A Hybrid Mobile Robot Architecture with Integrated Planning and Control

Kian Hsiang Low
Dept. Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543, Singapore
lowkianh@comp.nus.edu.sg

Wee Kheng Leow
Dept. Computer Science
National University of Singapore
3 Science Drive 2
Singapore 117543, Singapore
leowwk@comp.nus.edu.sg

Marcelo H. Ang, Jr.
Dept. Mechanical Engineering
National University of Singapore
10 Kent Ridge Crescent
Singapore 119260, Singapore
mpeangh@nus.edu.sg

ABSTRACT

Research in the planning and control of mobile robots has received much attention in the past two decades. Two basic approaches have emerged from these research efforts: deliberative vs. reactive. These two approaches can be distinguished by their different usage of sensed data and global knowledge, speed of response, reasoning capability, and complexity of computation. Their strengths are complementary and their weaknesses can be mitigated by combining the two approaches in a hybrid architecture. This paper describes a method for goal-directed, collision-free navigation in unpredictable environments that employs a behavior-based hybrid architecture with asynchronously operating behavioral modules. It differs from existing hybrid architectures in two important ways: (1) the planning module produces a sequence of checkpoints instead of a conventional complete path, and (2) in addition to obstacle avoidance, the reactive module also performs target reaching under the control of a self-organizing neural network. The neural network is trained to perform fine, smooth motor control that moves the robot through the checkpoints. These two aspects facilitate a tight integration between high-level planning and low-level control, which permits real-time performance and easy path modification even when the robot is en route to the goal position.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence; I.2.6 [**Artificial Intelligence**]: Learning—*connectionism and neural nets*; I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Design, Experimentation, Performance, Reliability, Theory

Keywords

hybrid agent architectures, learning, mobile agents, perception and action in agents, performance, self-organizing systems

1. INTRODUCTION

Research in the planning and control of mobile robots has received much attention in the past two decades. Two basic approaches have emerged from these research efforts: deliberative vs. reactive. The top-down deliberative (planner-based) approach [9, 15, 24, 28] uses a global world model to generate the most appropriate sequence of actions for the agent to reach a specified goal. By performing deliberate planning, this approach can generate optimal sequences in a static complex environment. Since the planning process typically takes some time to execute, this approach cannot react quickly to unforeseen changes in the environment.

The bottom-up reactive approach [8, 11] directly couples sensed data to motor actions. It can thus respond robustly and timely to unexpected obstacles and unforeseen changes in the environment. However, since it lacks global knowledge about the environment, the sequence of actions produced may not be globally optimal. It may also fail to react correctly in a complex environment.

These two approaches can be distinguished by their different usage of sensed data and global knowledge, speed of response, reasoning capability, and complexity of computation. Their strengths are complementary and their weaknesses can be mitigated by combining the two approaches in a hybrid architecture. Among the existing hybrid frameworks, [2, 3, 12, 14, 16, 29, 32, 33] emphasize high-level task planning. On the other hand, [5, 6, 10, 23, 30] focus on integrating low-level reactive motor control with motion path planning.

This paper describes a method for goal-directed, collision-free navigation in complex, unpredictable environments that employs a behavior-based hybrid architecture with asynchronously operating behavioral modules. It differs from most existing hybrid architectures in three important ways:

1. Our architecture is one of very few frameworks that perform continuous response encoding [1] (infinite set of responses) rather than discrete response encoding (finite, enumerated set of responses). This method can produce very low-level velocity or torque control

of motors to perform fine, smooth motion control.

2. In our framework, the planning module produces a sequence of checkpoints instead of a complete motion path. The constraint of adhering strictly to a generated path no longer exists. This approach is adopted in [23] as well.

3. The real-time performance of existing hybrid architectures is still not optimal because the capability of the reactive components has not been fully exploited. In extreme cases, the workload of the high-level planning module far exceeds that of the low-level reactive module (e.g., [7, 18, 29, 34]). The planning module plans the exact motion path and generates the detailed sequence of actions to be executed by the actuators. The reactive module performs a single task, i.e., obstacle avoidance, by making minor modifications to an otherwise good course of action.

   A good example of exploiting the strengths of reactive robotics is the *compliant motion control* method of Jamisola et al. [19] for controlling a reactive robot manipulator to perform polishing task. By sensing and reacting to the forces acting on the manipulator, this method can provide real-time, fine, and smooth motion control for the manipulator to follow the contours of the surface to be polished.

   Our hybrid architecture is another example. In addition to obstacle avoidance, the reactive module in our architecture also performs target reaching under the control of a self-organizing neural network. The neural network is trained to perform real-time, fine, and smooth motor control that moves the robot through the checkpoints planned by the planning module.

These characteristics facilitate a tight integration between high-level planning and low-level control, which permits real-time performance and easy path modification even when the robot is en route to the goal position.

2. INTEGRATED FRAMEWORK

2.1 Overview

Our integrated framework consists of two main blocks with four modules (Fig. 1). At the highest level, the deliberative planning module produces a sequence of checkpoints from the start point to the goal using a variation of the cell decomposition method in [30]. The main difference is that our algorithm operates in the robot's workspace instead of the configuration space. This is unlike conventional planning algorithms (e.g., [22, 25]) that plot detailed paths. Also, the current implementation focuses on achieving a single goal at a time. To achieve multiple, possibly conflicting goals, high-level planning algorithms may be used to order the goals.

The reactive block consists of three levels. At the first level, the *target reaching* module determines the motion path between checkpoints. It senses the checkpoint state relative to the current state and outputs appropriate motor control signals. It contains a self-organizing neural network which is trained to produce a sequence of low-level (motor velocity) control commands to move the robot from one checkpoint to the next.

The next lower-level module, *obstacle avoidance*, senses the presence of local unforeseen or moving obstacles and produces additional motor control commands to repel the robot away from the obstacles.

The lowest-level *homeostatic control* module senses the internal state of the robot to maintain internal stability by coordinated responses that automatically compensate for environmental changes [1]. An example of this low-level control is operational space control [19, 20], which senses the internal joint angles, velocities, and forces to regulate the dynamic behavior of a robot manipulator during task execution. This module is not strictly required for mobile robot navigation, but is crucial for mobile manipulation tasks [21] where the robot is manipulating an object or the environment while its base is in motion.

The three lower-level modules constitute a reactive model of motion control. The *command fusion* module combines the control commands from the reactive components into a final command that is sent to the actuators.

All the modules operate asynchronously at different rates. The planning module typically operates at the time scale of several seconds or minutes depending on task complexity. The target reaching module operates at about 1 second between servo ticks while the obstacle avoidance module operates at intervals of 100 ms. The homeostatic control module operates at 1 ms interval. The command fusion module is activated as and when control commands are generated (see Section 2.5 for details). The asynchronous execution of modules is the key to preserving reactive capabilities while allowing improvement of performance by the deliberative planner. In fact, the planner can be removed and the resulting decapitated architecture degrades to a purely reactive system capable of less complex motion tasks.

The hybrid architecture is applicable to both mobile robot and robot manipulator. This paper focuses on the target reaching and obstacle avoidance modules and their integration with the planning module, with specific application to mobile robot target reaching task. This task is performed by an *Extended Kohonen Map* (EKM) [31] which is trained to produce a sequence of motor velocity commands. The method utilized in the planning module for generating checkpoints is described in a separate paper [26]. The next section describes the control method, which is achieved through *indirect mapping* of sensory input to motor control. The advantages of indirect-mapping EKM over direct-mapping EKM [17, 35] will be discussed in the following sections.

2.2 Indirect Mapping

Our indirect-mapping EKM adopts an egocentric representation of the sensory input vector $\mathbf{u}_p = (\alpha, d)^T$ where α and d are the direction and the distance of a checkpoint relative to the robot's current location and heading. At the goal state at time T, $\mathbf{u}_p(T) = (\alpha, 0)^T$ for any α.

If sensorimotor coordination is a linear problem, then the motor control vector $\mathbf{c}_p$ would be related to the sensory input vector $\mathbf{u}_p$ by the linear equation

$$\mathbf{c}_p = \mathbf{M}\mathbf{u}_p \tag{1}$$

where $\mathbf{M}$ is a matrix of motor control parameters. The control problem would be reduced to one of determining $\mathbf{M}$ from training samples.

In practice, however, sensorimotor coordination is a non-

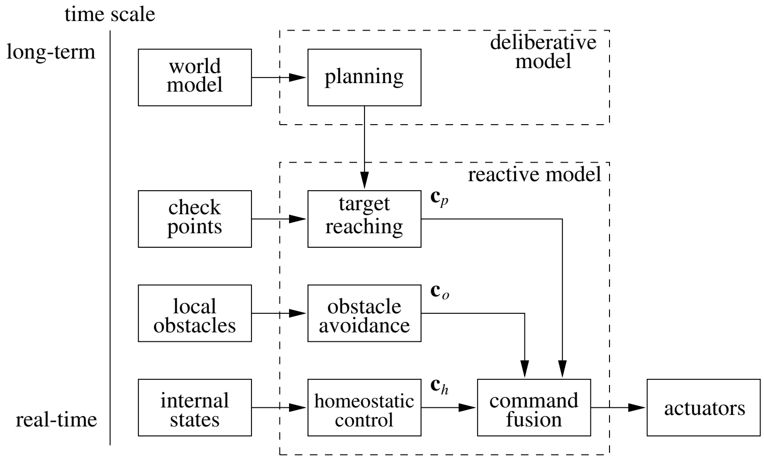

Figure 1: A hybrid architecture for integrated planning and control. It combines the planning module of a deliberative model and the behavioral-based control of a reactive model.

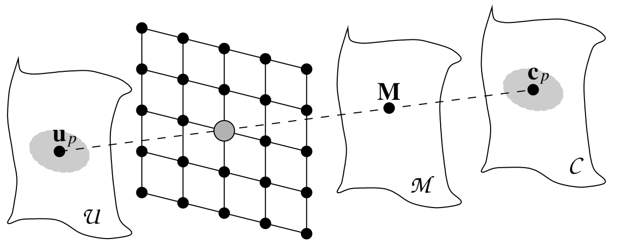

Figure 2: Target reaching module. EKM neurons map the sensory input space $\mathcal{U}$ indirectly to the motor control space $\mathcal{C}$ through the control parameter space $\mathcal{M}$.

linear problem because a real motor takes a finite but nonzero amount of time to accelerate or decelerate in order to change speed. This nonlinear problem is further complicated in non-holonomic robots. To solve the nonlinear problem, the EKM is trained to partition the sensory input space $\mathcal{U}$ into locally linear regions. Each neuron i in the EKM has a sensory weight vector $\mathbf{w}_i$ that encodes the region in $\mathcal{U}$ centered at $\mathbf{w}_i$. It also has a set of output weights which encode the outputs produced by the neuron. However, unlike existing methods (e.g., [17, 35]), the output weights $\mathbf{M}_i$ of neuron i represent control parameters in the parameter space $\mathcal{M}$ instead of the motor control vector (Fig. 2). The control parameter matrix $\mathbf{M}_i$ is mapped to the actual motor control vector $\mathbf{c}_p$ by the linear model of Eq. 1. With indirect-mapping EKM, motor control is performed as follows:

Motor Control

Given a sensory input vector $\mathbf{u}_p$,

1. Determine the winning neuron k.
   The winning neuron k is the neuron whose sensory weight vector $\mathbf{w}_k = (\alpha_k, d_k)^T$ is nearest to the input $\mathbf{u}_p = (\alpha, d)^T$:
   $$D(\mathbf{u}_p, \mathbf{w}_k) = \min_{i \in \mathcal{A}(\alpha)} D(\mathbf{u}_p, \mathbf{w}_i) \ . \tag{2}$$
   The difference $D(\mathbf{u}_p, \mathbf{w}_i)$ is a weighted difference between $\mathbf{u}_p$ and $\mathbf{w}_i$:
   $$D(\mathbf{u}_p, \mathbf{w}_i) = (\gamma_\alpha(\alpha - \alpha_i)^2 + \gamma_d(d - d_i)^2)^{1/2} \tag{3}$$
   where γ_α and γ_d are constant parameters. The minimum in Eq. 2 is taken over the set $\mathcal{A}(\alpha)$ of neurons encoding very similar angles as α:
   $$\begin{gathered} |\alpha - \alpha_i| \leq |\alpha - \alpha_j|, \\ \text{for each pair } i \in \mathcal{A}(\alpha), j \notin \mathcal{A}(\alpha) \ . \end{gathered} \tag{4}$$
   In other words, direction has priority over distance in the competition between EKM neurons. This method allows the robot to quickly orientate itself to face the target while moving towards it [35].

2. Compute motor control vector $\mathbf{c}_p$ for target reaching:
   $$\mathbf{c}_p = \begin{cases} \mathbf{M}_k \mathbf{u}_p & \text{if } -\mathbf{c}^* \leq \mathbf{M}_k \mathbf{u}_p \leq \mathbf{c}^* \\ \mathbf{M}_k \mathbf{w}_k & \text{otherwise.} \end{cases} \tag{5}$$
   The constant vector $\mathbf{c}^*$ denotes the upper limit of physically realizable motor control signal. For instance, for the Khepera robots (http://www.k-team.com/robots/khepera/), $\mathbf{c}_p$ consists of the motor speeds v_l and v_r of the robot's left and right wheels. In this case, we define $\mathbf{c}_p \leq \mathbf{c}^*$ if $v_l \leq v_l^*$ and $v_r \leq v_r^*$. Note that if $\mathbf{c}_p$ is beyond $\mathbf{c}^*$, simply saturating the wheel speeds does not work. For example, if the target is far away and not aligned with the robot's heading, then saturating both wheel speeds only moves the robot forward. Without correcting the robot's heading, the robot will not be able to reach the target.

The motor control algorithm is applied at each time step t to compute the motor control vector $\mathbf{c}_p(t)$ for the current sensory input $\mathbf{u}_p(t)$. It is repeated until the robot reaches the goal state $\mathbf{u}_p(T)$ at time step T.

The direct-mapping approach [17, 35] maps all the sensory inputs $\mathbf{u}_p$ in a region in the sensory input space $\mathcal{U}$, represented by a neuron k, to the same discrete point $\mathbf{c}_k$ in the motor output space $\mathcal{C}$, i.e., $\mathbf{c}_p = \mathbf{c}_k$. As a result, only a small number of points in $\mathcal{C}$ are represented by the neurons' outputs, i.e., the motor output space is very sparsely sampled. In contrast, the indirect approach maps each $\mathbf{u}_p$ in a locally linear region in $\mathcal{U}$ to a different point $\mathbf{c}_p$ in $\mathcal{C}$ through Eq. 5. Since this mapping is linear and continuous, the indirect approach maps a region in $\mathcal{U}$ to a region in $\mathcal{C}$, thus providing finer and smoother control of the robot's motion than does direct mapping.

2.3 Self-Organization of Indirect Mapping

In contrast to most existing methods, online training is adopted for the indirect-mapping EKM. Initially, the EKM has not been trained and the motor control vectors $\mathbf{c}_p$ generated are inaccurate. Nevertheless, the EKM self-organizes, using these control vectors $\mathbf{c}_p$ and the corresponding robot displacements $\mathbf{v}$ produced by $\mathbf{c}_p$, to map $\mathbf{v}$ to $\mathbf{c}_p$ indirectly. As the robot moves around and learns the correct mapping, its sensorimotor control becomes more accurate. At this stage, the same online training can still be performed, and it mainly fine tunes the indirect mapping. The self-organized training algorithm (in an obstacle-free environment) can be summarized as follows:

Self-Organized Training

Repeat

1. Get sensory input $\mathbf{u}_p$.
2. Execute motor control algorithm and move robot.
3. Get new sensory input $\mathbf{u}'_p$ and compute actual displacement $\mathbf{v}$ as a difference between $\mathbf{u}'_p$ and $\mathbf{u}_p$.
4. Use $\mathbf{v}$ as the training input to determine the winning neuron k (same as Step 1 of Motor Control).
5. Adjust the weights $\mathbf{w}_i$ of neurons i in the neighborhood $\mathcal{N}_k$ of the winning neuron k towards $\mathbf{v}$:
$$\Delta \mathbf{w}_i = \eta\, G(k,i)(\mathbf{v} - \mathbf{w}_i) \tag{6}$$
where $G(k,i)$ is a Gaussian function of the distance between the positions of neurons k and i in the EKM, and η is a constant learning rate.
6. Update the weights $\mathbf{M}_i$ of neurons i in the neighborhood $\mathcal{N}_k$ to minimize the error e:
$$e = \frac{1}{2} G(k,i) \|\mathbf{c}_p - \mathbf{M}_i \mathbf{v}\|^2 \ . \tag{7}$$
That is, apply gradient descent to obtain
$$\Delta \mathbf{M}_i = -\eta \frac{\partial e}{\partial \mathbf{M}_i} = \eta\, G(k,i)(\mathbf{c}_p - \mathbf{M}_i \mathbf{v})\mathbf{v}^T \ . \tag{8}$$

At each training cycle, the weights of the winning neuron k and its neighboring neurons i are modified. The amount of modification is proportional to the distance $G(k,i)$ between the neurons in the EKM. The input weights $\mathbf{w}_i$ are updated towards the actual displacement $\mathbf{v}$ and the control parameters $\mathbf{M}_i$ are updated so that they map the displacement $\mathbf{v}$ to the corresponding motor control $\mathbf{c}_p$.

After self-organization has converged, the neurons will stabilize in a state such that $\mathbf{v} = \mathbf{w}_i$ and $\mathbf{c}_p = \mathbf{M}_i \mathbf{v} = \mathbf{M}_i \mathbf{w}_i$. For any winning neuron k, given the sensory input $\mathbf{u}_p = \mathbf{w}_k$, the neuron will produce a motor control output $\mathbf{c}_p = \mathbf{M}_k \mathbf{w}_k$ which yields a desired displacement of $\mathbf{v} = \mathbf{w}_k$. For a sensory input $\mathbf{u}_p \neq \mathbf{w}_k$ but close to $\mathbf{w}_k$, the motor control output $\mathbf{c}_p = \mathbf{M}_k \mathbf{u}_p$ produced by neuron k will still yield the correct displacement if linearity holds within the input region

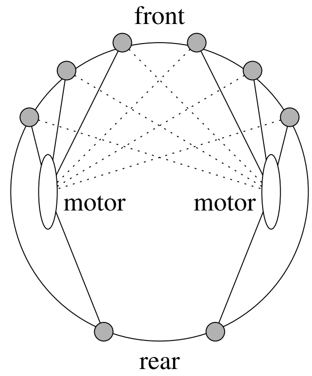

Figure 3: Obstacle avoidance. The connections between the forward-facing sensors on one side of the robot's body with the motor on the opposite side have large negative values (dotted lines), while the other connections have small positive values (solid lines).

that activates neuron k. Therefore, given enough neurons to produce an approximate linearization of the sensory input space $\mathcal{U}$, the indirect-mapping EKM can produce finer and smoother motion control than that of direct-mapping EKM.

2.4 Obstacle Avoidance

The reactive obstacle avoidance module adopts the architecture of Braitenberg's Type-3C vehicle [4]. Given a set $\mathbf{u}_o$ of sensor inputs, the motor velocity $\mathbf{c}_o$ for obstacle avoidance is computed as:
$$\mathbf{c}_o = \mathbf{Z}\, \mathbf{u}_o \tag{9}$$
where $\mathbf{Z} = [z_{ij}]$ is the control matrix. The matrix elements z_{ij} that link the forward-facing sensors on one side of the robot's body with the motor on the opposite side have large negative values, while the other matrix elements have small positive values (Fig. 3). When the robot senses the presence of an obstacle, say, in front and on the left, the right motor will rotate backward faster than the left motor's rotation forward, thus turning the robot away from the obstacle.

A similar approach is adopted in the architecture of Decugis and Ferber [13]. The main difference is that Decugis and Ferber assigned different sets of weights to different behaviors including obstacle avoidance, left wall following, right wall following, corridor following, left static turn, right static turn, and forward move. On the other hand, our method uses only one set of weights. It removes the need to recognize different situations and select different behaviors, and thus simplifies the reactive modules. As will be seen in Section 3.2, different behaviors can still emerge naturally from the interaction between the robot and the environment.

2.5 Command Fusion

The motor control for obstacle avoidance $\mathbf{c}_o$ is added to the motor control for target reaching $\mathbf{c}_p$ to produce the final motor control signal $\mathbf{c}$:
$$\mathbf{c} = \beta\, \mathbf{c}_p + (1-\beta)\, \mathbf{c}_o \tag{10}$$
where β is a constant parameter. The homeostatic control $\mathbf{c}_h$ of the motors is omitted. Equation 10 is analogous to the potential fields method for obstacle avoidance [10, 22]

and is able to overcome small unforeseen obstacles and non-adversarial moving obstacles.

Recall that the target reaching and obstacle avoidance modules run at different rates. Each time one of the modules produces a new motor control signal, it updates a global motor state, which then causes the combined motor control signal $\mathbf{c}$ to be sent to the robot's wheels to drive the robot. In the absence of obstacles, the motor control signal will be sent at regular intervals. In the presence of obstacles, additional control signal may be sent as and when obstacles are detected. This method allows the robot to run as smoothly as possible and to make adjustments only when necessary.

3. EXPERIMENTS AND DISCUSSIONS

3.1 Quantitative Evaluation

Experiments were conducted to assess both the quantitative and qualitative performance of the hybrid architecture for mobile robot navigation. The experiments were performed using Webots (http://www.cyberbotics.com), the simulator for Khepera mobile robots. In the experiments, EKMs with 15×15 neurons were trained in an obstacle-free environment. Each training/testing trial took 100,000 time steps and each time step for target reaching control lasted 1.024 sec. During training, the weights of the EKM were initialized to correspond to regularly spaced locations in the sensory input space $\mathcal{U}$. The robot began the training at the origin and a randomly selected sequence of checkpoints were presented. The robot's task was to move to the checkpoints, one at a time, and weight modification was performed at each time step after the robot had made a move. At each time interval of 10,000 steps during training, a fixed testing process was conducted. In each test, the robot began at the origin and was presented with 50 random target locations in sequence. The robot's task was to move to each of the target locations (this time, no training was performed). The above training/testing trial was repeated five times and testing performance was averaged over the five trials.

The testing performance index measured in the above trials is the *mean positioning error* E, which is the average distance ε_i between the center of the robot and the ith target location after it has come to a stop (i.e., motor control $\mathbf{c} = \mathbf{0}$):

$$E = \frac{1}{RN} \sum_i \varepsilon_i \tag{11}$$

where R is the number of trials and N is the number of testing target locations. Experimental results (Fig. 4) show that, with indirect mapping, the mean positioning error E began to stabilize at 50,000 time steps. This implies that the self-organization of EKM also began to stabilize at 50,000 time steps. At the end of 100,000 time steps, the robot driven by the trained EKM had a mean positioning error of 3 mm. In comparison, the same EKM that adopted the direct-mapping method stabilized at about the same time but the direct-mapping robot had a mean positioning error of 8 mm.

The radius of the robot is 25 mm. So, it is reasonable to regard the robot to have reached (and touched) a target if the distance-to-target ε is less than 25 mm. The next two performance indices are based on this target reaching criterion.

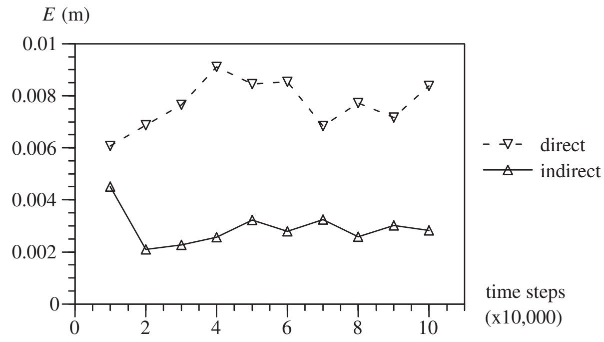

Figure 4: Mean positioning error at various training stages.

The *target reaching probability* $P(\varepsilon)$ measures the probability of the robot reaching closer than a distance of ε (with or without stopping) from the target locations after training. The *normalized time-to-target* $T(\varepsilon)$ measures how long it takes the robot to reach closer than a distance of ε after training:

$$T(\varepsilon) = \frac{1}{RN} \sum_i \tilde{t}_i(\varepsilon) \ , \quad \tilde{t}_i(\varepsilon) = \frac{t_i(\varepsilon)}{l_i} \tag{12}$$

where $t_i(\varepsilon)$ is the earliest time it takes the robot to reach closer than a distance of ε from the ith target location, l_i is the straight line distance between target locations $i-1$ and i, and $\tilde{t}_i$ is the normalized time taken to reach target i. That is, normalized time-to-target measures the average amount of time the robot takes to travel a distance of 1 m.

Experimental results (Fig. 5) show that, with indirect mapping method, the robot can get very close to a target with high probability (> 0.9) and could reach the targets in about 9 time steps. In contrast, with direct mapping, the robot has a lower probability (< 0.9) of reaching close to the targets and took about 17.5 time steps to reach them.

The above test results show that, with indirect mapping, the robot can reach the targets faster and closer than with direct mapping. The advantages of indirect mapping can also be assessed from the results of self-organization. Figure 6 illustrates the EKMs at the end of one of the five training trials. The neurons in the direct-mapping EKM were clustered into four clusters: $d = 0$ and $\alpha = -3, 0, +3$ radian. Although the neurons in the indirect-mapping EKM also clustered in a similar manner, its neurons were more spread out. Moreover, they sampled distances up to 0.16 m whereas direct-mapping neurons sampled distances only up to 0.11 m. Note that 0.16 m is the furthest that a Khepera robot can move in a single time step of 1 second. Therefore, indirect-mapping EKM samples the sensory input space more completely than does the direct-mapping EKM, and produces finer control of the robot.

3.2 Qualitative Evaluation

The robot's performance was also qualitatively assessed in an environment under three unforeseen conditions: (1) static obstacle, (2) moving obstacle, and (3) unexpected change in environment. The environment consisted of three rooms connected by two doorways (Figs. 7–9). The robot began in the left-most room and was tasked to move to the right-

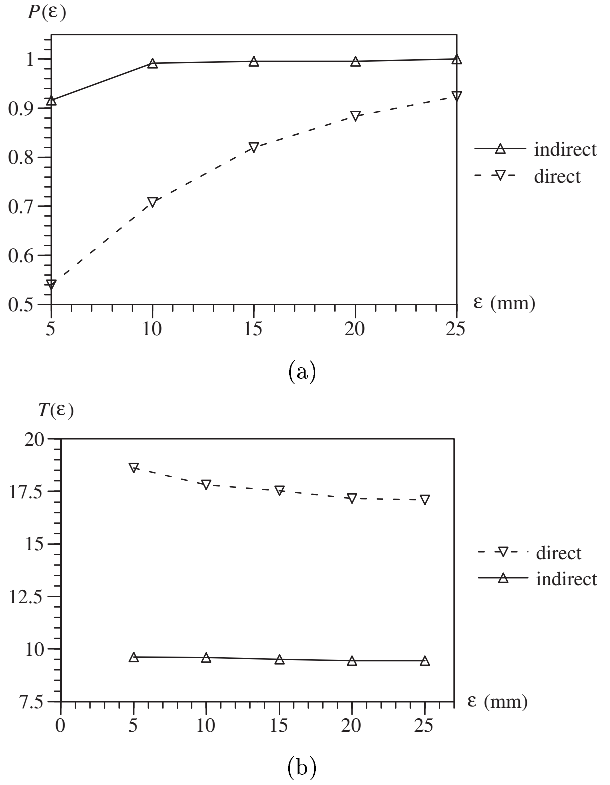

Figure 5: Performance comparison after training. (a) Target reaching probability. (b) Normalized time-to-target.

most room via three checkpoints. The robot was regarded to have reached a checkpoint if it was less than 5 mm from the checkpoint. The robot was required to stop at the goal. The target reaching module ran at 1.024 sec interval while the obstacle avoidance module ran at 0.128 sec interval.

In the first test (Fig. 7), an unforeseen static obstacle was placed in the middle room between the first and second checkpoint. Just before reaching the first checkpoint, the robot detected a wall on its right and deviated slightly to the left to reach the first checkpoint. While moving towards the second checkpoint, the robot detected the unforeseen static obstacle. It reacted to the presence of the obstacle by going around the obstacle to reach the second checkpoint. At the second checkpoint, the robot made a sharp turn due to the direction of the goal and headed towards it.

In the second test (Fig. 8), a mobile robot, following an anti-clockwise circular path, served as the moving obstacle. When the target reaching robot first met the obstacle on its left, it tried to avoid by turning right. Subsequently, it encountered the obstacle on its right and was diverted to the left before it moved out of the obstacle's path, headed towards the second checkpoint, and finally towards the goal.

In the last test (Fig. 9), the same checkpoints as the previous tests were initially planned for the robot. However, while the robot was en route to the second checkpoint, the high-level planning module realized that the environment had changed. A new checkpoint was planned and given to the target reaching module. Consequently, the target reaching

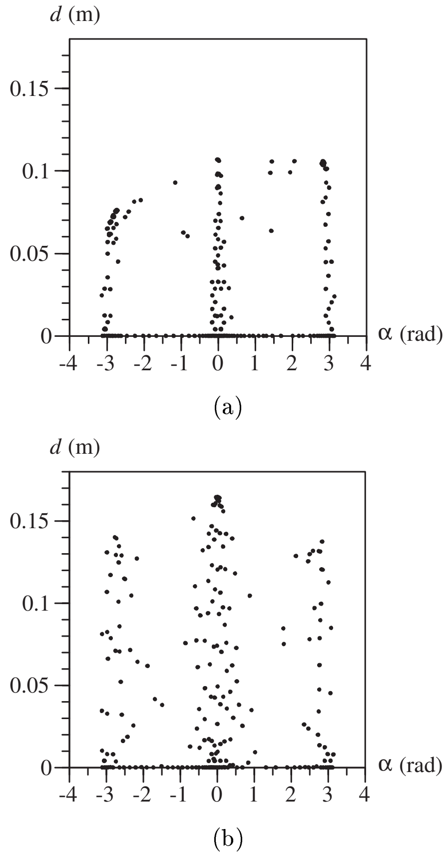

Figure 6: Self-organization results of EKM using (a) direct and (b) indirect mapping. Each dot denotes the weights $\mathbf{w}_i = (\alpha_i, d_i)^T$ of a neuron.

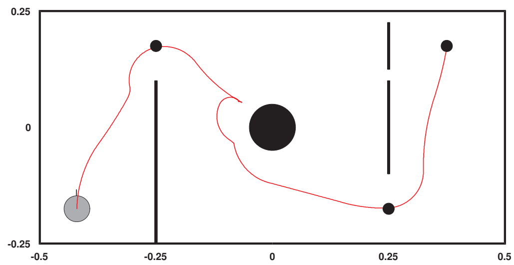

Figure 7: Motion of robot (gray) in an environment with an unforeseen static obstacle (black). The checkpoints (small black dots) are located at the doorways and the goal position. The robot was able to go around the obstacle to reach the goal. The robot, obstacle, and rooms are drawn to the same scale.

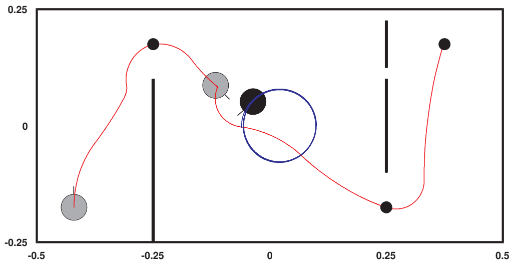

Figure 8: Motion of robot (gray) in an environment with an obstacle (black) moving in an anti-clockwise circular path. The robot was able to negotiate past the moving obstacle and proceed to the second checkpoint.

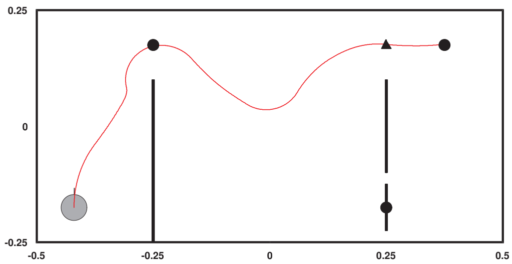

Figure 9: Motion of robot (gray) in an environment that changed. The initial checkpoints were planned with the assumption that the lower doorway was opened. The planning module changed the second checkpoint to a new checkpoint (triangle) while the robot was en route to the old one. The target reaching module was able to react immediately and it changed the robot's heading.

module changed the heading of the robot en route, so that it could reach the goal through the new checkpoint. This test clearly demonstrates the advantage of our integrated approach. First, a plan can be easily modified by changing only the checkpoints. Second, the target reaching module can react immediately to the change of a checkpoint, and produce a course change at ease. In contrast, existing architectures that plan the entire path need to make detailed modifications to the path such as how to turn the robot around, taking into account the robot's current motion.

In all cases, the robot under the control of the trained EKM was able to move to the checkpoints successfully. The paths taken by the robot between checkpoints were not perfectly straight due to several realistic constraints. The two-wheeled robot was non-holonomic. The motor control injections by the obstacle avoidance and target reaching modules were not strictly continuous, but at discrete servo intervals. Furthermore, the Webots simulator automatically injected noise into the sensor inputs and motor outputs, which offered realistic simulation of low-level robot control. Nevertheless, the paths taken were quite close to straight lines.

Our hybrid architecture implemented only two reactive modules, compared to seven in that of Decugis and Ferber [13]. Nevertheless, the robot still exhibits a rich set of behaviors including moving forward, turning, avoiding static and moving obstacles, and reacting immediately to change of path. These behaviors emerge from the interaction between the robot and the environment.

4. CONCLUSION

A hybrid architecture for integrated planning and control is presented in this paper. It differs from existing hybrid architectures in the following ways. First, our architecture is one of very few frameworks that perform continuous response encoding that permits fine, smooth motion control. Second, its planning module produces a sequence of checkpoints instead of a complete path. Third, it integrates the planning module with two reactive modules, i.e., target reaching and obstacle avoidance. The target reaching module is controlled by a neural network, which is trained online to move the robot through the checkpoints. The neural network can be easily trained to control different mobile robots, thus providing flexibility and adaptability that are lacking in many hard-wired reactive controllers. Although a simple local reactive strategy (i.e., Braitenberg's Type-3C vehicle) is employed in the obstacle avoidance module, it enables the robot to overcome unforeseen convex obstacles (static and dynamic). In another paper [27], we show that the obstacle avoidance capabilities can be further enhanced by the cooperation of multiple EKMs to permit the negotiation of unexpected concave obstacles.

Quantitative experimental results show that the neural network can perform fine control of the motion of a mobile robot very accurately and efficiently. In addition, qualitative test results show that the low-level reactive control modules can be seamlessly integrated with the high-level planning module. In particular, changes to the robot's heading can be easily made at every level even when the robot is en route to the goal position. Our continuing research goal is to apply the integrated framework to the planning and control of static as well as mobile robot manipulator.

5. ACKNOWLEDGMENT

This research was supported by NUS ARF R-252-000-018-112.

6. REFERENCES

[1] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, 1998.

[2] R. C. Arkin and T. Balch. AuRA: Principles and practices in review. *J. Experimental and Theoretical AI*, 2-3:175–189, 1997.

[3] R. P. Bonasso, R. J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *J. Experimental and Theoretical AI*, 9(1), 1997.

[4] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, 1984.

[5] O. Brock and L. E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *Proc. ICRA*, volume 2, pages 1469–1474, 2001.

[6] O. Brock and O. Khatib. Executing motion plans for robots with many degrees of freedom in dynamic environments. In *Proc. ICRA*, volume 1, pages 1–6, 1998.

[7] O. Brock and O. Khatib. High-speed navigation using the global dynamic window approach. In *Proc. ICRA*, pages 341–346, 1999.

[8] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. Robotics and Automation*, 2(1):14–23, 1986.

[9] R. Chatila and J.-C. Laumon. Position referencing and consistent world modeling for mobile robots. In *Proc. ICRA*, 1985.

[10] W. Choi and J.-C. Latombe. A reactive architecture for planning and executing robot motions with incomplete knowledge. In *Proc. IROS*, volume 1, pages 24–29, 1991.

[11] J. H. Connell. *Minimalistic Mobile Robotics: A Colony Architecture for an Artificial Creature*. Academic Press, 1990.

[12] J. H. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proc. ICRA*, pages 2719–2724, 1992.

[13] V. Decugis and J. Ferber. Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture. In *Proc. ICAA*, pages 354–361, 1998.

[14] E. Gat. On three-layer architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. AAAI Press, 1998.

[15] G. Giralt, R. Chatila, and M. Vaisset. An integrated navigation and motion control system for autonomous multi-sensory mobile robots. In *Proc. 1st Int. Symp. on Robotics Research*, 1983.

[16] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In *Proc. ICAA*, pages 363–370, 1997.

[17] J. Heikkonen, P. Koikkalainen, and E. Oja. From situations to actions: Motion behavior learning by self-organization. In *Proc. ICANN*, pages 262–267, 1993.

[18] H. Hu and M. Brady. A bayesian approach to real-time obstacle avoidance for a mobile robot. *Autonomous Robots*, 1:69–92, 1994.

[19] R. Jamisola, M. H. Ang Jr., D. N. Oetomo, T. M. Lim, O. Khatib, and S. Y. Lim. Operational space formulation implementation to aircraft canopy polishing application using a mobile manipulator. In *Proc. ICRA*, 2002.

[20] O. Khatib. A unified approach to motion and force control of robot manipulators: The operational space formulation. *IEEE J. Robotics and Automation*, 3(1):43–53, 1987.

[21] O. Khatib. Mobile manipulation: The robotic assistant. *Robotics and Autonomous Systems*, 26:175–183, 1999.

[22] D. E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proc. ICRA*, pages 1–6, 1987.

[23] B. H. Krogh and C. E. Thorpe. Integrated path planning and dynamic steering control for autonomous vehicles. In *Proc. ICRA*, pages 1664–1669, 1986.

[24] J. E. Laird and P. S. Rosenbloom. Integrating, execution, planning, and learning in soar for external environments. In *Proc. AAAI*, pages 1022–1029, 1990.

[25] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic, Boston, 1991.

[26] K. H. Low. Integrated planning and control of mobile robot with Extended Kohonen Maps. Master's thesis, National University of Singapore, 2002.

[27] K. H. Low, W. K. Leow, and M. H. Ang Jr. Enhancing the reactive capabilities of integrated planning and control with self-organizing neural networks. In *Proc. IROS*, 2002. (submitted).

[28] H. P. Moravec and D. W. Cho. A bayesian method for certainty grids. In *Proc. AAAI Spring Symp. on Robot Navigation*, pages 57–60, 1989.

[29] L. D. Pyeatt and A. E. Howe. Integrating POMDP and reinforcement learning for a two layer simulated robot architecture. In *Proc. ICAA*, pages 168–174, 1999.

[30] S. Quinlan and O. Khatib. Towards real-time execution of motion tasks. In R. Chatila and G. Hirzinger, editors, *Experimental Robotics 2*. Springer-Verlag, 1993.

[31] H. Ritter and K. Schulten. Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In R. Eckmiller and E. von der Marlsburg, editors, *Neural Computers*. Springer, 1987.

[32] R. Simmons, R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan. A layered architecture for office delivery robots. In *Proc. ICAA*, pages 245–252, 1997.

[33] I. Soto, M. Garijo, C. A. Iglesias, and M. Ramos. An agent architecture to fulfill real-time requirements. In *Proc. ICAA*, pages 475–482, 2000.

[34] S. Thrun, A. Buecken, W. Burgard, D. Fox, T. Froehlinghaus, D. Henning, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, pages 21–52. MIT Press, 1998.

[35] C. Versino and L. M. Gambardella. Learning the visuomotor coordination of a mobile robot by using the invertible Kohonen map. In *Proc. Int. Workshop on Neural Networks*, 1995.