PAPER

Reasoning on the Self-Organizing Incremental Associative Memory for Online Robot Path Planning

Aram KAWEWONG[†a)], **Yutaro HONDA**[†], **Manabu TSUBOYAMA**[†], ***Nonmembers***, ***and*** **Osamu HASEGAWA**[†,††], ***Member***

SUMMARY Robot path-planning is one of the important issues in robotic navigation. This paper presents a novel robot path-planning approach based on the associative memory using Self-Organizing Incremental Neural Networks (SOINN). By the proposed method, an environment is first autonomously divided into a set of path-fragments by junctions. Each fragment is represented by a sequence of preliminarily generated common patterns (CPs). In an online manner, a robot regards the current path as the associative path-fragments, each connected by junctions. The reasoning technique is additionally proposed for decision making at each junction to speed up the exploration time. Distinct from other methods, our method does not ignore the important information about the regions between junctions (path-fragments). The resultant number of path-fragments is also less than other method. Evaluation is done via Webots physical 3D-simulated and real robot experiments, where only distance sensors are available. Results show that our method can represent the environment effectively; it enables the robot to solve the goal-oriented navigation problem in only *one* episode, which is actually less than that necessary for most of the Reinforcement Learning (RL) based methods. The running time is proved finite and scales well with the environment. The resultant number of path-fragments matches well to the environment.

key words: *neural networks, associative memory, path-planning, reinforcement learning (RL)*

1. Introduction

A robot path planning is an important and attractive topic of the robotic research field. Reaching a previously unknown destination in unfamiliar environments in the shortest time is the most common objective. Approaches based on various frameworks and having their respective advantages and disadvantages have been proposed to solve the problem.

Path planning can be done in various ways. Reinforcement Learning (RL) formulates this problem as a robot interacting with either a fully or partially observable Markov Decision Process (MDP or POMDP) [3], [4]. With the appropriated reward and value functions, the shortest path, if it exists, is guaranteed to be found. However, the reward function is not always easy to specify manually [30]. Moreover, RL fundamentally requires numerous episodes for maximum reward convergence [7]–[9], [21], [22] although the number varies depending on the complexity of environments and the functions. Probabilistic Roadmaps (PRM) randomly samples a robot's C-Space, and constructs a roadmap graph that captures the free space connectivity [27]. The key to PRM is the sampling: some methods have been proposed to improve the sampling strategy [28]. Nonetheless, sampling is not appropriate for the navigation of real robots with low degrees of freedom (DOFs) in a completely unknown map. Sensorimotor Mapping (SM) has recently been reported as successful in performing predictive motor control by coupling sensor and motor signals in a joint representational layer [29]. Actually, SM is proposed for simulation in the brain: it must perform random exploration to couple the sensor and motor, which generally costs many iteration (100,000 iterations in a previous study [29]). Rapidly Expanding Random Trees (RRT) is another popular means for robot path planning ([11] and [12]). A recent study [12] uses a viability filter to improve the random sampling so that the running time is reduced drastically. However, RRT is suited to planning for complex robots such as humanoid robots [37], [38]. To expand the node randomly into potential paths means that the robot must move numerous times.

Another approach to solve the navigation problem is to create the map and then find the shortest path. By this approach, main concern lays in finding the most effective way to map the environment while simultaneously localizing the robot's position relative to the map. If just the accurate map were acquired, then the shortest path would be easily obtainable from the occupancy grid map [16]. However, to acquire such map in an unknown environment is not easy. This call into the popular Simultaneous Localization and Mapping, or SLAM, problem [19]. The SLAM problem generally possesses a continuous and a discrete component, giving rise to two sources of problems: (1) the high dimensionality of the continuous parameter space (curse of dimensionality), and (2) numerous discrete correspondence variables (curse of history). These problems render the metric SLAM as necessitating high memory consuming and difficult for loop closing.

The Topological Map (TM) is a map-building approach which consumes considerably less memory than that of metric map and suits to solving the loop closing problem [13], [14], [17], [36]. It requires junctions, or places, to be distinctive and that they must be detected correctly whenever the robot passes close to them. However, the pure TM is unsuitable to the SLAM problem because most sensory data

Manuscript received May 15, 2009.
Manuscript revised September 15, 2009.

[†]The authors are with the Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, Yokohama-shi, 226–8503 Japan.

[††]The author is with Imaging Science and Engineering Laboratory, Tokyo Institute of Technology, Yokohama-shi, 226–8503 Japan.

a) E-mail: kawewong.a.aa@m.titech.ac.jp

DOI: 10.1587/transinf.E93.D.569

are simply ignored. This motivates researchers to propose hybrid maps in which topological nodes contain local metric information [15], [16], [18], [20]. The hybrid is currently considered as state-of-the-art for the SLAM problem [23], [24].

Nevertheless, most of the described methods pose some drawbacks. Metrical approaches consume much computational cost. The hybrid map-building capture only the information of the topological node (i.e. junction), while ignoring most of information about the edge [13], [16] (regions between junctions). Reinforcement learning requires a number of episodes for trial and errors. Also, to move from place-to-place, most of these method, except for Spatial Semantic Hierarchy (SSH) based method, needs to generate the proper action by some means, i.e. visual servoing [37]. This requires the robot to keep a number of raw images as the reference images for servoing. By considering all of these drawbacks, we want to achieve the path-planning system which can

1) keeps all information both for junction and path-fragments, while consuming memory less than pure metrical approach,
2) divide the path-fragments reasonably so that the number of fragments matches well to the environment,
3) reasonably plan the path so that the exploration time is shortened.
4) be combined with the spatial semantic hierarchy (SSH) so that it does not need to rely on a large database of reference images for visual-servoing.

As a solution, we combine the concept of Spatial Semantic Hierarchy (SSH) [31] with the associative memory to address this issue. First, we employ the Self-Organizing Incremental Neural Networks [5] as the online clustering tools for generating the "common patterns", or CP for short, for representing the path. Each obtained observation along the path is represented by the most appropriated CP. In other words, robot's "path" consists of a sequence of CPs. The sequence is further divided into sub-sequences, which we call "path-fragment", by junction (a single path comprises a sequence of path-fragments connected by junctions). Representing path with these common patterns saves much memory. This satisfies the first requirement. Dividing the path-fragments based on detected junction results in reasonable number of fragments. This solves the second problem. Reasoning function is proposed to enable the robot to reasonably determine the way to continue, so that the exploration time could be shortened. This satisfies the third requirement. By letting the basic movement be controlled by lower modules, our method becomes the SSH-based method. Since the robot can autonomously follow the path, and partition the path into fragments based on junctions, reference images are no longer needed for visual-servoing. This satisfies the last requirement.

Figure 1 show the simple example of reasoning at junction. Given A $\xrightarrow{\Delta_{AB}}$ B $\xrightarrow{\Delta_{BC}}$ C $\xrightarrow{\Delta_{CD}}$ D $\xrightarrow{\Delta_{DA}}$ A as the path taken by the robot where A, B, C and D are path-fragments

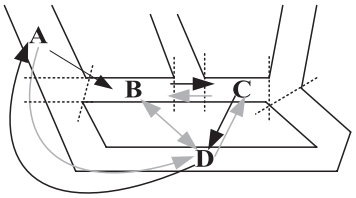


Fig. 1 The reasoning technique for generating the new transitions between the path-fragments.

(divided by dash line), the new transitions Δ_{BA}, Δ_{CB}, Δ_{DC}, Δ_{AD}, Δ_{DB}, Δ_{BD} (marked in gray color) can be generated by reasoning. This technique is simple but effective for speeding up the exploration time. The technique is specifically compatible with SSH.

2. Related Works

Recently, path planning and SLAM shares a number of common problems and objectives, and thus we need to clearly contrast our method to theirs. Comparing to reinforcement learning based methods, out method is considerably faster because the reasoning and SSH concept allow us to skip the complexity in movement generation. Our work seems to share some similar concept with the hybrid SSH of [31]; path fragments are divided by junctions, and the basic movements of robot are assumed to be controlled by other module. However, the main difference is that our method tries to keep the information about both the junctions and path-fragments (regions between junctions). In other words, our method uses the common patterns to compress the representation so that it, like metrical approach, does not lose important information between nodes, while, like topological approach, consumes less memory.

The common pattern which is used to represent the path in this work share the same concept as that of popular Bag-of-words in computer vision [32]. A dictionary must be generated either online or offline. In vision, the scene is represented by the set of words. In this work, we treat a single path-fragment as a single scene. A single path-fragment are converted into the sequence of common patterns in the same spirit as a single scene is represented by a set of unordered visual words.

Again, we confirm that this paper focus on the fast path-planning system by using associative memory and the reasoning at junction. We do not specifically address the problem of perceptual-aliasing (two different places which look very similar). What we achieve here is the success in applying the associative memory to represent the path, so that the shortest path can be retrieved successfully. Representing the path-fragments with common-patterns allow us to save memory in keeping all information both about junctions and path-fragments. The result show that the proposed method can successfully navigate the robot to the goal for every trials in all mazes. Also, the system has been successfully implemented on the real mobile robot.

3. Backgrounds

From among related works described in Sect. 2, some other works contributed to development of the proposed method.

3.1 Overview of SOINN and SOINN-AM

The Self-Organizing and Incremental Neural Networks, SOINN [5], is an two-layered unsupervised clustering method proposed for data clustering (unsupervised classification) and topology learning. The first layer is used to generate a topological structure of presented patterns. The second layer outputs the number of clusters and gives prototype vectors of the distribution of presented patterns. The learning algorithms of both layers are almost identical, but the inputs into them are different. The input to the first layer is the data presented to SOINN. After the first layer finishes learning them, the second layer obtains exactly the same vectors as the weights of the nodes which have been generated in the first layer. The distance are derived both between the input and the nearest node and between the input and the second-nearest node when each layer obtains input. A new node with equal weight to that of the input is generated if the distance are sufficiently large. Otherwise, a new edge is generated between the nearest and the second nearest node; the weights of the nearest node and its neighbors are updated.

Network growth is an important feature to adapt to non-stationary environments. Many conventional clustering algorithms such as k-means demand that a user predetermine how many clusters are to be generated. For a topology learning problem, a user of many conventional methods like Kohonen Feature Map (KFM) [10] must decide the number of nodes in advance. However, SOINN chooses the number both clusters and nodes adaptively during learning. SOINN not only generate new nodes but also eliminates unnecessary nodes. This property renders SOINN immune to noise.

Later, Sudo et al., [33] proposed the SOINN-based associative memory named SOINN-AM. This method is capable of retrieving the right associated patterns given only one pattern. SOINN-AM offers main advantage of the highly compact memory because it does not need to store a number of associations of the similar patterns. Also, the use of SOINN-AM, does not require determination of the number of directions in advance (see [33] for more detail). In this work, we modify the SOINN-AM to be suitable to robotic. Unlike the original SOINN-AM, our method does not limit to associative pairs. Specifically, our method create the association $A \xrightarrow{\Delta_{AB}} B \xrightarrow{\Delta_{BC}} C \xrightarrow{\Delta_{CD}} D \cdots$ instead of $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$, ... This association can be regarded as the path. This association can be duplicated according to the walks of the robot, and so that the optimization process is needed.

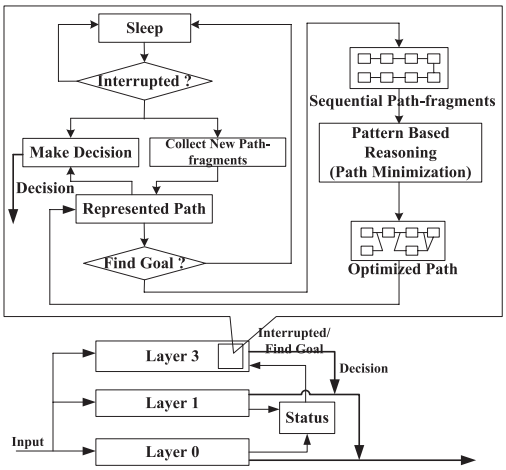


Fig. 2 Proposed method and its relation to SSH&SA.

3.2 Overview SA and SSH

The idea of Subsumption Architecture (SA) [6] has been well known in the robotic research field. According to the SA concept, a problem is divided into multiple layers. Each layer does its own task independently in parallel. The signals from the upper layers can suppress those from lower layers.

Later, Kuipers [38] propose the spatial semantic hierarchy (SSH) for using in mobile robot. The SSH is developed upon the former SA. It becomes more practical for real mobile robot. Based on the concept of SSH, we assume that the robot is controlled by the basic layers (or the control laws in [38]): layers 0 and layer 1. Layer 0 prevents the robot from colliding with obstacles and moves the robot away from oncoming objects. Layer 1 forces the robot to wander aimlessly in the *forward* direction. The proposed method resides in layer 3: the task of path planning.

Figure 2 shows where to put our method in the Subsumption Architecture of a mobile robot. The proposed method would be located in layer 3 module. Because it is at layer 3, the method functions based on layers 0 and 1; namely, the robot is at least capable of following the path without collision. Layer 3 sleeps until it either is interrupted by layers 0 and 1, or obtains the bizarre input. Once activated, it makes a decision for the robot, and collects the path-fragment representation. It repeats this until the goal is found, then it finally optimizes the map. This is called offline Common Pattern-Based Map, or CPM for short. The online CPM is described in Sect. 6.

4. Proposed Method

We believe that almost all environments share some common characteristics. Using the same concept, if the robot were able to recognize such common characteristics, it would be able to represent an unfamiliar environment as a combination of such characteristics, which results in

compact memory for representation. However, the robot must realize these characteristics in an unsupervised manner, which can be done using an unsupervised clustering tool. For our choice, we select the Self-Organizing and Incremental Neural Networks (SOINN) [5] because it is suitable to noisy environment and is easily implemented. These common characteristics are called *Common Patterns* or *CPs*. For later reference, we designate the set of these CPs as C. Note that our CP is based on the same concept as the word in Bag-of-words (BoW) approach. Because BoW is generally used with visual data, we use the term "CP" instead of "word".

Based on the generated CPs, the robot can represent the path with a sequence of CPs. More precisely, at every time step, given a sequence of observation obtained up to time t, each observation is classified to the corresponded CP. Once the "critical" CPs are detected (i.e. junction in this paper), layer 3 activates and wraps up all obtained sequential CPs and divide the path into fragments. That is to say, the junction detection signal from the lower layers is used to divide the path fragment. Then, layer 3 suppresses the signal of lower layers and directly instructs the robot to perform some specific action (i.e. turn left or right until spotting space); then it lets the lower layers regain the control of the robot. This process can be stopped under either one of two conditions: (1) the goal has been reached or found (so that the map is sufficient for navigation), or (2) all path-fragments in the map have been explored (Goal must also be found). A map generated based on CPs is called a *Common-Patterns Based Map* or *CPM*.

The number of CPs for representing any path fragments could vary depending on the size of the fragment and the behavior of lower layers. Namely, if the path fragment was short while the robot makes observation too frequently, then the number of required CPs would be high. On the contrary, if the robot made the observation only a few times while the fragment is very large, then the number of CPs would be too small and contains too less information. As such, the behavior of layers (i.e. observing rate) should match to the size of the robot and the environment, and this, for the time being, must be set by users. If all sensors of the robot are sufficiently accurate and informative, then only a few observations are enough. Otherwise, the robot should frequently make observations to gain enough information.

Figure 3 shows an outline of the proposed approach. The approach is consisted of four layers of network. The first layer obtains the raw inputs and clusters them to form the CPs. The generated CPs are put into the second layer. During the path planning, the robot get the observation by the first layer and pass it to the second layer to obtain the nearest CPs, and generate a new CP, if necessary. The third layer is the space of path-fragment. Each fragment comprises a sequence of CPs. The fourth layer is introduced for the case in which the robot must experience more than one environment. A single environment is represented by the associated path-fragments.

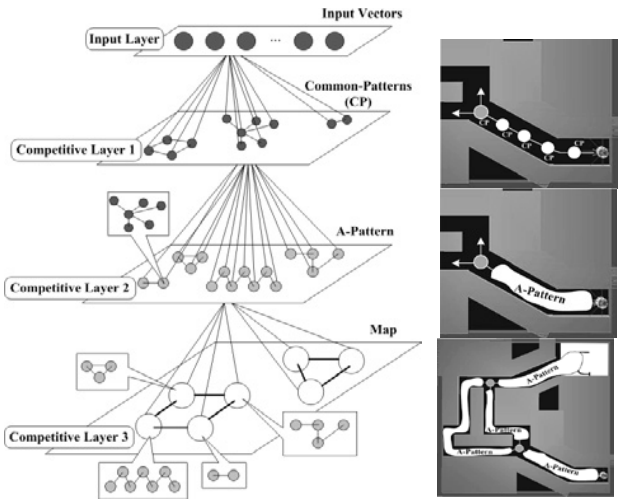


Fig. 3 (Left) The illustration of our proposing system. (Right) The illustration of how the proposed architecture can be applied to the navigation in maze.

4.1 Generating CPs

Because the testing environment in this work is not large-scale, we generate the dictionary of CPs in an unsupervised *offline* manner. An additional environment (different from those used for testing in Sect. 5) is provided for the robot to create the CP-dictionary. In this environment, the robot incrementally observes feature vectors corresponding to every time step t. In each observation, the robot sweeps its *eight* distance sensors 360° (see www.e-puck.org) with angular velocity of θ rad/s ($\theta = 2\pi$ rad/s in this work, thereby 1 s is sacrificed for making observations). After the observation, the robot moves forward for an additional 0.5 s. Therefore, *ca*. 1.5 s is required for a single step. It is important to note that this observation method is not fixed. Any other observation technique would also be fine as long as it yields the input vectors. For each sensory input (which come from 360° sweeping), the robot samples the sensor value (0–3800) 100 times (i.e. for 1 s of one sweep, the robot takes input every 1/100 s). Therefore, the robot gains eight 100-dimensional vectors (from eight sensors) in each observation. These vectors are reduced to one single 100-dimensional vector $\vec{v}$ by averaging. Then, $\vec{v}$ would be segmented based on the dropping rate of its elements' values. Given the following:

$\vec{v}$: The n_v-dimensional input vector $(n_v = 100)$
δ: The estimated error set by the user
$S_{max} = \max_{k=1:n_v}(|v^{(k)} - v^{(k+1)}|)$
$S_{min} = \min_{k=1:n_v}(|v^{(k)} - v^{(k+1)}|)$

$$f_p = \frac{\sum_{j=1}^{\delta} |v^{(i)} - v^{(i+j)}|}{\delta}, \quad \text{where} \in \mathrm{I}^+,\ i \le n_v - \delta,$$

Then i is the segmentation point if $f_p > \frac{S_{max}+S_{min}}{2}$. This process is repeated until all possible segmentation points are obtained. The vector is then segmented corresponding to

the number of segmentation points. Each segment's size is pruned down to one by averaging all values; thereby, only one real value represents one segment. We arbitrarily assume that the number of segments for the real-world navigation problem would be less than 10 (five-way junction at most). The vector's size might be more than ten if the junction size is larger than five ways. For that reason, a 10-dimensional vector is generated by concatenating all segments' averaging values. When the number of segmentation is less than the designated size of the vector, i.e. 10, all undefined elements are set to zero. The additional dimension of vector is added for landmark detection value. This results in 11-d vector. In this work, unless the robot reaches the start or goal, this value would be set to zero (*no landmarks*). These 11-dimensional vectors are input continuously to SOINN in an *on-line* manner. For SOINN, λ and age_{dead} are both set to 20 according to a previous report [5]. After a period of time, the robot stops wandering and a set of clusters is derived. A representative vector of each cluster is a common-pattern CP.

By this feature extraction, the junction can be simply detected by considering those CPs that have spaces more than two. The dead-end can be detected by considering those CPs that have only one space. In other words, a single path-fragment is represented by a sequence of CPs, while the junction is represented by only single CP. The branches of the junction is numbered counter-clockwise starting from the robot's current direction.

4.2 Mapping the Environment

As the robot is capable of segmenting the path into path-fragments, the robot gradually and incrementally represents a new path fragment until either all fragments have been explored or the goal has been found. Figure 4 illustrates the simple path-fragment representation. Given that the robot start from the 3-junctions and walk in the direction indicated by the arrow, the robot constantly make and observation and represent each observation by the corresponding CP. The robot repeats this until the junction CP is spotted (i.e. 4-junctions represented by CP-5 in this example). Thus, the first path-fragment is represented by the CP sequence 2-3-4-3, and the second fragment is represented by 3-3-3-6-3-7-4-7-3-3-3-3-3-3. Note that, in addition to CP-junctions, other CP (some extra landmarks) might also be used to divide fragment as well.

Let C be the set of CPs; $C = \{c_1, c_2, \ldots, c_{n_c}\}$, n_c is the number of CPs in the dictionary. In the first episode, the robot wanders until it finds the goal. While searching, the part fragment is represented using a sequence of CPs called *A-Pattern*.

Definition 1: Given a sequence of CPs, $\mathcal{S} = \{s_1, s_2, \ldots, s_{n_s}\}$, where $s_i \in C$, $i \in \mathrm{I}^+$, $i \leq n_s$, and $\mathcal{J}$ as a set of junction CPs, where $\mathcal{J} \subseteq C$, $\mathcal{S}$ is the *A-Pattern* denoted by $\mathbb{A} = \mathcal{S}$ if and only if

1) $s_0,\ s_{n_s+1} \in \mathcal{J}$, and

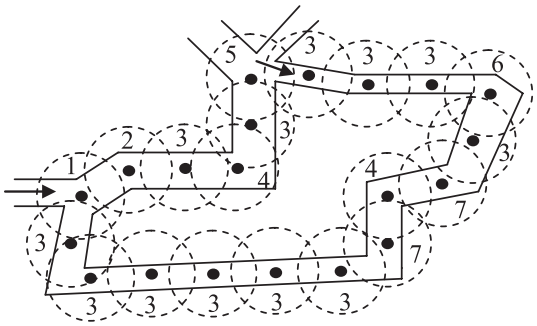


Fig. 4 Path-fragment representation. The number indicate the index of CP in the dictionary. Black points are the observation points done by robot. The dashed circle is the typical radius of sensor sweeps. The robot walks in the direction indicated by the arrows.

2) For every i, $\{s_i\} \cap \mathcal{J} = \emptyset$.

Once the goal is found, the robot derives the *Common-Patterns Based Map*, abbreviated as CPM and denoted by $\mathcal{M}$; $\mathcal{M} = \{\mathbb{A}_1, \ldots, \mathbb{A}_{n_p}\}$, where n_p is the number of all distinctive path-fragments.

Using no prior information about position, the robot has no other way but to search randomly for the unknown destination while mapping the environment. The observation is made every time step. From the start point, the robot follows the path under the control of layer 0 and 1. The observation is made constantly and represented by the corresponding CP. Once the CP-junction is detected, layer 3 activates, divides the path-fragment, and stores the fragments into the map. The new fragment and the previous fragment are connected by junction together with the performed transition. The robot repeats this until the goal has been found. The map would be considered as completed, if the robot has traversed every path-fragment before the goal is found (every way of every junctions have been explored). However, the map might be incomplete if the robot found the goal before disclosing all path-fragment. It is noteworthy that only the complete map can guarantee the shortest path. However, in some cases where the environment is very large, exploring all path-fragments may consume more time than simply follow the path optimized from the uncompleted map.

The robot searches for the unknown destination and gradually represents the path-fragment. Once the goal is found, the CP-Map (CPM) would be obtained. The robot optimizes this CPM based on reasons and then plan the shortest path based on the optimized CPM.

4.3 Planning the Path

Now that the robot has found the goal and obtained the CPM, it must optimize the CPM. The optimization is done after the goal has been found. Therefore, this is called an *offline* CPM. The online CPM is described in Sect. 6. Before continuing this explanation in greater detail, some assumptions must be clarified. According to the scope of this study, the robot has only eight distance sensors (no vision), the environment must satisfy the following constraints.

1) Every path-fragment in the environment is *distinctive*: we assume that, in the real world, even very similar places always include some differences which distinguish them from each other. This constraint might seem to be too strict as some may argue that the real-world always contain many place which look similar, and the perceptual-aliasing problem is very common. However, this paper proposes the architecture for applying SOINN-AM for path-planning and the reasoning at the junction which improve the path-planning behavior. We do not focus on the SLAM problem. Any path-fragment can be made distinctive in a number of ways. Combining proximity sensors and cameras enable to robot to efficiently capture the unique characteristics of the place which looks similar, so that they become distinguishable. Another good choice is to add the odometry data, so that the robot can also localize itself. This can further enable the robot to find get the distinctive path-fragment.
2) Every path-fragment in the environment is *asymmetrical*. It is assumed that the entrance and the exit of any path-fragment would differ somehow to be precise. Particularly, the plus/minus (+/−) signs can be assigned to $\mathbb{A}$ to distinguish its entrance and exit. This constraint can also be relaxed in the same way as the previous constraint.

Given $\mathcal{M} = \{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n_p}\}$ as the CPM, the minus sign (−) denotes the entrance direction of the path-fragment $\mathbb{A}_i$, where $i \in I^+$, $i \leq n_p$, whereas a plus sign (+) denotes the exit direction. For any path-fragment, the signs are assigned only at the *first* time of traversal. Given *junc*() as the junction; $junc(\mathbb{A}_i^-)$ is the junction before $\mathbb{A}_i$, and $junc(\mathbb{A}_i^+)$ is the junction after $\mathbb{A}_i$. According to Def. 1, each $\mathbb{A}_i$ is connected by the junction. The transitions (decision made by layer 3) which bring the robot from $\mathbb{A}_i$ to $\mathbb{A}_{i+1}$ have been realized by the robot. Namely, from $\mathbb{A}_i$ to $\mathbb{A}_{i+1}$, there exists a transition Δ_i, where $i \in I^+$, $i < n_p$. A transition Δ can be either a real value or vector. However, in this study, we simply define the transition as the integer value indicating the index of choice made by Layer 3 at the junction. The algorithm is divided into two steps.

First, 1) based on the derived $\mathcal{M}$, in which its element $\mathbb{A}_i$ is connected by the initial transition Δ_i, the robot finds the inversed transition Δ_i^{-1} using the function Λ_1. Given N_w as the number of the spaces at the junction, $junc(\mathbb{A}_i^+)$, where

$$junc(\mathbb{A}_i^+) = junc(\mathbb{A}_{i+1}^-), \tag{1}$$

$$\Lambda_1(\Delta_i) = N_w - \Delta_i + 1 = \Delta_i^{-1}, \tag{2}$$

Second, (2) now that the robot knows all bidirectional transitions for $\mathbb{A}$, it starts to search for the duplicated $\mathbb{A}$ in $\mathcal{M}$. The real world usually contains multiple loops. Therefore, it is likely that the robot passes the same junction more than once. Those similar junctions must be *unified* to reconnect $\mathbb{A}$ and eliminate all duplicated $\mathbb{A}$.

Definition 2: Given $\mathcal{M} = \{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n_p}\}$, $\mathcal{M}$ is optimized if and only if, for every $\mathbb{A}_i$ in $\mathcal{M}$, $\mathbb{A}_i$ is distinctive, where $i \in I^+$, $i \leq n_p$. The optimized $\mathcal{M}$ is denoted as $\mathcal{M}^*$.

Definition 3: Given $\mathcal{M}^* = \{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n_p}\}$, and a set of corresponding junctions $\mathcal{T} = \{\mathbb{T}_1, \mathbb{T}_2, \ldots, \mathbb{T}_{n_p-1}\}$, where $\mathbb{T}_i \in \mathcal{J}$, $i \in I^+$, $i < n_p$, $\mathcal{M}^*$ is **complete** if and only if

1) every port (gateway) of $\mathbb{T}_i$, where $i \in I^+$, $i < n_p$, has been filled, and
2) $\mathcal{M}$ contains an A-Pattern which represents the goal path-fragment.

To recognize similar junctions, it must consider from connected $\mathbb{A}$. Specifically, given that $\mathbb{A}_1$ is approximately identical to $\mathbb{A}_2$, then $junc(\mathbb{A}_1^+) = junc(\mathbb{A}_2^+)$ and $junc(\mathbb{A}_1^-) = junc(\mathbb{A}_2^-)$. The similarity between $\mathbb{A}_1$ and $\mathbb{A}_2$ is determined by similarity function f_S given the following.

$\ell(\mathbb{A}_1)$: length of $\mathbb{A} = \{a_1, a_2, \ldots, a_{\ell(\mathbb{A}_1)}\}$, $a \in C$

$\ell(\mathbb{A}_2)$: length of $\mathbb{A} = \{a'_1, a'_2, \ldots, a'_{\ell(\mathbb{A}_2)}\}$, $a' \in C$

d_{max}: maximun distance among all CPs

$$\varepsilon_b = \frac{\sum_i^{min(\ell(\mathbb{A}_1),\ell(\mathbb{A}_2))} E}{min(\ell(\mathbb{A}_1), \ell(\mathbb{A}_2))} \tag{3}$$

$$E = \sqrt{\sum_{j=1}^{min(\ell(\mathbb{A}_1),\ell(\mathbb{A}_2))} \left(a_j - a'_j\right)^2} \tag{4}$$

$$f_S = 1 - \frac{\varepsilon_b}{d_{max}}, \quad \delta \in [0, 1] \tag{5}$$

Then $\mathbb{A}_1 \cong \mathbb{A}_2$ if $f_S > \mu$. Parameter μ is determined by the user. In this work, our Webots simulated robot contains 10% slip noise at its wheels, we simply set $\mu = 0.9$. The inverse of $\mathbb{A}_2$, namely $\mathbb{A}_2^{-1}$, must also be compared with $\mathbb{A}_1$. Figure 5 (a) illustrates this. Two path-fragments, $\mathbb{A}_i$ and $\mathbb{A}_j$, are detected as similar. If $\mathbb{A}_i \cong \mathbb{A}_j$, then the connecting is performed in the left branch. If $\mathbb{A}_i \cong \mathbb{A}_j^{-1}$, then the connecting is performed in the right branch. Given the following:

$\mathcal{M}$: A CPM (sequence of all gained A-Pattern)

$\mathbb{A}_1$: The A-Pattern being considered, $\mathbb{A}_1 \in \mathcal{M}$

$\mathbb{A}_2$: The similar A-Pattern comparing with $\mathbb{A}_1$, $\mathbb{A}_2 \in \mathcal{M}$

$n_{junc(\mathbb{A}_1^-)}$: a number of spaces of $junc(\mathbb{A}_1^-)$

$\mathcal{U}^{\{-,+\}}$: a set of A-Pattern connected to $junc\left(\mathbb{A}_1^{\{-,+\}}\right)$

$$\mathcal{U}^- = \left\{\mathbb{U}_1^-, \mathbb{U}_2^-, \ldots, \mathbb{U}_{n(\mathcal{U}^-)}^-\right\},$$

$$\mathcal{U}^+ = \left\{\mathbb{U}_1^+, \mathbb{U}_2^+, \ldots, \mathbb{U}_{n(\mathcal{U}^+)}^+\right\}, \text{ where } \mathbb{U} \in \mathcal{M}.$$

$n(\mathcal{U}^{\{-,+\}})$: a number of member in $\mathcal{U}^{\{-,+\}}$

$$\boldsymbol{n}\left(\boldsymbol{\mathcal{U}}^{\{-,+\}}\right) \in \boldsymbol{I}^+, \ \boldsymbol{n}\left(\boldsymbol{\mathcal{U}}^{\{-,+\}}\right) < \boldsymbol{n}_{\boldsymbol{junc}\left(\mathbb{A}_1^{\{-,+\}}\right)}$$

$\mathbb{R}^{\{-,+\}}$: a A-Pattern connected to $junc\left(\mathbb{A}_2^{\{-,+\}}\right)$

$\Delta_{\mathbb{X}\to\mathbb{Y}}$: a transition bringing $\mathbb{X}$ to $\mathbb{Y}$, where $\mathbb{X}$, $\mathbb{Y} \in \mathcal{M}$

Λ_2: a transition function between A-Pattterns

$$\Lambda_2(\mathbb{X}, \mathbb{Y}) = \Delta_{\mathbb{X}\to\mathbb{Y}} \tag{6}$$

Considering $\mathbb{A}_1$, if there already existed k_1 path-fragments connected to $junc(\mathbb{A}_1^-)$, and k_2 path-fragments connected to $junc(\mathbb{A}_1^+)$ at the time $\mathbb{A}_1$ is being considered,

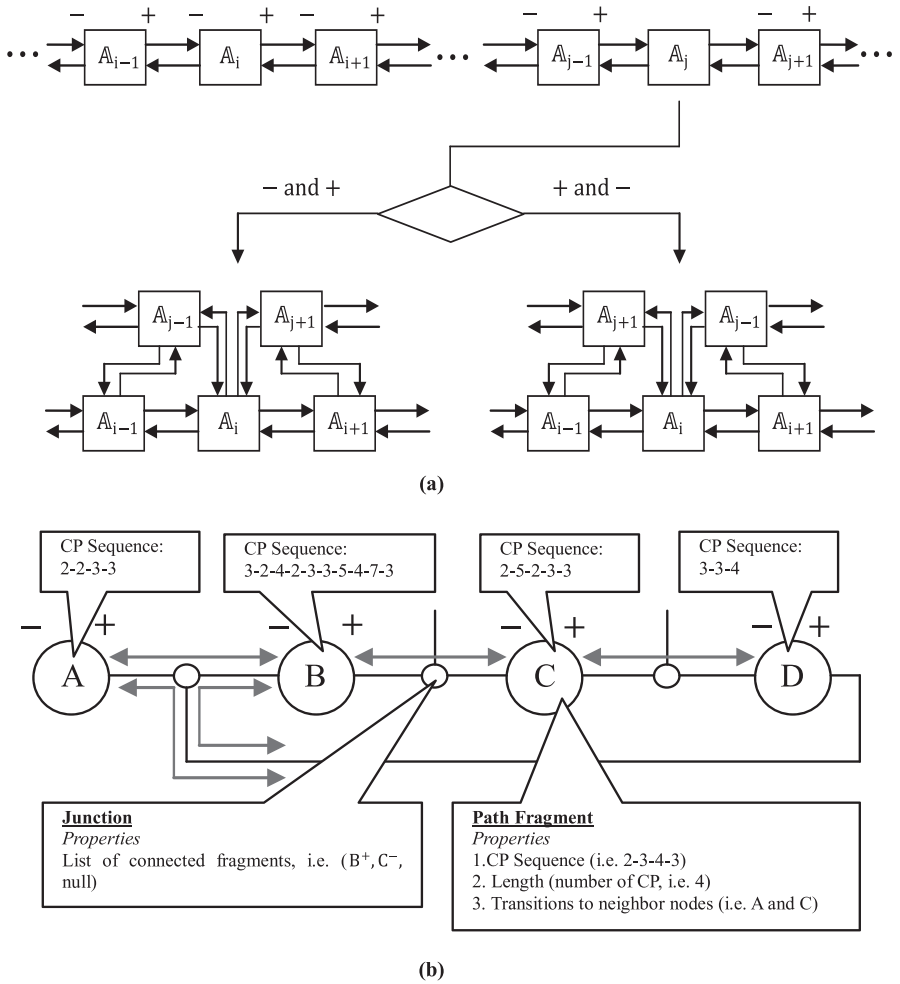


Fig. 5 (a) The A-Pattern $\mathbb{A}_j$ is considered to be similar to $\mathbb{A}_i$ either in inversed or non-inversed. The left branch is for non-invert similarity and the right branch is for the invert similarity. (b) The typical path representation from Fig. 1.

then the transitions $\Delta_{\mathbb{U}_i^- \to \mathbb{A}_1}$, for every $i \in I^+$, $i \leq k_1$, and transitions $\Delta_{\mathbb{A}_1 \to \mathbb{U}_j^+}$, for every $j \in I^+$, $j \leq k_2$, have already been realized. Also, according to inversed transition function of (2), the invert transitions are also derived as follows:-

$$\Lambda_1\left(\Delta_{\mathbb{U}_i^- \to \mathbb{A}_1}\right) = \Delta^{-1}_{\mathbb{U}_i^- \to \mathbb{A}_1} = \Delta_{\mathbb{A}_1 \to \mathbb{U}_i^-} \tag{7}$$

$$\Lambda_1\left(\Delta_{\mathbb{A}_1 \to \mathbb{U}_j^+}\right) = \Delta^{-1}_{\mathbb{A}_1 \to \mathbb{U}_j^+} = \Delta_{\mathbb{U}_j^+ \to \mathbb{A}_1} \tag{8}$$

In case of $\mathbb{A}_1 \cong \mathbb{A}_2$, there existed two sub-cases.

1) First, if $\mathbb{U}_i^- \ncong \mathbb{R}^-$ for every i, then $\mathbb{R}^-$ would be added to the junction: $junc(\mathbb{A}_1^-)$.
2) Second, if $\mathbb{U}_j^+ \ncong \mathbb{R}^+$ for every j, then $\mathbb{R}^+$ would be added to the junction: $junc(\mathbb{A}_1^+)$.

Regarding to $\mathbb{A}_2$, there exists two forward transitions; $\Delta_{\mathbb{R}^- \to \mathbb{A}_2}$ and $\Delta_{\mathbb{A}_2 \to \mathbb{R}^+}$, and two inversed transitions derived by the inversed function (2); $\Delta_{\mathbb{A}_2 \to \mathbb{R}^-}$ and $\Delta_{\mathbb{R}^+ \to \mathbb{A}_2}$.

For the first sub-case, the following pertains.

$$\begin{aligned}\Delta_{\mathbb{U}_i^- \to \mathbb{R}^-} &= \Lambda_2(\mathbb{U}_i^-, \mathbb{R}^-).\\ &= (\Delta_{\mathbb{U}_i^- \to \mathbb{A}_1} + \Delta_{\mathbb{A}_2 \to \mathbb{R}^-})\ mod(n_{junc(\mathbb{A}_1^-)})\end{aligned} \tag{9}$$

$$\begin{aligned}\Delta_{\mathbb{R}^- \to \mathbb{U}_i^-} &= \Delta^{-1}_{\mathbb{U}_i^- \to \mathbb{R}^-} = \Lambda_1(\Delta_{\mathbb{U}_i^- \to \mathbb{R}^-})\\ \Delta_{\mathbb{R}^- \to \mathbb{A}_1} &= \Delta_{\mathbb{R}^- \to \mathbb{A}_2}\\ \Delta_{\mathbb{A}_1 \to \mathbb{R}^-} &= \Delta_{\mathbb{A}_2 \to \mathbb{R}^-}\end{aligned}$$

Because $\mathbb{R}^-$ is connected to the $junc(\mathbb{A}_1^-)$, we assign $\mathbb{R}^- = \mathbb{U}_{k_1+1}^-$ and add it to the set $\mathcal{U}^-$; $\mathcal{U}^- \cup \left\{\mathbb{U}_{k_1+1}^-\right\}$.

The second sub-case includes the following.

$$\begin{aligned}\Delta_{\mathbb{U}_j^+ \to \mathbb{R}^+} &= \Lambda_2(\mathbb{U}_j^+, \mathbb{R}^+)\\ &= (\Delta_{\mathbb{U}_j^+ \to \mathbb{A}_1} + \Delta_{\mathbb{A}_2 \to \mathbb{R}^+})\ mod(n_{junc(\mathbb{A}_1^+)})\end{aligned} \tag{10}$$

$$\begin{aligned}\Delta_{\mathbb{R}^+ \to \mathbb{U}_j^+} &= \Delta^{-1}_{\mathbb{U}_j^+ \to \mathbb{R}^+} = \Lambda_1(\Delta_{\mathbb{U}_j^+ \to \mathbb{R}^+})\\ \Delta_{\mathbb{R}^+ \to \mathbb{A}_1} &= \Delta_{\mathbb{R}^+ \to \mathbb{A}_2}\\ \Delta_{\mathbb{A}_1 \to \mathbb{R}^+} &= \Delta_{\mathbb{A}_2 \to \mathbb{R}^+}\end{aligned}$$

As in the first sub-case, $\mathbb{R}^+$ is connected to the $junc(\mathbb{A}_1^+)$ and $\mathbb{R}^+$ is assigned to $\mathbb{U}_{k_1+1}^-$. Then, we make a union: $\mathcal{U}^+ \cup \left\{\mathbb{U}_{k_2+1}^+\right\}$.

For $\mathbb{A}_1 \cong \mathbb{A}_2^{-1}$ (invert similarity), the connection is done in mostly the same way as that of the first case: $junc(\mathbb{A}_1^+) =$

$junc(\mathbb{A}_2^-)$ and $junc(\mathbb{A}_1^-) = junc(\mathbb{A}_2^+)$.

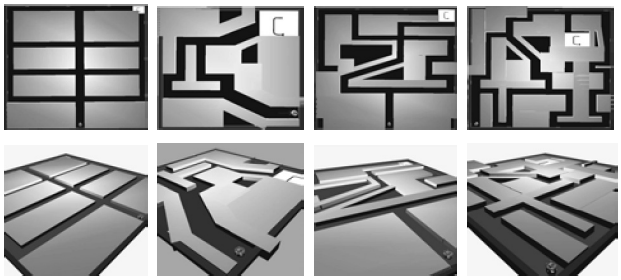
The robot repeats checking the similar A-Pattern and fills the new path-fragments to the junction. The robot then considers the next pattern $\mathbb{A}_{i+1}$ and fills the junction $junc(\mathbb{A}_{i+1}^+)$ and $junc(\mathbb{A}_{i+1}^-)$. This process is repeated until no similar pattern remains.

After the optimization, the path would be represented in form of associative path-fragments. Figure 5 (b) show the typical sample of optimized CPM (path) representation based on the given walks in Fig. 1. Each path-fragment node contains three properties. The CP sequence is used for path-fragment matching for loop-closure detection. The length of the fragments is used to estimate how many observation the robot can skip, so that it can travel directly from junction-to-junction. The transitions stored in the node let the robot know which way it need to select in order to reach the expected fragment. The junction node keeps only the fragments connecting to it, including the direction (−, +). Based on this representation, the robot can plan the path in the similar way as human. For example, given C as the starting point, and A as the goal. Two path exist: C → B → A and C → D → A. The first path take about 10 CPs to pass B, plus some more steps in A, while the second path take only 3 CPs to pass D. Thus, the shortest path is the latter one. Once the shortest path has been retrieved, the corresponding transitions $\Delta_{C^+\to D^-}$ and $\Delta_{D^+\to A^+}$ are retrieved. These information can be finally interpreted as start from C, follow the path until reaching the junction, perform $\Delta_{C^+\to D^-}$, follow the path by skipping the observation for 3 CPs until reaching another junction, perform $\Delta_{D^+\to A^+}$, and make observation again to confirm the correct destination.

5. Experiment1: Simulation

The simulated experiment is described in this section. This experiment is conducted to prove that our system can successfully navigate the robot from the start to the goal with markedly short time. The results would show the improvement of path done by our method. The 3-D physical simulator used in this experiment is Webots [1], a realistic physical 3-D simulator that enables a straightforward transfer to real robots [2]. Figure 7 (a) shows the simulated e-puck. Instead of the real value of distance sensors used by the real e-puck, we increased the sensor range slightly from *ca*. 12.8 cm to *ca*. 25 cm. For CP-dictionary generation, a robot is given a simple simulated learning maze shown in Fig. 6 (a). According to the setup, the robot can learn the CPs for *ca*. 1500 s; thereby 1000 steps are taken and 1000 input vectors are processed by SOINN. Regarding the results, 31 clusters are obtained with 102 nodes (neurons) in all. This requires only a small amount of memory.

The testing is done 10 times corresponding to each maze (Maze-1, Maze-2, and Maze-3 as shown in Fig. 6 (b-c)). The result of a typical CPM generated from Maze-3 of 6th Trial is depicted in Fig. 8 (a). The optimized CPM is also shown. In Fig. 8 (b) and (c) show the path optimization based on reasoning. The path-fragment which has been passed in only one direction (i.e., A3) could be reasoned to generate the transition of another direction. Results of navigation problem solving are shown by the graphs in Fig. 9, where panels (a–c) show results in the aspect of time, and panel (d) shows the quality of improvement.



(a) Maze-L **(b)** Maze-1 **(c)** Maze-2 **(d)** Maze-3

Fig. 6 Mazes simulated by Webots. Illustrated in the second row is the side-view of the simulated mazes. (a) A maze for CRPs generation in the Learning Process. (b) The first maze for living with $1 \times 1\,m^2$. (c) The second testing maze containing both a T-junction and Cross-junction. The size is $3 \times 3\,m^2$. (c) The last testing complex maze with size $3 \times 3\,m^2$.

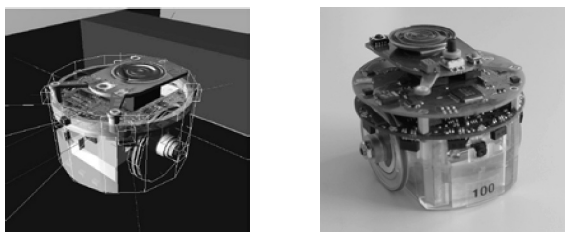


Fig. 7 Simulated e-puck and the real e-puck.

In the aspect of robotic mapping, the loops can be closed properly with neither false negative nor positive. As depicted in Fig. 8, the path that is taken is created by arbitrary exploration: it therefore contains some loops (duplicated path-fragments). Consequently, the map is optimized. Finally, the optimized CPM, $\mathcal{M}^*$, is derived (lower left of Fig. 8 (a)). In this typical case, based on Def. 3, $\mathcal{M}^*$ is complete. For every trial of each Maze, map optimization can be done. Considering the right picture of Fig. 8 (a), for instance, path-fragments A_3 and A_5 can be detected as duplicated (loop-closing) and the junctions are unions. However, it is noteworthy that some trials exist for which CPM is not complete. For example, in the 4th trial of Maze-3, the robot fortunately selected the shortest route to find the goal ($A_s \to A_5 \to A_3 \to A_G$ of Fig. 8 (a)); in this case, the obtained CPM contains A_s, A_3, A_5 and A_G.

Regarding navigation problems, reasoning on CPM offers many advantages. The improvement shown in panels (d) of Fig. 9 is considerable. Considering time, the reduction rate is more than 50%, perhaps more than 90%. For instance, in some trials, the robot unfortunately spent a long time searching for the goal in the first episode (Fig. 9 (c)-6th trial of M3-EP1). In such cases, the robot can reduce the time based on this "unlucky path" by 95% (Fig. 9 (d)-6th trial of M3-T). Based on the fact that the robot has no

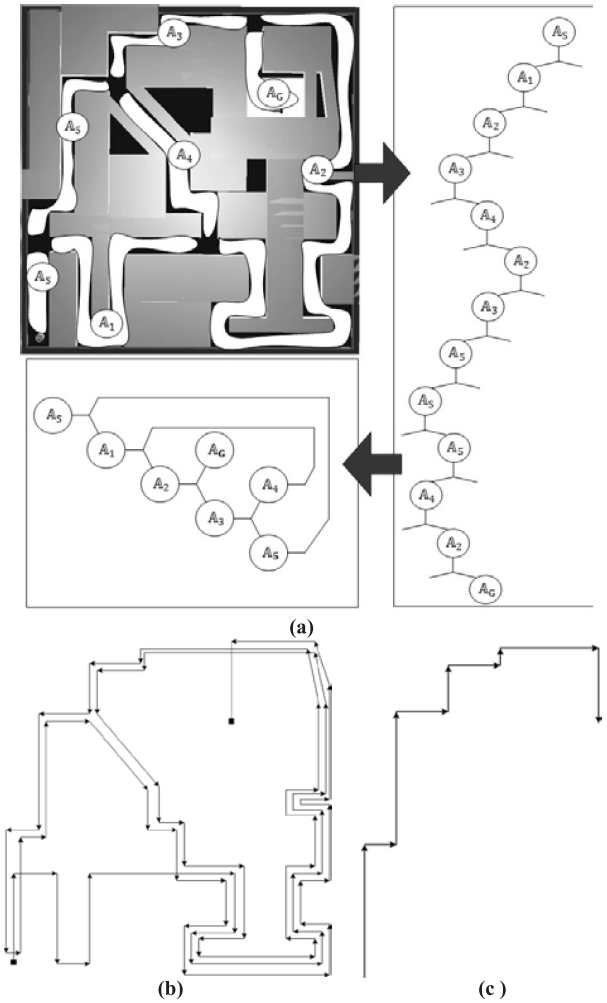


Fig. 8 The illustration of result of 6th trial in Maze-3. (a) The CPM is generated and then optimized. (b) The route in the first episode. (c) The route in the second episode optimized by reasoning.

guessing ability, although the total time for solving the problem is greater than 600 s (Fig. 9 (c)-6th Trial of Maze3), the reduction rate is also as high as 95%. Consequently, if the robot *fortunately* found the goal quickly, the reduction rate of time is greater than 50%, but if the robot *unfortunately* took a long time to find the goal in the first episode, the reduction is consequently increased to 90%. This result underscores the phenomenon of path improvement: the performance benefits from serendipity.

The comparison between our method and the random walk may seem to be non-surprising. However, this result shows that the shortest path can be successfully retrieved for every trials. The approximated length of CPs for skipping can significantly speed-up the walks in the second episode. Regardless of how the robot walk in the first episode, the path can be optimized, resulting in reliable shortened path in the second episode. The retrieved path in also in the form which is very suitable to the SSH-based methods.

Comparing to reinforcement learning-based method, out method is clearly faster as it always require only

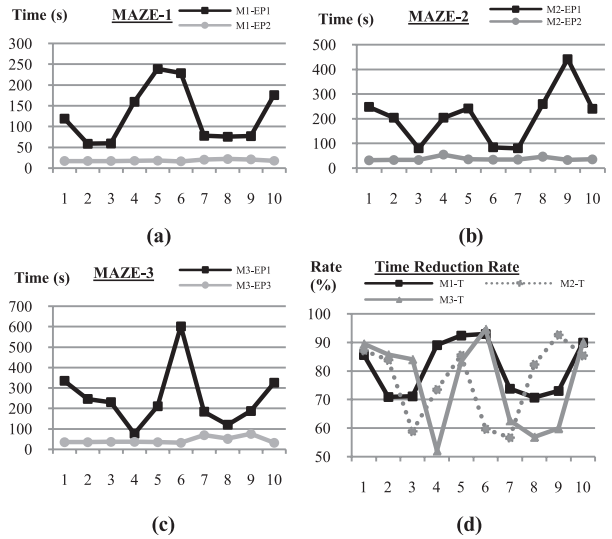


Fig. 9 Results of mazes with simulated Webots. The *x*-Axis for all graphs shows the trial index. (a–c) The result is shown in the aspect of time corresponding to each individual maze. M*i*-EP*j* denotes Maze-*i* in Episode-*j*. (d) The rate of time reduction between episode 1 and 2 is shown. T denotes Time. The reduction rate shows the quality of path improvement.

2 episodes for finding the best path in respect to the path it took in the first episode. This does not mean that our method outperforms the RL in general cases. RL is a general method for general paths or motion planning. It can also efficiently deals with very complex motion planning. Our method, however, is especially suitable to simple path-planning.

Nonetheless, by analyzing the proposed method in detail, two disadvantages of our method are readily apparent. First, (i) the search performed in the first episode might be locked in infinite loops of duplicated path-fragments. Considering graphs depicted in panels (c) of Fig. 9 in trial no. 6, the robot unfortunately repeated the same path-fragment too many times, which results in a long time used for searching. Second, (ii) if the goal was reached before all path-fragments were disclosed, then the path may probably not the real shortest path, though the robot can walk on such path faster in the second episodes. These drawbacks can be eliminated using Online CPM.

6. Online Mapping

The Map Optimization Algorithm processes the obtained $\mathcal{M}$ only after the end of episode 1. Unfortunately, it cannot be guaranteed that the goal would always be found in acceptable time as long as the robot searches in an arbitrary manner. The probability of being stuck in a loop, although very small, exists. For a solution, the robot is expected to consider the obtained map $\mathcal{M}_t$ at any current time (online) for making a decision at the junction to find the goal as quickly as possible and avoid traversing to duplicated path-fragments.

Figure 8 shows a typical CPM generated from Maze-3 in the 6th trial. The map contains too many non-essential

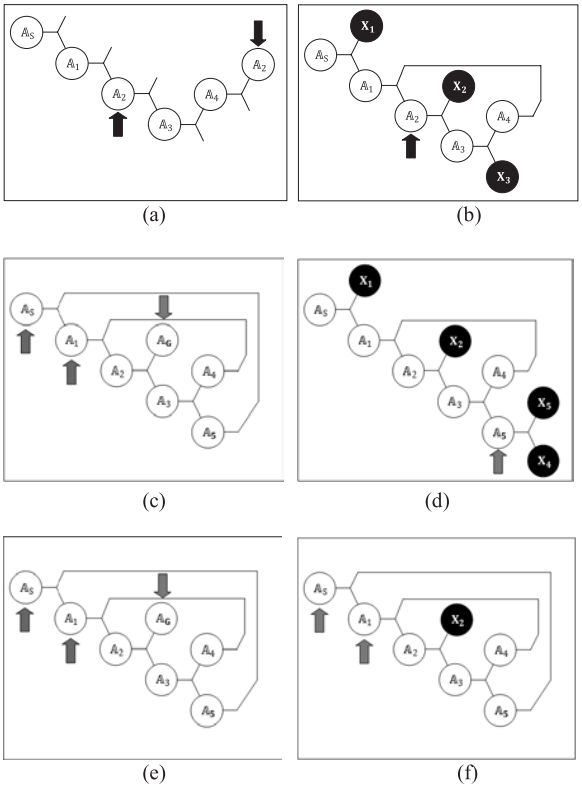


Fig. 10 Illustration of online CPM-building. (a) Duplication detected (b) Robot localization (c) Explore node X_2 and reach the goal (d) Explore node X_3 and found two more unexplored path-fragment (e) The complete optimized CPM (f) Explore node X_4 or X_5. Both of them lead to $junc(\mathbb{A}_1^-)$.

duplicated path-fragments (i.e. A_3). Consider Fig. 10, if the map were considered before the robot makes a decision (for entering a new path-fragment), it would know that A_2 is duplicated, as shown by the black arrows in panel (a). Consider panel (b): the robot connects path-fragment A_4 to $junc(\mathbb{A}_1^+) = junc(\mathbb{A}_2^-)$ and localizes itself at the path-fragment A_2 where it is going to make a decision at $junc(\mathbb{A}_2^+)$. At this step, the map contains three unexplored path-fragments connecting to $junc(\mathbb{A}_S^+)$, $junc(\mathbb{A}_2^+)$ and $junc(\mathbb{A}_3^+)$; namely, X_1, X_2 and X_3 (A_3 is not available as a choice because it has already been realized). The robot can randomly choose one to explore: it might walk back to A_1 and explore X_1, or it might pass A_3 to explore X_3. However, in this study, we let the robot move forward, so that it chooses either X_2 or X_3. If X_3 is chosen, the new path-fragment is A_5 (shown by the arrow in panel (d)) and two more unexplored path-fragments are found: X_4 and X_5. By continuing exploration, either to X_4 or X_5, the robot ends at $junc(\mathbb{A}_1^-)$ and they are both revealed simultaneously (indicated by arrows in panel (f)) with only unexplored path-fragments left: a goal. After exploration, the map would be completed as shown in panel (e). Then the robot luckily finds the goal, as depicted in panel (c) if X_2 is chosen. In this case, the robot might stop and remain satisfied with the incomplete map, which is the case of our study, where the main task is navigation. However, if the task is to obtain the complete map, the robot can continue exploration on X_3 and it will finally end up at the complete map in panel (e).

Using the online path planning technique, it can be guaranteed that the robot will always find the goal within a finite time according to the following theorem.

Theorem 1: *Given that all path-fragments in the environment are distinctive and asymmetrical, the time required for reaching the unknown goal is finite, as*

$$t = \sum_{i=1}^{n_p}(\mathrm{T}_i) + \max_i(\mathrm{T}_i) \cdot \left[\frac{n_A(n_A+1)}{2} - \frac{n_p(n_p+1)}{2}\right],$$

where n_A is the number of path-fragments in the environment, n_p is the number path-fragments the robot passes before detecting duplication, and T_k is the time required for passing the k^{th} path-fragment.

For the second disadvantage, based on the solution described in the previous paragraph, the robot can always find the shortest path. Once the goal is found, the robot would derive the optimized CPM, $\mathcal{M}^*$, either complete or incomplete. If the map is complete, then the robot can exactly retrieve the shortest path from the map. Hence, we focus on the latter case. For the incomplete CPM, the current path may not be the shortest one. As for the solution, during the way back of the robot, it would be worthwhile to explore the map more to see if there would be another shorter path. Particularly, assume that the robot finds the goal at time step t and that map $\mathcal{M}_t$ is derived, given n_J as the number of junctions $\mathbb{J} = \{J_1, J_2, \ldots, J_{n_J}\}$ corresponding to $\mathcal{M}_t = \{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{n+1}\}$ before the robot follows the map pass from $\mathbb{A}_{i+1}$ to $\mathbb{A}_i$ where $i \in I^+$, $i \leq n$, the robot tries to enter another different path-fragment $\mathbb{A}_i'$ at junction J_i. As long as the new path length is less than the old path length, the robot continues moving along the new path. In contrast, if the robot spent more steps to seek the goal in the new path than on the old path, it would exactly realize that this path will never be shorter than the previous one. Therefore it stops and goes back to the old path. In simpler view, this technique would be mostly like the creation of complete CPM as long as the length of path is not longer than the current one.

7. Experiment2: Real Robot

The experiment is done on the real e-puck robot. The setting differs slightly from that of the simulated experiment. Distinct from the first experiment, this experiment is conducted to show that out method is applicable to the real-robot. The maze would not be much complex. Also, the online mapping has been implemented. The velocity is set to $\frac{3\pi}{5}$ rad/s; thereby, one observation takes *ca.* 3 s. The learning time of 90 min (5400 s) is provided to the robot to learn the environment presented in Fig. 11 (a) Regarding testing, the maze is modified from the learning maze. According to Fig. 11 (c), the start point is at the upper right, whereas the goal is at the lower right dead-end.

For the result, the robot moves from the start point to search for the goal, which takes 20 CPs (*ca.* 121 s). After

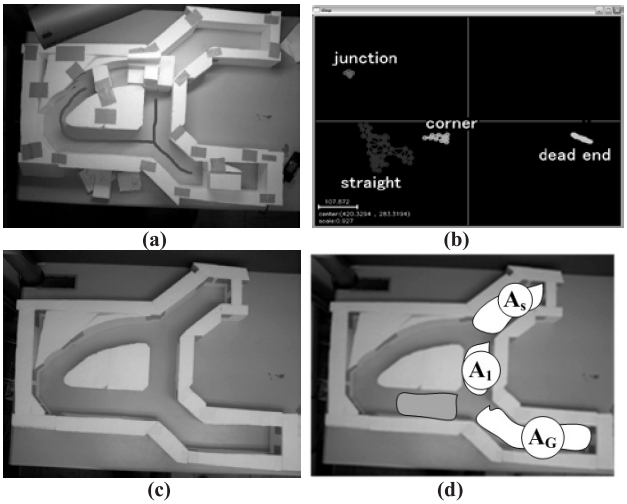


Fig. 11 (a) The real environment for learning CRPs (Learning Process). The line shows the available path-fragment. (b) Depiction of CRP generation using a real e-puck. (c) The testing environment. (d) The CPM result.

the robot locates the goal, it must walk back (independently) to the starting point to try reaching the goal again. On the way back, the robot tries a new path-fragment. However, the robot found that the length of the new path-fragment will never be shorter than the latest one and it gives up. The robot moves back to the junction and follows the path back to the starting point. Therefore, the robot spends 33 CPs in all to seek the goal (20 CPs for reaching the goal and 13 CPs for traveling back including trying new path-fragments). Then the robot moves to the goal again. The movement becomes smooth. The number of steps is reduced to 4 (*ca*. 44 s). Considering the time, the robot reduces the time from *ca*. 121 s to *ca*. 44 s. Regarding the aspect of improvement, the robot reduces the number of steps by 80% and time by 63.636%. The rate of reduction increases according to the maze complexity. Figure 11 (b) shows CPs generated by SOINN.

The CPM result is presented in Fig. 11 (d). This environment contains four path-fragments which are connected by two junctions. Each path-fragment is represented using a sequence of CP. The task is goal-oriented navigation. Therefore, the robot allocates no interest to path-fragment which lead to a longer path to the goal than those already found (gray color). Considering Fig. 11 (d), once the robot reaches the goal, it must go back to A_s. On the way back, it tries a new path-fragment at $junc(\mathbb{A}_G^-)$ (gray path-fragment in Fig. 11 (d)). However, it was found that the path-fragment's length is greater than that of A1, so the robot stops and follows the route back. At $junc(\mathbb{A}_S^+)$, it does not try the new path-fragment because the junction connects to A_s.

8. Discussion

Combining the application of SOINN-AM, the reasoning technique for decision making at junction, and the SSH architecture has been shown to be effective for robotic path planning. The use of CP enables the robot to remember all information like that of metric map, but with much lower storage. Advantages of using the common pattern for compact representation has been proved successful both in computer vision [32] and robotic appearance based navigation [34]. In our method, instead of image, we represent a whole path-fragment with a set of "ordered" CPs (or words). There is a room for improvement here. In this paper, we simply determine the similarity between path fragments by direct comparison because the maze is not too complex. One can use the 'term frequency-inverse document frequency', *tf-idf*, for path-fragment matching in more complex environment. Because the implementation of CP is, in some sense, similar to that of text retrieval approach, any techniques of such approach can be used with our method.

Additionally, the path partitioning at junction allows the robot to be free from the problem of large image database for visual-servoing. Many other methods divide the path fragment based on the appearance [35], so that the robot needs to keeps the images for action generation. Also, dividing the fragment in such manner may result in too great number of node in the topological map [35], which seriously affect the image retrieving time in long run. In contrast, our method output the number of fragments which matches well to the environment. According to theorem 1, if distinctive path-fragments in the environment, n_A, were numerous, then the environment would generally be considered complex because the number of junctions n_J directly depends on n_A: $n_J \propto n_A$. In addition, if the robot were unfortunate, it would end up in the duplicated path-fragments too early; thereby, the initial length of incomplete path n_p would be small. It would take time in finding the goal. On the other hand, if the environment is not too complex and the robot is not too unfortunate, the goal would be located quickly; a short path would be derived. In addition, the luck of the robot in maximizing n_p might be done in various ways such as adding guessing capability to the robot or activating a camera. Our method divides the path fragment based on junction like the work of Kuipers [16]. However, our method did not ignore the information of the path-fragment.

One may question about how well the current robot can detect the junction, especially in the outdoor scenes where the obstacles does not always exist. Our method in its current form cannot be applied to the path-planning in widely opened area in outdoors. For example, navigating the robot in the football field where there is no clear path of streets, the junction cannot be detected. This will results in failure in our path fragment definition. In other words, our proposed method works well in the environment where the junction can be detected and the clear path or road exists. Otherwise, the method cannot segment the map into path-fragments.

Another question arises with regard to the SOINN's resistance against sensory noises and motor noise. The experiments conducted in this study includes both sensory and motor noise. Generally, motor-noise can results in sensory-noise. The trajectories of the robot is very unlikely to be exactly similar in the real-world environment. Our method

can cope with these noises by using SOINN. Our system can correctly identify the robot's location regardless of the small difference in the position of the robot because it takes into account many observations to identify the path fragment. In other words, our system does accurately localize the robot's position by the exact coordinate but the path-fragment the robot is visiting. Doing this can eliminate both the noise from observation and the noise from different observing position. This also explains why representing the path fragment in this manner works well even under the perturbation of robot trajectory. Even though the length of the same path-fragment is slightly different because of motor perturbation, the fragment can still be correctly recognized as long as most of the CPs are similar. For example, the path-fragment represented by an A-Patterns 1-2-2-3-4-4 can still be recognized as similar to 2-2-3-4-4 because the number of matched CPs are sufficient in respect to Eq. (5) for being the same path-fragment. The result of clustering as shown in Fig. 5 shows that the noise generated from robot's motor can be eliminated by SOINNs.

Furthermore, considering the problem in which the starting point is changed, it is solvable based on our method. Because the robot has already obtained the complete optimized CPM, $\mathcal{M}^*$, it can find the shortest path between any pairs of path-fragments. However, there might be one remaining problem if the robot is left to start at the middle of path-fragment. In this case, the robot will not be able to recognize the path-fragment correctly and therefore cannot localize itself in CPM. Fortunately, the robot can solve this problem easily by starting the discovery process from a junction. Namely, if it is left at the middle of the path-fragment, then the representing CPs would not be junction CPs and the robot continues moving forward until the first junction is found. Once found, it can arbitrarily select the next path-fragment to enter and discover it. Once the robot becomes oriented in that manner, it can find the shortest path to reach the goal. This causes a small overhead in which the robot needs to neglect the first path-fragments and start observing from the next junction. However, the overhead time required for doing this will always be less than the time necessary for traversing the largest (longest) path-fragment.

Another point that should be further described about is the comparison between our method and the others. As described previously, our method is especially suitable to path-planning for simple-shape robotics. Reinforcement leaning and RRT [37], [38] is usually used for complex path-planning. It requires the good low-level controller to efficiently drive the robot along the path. Conceptually, our method can also be applied to 3-dimensional world in some senses. The path fragment is represented by a set of CPs. Each CP is obtained by clustering a number of feature vectors. These feature vectors are the observations the robot observes from the environment. These observations can be observed regardless of the dimensions of the environments. In other words, our proposed method can be applied to either 2-D or 3-D world depending on the designed low-level controller. However, in this study, the controller is a simple controller designed for 2-D environment so that the system is application to only 2-D path-planning. Nevertheless, we expect that our method can also be combined with the more efficient 3-D controller. Our method makes use of the reasoning on connection between path-fragments, so that it can automatically generate some new previously unvisited paths and can speed up the robot movement by estimating the length of each path-fragments. That is to say, as long as the junction detection can be efficiently done, our method would clearly perform faster that other path-planning method. Nonetheless, our method is based on the high-level concepts (assuming that low-level controller performs well), so that it might be currently not applicable to complex environment with complex robots, since such good controllers are very difficult to obtain.

The proposed method, although it has been adapted to online path planning, retains an important disadvantage. As described previously, the CPs are not incrementally learnt. Although it might be said that the junction with size of more than five-way is rare, such path-fragments do exist. For the most suitable solution, the CPs must be learned incrementally. When the robot detects any new environment, it is expected to be capable of deciding if the current path-fragment is best represented by the existed CP, or if a new CP is best added. Solving the problem in this manner will enable the robot incrementally to gain CP unlimitedly, which is suited to life-long planning.

We believe that one can integrate the proposed path-planning the post-processing of the mapping method. The proposed simple reasoning technique would be very efficient for the SSH-based SLAM.

Acknowledgments

This research was supported by the Industrial Technology Research Grant Program in 2004 from the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

References

[1] O. Michel, "WebotsTM: Professional mobile robot simulation," International J. Advanced Robotic Systems, vol.1, no.1, pp.39–42, 2004.
[2] O. Michel, "Webots: Symbiosis between virtual and real mobile robots," Proc. Int. Conf. on Virtual World, pp.254–263, 1998.
[3] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction (Hard Cover), ch. 3, MIT Press, Cambridge, 1998.
[4] S. Thrun, W. Burgard, and D. Fox, Probabilistic Robotics (Hard Cover), ch. 10, MIT Press, Cambridge, 2005.
[5] F. Shen and O. Hasegawa, "An incremental network for online unsupervised classification and topology learning," Neural Netw., vol.19, no.1, pp.90–106, 2005.
[6] R. Brooks, "A robust layered control system for a mobile robot," IEEE J. Robotics and Automation, vol.RA-2, no.1, pp.14–23, 1986.
[7] A. Arleo, F. Smeraldi, and W. Gerstner, "Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning," IEEE Trans. Neural Netw., vol.15, no.3, pp.639–652, May 2004.
[8] T. Strosslin, D. Sheynikhovich, R. Chavarriaga, and W. Gerstner,

"Robust self-localisation and navigation based on hippocampal place cells," Neural Netw., vol.18, no.9, pp.1125–1140, Nov. 2005.
[9] A. Arleo, F. Smeraldi, S. Hug, and W. Gerstner, "Place cells and spatial navigation based on vision, path integration, and reinforcement learning," Proc. Neural Information Processing Systems, pp.89–95, 2001.
[10] H. Ichiki, M. Hagiwara, and M. Nakagawa, "Kohonen feature maps as a supervised learning machine," Proc. IEEE Int'l Conf. Neural Networks, pp.1944–1948, 1993.
[11] M. Kalisiak and M. Panne, "RRT-blossom: RRT with a local flood-fill behavior," Proc. Int'l Conf. Robotics and Automation, pp.1237–1242, 2006.
[12] M. Kalisiak and M. Panne, "Faster motion planning using learned local viability models," Proc. Int'l Conf. Robotics and Automation, pp.2700–2705, 2007.
[13] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization," IEEE Trans. Robot. Autom., vol.17, no.2, pp.125–137, April 2001.
[14] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," IEEE Trans. Robot. Autom., vol.7, no.6, pp.859–865, Dec. 1991.
[15] J. Modayil, P. Beeson, and B. Kuipers, "Using the topological skeleton for scalable global metrical map-building," Proc. Int'l Conf. Intelligent Robots and Systems, pp.1530–1536, 2004.
[16] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy," Proc. Int'l Conf. Robotics and Automation, pp.4845–4851, 2004.
[17] F. Savelli and B. Kuipers, "Loop-closing and planarity in topological map-building," Proc. Int'l Conf. Intelligent Robots and Systems, pp.1511–1517, 2004.
[18] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "A new approach for large-scale localization and mapping: Hybrid metric-topological SLAM," Proc. Int'l Conf. Robotics and Automation, pp.2061–2067, 2007.
[19] M. Dissanayake, P. Newman, S. Clark, H. Durrant-whyte, and M. Csorba, "A solution to the simultaneous localization and map building," IEEE Trans. Robot. Autom., vol.17, no.3, pp.229–241, June 2001.
[20] S. Thrun and A. Bucken, "Integrating grid-based and topological maps for mobile robot navigation," Proc. 13th Nat. Conf. Advanced Artificial Intelligence, pp.944–950, 1996.
[21] G.Z. Grudic, V. Kumar, and L. Ungar, "Using policy gradient reinforcement learning on autonomous robot controllers," Proc. Int'l Conf. Intelligent Robots and Systems, pp.406–411, 2003.
[22] H. Igarashi, "Path planning of a mobile robot by optimization and reinforcement learning," Artif. Life Robotics, vol.6, no.2, pp.59–65, 2002.
[23] H.J. Chang, C. Lee, L. Yung-Hsiang, and Y.C. Hu, "P-SLAM: Simultaneous localization and mapping with environmental-structure prediction," IEEE Trans. Robotics, vol.23, no.2, pp.281–293, April 2007.
[24] J. Blanco, J. Fernandez-Madrigal, and J. Gonzalez, "Toward a unified Bayesian approach to hybrid metric-topological SLAM," IEEE Trans. Robotics, vol.24, no.2, pp.259–270, April 2008.
[25] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Trans. Robot. Autom., vol.12, no.4, pp.566–580, Aug. 1996.
[26] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. Reif, "Narrow passage sampling for probabilistic roadmap planning," IEEE Trans. Robotics, vol.21, no.6, pp.1105–1115, Dec. 2005.
[27] M. Toussaint, "A sensorimotor: Modulating lateral interactions for anticipation and planning," Neural Comput., vol.18, pp.1132–1155, 2006.
[28] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," Proc. Int'l Conf. Machine Learning, pp.1–8, 2004.
[29] E. Remolina and B. Kuipers, "Towards a general theory of topological maps," Artif. Intell., vol.152, pp.47–104, 2004.
[30] T. Goedeme, M. Nuttin, T. Tuytelaars, and L.V. Gool, "Omnidirectional vision based topological navigation," Int. J. Comput. Vis., vol.74, no.3, pp.219–236, 2007.
[31] B. Kuipers, "The spatial semantic hierarchy," Artif. Intell., vol.119, no.1-2, pp.191–233, 2000.
[32] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," Proc. IEEE Int. Conf. Computer Vision, pp.1470–1477, 2003.
[33] A. Sudo, A. Sato, and O. Hasegawa, "Associative memory for online learning in noisy environments using self-organizing incremental neural network," IEEE Trans. Neural Netw., vol.20, no.6, pp.964–972, 2009.
[34] M. Cummins and P. Newman, "FAB-MAP: Probabilistic localization and mapping in the space of appearance," Int. J. Robot. Res., vol.27, pp.647–665, 2008.
[35] C. Valgren and A. Lilienthal, "Incremental spectral clustering and seasons: Appearance-based localization in outdoor environments," Proc. IEEE Int'l Conf. Robotics and Automation, pp.1856–1861, 2008.
[36] C. Rasmussen, "Combining laser range, color, and texture cues for autonomous road following," Proc. IEEE Int. Conf. Robotics and Automation, pp.4320–4325, 2002
[37] S. Kagami, J.J. Kuffner, K. Nishiwaki, K. Okada, M. Inaba, and H. Inoue, "Humanoid arm motion planning using stereo vision and RRT search," Proc. IEEE/RSJ Int'l Conf. Intelligent Robots and Systems, pp.2167–2172, 2003.
[38] S. Kagami, J.J. Kuffner, K. Nishiwaki, K. Okada, M. Inaba, and H. Inoue, "Humanoid arm motion planning based on RRT search of 3D depth map," J. Robotics and Mechatronics, vol.15, no.2, pp.200–207, 2003.

Appendix

Proof of Theorem 1: Given n_A as the number of all distinct path-fragments in the environment, let $\mathbb{T}$ be the set of macro-time-steps T required for passing path-fragments corresponding to the sequence of path-fragments in the map $\mathcal{M}$. Starting from the start point, the robot randomly makes decisions until it passes the duplicated path-fragment at macro time step $\mathrm{T_k}$; it means that $\mathbb{T} = \{\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_{k-1}\}$, $\mathcal{M} = \{\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{k-1}\}$, $\mathbb{J} = \{\mathrm{J}_1, \mathrm{J}_2, \ldots, \mathrm{J}_{k-2}\}$. Because n_p is the length of map $\mathcal{M}$, it is inferred that $n_p = \mathrm{k} - 1$. Consequently, the number of unknown path-fragments is $n_A - n_p$ and the time spent for the first search is $\sum_{i=1}^{n_p} \mathrm{T}_i$. Every time the robot passes the duplicated path-fragment, it performs either of the following.

1) If the current junction has some remaining unknown ports, it randomly chooses one and moves on.
2) If the current junction has no unknown ports left, it continues forward until reaching a junction that has some remaining unknown ports.

In doing so, every time the robot finds duplicated path-fragments, it is guaranteed that one more unknown path-fragments would be revealed. Namely, $\mathcal{M} \cup \{\mathbb{A}_{\mathrm{new}}\}$ within maximum time $\sum_{i=1}^{n_p} \mathrm{T}_i$. Therefore, if the path-fragments in closed environment n_A are finite in number, then $\mathrm{n}(\mathcal{M})$ equals n_A within time

$$\sum_{j=1}^{n_A-n_p}\left(\sum_{i=1}^{n_{p_j}} \mathrm{T}_i\right) \leq \sum_{i=1}^{n_A-n_p}\left(n_{p_j} \cdot \max_{1\leq i\leq n_{p_j}} \mathrm{T}_i\right).$$

The number of path-fragments is finite. Therefore, the maximum length of $\mathcal{M}$, n_{p_j} is finite. Because every path-fragment has limited length, the time for passing each path-fragment is limited, as is the maximum value. Furthermore, noticing that n($\mathcal{M}$) increases by 1 for each iteration, it can be inferred that

$$\begin{aligned}\sum_{i=1}^{n_A-n_p}\left(n_{p_j} \cdot \max_i(\mathrm{T}_i)\right) &= \max_i(\mathrm{T}_i) \cdot \sum_{i=1}^{n_A-n_p}(n_{p_j}) \\ &= \max_i(\mathrm{T}_i) \cdot \frac{(n_A-n_p)(n_A+n_p+1)}{2} \\ &= \max_i(\mathrm{T}_i) \cdot \left[\frac{n_A(n_A+1)}{2} - \frac{n_p(n_p+1)}{2}\right].\end{aligned}$$

We conclude that the time for searching is bounded to limited time.

Aram Kawewong received the B.S. degree in Computer Engineering from Chulalongkorn University in 2005, and M.S. degree from Tokyo Institute of Technology in 2008. He is currently the PhD student of Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology. His research focuses on Visual SLAM and Robotic Vision.

Yutaro Honda received the B.S. degree in Computer Science from Tokyo Institute of Technology in 2008. He is currently the master student of Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology. His current research focuses on mobile robot navigation, and interaction between robot and human.

Manabu Tsuboyama received the B.S. degree in Electrical Engineering from Tokyo University of Agriculture and Technology in 2007, and M.S. degree in Computer Engineering from Tokyo Institute of Technology in 2009. He is currently working for Keyence Co. Ltd. in Japan. He is interested in autonomous mobile robot path and movement planning.

Osamu Hasegawa received his Dr.Eng. degree in electronic engineering from the University of Tokyo in 1993. He was a research scientist at Electrotechnical Lab (ETL) from 1993 to 1999 and at Advanced Industrial Science and Technology (AIST) from 2000 to 2002. From 1999 to 2000 he was a visiting scientist at the Robotics Institute, Carnegie Mellon University. In 2002 he became a faculty member of the Imaging Science and Engineering Lab, Tokyo Institute of Technology. In 2002, he was jointly appointed researcher at PRESTO, Japan Science and Technology Agency (JST). He is a member of the IEEE Computer Society and IPSJ.