Integrated Planning and Control of Mobile Robot with Self-Organizing Neural Network

Kian Hsiang Low†, Wee Kheng Leow†, and Marcelo H. Ang, Jr.‡
†Dept. of Computer Science, National University of Singapore
3 Science Drive 2, Singapore 117543, Singapore
‡Dept. of Mechanical Engineering, National University of Singapore
10 Kent Ridge Crescent, Singapore 119260, Singapore
†lowkianh, leowwk@comp.nus.edu.sg, ‡mpeangh@nus.edu.sg

Abstract

Despite the many significant advances made in robotics research, few works have focused on the tight integration of task planning and motion control. Most integration works involve the task planner providing discrete commands to the low-level controller, which performs kinematics and control computations to command the motor and joint actuators. This paper presents a framework of the integrated planning and control for mobile robot navigation. Unlike existing integrated approaches, it produces a sequence of checkpoints instead of a complete path at the planning level. At the motion control level, a neural network is trained to perform motor control that moves the robot from one checkpoint to the next. This method allows for a tight integration between high-level planning and low-level control, which permits real-time performance and easy modification of motion path while the robot is enroute to the goal position.

1 Introduction

There have been many advances in robotics over the past two decades, yet it is not pervasive in our daily lives. This is mainly due to the difficulties of achieving real-time performance and precise, smooth control while executing a complex task in our unstructured world. Nevertheless, a fair amount of robotics research has proceeded along two separate directions: high-level task planning and low-level motion control. Developments in high-level planning have largely ignored the details of low-level control. This perpetual gap impedes progress in achieving our ultimate goal.

To illustrate the point, consider the problem of executing a collision-free motion to an arbitrary goal in a complex environment, which can be solved by two contrasting approaches. Using a world model, high-level motion planning can determine a feasible sequence of collision-free motion that achieves a specified goal even in the presence of complex obstacles. However, it runs into difficulties when unforeseen or moving obstacles obstruct the generated path because real-time replanning that reacts to these situations can be too computationally expensive. On the other extreme of the motion control spectrum, low-level reactive control relies on the direct coupling of current sensor data to an appropriate motion. Thus, it is extremely robust in reacting to uncertainties and unexpected obstacles but may fail to achieve a globally specified goal. We can observe that both approaches have limitations, which are rather complementary. A union of the two can potentially mitigate their respective drawbacks and yield the best of both worlds.

The work presented in this paper is a step towards the full integration of task planning and motion control, motivated by Khatib's seminal work [11] on robot planning and control. This work and a few other integrated architectures [2, 5] have utilized methods based on potential fields [9] in their reactive control algorithms, while their planning and interface techniques differ. Potential field methods are implementations of *continuous response encoding* [1] (infinite set of responses), which makes low-level control possible. Our architecture adopts another form of continuous response encoding that can produce very low-level velocity or torque control of motor and joint actuators to perform fine, smooth motion control. This is in contrast to behavior-based architectures [6, 16, 17] and hybrid deliberative/reactive architectures [1, 3, 14] that employ *discrete response encoding* (finite, enumerated set of responses) and, thus, do not emphasize the significance of low-level control.

Our framework differs from existing architectures in two important ways: (1) The planning module produces a sequence of checkpoints instead of a conventional complete path, thus the constraint of adhering strictly to a generated path no longer exists. (2) A 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 con-

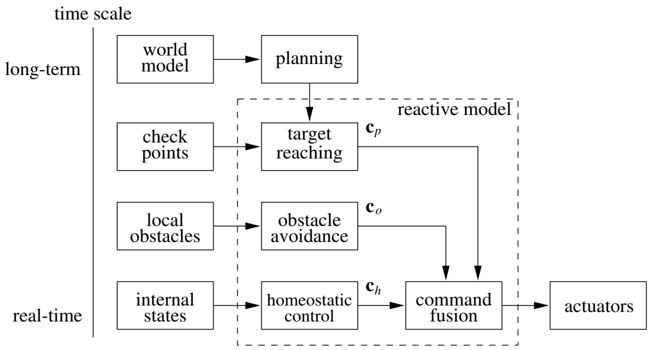

Figure 1: A framework for integrated planning and control that combines a deliberative planning model and a behavioral-based reactive model.

trol, which permits real-time performance and easy path modification while the robot is enroute to the goal position.

2 Integrated Framework

2.1 Overview

Our integrated framework consists of 4 levels (Fig. 1). At the highest level, the *planning* module produces a sequence of checkpoints from the start point to the goal using a variation of the cell decomposition method in [15]. The main difference is that our algorithm operates in the robot's workspace instead of the configuration space. This is unlike conventional planning algorithms [12, 13] which plot the detailed path.

The motion path between checkpoints is determined by the *target reaching* module. It senses the checkpoint state relative to the current state and outputs appropriate motor control signals. For mobile robot navigation, this module contains a 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 obstacles.

The lowest-level *homeostatic control* module senses the internal states of the robot to maintain internal stability by coordinated responses that automatically compensate for environmental changes [1]. This module is not strictly required for mobile robot navigation, but is crucial for mobile manipulation tasks [10] 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.

The integrated framework is applicable to both mobile robot and robot manipulator. This paper focuses on the two lower-level modules (target reaching and obstacle avoidance) and their integration with the planning module, with specific application to target reaching by a nonholonomic mobile robot. This task is performed by an *extended Kohonen map* (EKM) which is trained to produce a sequence of motor velocity commands. 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 [8, 18] 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 nonlinear problem. 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., [8, 18]), the output weights $\mathbf{M}_i$ of neuron i represents 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:

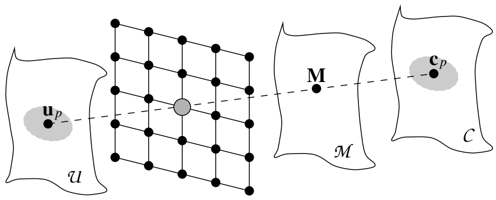

Figure 2: Target reaching module. EKM neurons map the sensory input space $\mathcal{U}$ indirectly to the motor control space $\mathcal{C}$ through the control parameter space $\mathcal{M}$.

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 [18].

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, $\mathbf{c}_p$ 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 [8, 18] 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 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

$$\Delta \mathbf{M}_i = -\eta \frac{\partial e}{\partial \mathbf{M}_i} = \eta \, G(k,i)(\mathbf{c}_p - \mathbf{M}_i \mathbf{v})\mathbf{v}^T \ . \quad (8)$$

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 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:

$$\mathbf{c}_o = \mathbf{Z}\,\mathbf{u}_o \quad (9)$$

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. 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.

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 $\mathbf{c}$:

$$\mathbf{c} = \beta\,\mathbf{c}_p + (1-\beta)\,\mathbf{c}_o \quad (10)$$

where β is a constant parameter. The homeostatic control of the motors is omitted. Equation 10 is analogous to the potential fields method for obstacle avoidance [7, 12] 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 integrated framework 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.

In the above trials, the robot's performance was assessed by measuring 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 $\mathbf{c} = \mathbf{0}$):

$$E = \frac{1}{RN} \sum_i \varepsilon_i \quad (11)$$

where R is the number of trials and N is the number of testing target locations. Experimental results (Fig. 3) show that, with indirect mapping, the self-organization of EKM 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

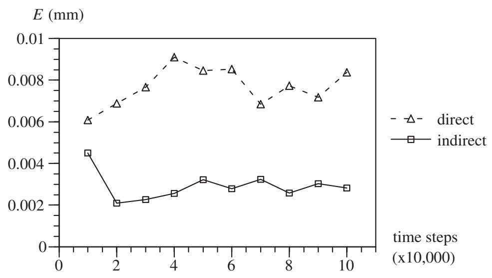

Figure 3: Mean positioning error at various training stages.

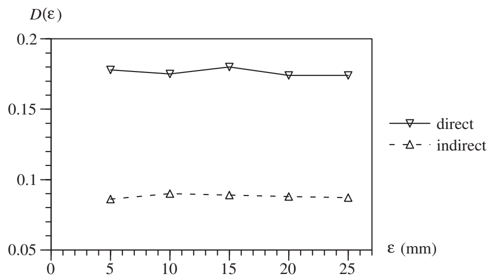

Figure 4: Mean deviation from straight line trajectory.

adopted the direct-mapping method stabilized at about the same time but the direct-mapping robot had a mean positioning error of 8 mm.

After training, the robot's performance in an obstacle-free environment was assessed by measuring the *mean deviation from straight-line trajectory* $D(\varepsilon)$. It measures how straight or wavy the robot's motion trajectory is:

$$D(\varepsilon) = \frac{1}{RN} \sum_i \tilde{\delta}_i(\varepsilon) \ , \quad \tilde{\delta}_i(\varepsilon) = \frac{|d_i(\varepsilon) - l_i|}{l_i} \qquad (12)$$

where $d_i(\varepsilon)$ is the distance traveled to reach closer than a distance of ε from target location i, l_i is the straight line distance to target i, and $\tilde{\delta}_i$ is the deviation from straight-line trajectory for target i. 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. Experimental results (Fig. 4) show that, with indirect mapping, the robot's trajectories deviated by less than 9% from perfect straight line trajectories. In contrast, with direct mapping, its trajectories deviated by about 18% from straight line. In summary, indirect-mapping EKM produces better motion control than does direct-mapping EKM.

3.2 Qualitative Evaluation

The robot's performance was also qualitatively assessed in an environment under two conditions: (1) moving obstacle and (2) unforeseen change in environment. The environment consisted of three rooms connected by two doorways

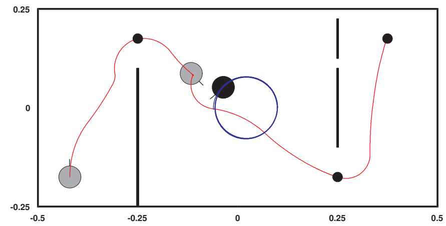

Figure 5: Motion of robot (gray) in an environment with an obstacle (black) moving in an anticlockwise circular path.

(Figs. 5–6). The robot began in the left-most room and was tasked to move to the right-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. 5), a mobile robot, following an anticlockwise circular path, served as the moving obstacle. When the 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 and headed towards the second checkpoint.

In the second test (Fig. 6), the same checkpoints as the previous tests were initially planned for the robot. However, while the robot was enroute 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 module changed the heading of the robot enroute, 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 turning the robot around while it is moving.

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.

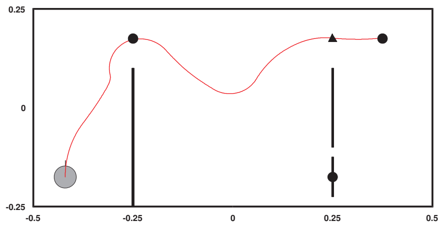

Figure 6: Motion of robot (gray) in an environment that changed. The initial checkpoints were planned with the assumption that the lower doorway was opened. The planning module changed the second checkpoint to a new checkpoint (triangle) while the robot was enroute to the old one. The target reaching module was able to react immediately and it changed the robot's heading.

4 Conclusion

A framework for integrated planning and control is presented in this paper. It differs from existing integrated frameworks in the following ways. It is one of very few integrated frameworks that perform continuous response encoding that permits fine, smooth motion control. A neural network is trained online to produce the fine, smooth control. It can be easily trained to control different mobile robots, thus providing flexibility and adaptability that are lacking in many hard-wired reactive controllers.

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 enroute 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.

Acknowledgment

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

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

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

[7] W. Choi and J.-C. Latombe. A reactive architecture for planning and executing robot motions with incomplete knowledge. In *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, volume 1, pages 24–29, 1991.

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

[9] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proc. ICRA*, pages 500–505, 1985.

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

[11] O. Khatib, S. Quinlan, and D. Williams. Robot planning and control. *Robotics and Autonomous Systems*, 21:249–261, 1997.

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

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

[14] F. R. Noreils and R. G. Chatila. Plan execution monitoring and control architecture for mobile robots. *IEEE Trans. on Robotics and Automation*, 11(2):255–266, 1995.

[15] 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.

[16] J. Rosenblatt. DAMN: A distributed architecture for mobile navigation. In *Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents*, pages 167–178, 1995.

[17] S. J. Rosenschein and L. P. Kaebling. The synthesis of machines with provable epistemic properties. In J. Halpern, editor, *Theoretical Aspects of Reasoning About Knowledge*, pages 83–98. Morgan Kaufman, Los Altos, CA, 1986.

[18] 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.