-based intelligent control system with exploration process: a quantum genetic algorithm approach) [9]. On the contrary, most ICSs are not provided with the mechanism of continuous learning, such as the adaptive neural network control system [13, 29], the quantum computing based ICS [23], the integrated intelligent fuzzy control system [26], and the adaptive neuro-fuzzy control system [19].

Bien et al. [3] presented an interactive smart house system which is a service-integrated complex system to assist older persons and/or people with disabilities. They developed a framework to realize human-friendly HRI (human-robot interaction) module, and the robotic tasks of HRI module can be partitioned into three groups. (1) A simplified set of commands of the user is proposed to deal with well-structured tasks autonomously through a task planning algorithm. (2) The robot makes use of human bio-signals as input of the HRI module, such as a hand gesture recognition system. (3) The probabilistic fuzzy rule-based life-long learning subsystem can provide reading of the user’s intentions by indirectly observing his/her behavioral patterns or can respond in the long run to inconsistent commands and changed environment. For example, the learning system can recommend to the resident favorite TV channel based on the acquired fuzzy rule-based knowledge. The proposed system provides friendly and multi-functional service to the people with disabilities to promote living quality. However, the complexity of the system may be not easy to be applied. Hagras et al. [15] developed a fuzzy-genetic system with hierarchical fuzzy controllers for the online learning and adaptation of an intelligent robotic navigator system. After learning the system parameters, the controller operates in its environment. If the controller fails to maintain the desired states, the online adaptation technique modifies the control rules. Learning can be applied for obstacle avoidance. When an obstacle is hit or approached, the robot returns to its pre-failure position to find the rules responsible for the failure and then corrects them. Only four control rules will be modified, and these are the rules which contributed mostly to the failure situation (collision). The approach of continuous learning is very flexible since it can be applied to various robots with the same sensor configuration and performing the same mission. However, this adaptive mechanism adjusts a fixed number of rules, which may prevent the robot from adapting to some complex environments.

A series of ICS models with continuous learning, i.e., SEICS [5], mSEICS [6], SyICS [7], qSyICS [8], and QRICS [9], have been developed. The main parts of these
frameworks are: 1) controller, 2) perception, and 3) self-adaptive mechanism. The controller generates the control actions to the plant, the controlled system, according to its control law. SEICS and mSEICS employ the fuzzy neural networks for implementing controller whose control laws are the fuzzy rules. In contrast to the fuzzy logic based control rules, SyICS, qSyICS and QRICS adopt the crisp logic based if-then rules which are called as rule-based controller. Perception deals with the sensory information obtained from the environment, and determines whether the system should keep on controlling task using the controller or switch to the self-adaptive mechanism to modify the control rules. The simple two-valued perception which indicates satisfactory and unsatisfactory controlling performance is applied in SEICS, qSyICS and QRICS. Another type of perception proposed in mSEICS and SyICS systems considers both the adaptability and efficiency of the controlling performance. The self-adaptive mechanism implements learning using EC techniques, such as genetic algorithms (GA) [5, 7], multi-objective genetic algorithm [6], quantum-inspired particle swarm optimization based genetic algorithm [8] and quantum-inspired genetic algorithms (QGA) [9], to obtain better control actions. It then transforms these actions into new control rules. To explain the continuous learning procedure, robot path planning is simulated for the above-mentioned models. If the robot collides with an obstacle, the adaptive mechanism is activated. The robot goes back several movements, and then employs an EC approach to obtain better movements, which are then transformed into a number of control rules. In contrast to the fuzzy-genetic system [15], the number of modified rules is not fixed, and future movements are planned in addition to the re-planning of past movements. The complexity of the control frameworks of the fuzzy-genetic system [15] is higher than those of these models since the former incorporates several fuzzy controllers for various robot behaviors. However, these methods cope with single task at one time, and are not suitable to schedule and perform multiple tasks.

He [18] proposed an adaptive dynamic programming (ADP) architecture with hierarchical learning which could represent multiple goals to integrate the optimization and prediction together. In general, an ADP architecture is similar to that of reinforcement learning which is basically consisted of action network and critic network. The hierarchical ADP proposed by He [18] further considers the reference network to provide multiple levels of internal reinforcement representations. The network parameters are trained on-line which can be applied to unknown environment and adapt different environments. Even through this method can deal with multiple goals at a time, the network parameters based control actions is lack of intuitive meaning and is not close to human language.

In order to deal with multiple tasks or complex behaviors, we developed an intelligent control system with hierarchical architecture, namely ICSHA. The abbreviation list of the proposed methods and other relevant technologies are shown in Table 1. From the whole architecture point of view, the proposed ICSHA has two major advantages.

1. It can automatically adjust the control rules to adapt environments, and
2. The hierarchical structure allows it to carry out multiple tasks.

ICSHA has three layers, namely the Planning Layer, the Executive Layer, and the Behavior Layer. The Planning Layer is applied to develop a subtask schedule which indicates the executing order for each subtask. For a problem with $n$ subtasks, the num-
ber of permutations is $n!$, and it is increased exponentially. Accordingly, to have an efficient algorithm to search the optimal permutation (or sequence) is important. We present an ant system with GA-inspired genetic operators, namely, ASGO, whose structure is based on the ant system [10, 11] and is suitable to resolve the permutation problem such as traveling salesman problem (TSP). ASGO has been demonstrated that it is superior to traditional ant system method as shown in Section 5.2. The Executive Layer superintends the schedule of subtask propagated from the Planning Layer, and it keeps track of which subtasks have been executed and which have not. The Behavior Layer that plays the role of controller (the core of the ICSHA) is applied to carry out the each subtask.

An improved version of QRICS [9], i.e., iQRICS, is presented for carrying out the control tasks in Behavior Layer. The iQRICS which is an ICS can adapt various environments through adjusting its control rules. Similar to the structure of QRICS, iQRICS has three basic functions: an if-then rule-based controller, a Percepter, and a qAdaptor. For an example of robotic path planning, if the Percepter, a mechanism for evaluating control performance, detects that the robot collides with an obstacle, the qAdaptor, an adaptive mechanism, will search better control rules and update the rule base. The major difference between iQRICS and QRICS is that we propose an improved QGA, namely iQGA, for iQRICS to discover better control actions. Nevertheless, QRICS adopts the HQGA (hybrid quantum genetic algorithm) to search better actions. The comparison results in Section 6 show that the performance of proposed iQGA is better than HQGA.

For demonstrating the proposed methods, we apply the ICSHA to the robot trash collection task, described in Section 5. The rest of this paper is organized as follows. Section 2 presents the general framework of ICSHA. Section 3 introduces the ASGO algorithm. In Section 4, the proposed iQRICS, is described. Section 5 presents comparison results of the proposed approaches and a simulation analysis of ICSHA for the trash collection task. Finally, Section 6 gives the conclusions and future work.

<table>
<thead>
<tr>
<th>Table 1. The abbreviation list of the relevant technologies.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abbreviation</strong></td>
</tr>
<tr>
<td>QGA</td>
</tr>
<tr>
<td>HQGA</td>
</tr>
<tr>
<td>QRICS</td>
</tr>
<tr>
<td>SEICS</td>
</tr>
<tr>
<td>SyICS</td>
</tr>
<tr>
<td>ICSHA</td>
</tr>
<tr>
<td>ASGO</td>
</tr>
<tr>
<td>iQGA</td>
</tr>
<tr>
<td>iQRICS</td>
</tr>
</tbody>
</table>

The last four methods, i.e., ICSHA, ASGO, iQGA and iQRICS, are proposed in this paper.

### 2. ARCHITECTURE OF ICSHA

This paper proposes a self-adaptive ICS with hierarchical architecture called IC-
SHA, shown in Fig. 1. The structure of ICSHA is inspired by the 3-T architecture [4], which is a mobile robot control architecture predominately used at NASA; however, the ICSHA approach is proposed for general intelligent control, not only aiming at robotics. The three-layered 3-T architecture can execute and supervise multiple tasks, and has been primarily applied for planetary rovers, underwater vehicles, and robot assistants for astronauts [24].

For conveniently describing the operation procedure of ICSHA, the robot trash collection task (as discussed in Section 5) is applied as an example. The robot has to move to collect all known litters and then drop them into trash bin, and subsequently moves back to robot’s home. As shown in Fig. 1, the top layer of ICSHA is the Planning Layer, which arranges the working schedule of subtasks for the robot. In this paper, a task is consisted of multiple subtasks. For each subtask, the robot has to move to pick up a designated litter at known positions according to the schedule. We propose the ASGO algo-
rithm that incorporates the ant systems, presented by Dorigo et al. [11], with several genetic operators (e.g., mutation operator) to determine a schedule for picking litter in order.

The schedule, which specifies the sequence of movements, is then passed to the middle layer, called the Executive Layer. The Executive Layer monitors which subtasks have been carried out and which have not. Each subtask is performed by the control module in the Behavior Layer. We present the iQRICS to accomplish the controlling assignments, i.e., driving the robot to a designated position to pick up litter.

For the iQRICS control module, the predefined if-then control rules are gathered in the rule base which provides the inference knowledge for the controller. According to specific sensory information, the Rule-Base Controller generates corresponding control actions for the robot. If the control performance evaluated by the Percepter is poor or inadaptable, e.g., the robot collides with an obstacle, the qAdaptor (an adaptive mechanism for discovering better and adaptable control rules by using quantum genetic algorithm and Rule Generator) is activated to obtain new rules to update the rule base. The adaptive mechanism of qAdaptor helps the control module improve its control rules to adapt environments. Within qAdaptor we propose the iQGA for finding new control actions by implementing the two-stage exploration process [11]. The Rules Generator is then applied to transform these actions into if-then rules. More details about the qAdaptor will be introduced in Section 4.

Once subtask $i$ has been executed completely, the Percepter sends a message to the Executive Layer, and subtask $i + 1$ is assigned to iQRICS in the Behavior Layer.

3. ANT SYSTEM WITH GENETIC OPERATORS

The ant system (AS) algorithm was first proposed by Dorigo [10]. It is inspired by the behavior of real ants in the wild in which the ant communicates indirectly with each other via the secretion of chemical pheromones [2]. Real ants are able to find the shortest path from the food source to their nest using pheromones [21]. In order to enhance the quality of solutions, we propose an advanced AS algorithm, ASGO, which incorporates the algorithmic structure of AS with genetic operators. ASGO uses three operators, namely the selection operator, the mutation operator, and the trimming operator, to increase the average quality and variety of solutions. The ant-based algorithms consider how to determine the next movement of an ant to result a shorter route, but the genetic operators carry out the reproduction for individuals and recombination of genes of an individual. ASGO combines the above two advantages. For an example of resolving the TSP, ASGO firstly determines a complete tour for each ant based on transition probability of AS method, and then employs the three operators to enhance the quality of solutions.

1. The selection operator reproduces the tours, and the shorter tour has the higher probability to be selected.
2. For a specific tour, the mutation operator inverts the visiting order between two selected cities.
3. By the trimming operating, only one tour is reserved among those same tours, and the
deleted tours will be regenerated randomly.

Section 5 demonstrates that the solutions obtained by ASGO are better than that of AS and ACS (the any colony system method proposed by Dorigo and Gambardella [12]). However, ASGO method requires more CPU time to carry out the extra genetic operators.

The procedure of ASGO is summarized in Algorithm 1. Algorithm 1 is similar to the approach for solving the TSP. Considering the robot trash collection task, ASGO is employed to obtain the shortest route required to gather all trash. The algorithm is simply stated as follows. At $t = 0$, an initialization phase takes place during which ants are positioned in different pieces of litter and $\tau_{ij}(0)$ (trail intensity) is set to a small positive constant $c$ on edges. The first element of the tabu list, which records the visited pieces of litter, for each ant is set as its starting position. In the robot trash collection task, the starting position for each ant is **Robot Home**. Every ant then moves from litter $i$ to litter $j$ according to the transition probability:

$$p_{ij}(t) = \begin{cases} \left[\frac{\tau_{ij}(t)}{\sum_{k \in \text{allowed}_k} \tau_{ik}(t)}\right]^\alpha \cdot \left[\frac{\eta_{ij}}{\sum_{k \in \text{allowed}_k} \eta_{ik}}\right]^\beta, & \text{if } j \in \text{allowed}_i \\ 0, & \text{otherwise} \end{cases}$$

(1)

where allowed$_i$ is the set of non-visited nodes for the $k$th ant, and $\eta_{ij} = 1/d_{ij}$ ($d_{ij}$ is the distance between litter $i$ and litter $j$) is the visibility, which specifies that the closer a piece of litter is the more it is desirable. $\alpha$ and $\beta$ are parameters that control the relative importance of trail intensity versus visibility ($\alpha$ and $\beta$ are set to 1 and 5, respectively, in our simulations based on prior experiments.). $\tau_{ij}$ is the trail intensity (shown on line 22 of Algorithm 1), which gives information about how many ants in the past have chosen the same edge ($i, j$). If $\alpha$ is set to 0, the trail intensity is no longer considered, and a stochastic greedy algorithm with multiple starting points is obtained [11].

After $n$ iterations (lines 4 to 11, Algorithm 1) all ants have completed a route, and their tabu lists (tabuk is the tabu list of the $k$th ant) will be full. Thereafter, the route length of the $k$th ant $L_k$ is computed. According to the roulette-wheel selection process [14], $m$ tabu lists (all tabu lists are denoted as a vector tabu) can be reproduced based on the route lengths. Shorter lengths have higher probabilities of being reproduced. The new tabu lists obtained by the selection operation, sTabu, is then subjected to the mutation operation. The mutation operation switches the visiting order of two randomly selected pieces of litter. For the robot trash collection task, assume that the visiting sequence for litter is: litter 2 $\rightarrow$ litter 1 $\rightarrow$ litter 3 $\rightarrow$ litter 5 $\rightarrow$ litter 4; then, litter 1 and litter 5 are randomly exchanged. After the mutation processing, the new sequence is: litter 2 $\rightarrow$ litter 5 $\rightarrow$ litter 3 $\rightarrow$ litter 1 $\rightarrow$ litter 4. The trimming operation modified from [22] reserves one tabu list from mergeTabu (obtained by merging sTabu and mTabu) among those with the same route length. The deleted tabu lists are regenerated randomly.

**Algorithm 1: Ant System with Genetic Operators (ASGO)**

1: $t \leftarrow 0$, $NC \leftarrow 0$, and initialize $\tau_{ij}$ with $\Delta \tau_{ij} = 0$
2: while $NC < NC_{\text{max}}$ do
3: \( s \leftarrow 1 \), and initialize \( \text{tabu}_k(s) \), \( k = 1, 2, \ldots, m \)
4: \( \textbf{while} \ s \leq n \ \textbf{do} \)
5: \( s \leftarrow s + 1 \)
6: \( \textbf{for} \ k = 1 \text{ to } m \ \textbf{do} \)
7: \( \text{Choose litter } j \text{ to move to, with probability } p_{ik}^s(t) \)
8: \( \text{tabu}_k(s) \leftarrow j \)
9: \( \textbf{end for} \)
10: \( t \leftarrow t + 1 \)
11: \( \textbf{end while} \)
12: \( \forall k, \text{ compute route length } L_k \text{ of } k\text{th ant based on } \text{tabu}_k \)
13: \( s\text{Tabu} \leftarrow \text{selectOperator}(\text{tabu}) \)
14: \( m\text{Tabu} \leftarrow \text{mutationOperator}(s\text{Tabu}) \)
15: \( \text{mergeTabu} \leftarrow s\text{Tabu} \cup m\text{Tabu} \)
16: \( \text{newTabu} \leftarrow \text{trimmingOperator}(\text{mergeTabu}) \)
17: \( \text{tabu} \leftarrow \text{find the best } m \text{ individuals from } \text{newTabu} \)
18: \( \textbf{for} \ k = 1 \text{ to } m \ \textbf{do} \)
19: \( \text{For every edge } (i, j), \text{ calculate } \Delta \tau_{ij}^k \)
20: \( \Delta \tau_{ij} \leftarrow \Delta \tau_{ij} + \Delta \tau_{ij}^k \)
21: \( \textbf{end for} \)
22: \( \text{For every edge } (i, j), \text{ calculate } \tau_{ij} = \rho \tau_{ij} + \Delta \tau_{ij} \)
23: \( NC \leftarrow NC + 1, \text{ and update the shortest route found} \)
24: \( \textbf{end while} \)
25: \( \textbf{return} \text{ shortest route} \)

After performing the three operators, \( \Delta \tau_{ij} \) (quantity per unit of length of the trail substance, \( i.e \), pheromone for real ants, laid on edge \((i, j)\)) is updated according to:

\[
\Delta \tau_{ij} = \sum_{k=1}^{m} \Delta \tau_{ij}^k
\]

where \( \Delta \tau_{ij}^k \) is the quantity per unit of length of the trail substance laid on edge \((i, j)\) by the \( k \)-th ant. \( \Delta \tau_{ij}^k = Q / L_k \) if the \( k \)-th ant uses edge \((i, j)\) in its route; otherwise, \( \Delta \tau_{ij}^k \) is set to 0. \( Q \) is a constant (in our simulations, \( Q = 100 \)). The trail intensity \( \tau_{ij} \) is updated according to line 22 of Algorithm 1, where \( \rho \) is a coefficient (\( \rho < 1 \)), and \((1 - \rho)\) represents the evaporation of the trail (\( \rho \) is set to 0.5 in our experiments). This process is repeated until the route counter reaches \( NC_{\text{max}} \).

4. iQRICS IN BEHAVIOR LAYER

The QRICS control model presented by Chiang [9] is improved to give iQRICS, as shown in Fig. 1, to fulfill the control assignments in the Behavior Layer. The major difference between iQRICS and QRICS is the iQGA. In contrast to HQGA within QRICS, iQGA is an advanced approach with higher efficiency (demonstrated in Section 5). iQRICS comprises a rule-based Controller, a Percepter, and a qAdaptor. The controller generates the control actions for the robot, and then the Percepter evaluates their perfor-
mance. If the effect is unsatisfactory, e.g., the robot collides with obstacles, the qAdaptor will be activated to search for better control rules; otherwise, the controller continues performing subtask $i$.

When the qAdaptor is enabled, the iQGA implements the two-stage exploration process to find better feasible control actions, i.e., a shorter path without collisions. The Rule Generator then transforms the results into if-then rules and updates the rule base. By using qAdaptor, iQRICS improves its control rules to adapt to the environment continuously.

### 4.1 Rule-based Controller

We will introduce the Rule-based Controller by using the robot trash collection task since it is the demonstrated application described in Section 5. We assume that the working space of the robot is $1500 \text{ cm} \times 1500 \text{ cm}$. Two input variables, $d$ and $\phi$, and one output variable, $\theta$, are defined. $d$ and $\phi$ are distance (within $[0, 2130] \text{ cm}$) and the included angle (within $[0, 360]$ degrees) between the start position of the robot and the target, respectively. $\theta$ denotes the steering angle of the robot within $[0, 360]$ degrees. Because there is no information about obstacles, the control rules are constructed based on an obstacle-free environment. Accordingly, the initial rules steer the robot forward from its current position to the target along a straight line. The input and output variables for the robot trash collection task are defined below.

1) There are 71 terms for $d$, i.e., $\{d_1, d_2, ..., d_{71}\}$, within $[0, 2130] \text{ cm}$, and $d_i \in [30(i-1), 30i] \text{ cm}, i = 1, 2, ..., 71$.
2) $\phi$ has 30 terms, i.e., $\{\phi_1, \phi_2, ..., \phi_{30}\}$, within $[0, 360]$ degrees. $\phi_i \in [6(i-1), 6i]$ degrees, $i = 1, 2$, and $\phi_i \in [12(i-2), 12(i-1)]$ degrees, $i = 3, 4, ..., 30$.
3) $\theta$ has 31 terms within $[0, 360]$ degrees, and $\theta_i = 12(i-1)$ degrees, $i = 1, 2, ..., 31$.

An example of an obstacle-free-based initial rule is:

\[
\text{If } d \text{ is } d_6 \text{ and } \phi \text{ is } \phi_9 \text{ Then } \theta \text{ is } \theta_9, \tag{3}
\]

where $d_6$, $\phi_9$, and $\theta_9$ are the sixth, ninth, and ninth terms for $d$, $\phi$, and $\theta$, respectively. The subscript values of $\phi$ and $\theta$ are the same, meaning that the steering angle of the robot inclines toward the target position.

### 4.2 Exploration Process

To imitate the human introspection process for resolving barriers, Chen and Chiang [6] presented a self-exploration process with three stages. Chiang [7] simplified the three-stage exploration process to a two-stage exploration process, which is employed in this paper. If the robot runs into an obstacle, the Percepter sends an “unsatisfactory” signal to activate qAdaptor. The exploration process is then enabled to find a shorter feasible path. The two stages of the exploration process are:

1) **Return stage** determines the number of steps that the robot should move back from its
collision point. In order to simplify the computational time, the number of steps is fixed at 3.

2) **Exploration stage** finds better feasible robot paths in order to steer the robot avoiding the obstacle. This stage is implemented using the iQGA, as described in the next section.

### 4.3 Proposed iQGA

A number of QGAs have been investigated [16, 22, 27]. In QGAs, a Q-bit representation represents a linear superposition. A Q-bit stored in a two-state quantum computer is the smallest unit of information. It may be in the “1” state, the “0” state, or in any superposition of the two states. The state of a Q-bit can be expressed by:

\[ |\Psi\rangle = \alpha |0\rangle + \beta |1\rangle \quad \text{and} \quad |\alpha|^2 + |\beta|^2 = 1 \quad (4) \]

**Algorithm 2: iQGA**

1. Generate initial Q-bit string based population \(Q = \{q_1, \ldots, q_n\}\) randomly
2. Make binary string based population \(P\) by measuring the state of \(Q\) and evaluate it
3. Record the best solution among \(P\) and denote it as \(b\)
4. for \(t = 1\) to \(\text{maxGen}\)
5. \(sQ \leftarrow \text{Selection}(Q)\)
6. \(rQ \leftarrow \text{QuantumRotationGate}(sQ)\)
7. \(cQ \leftarrow \text{Crossover}(sQ)\)
8. \(dQ \leftarrow \text{Deletion}(sQ)\)
9. \(pQ \leftarrow \text{Pull}(sQ)\)
10. \(mQ \leftarrow rQ \cup cQ \cup dQ \cup pQ\)
11. \(mP \leftarrow \text{measure the state of } mQ \text{ and evaluate it}\)
12. \([P, Q] \leftarrow \text{derive the best } n \text{ individuals from } mP\)
13. Update the best solution \(b\)
14. end for
15. return best solution \(b\)

where \(\alpha\) and \(\beta\) are complex numbers that specify the probability amplitudes of the corresponding states. \(|\alpha|^2\) and \(|\beta|^2\) are the probabilities that the Q-bit will be in the “1” and “0” states, respectively.

For a QGA, within a population, an individual with \(m\) Q-bits is defined as:

\[
\begin{bmatrix}
\alpha_1 & \alpha_2 & \cdots & \alpha_m \\
\beta_1 & \beta_2 & \cdots & \beta_m
\end{bmatrix}
\]

where \(|\alpha_i|^2 + |\beta_i|^2 = 1\), \(i = 1, 2, \ldots, m\). In general, the length of a Q-bit representation is fixed; however, the iQGA has variable-length individuals, as summarized in Algorithm 2.

For the robot trash collection task, the iQGA is employed to find shorter feasible robot paths, which are made up of a number of steering angles (\(\theta\)) of the robot, to avoid obstacles. As shown in Algorithm 2, the initial population \(Q = \{q_1, \ldots, q_n\}\) is generated, where \(q_i\) denotes the \(i\)th Q-bit individual defined as:
According to experimental results, the population size \( n \) is set to 30. For variable-length individual \( \mathbf{q}_i \), four Q-bits are used to encode the output variable, i.e., steering angle \( \theta \). Each \( \theta \) guides the robot for one movement. It is assumed that the number of movement steps for the robot is between 5 to 10, and thus \( m \) (as shown in Eq. (6)) is between 20 (5×4) and 40 (10×4). Each \( \alpha_j \) and \( \beta_j \), \( i = 1, 2, \ldots, m \), are initialized with \( \frac{1}{\sqrt{2}} \).

As shown in line 2 of Algorithm 2, the initial population \( Q \) is measured to determine the corresponding binary string of a Q-bit individual. Hence, a random number \( r \) is generated. If \( r < |\alpha|^2 \), the corresponding binary bit is set to “0”; otherwise, the binary bit is set to “1.” Next, the fitness function proposed by Chiang [9] is slightly modified to evaluate an individual:

\[
\text{fit} = (1 - (w \cdot D + (1 - w) \cdot L)) \cdot e^{-2N_c},
\]  

where \( w = 0.15 \cdot e^D + 0.85 \). \( D \) specifies the distance between the trash bin and the final position of the robot, and \( N_c \) is the number of collisions. \( L \) is the length of an individual (5 ≤ \( L \) ≤ 10). Thereafter, the fit shown in Eq. (7) is normalized to be within \([0, 1]\). After evaluating the individuals, the best solution is denoted as \( b \) and stored.

Lines 4 to 14 of Algorithm 2 describe the main process for evolving new populations. \( \text{maxGen} \) denotes the maximum number of evolutions, which was set to 100 in the simulation. Five operations, lines 5 to 9, are used for manipulating the Q-bit strings to form new individuals. The first genetic operation is the selection operation, which is similar to that in the HQGA. However, in the iQGA, to reproduce the individuals of the \( Q \) population, the selection operation selects individuals from the best 50% of the population to be duplicated in the new population \( sQ \).

The second operation introduces the \( sQ \) population for carrying out quantum rotation gate to produce \( rQ \). A rotation gate \( U(\theta) \) is used to update a Q-bit individual. The \( i \)th Q-bit \((\alpha_i, \beta_i), j = 1, 2, \ldots, n, \) and \( i = 1, 2, \ldots, m \), is updated as follows [28]:

\[
\begin{bmatrix}
\alpha_i' \\
\beta_i'
\end{bmatrix} = U(\theta_i) \begin{bmatrix}
\alpha_i \\
\beta_i
\end{bmatrix} = \begin{bmatrix}
\cos(\theta_i) & -\sin(\theta_i) \\
\sin(\theta_i) & \cos(\theta_i)
\end{bmatrix} \begin{bmatrix}
\alpha_i \\
\beta_i
\end{bmatrix}
\]  

where \( \theta_i = s(\alpha_i, \beta_i)\Delta\theta_i \) is the rotation angle. \( s(\alpha_i, \beta_i) \) is the sign of \( \theta_i \), and \( \Delta\theta_i \) is the magnitude of \( \theta_i \), whose lookup table is listed in Table 2. The table is simpler than that proposed for the HQGA to reduce the computational time.

<table>
<thead>
<tr>
<th>( r_i )</th>
<th>( b_i )</th>
<th>( \Delta\theta_i )</th>
<th>( s(\alpha_i, \beta_i) )</th>
<th>( \alpha, \beta &gt; 0 )</th>
<th>( \alpha, \beta &lt; 0 )</th>
<th>( \alpha = 0 )</th>
<th>( \beta = 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.2\pi</td>
<td>-1</td>
<td>+1</td>
<td>1/−1*</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.5\pi</td>
<td>-1</td>
<td>+1</td>
<td>1/−1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.5\pi</td>
<td>-1</td>
<td>+1</td>
<td>1/−1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.2\pi</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>1/−1</td>
<td></td>
</tr>
</tbody>
</table>

*"1/−1" indicates that the value can be 1 or −1.

\[
\mathbf{q}_i = \begin{bmatrix}
\alpha_{j1} & \alpha_{j2} & \ldots & \alpha_{jm} \\
\beta_{j1} & \beta_{j2} & \ldots & \beta_{jm}
\end{bmatrix}
\]  

(6)
In Table 2, \( b_i \) and \( r_i \) are the \( i \)th binary bits of the best solution \( b \) and the binary solution \( r \), respectively. Since the lengths of individuals vary, only the first \( l \)-Q-bit of a binary solution is updated. \( l = \min (|r|, |b|) \), where \(|r|\) and \(|b|\) denote the lengths of individuals for the binary and best solutions, respectively. If \( f(r) \geq f(b) \), Q-bit \((\alpha_i, \beta_i)\) does not need to be updated since the current solution is better. If \( f(r) < f(b) \) and \(|r| > |b|\), the surplus Q-bits of the binary solution are deleted.

If \( f(r) < f(b) \) and \(|r| < |b|\), the insufficient part of the Q-bit string of the binary solution is supplemented randomly.

The third operation is crossover, which is a two-point crossover process for producing a new population \( cQ \). The two crossover points for different parent-individual pairs can have different positions. In contrast, in the HQGA, the positions of the two crossover points have to be the same. To process the crossover operation, two random crossover points \( c_1 \) and \( c_2 \) are generated, and then the Q-bit strings within \( c_1 \) and \( c_2 \) are exchanged for the two parent-individual pairs.

The forth operator is deletion, which is used to generate a new population \( dQ \). At first, a deletion point \( d \) is randomly generated within an individual, and then the number of Q-bits to be deleted is randomly determined. In the HQGA, the deleted Q-bit string occurs on the last Q-bits of an individual after point \( d \), but in the iQGA, the deleted string may be occurring inside an individual. For the robot trash collection task, the deleted Q-bit string is based on four Q-bits (represents \( \theta \)).

The fifth operation is the pull operator, which is used to generate \( pQ \). The pull operator is developed to lengthen the robot path. The number of new steering angles \((\theta)\) that should be added to the individual is first randomly determined. The pull point \( p \) is selected randomly. The new Q-bit string is then generated randomly and to inserted into the individual after point \( p \). The operation is based on four Q-bits. In contrast to the pull operation of the HQGA, the new Q-bit string is only connected with the last Q-bit, and cannot be inside of an individual.

After performing the five operations, the populations \( rQ \), \( cQ \), \( dQ \), and \( pQ \) are integrated into \( mQ \). Next, \( mQ \) is evaluated and the best \( n \) individuals according to fitness functions shown in Eq. (7) are obtained and recorded. The for-loop is repeated until the counter \( t \) reaches \( \text{maxGen} \).

4.4 Rule Generator

The procedure of Rule Generator modified from [9] is described as follows.

**Step 1:** Decode the best solution \( b \) obtained by the iQGA into \( \{ \theta_1, \theta_2, \ldots, \theta_n \} \), where \( n \in [5, 10] \).

**Step 2:** For all \( \theta_i \), evaluate the robot path, and then calculate its corresponding \( d_i \) and \( \phi_i \).

**Step 3:** For each pair \((d_i, \phi_i, \theta_i)\), form the \( i \)th if-then rule as:

\[
\text{If } d \text{ is } d_i \text{ and } \phi \text{ is } \phi_i \text{ Then } \theta \text{ is } \theta_i.
\]

(9)

**Step 4:** If two rules conflict, the rule whose corresponding robot path leads to the farthest distance between the robot and an obstacle is selected. In iQRICS, the rule with the smaller subscript \( i \) is selected, which may result in a path that is closer
to an obstacle. Steps 3 and 4 are repeated until all data pairs are transformed into if-then rules.

5. SIMULATION RESULTS

An illustration of the robot trash collection task is shown in Fig. 2. The environment includes a number of unknown obstacles and several pieces of litter with given positions. The robot, which is initially at Robot Home, has to plan a visiting schedule, and then move to collect each piece of litter in order according to the schedule. Finally, the robot moves to the trash bin at a given position to deposit the litter, and then moves back to Robot Home. It is here assumed that the positions of the pieces of litter are known to make path planning relatively simple. If the positions of the pieces of litters are unknown, goal seeking [15] and robot exploration [25] must be considered, which is relatively complex.

![Fig. 2. Illustration of robot trash collection task.](image1)

In this paper, the adaptive capability of the robot is emphasized. As shown in Fig. 2, the visiting schedule of litter may be litter 2 → litter 1 → litter 3 → litter 5 → litter 4. The initial position of the rectangular robot with size 26 cm × 26 cm is (25, 25), and the rectangular trash bin with size 60 cm × 60 cm is located at (1470, 1470).

The computer programs were written in MATLAB version 7.12 (R2011a). A computer with a 3.4-GHz CPU and 2 GB of RAM running Windows XP Professional (SP3) was used.

![Fig. 3. Decomposition of perception regions into three sectors.](image2)
5.1 Design of ICSHA for Controlling Robot

It is assumed that the robot has several sonar sensors for detecting obstacles. The perception range is divided into three circular side sectors, whose radius is 50 cm, as shown in Fig. 3. Vector variable $\textbf{IR}$ is used to record whether the sonar sensors perceive obstacles.

$$\textbf{IR} = [s_1, s_2, s_3]$$  \hspace{1cm} (10)

where $s_j = 1$ indicates that the sonar at the $j$th sector has detected at least one obstacle; otherwise, $s_j = 0$. In iQRICS, the if-then rules steer the robot toward the target. When an obstacle is detected, the robot has to turn an angle $\theta'$ according to Table 3. The ICSHA computational procedure proposed for the robot trash collection task is summarized in Algorithm 3.

The initial control rules are set up according to Eq. (3) (see Section 4.1). The environment information includes the locations of pieces of trash, the robot home (initial position of the robot), and the trash bin.

Table 3. Adjusted rules for steering angle based on sonar sensors.

<table>
<thead>
<tr>
<th>$[s_1, s_2, s_3]$</th>
<th>Adjusted $\theta'$ ($0^\circ \leq \theta \leq 180^\circ$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0, 0, 0]</td>
<td>$\theta' = \theta$</td>
</tr>
<tr>
<td>[0, 0, 1]</td>
<td>$\theta' = \theta - 12^\circ$</td>
</tr>
<tr>
<td>[0, 1, 0]</td>
<td>If $\theta \leq 180^\circ$, $\theta' = \theta - 48^\circ$; otherwise, $\theta' = \theta + 48^\circ$.</td>
</tr>
<tr>
<td>[0, 1, 1]</td>
<td>$\theta' = \theta - 48^\circ$.</td>
</tr>
<tr>
<td>[1, 0, 0]</td>
<td>$\theta' = \theta + 12^\circ$.</td>
</tr>
<tr>
<td>[1, 0, 1]</td>
<td>$\theta' = \theta$.</td>
</tr>
<tr>
<td>[1, 1, 0]</td>
<td>$\theta' = \theta + 60^\circ$.</td>
</tr>
<tr>
<td>[1, 1, 1]</td>
<td>If $x_r \geq x_t$, $\theta' = \theta + 90^\circ$; otherwise, $\theta' = \theta - 90^\circ$.</td>
</tr>
</tbody>
</table>

$x_r$ and $x_t$ denote the $x$-coordinates of the robot and target, respectively.

In the ICSHA main process, the ASGO approach presented in Algorithm 1 is used to obtain the visiting schedule $VS$ that indicates the picking order for each piece of litter (line 3). After all litters have been picked up, the robot deposits the trash in the trash bin and moves back to the robot home. On line 4, the visiting schedule incorporates the locations of the trash bin ($\text{loc}[\text{Trash Bin}]$) and the robot home ($\text{loc}[\text{Robot Home}]$). In Algorithm 3, the robot path $P$ and adaptable control rules $R$ are determined for completing the task. On line 5, the initial robot path is the empty set $\phi$ ($P \leftarrow \phi$). The iQRICS drives the robot to pick up each pieces of litter in order. If the qAdaptor within iQRICS is activated to obtain new rules, the control rules are updated as $R'$. $VS$ (1) indicates the first member of $VS$, which is the first subtask the robot should carry out. For finishing each subtask, i.e., picking up a piece of litter, the robot path is recorded to $p_i$, and each $p_i$ is incorporated into $P$. After the above process has been completed, the new control rules $R$ and entire robot path $P$ are obtained.
Algorithm 3: ICSHA process for trash collection task
1: Predefine if-then rules $R$ for robot and give trash positions
2: Determine the locations of robot home and trash bin
3: Visiting Schedule $VS \leftarrow$ run ASGO method (Algorithm 1)
4: $VS \leftarrow VS \cup \text{loc}[\text{Trash Bin}] \cup \text{loc}[\text{Robot Home}]
5: Let robotic path $P \leftarrow \phi$
6: while $VS$ is not empty do
7: $[R', p_i] \leftarrow \text{iQRICS}(R, VS(1), \text{environment information})$
8: Let $R \leftarrow R', P \leftarrow P \cup p_i$, and $VS(1) \leftarrow \phi$
9: end while
10: return control rules $R$ and robotic path $P$

5.2 Comparison of Performance for Scheduling the Visiting Order of Litters

To demonstrate the performance of the proposed ASGO approach, two comparisons among the ASGO, AS [11] and ACS [12] methods were conducted.

(1) Experiment 1: Comparison for 25 pieces of litter with various numbers of ants
   Twenty-five pieces of litter were randomly generated in the workspace. The parameter settings of ASGO are described in Section 3. Thirty runs were executed. For each run, the computational iterations ($NC_{\text{max}} = 2500/(\text{number of ants})$. The numbers of ants are varied from 5 to 25. Figure 4 shows the comparison results of ASGO, AS and ACS. The ASGO outperforms (shorter route length) AS and ACS in all five experiments. For AS, the results show that the route shortens with increasing number of ants. For ASGO, the shortest path was obtained with 10 ants. In addition, the best tours of ACS of all five experiments are almost the same, and all of them are shorter than that of AS.

(2) Experiment 2: Comparison for different trash sets with a given number of ants
   In the second experiment, trash sets with 15, 20, and 25 pieces of litter were randomly created, respectively. The parameter settings are the same as those for experiment 1, except $NC_{\text{max}} = 100$ and the number of ants = 10. The comparison results are listed in
Table 4. ASGO produces shorter routes in terms of the best route for all trash sets, but its average CPU time is slightly higher (0.15 seconds on average) than that of AS and ACS. The best route for all trash sets between AS and ACS are close, but the performance of average route lengths for all trash sets of ACS are worst. Besides, comparing with other methods, the average CPU time of ACS is least.

### Table 4. A comparison of ASGO and AS. Each experiment has 30 runs, and each run carries out 100 iterations.

<table>
<thead>
<tr>
<th>Pieces of litter</th>
<th>ASGO</th>
<th></th>
<th></th>
<th>AS</th>
<th></th>
<th></th>
<th>ACS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>3550</td>
<td>4314</td>
<td>1.154</td>
<td>3550</td>
<td>4017</td>
<td>1.030</td>
<td>3550</td>
<td>4960</td>
</tr>
<tr>
<td>20</td>
<td>4978</td>
<td>5872</td>
<td>1.610</td>
<td>4990</td>
<td>5590</td>
<td>1.503</td>
<td>5001</td>
<td>7231</td>
</tr>
<tr>
<td>25</td>
<td>4516</td>
<td>5579</td>
<td>2.107</td>
<td>4532</td>
<td>5248</td>
<td>1.938</td>
<td>4530</td>
<td>6317</td>
</tr>
</tbody>
</table>

*Best route length (cm). *Average route length (cm). *Average CPU time (seconds). Results in bold are the best values in a specific comparison index. All data in the table are averages of over 30 runs. The numbers of ants for all experiments are 10.

5.3 Comparison of Performance for Exploring New Robotic Paths

In this section, two experiments that demonstrate the performance of the proposed iQGA approach are described.

(1) Experiment 1: Comparison for four types of QGA

In the first experiment, a comparison of the iQGA with the HQGA [9], the tQGA [22], and the cQGA [27] was conducted. The tQGA, proposed by Li and Wang (page 583, [22]) is a benchmark used for comparing QGA methods. Similarly, the cQGA is a benchmark method proposed by Wang and Li (pages 26-27, [27]). In the experiment, when the best solution did not change in a certain number of consecutive generations, the best solution was kept and the others were replaced by randomly generated solutions.

In the simulation case, the robot had to move from (450, 450) to (450, 1425) without collisions in an environment with 20 obstacles. The population size and number of evolutionary generations maxGen were both set to 30. The number of movement steps of the robot was limited to within [25, 35], and thus 25 to 35 output variables (steering angles) were determined. The robot moved a fixed distance (33.75 cm) for each step. The experimental results are listed in Table 5. In terms of fitness value, the iQGA had the best quality over 30 runs. The cQGA requires the least CPU time, and the iQGA requires the most CPU time.

### Table 5. Comparison results for four types of QGA (30 runs were used for each method).

<table>
<thead>
<tr>
<th>Method</th>
<th>Fitness values (within [0, 1])</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>iHQGA</td>
<td>0.9987</td>
<td>0.8620</td>
</tr>
<tr>
<td>HQGA</td>
<td>0.9819</td>
<td>0.8323</td>
</tr>
<tr>
<td>tQGA</td>
<td>0.9099</td>
<td>0.6875</td>
</tr>
<tr>
<td>cQGA</td>
<td>0.8664</td>
<td>0.5491</td>
</tr>
</tbody>
</table>

Results in bold are the best values in a specific comparison index. All data are the averages of 30 runs. A higher fitness value indicates better robot path.
In the second experiment, a comparison of the iQGA with the tQGA and the cQGA methods was conducted in three different environments. The start and target positions of the robot in the environments with 15, 20, and 25 obstacles were: (1) (250, 450) and (700, 1000); (2) (800, 600) and (300, 1100); (3) (1000, 800) and (300, 100), respectively. The ranges of movement steps of the robot in the environments with 15, 20, and 25 obstacles were set to [25, 40], [15, 35], and [30, 50], respectively. The movement distance for each step was fixed at 25 cm in all environments. The population size and \( maxGen \) were 30 and 250, respectively.

<table>
<thead>
<tr>
<th>Number of obstacles</th>
<th>Method</th>
<th>Fitness values (within [0, 1])</th>
<th>Avg. CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
<td>Average</td>
</tr>
<tr>
<td>15</td>
<td>iQGA</td>
<td>0.9998</td>
<td>0.9654</td>
</tr>
<tr>
<td></td>
<td>tQGA</td>
<td>0.7320</td>
<td>0.9715</td>
</tr>
<tr>
<td></td>
<td>cQGA</td>
<td>0.9832</td>
<td>0.6979</td>
</tr>
<tr>
<td>20</td>
<td>iQGA</td>
<td>1.0000</td>
<td>0.9234</td>
</tr>
<tr>
<td></td>
<td>tQGA</td>
<td>0.9141</td>
<td>0.4879</td>
</tr>
<tr>
<td></td>
<td>cQGA</td>
<td>0.9851</td>
<td>0.4886</td>
</tr>
<tr>
<td>25</td>
<td>iQGA</td>
<td>0.9997</td>
<td>0.8632</td>
</tr>
<tr>
<td></td>
<td>tQGA</td>
<td>0.8634</td>
<td>0.5847</td>
</tr>
<tr>
<td></td>
<td>cQGA</td>
<td>0.8817</td>
<td>0.5809</td>
</tr>
</tbody>
</table>

Results in bold are the best values in a specific comparison index. All data are the averages of 30 runs. A higher fitness value indicates better robot path.

The results, as shown in Table 6, show that the best fitness values obtained by the iQGA in all environments are almost 1.0, giving the iQGA the highest performance. The worst fitness values of the iQGA are close to 0.5 in all experiments, giving the iQGA the highest performance. The cQGA required the least CPU time on average.

### 5.4 Comparison Between iQRICS and Other Methods

In order to demonstrate the adaptability of iQRICS, a comparison between iQRICS with sonar sensors, iQRICS without sonar sensors, QRICS [9], SyICS [7], and SEICS [5] was conducted. All methods except the iQRICS with sonar sensors used touch sensors to detect collisions. In an environment with 20 obstacles, the robot had to move from (25, 25) to (1470, 1470) without collisions. The fixed movement distance of the robot was 33.75 cm for all method except iQRICS without sonar sensors, which had a movement distance of 25 cm to increase performance.

The simulation results are summarized in Fig. 5. The initial control rules for all methods were established so that the robot could move toward the target along a straight line. The original robot paths of all methods except iQRICS with sonar (Fig. 5 (a)) led to collisions. This shows that the sonar can effectively steer the robot away from obstacles. Two simulation figures, the original path and the well-adapted path, are shown for
Fig. 5. Comparison results of iQRICS with sensors, iQRICS without sensors, QRICS, SyICS, and SEICS in an environment with 20 obstacles.
iQRICS without sonar, QRICS, and SyICS. Since SEICS could not find a path without collisions, it does not have a well-adapted path, which is a robot path without collisions obtained after the control rules have been adjusted by the adaptive mechanism.

Table 7 shows comparison results in terms of adaptability. Except for iQRICS with sonar, the approach, iQRICS without sonar, has the best adaptability since it only requires one adaptation, and updates the least number of rules. iQRICS does not have any collision in the original path, and thus it obtains the best path. An adaptation of the robot is defined as one complete run of the control module. QRIS requires the most number of adaptations (i.e., 5 runs) to achieve a well-adapted path. In addition, iQRICS without sonar has the shortest path lengths for the original and well-adapted paths.

Table 7. Comparison results of control modules in an environment.

<table>
<thead>
<tr>
<th>Method</th>
<th>No. of adaptations</th>
<th>No. of adjusted rules</th>
<th>Original path length (cm)</th>
<th>Well-adapted path length</th>
</tr>
</thead>
<tbody>
<tr>
<td>iQRICS with sonar</td>
<td>0</td>
<td>0</td>
<td>2295</td>
<td>–</td>
</tr>
<tr>
<td>iQRICS without sonar</td>
<td>1</td>
<td>16</td>
<td>2200</td>
<td>2200</td>
</tr>
<tr>
<td>QRICS</td>
<td>5</td>
<td>17</td>
<td>2228</td>
<td>2228</td>
</tr>
<tr>
<td>SyICS</td>
<td>2</td>
<td>51</td>
<td>2550</td>
<td>2378</td>
</tr>
<tr>
<td>SEICS</td>
<td>–</td>
<td>–</td>
<td>2363</td>
<td>–</td>
</tr>
</tbody>
</table>

Results in bold are the best values in a specific comparison index. Number of adaptations indicates the number of runs required to obtain a path without collisions.

It can thus be concluded that the proposed iQRICS, with or without sonar, has the best path without collision or best adaptability of the methods tested.

5.5 Simulation Analysis of ICSHA

In this section, the simulation results for the proposed ICSHA approach for completing the trash collection task are presented. A regular environment was constructed with 10 pieces of litter and 10 unknown obstacles, as shown in Fig. 6. The robot had to move from (25, 25) to collect the trash at known positions, move to the trash bin to deposit the trash, and then go back to the robot home at (25, 25). To show the adaptive behavior of robot, the radius of the sonar sensors was reduced to 33 cm. There was one collision, as shown in Fig. 6 (a); however, when the radius of the sonar sensors was increased to 50 cm, there were no collisions. When the robot moves from litter 6 to litter 7, it collides with an obstacle. Thus, the qAdaptor is activated to obtain adaptable rules to update rule base. According to the exploration process described in Section 4.2, the robot firstly moves back three steps and the iQGA searches a new path consisted of 14 moving steps in order to avoid the obstacle. The new path is then to be transformed into 20 if-then rules to renew the rule base.

The litter was gathered from Litter 1 to Litter 10 in order. Fig. 6 (b) shows the well-adapted trajectory after two adaptations. The lengths of the original path and the well-adapted path are 6581 cm and 6624 cm, respectively. ICSHA can automatically adjust its control rules to adapt to the environment. For example, the sensing range of sonar sensors can be ignored because the adaptive mechanism (qAdaptor) will find suitable rules. ICSHA will adjust its rule base to adapt to complex environments.
6. CONCLUSIONS

A framework of self-adaptive intelligent control system with hierarchical architecture, namely ICSHA, has been proposed. ICSHA is an intelligent control system that can adjust its control rules to adapt various environments. The hierarchical ICSHA has three layers: the Planning Layer, the Executive Layer, and the Behavior Layer. In Planning Layer, an evolutionary algorithm, ASGO, is proposed to obtain the best schedule for determining the executive order of all subtasks. This schedule is then sent to the Executive Layer, which creates an agenda of subtasks. A control module, iQRICS, in the Behavior Layer carries out a subtask assigned by the Executive Layer.

A robot trash collection task with known positions of litter was constructed to demonstrate the proposed method. The performances of the proposed ASGO, iQGA, and iQRICS approaches within ICSHA were demonstrated via comparisons with other methods. The proposed methods outperform the other methods in terms of solution quality. Moreover, the simulation results demonstrate the adaptability of ICSHA. An advantage to using a self-adaptive mechanism, i.e., the qAdaptor in iQRICS, is that the initial rules can be simply or crudely set up.

In future study, we may assume that the litters are with unknown positions for more approaching the practical applications, e.g., park cleaning.

ACKNOWLEDGMENT

We would like to thank Dr. Ju-Yin Cheng for her assistance in preparing the English version of this manuscript.

REFERENCES

2. A. A. Bakhtiari, H. Navid, J. Mehri, and D. D. Bochtii, “Optimal route planning of agricultural field operations using ant colony optimization,” Agricultural Engineer-


Cheng-Hsiung Chiang (姜正雄) received his MS degree in Industrial Engineering from National Tsing Hua University, Taiwan, and Ph.D. degree in Industrial Management from National Cheng Kung University, Taiwan, in 1997 and 2003, respectively. He is currently an Assistant Professor of Information Management with Hsuan Chuang University, Taiwan. He is a member of IEEE. His main research interests include intelligent control, robotic path planning, and the related fields of computational intelligence.

Liang-Hsuan Chen (陳梁軒) received the B.S. and M.S. degrees in Industrial Management from National Cheng Kung University, Tainan, Taiwan, in 1980 and 1982, respectively, and the Ph.D. degree in Industrial Engineering from the University of Missouri, Columbia. He is currently a Distinguished Professor of Industrial and Information Management with National Cheng

Dr. Chen was the Editor-in-Chief of the Journal of the Chinese Institute of Industrial Engineers in 2005 and 2006. He received the Outstanding Research Award of management science from the National Science Council of Taiwan in 2000. He served as the Committee Chair of the management area in the National Science Council of Taiwan from 2009 to 2011.