$\varepsilon^\star$: An Online Coverage Path Planning Algorithm

Junnan Song[†] **Shalabh Gupta**[†⋆]

***Abstract*—The paper presents an algorithm, called $\varepsilon^\star$, for online *Coverage Path Planning* (CPP) of unknown environments. The algorithm is built upon the concept of an *Exploratory Turing Machine* (ETM) which acts as a supervisor to the autonomous vehicle to guide it with adaptive navigation commands. The ETM generates a coverage path online using *Multiscale Adaptive Potential Surfaces* (MAPS) which are hierarchically structured and dynamically updated based on sensor information. The $\varepsilon^\star$-algorithm is computationally efficient, guarantees complete coverage, and does not suffer from the local extrema problem. Its performance is validated by: i) high-fidelity simulations on Player/Stage and ii) actual experiments in a laboratory setting on mobile robots.**

1. Introduction

Typical operations of autonomous vehicles (AVs) that require *Coverage Path Planning* (CPP) [1][2] include floor cleaning [3], lawn mowing [4], map generation [5], oil spill cleaning [6], demining [7][8], etc. Often, these operations are conducted in either completely unknown or partially known environments. Therefore, it is essential to utilize sensor-based methods which enable online planning for complete coverage [9][10].

A variety of coverage algorithms exist in literature; a review of such algorithms is presented in [11]. The CPP methods are categorized into two types: offline or online (i.e. sensor-based). While offline approaches [12] assume the environment to be *a priori* known, online approaches [13] compute the coverage path *in situ* based on sensor information. Independently, CPP methods are also characterized as randomized or systematic. Random strategies follow simple behavior-based rules, requiring neither localization system nor costly computational resources; however, they generate strongly overlapped trajectories. In contrast, the systematic coverage strategies are typically based on cellular decomposition [13] of the search area into cells of varying shapes. Lumelsky et al. [1] decomposed the area into fixed-width cells and presented the *sightseer* and the *seed-spreader* strategies for coverage. This algorithm was later improved by Hert et al. [2] by reducing the upper bound on path length; however, these algorithms are limited to a small set of obstacle geometries.

Zelinsky et al. [12] used a grid of equal-sized cells to partition an *a priori* known area and assigned a potential to each cell; then the coverage path was generated along the steepest ascent from the start to the goal. Koenig et al. [14] used the so-called 'ant robots' with limited sensing and computational capabilities to scan unknown areas. Gabriely and Rimon [15] used the *Spanning Tree Covering* (STC) algorithm for online coverage, which was later improved to *Full Spiral STC* (FS-STC) algorithm [16]. Gonzalez et.al. [17] proposed the *Backtracking Spiral Algorithm* (BSA), which utilizes a spiral filling path for online coverage. Both STC and BSA generate spiral paths, and this limits their application when turning is regarded expensive and undesired. More recently, Acar and Choset [13] developed a sensor-based coverage method that detects the critical points on obstacles to divide the area into cells; coverage is then achieved via back and forth motion in each cell. However, this method relies on correct detection and pairing of the IN and OUT critical points [18], which could be difficult in complex environments. Furthermore, this method cannot function in rectilinear environments.

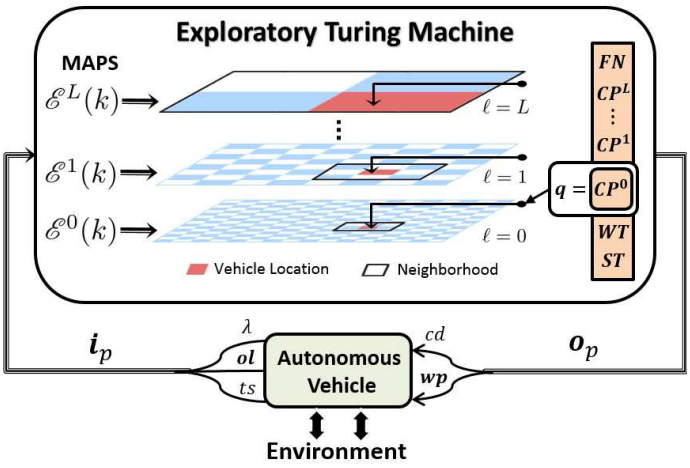


Figure 1: ETM as a supervisor of the autonomous vehicle

This paper presents the $\varepsilon^\star$-algorithm (that stands for ε-STAR or "*ε-coverage via Structural Transitions to Abstract Resolutions*"), where ε refers to the cell resolution. As shown in Fig. 1, the algorithm utilizes an *Exploratory Turing Machine* (ETM), that consists of a two-dimensional multilevel tape formed by *Multiscale Adaptive Potential Surfaces* (MAPS). The ETM stores and updates the information corresponding to unexplored, explored, and obstacle-occupied regions, as time-varying potentials on MAPS. In essence, it takes advantage of both the potential field-based and sensor-based planning methods by incrementally building the MAPS using real-time sensor measurements. While, by default the ETM uses the lowest level of MAPS for generating the coverage path online, it switches to higher levels as needed to escape from a local extremum. The ETM acts as a supervisor to the AV and guides it with adaptive navigation commands.

The advantages of $\varepsilon^\star$-algorithm in comparison to existing online methods are that it produces the desired back and forth motion and does not rely on critical points detection. Furthermore, the algorithm is computationally efficient, guarantees complete coverage, and does not suffer from the local extremum problem. The $\varepsilon^\star$-algorithm is validated via: i) high-fidelity simulations on Player/Stage and ii) real experiments in a laboratory setting.

The rest of the paper is organized as follows. Section 2 describes the CPP problem. Section 3 presents the details of the $\varepsilon^\star$-algorithm, while Section 4 presents the results. The paper is concluded in Section 5 with suggestions on future work.

[†] Department of Electrical & Computer Engineering, University of Connecticut, Storrs, CT 06269, USA

[⋆] Corresponding author, email: shalabh.gupta@uconn.edu

The paper includes video files that show simulation and experimental results.

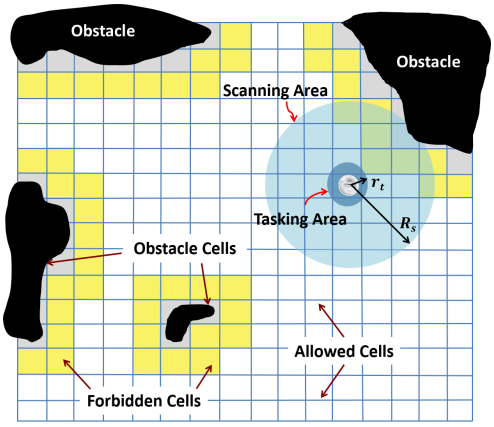


Figure 2: An autonomous vehicle working in its environment.

2. Problem Description

This section presents the concept of ε-coverage of an environment that is populated with unknown obstacles of arbitrary shapes. The autonomous vehicle as shown in Fig. 2 contains: i) a localization device (e.g. GPS), ii) range detectors (e.g. a laser scanner) to detect obstacles within a circular region of radius $R_s \in \mathbb{R}^+$, and iii) a task-specific sensor for performing its main task (e.g., cleaning) with a circular area of radius $r_t \leq R_s$. For operation in GPS-denied environments, the $\varepsilon^\star$-algorithm can be integrated with a SLAM algorithm as in [19] to achieve coverage.

Let $\mathscr{A} \subset \mathbb{R}^2$ be the estimated region which includes the desired area to be covered. First we construct a tiling on $\mathscr{A}$ as follows.

Definition 2.1 (Tiling). *A set $\mathscr{T} = \{\tau_\alpha \subset \mathbb{R}^2, \alpha = 1,\ldots|\mathscr{T}|\}$ is called a tiling of $\mathscr{A}$ if its elements: i) have mutually exclusive interiors, i.e. $\tau_\alpha^\circ \bigcap \tau_\beta^\circ = \emptyset$, $\forall \alpha,\beta \in \{1,\ldots|\mathscr{T}|\}$, $\alpha \neq \beta$, where $^\circ$ denotes an interior, and ii) form an exact cover of $\mathscr{A}$, i.e. $\mathscr{A} = \bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha$. If an exact cover is not possible (e.g., square tiles cannot exactly cover a circular region), condition ii) can be relaxed to $\mathscr{A} \subseteq \bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha$, to form a minimal tiling of $\mathscr{A}$, s.t. removal of any single tile destroys the covering property.*

The tiling formed by square tiles of side ε is called an *ε-cell tiling*. It is recommended that an ε-cell should be atleast big enough such that it can contain the AV and small enough such that the tasking sensor is able to cover it when the AV passes through it. Within these two bounds, the choice of ε depends on the following factors. A smaller ε provides a better approximation of the search area and its obstacles. On the other hand, a larger ε reduces the computational complexity by requiring less number of ε-cells to cover the area and it also provides improved robustness to uncertainties for localization within a cell.

The tiling $\mathscr{T}$ is partitioned into three sets: i) *obstacle* ($\mathscr{T}^o$), ii) *forbidden* ($\mathscr{T}^f$), and iii) *allowed* ($\mathscr{T}^a$), as shown in Fig. 2. While the obstacles cells are occupied by obstacles, the forbidden cells create a buffer around the obstacles to prevent collisions due to inertia or large turning radius of the AV. The remaining cells are allowed which are desired to be covered. The AV dynamically discovers the obstacles online, updates the obstacle and forbidden cells, and performs tasks in the allowed cells. Now we define the concept of ε-coverage of the allowed cells.

Definition 2.2 (ε-Coverage). *Let $\mathscr{R}(\mathscr{T}^a)$ denote the total area of the allowed cells in $\mathscr{T}^a \subseteq \mathscr{T}$. Let $\tau(k) \in \mathscr{T}^a$ be the ε-cell that is visited and explored by the tasking sensor of the AV at time k. Then, the area $\mathscr{A}$ is said to achieve ε-coverage if $\exists\ K \in \mathbb{Z}^+$ s.t. the sequence $\{\tau(k),\ k = 1,\ldots K\}$ covers $\mathscr{R}(\mathscr{T}^a)$, i.e.*

$$\mathscr{R}(\mathscr{T}^a) \subseteq \bigcup_{k=1}^{K} \tau(k). \tag{1}$$

Remark 2.1. *ε-coverage achieves complete coverage if the tasking sensor completely covers each ε-cell visited by the AV.*

3. The $\varepsilon^\star$-Algorithm

The $\varepsilon^\star$-algorithm utilizes the concept of ETM for ε-coverage of unknown environments. As shown in Fig. 1, the ETM constantly takes feedback from the AV and in turn acts as its supervisor to guide it with operational commands and navigation waypoints; thus, it falls in the category of *Interactive Transition Systems* [20][21]. The ETM consists of a single tape head and a two-dimensional multilevel tape formed by MAPS (see Section 3-A), which act as guidance surfaces for decision-making. Formally, the ETM is defined as follows.

Definition 3.1 (Exploratory Turing Machine). *An Exploratory Turing Machine is a 7-tuple $M = (Q, \Xi, I_p, O_p, \delta, q_0, F)$ where:*

- *$Q = \{ST, CP^0, ..., CP^L, WT, FN\}$ is the set of machine states, where $ST \equiv$'Start', $CP \equiv$'Compute', $WT \equiv$'Wait', and $FN \equiv$'Finish'. The superscript on CP specifies the level of MAPS at which the head is operating. The WT state implies waiting for the AV to finish its task in the current cell.*
- *$\Xi = \{\Xi^\ell : \ell = 0, 1, ...L\}$, where $\Xi^\ell = \{\Xi^\ell_{min}, ...\Xi^\ell_{max}\}$ is the set of potential values that can be encoded on each cell at Level ℓ of the MAPS.*
- *I_p is the set of all input parameters which describe the feedback information received from the AV. The input vector $\boldsymbol{i}_p \in I_p$ consists of:*
  - i. *$\lambda \in \{1,\ldots|\mathscr{T}|\}$: Index of the ε-cell where the AV is currently located on tiling $\mathscr{T}$. It is computed using the onboard positioning system.*
  - ii. *$\boldsymbol{ol} \subset \{1,\ldots|\mathscr{T}|\}$: Vector of the obstacle locations which consists of the indices of all ε-cells where obstacles are detected using the range detectors.*
  - iii. *$ts \in \{cm, ic\}$: Task status of the AV in its current ε-cell, where $cm \equiv$'Complete' and $ic \equiv$'Incomplete'.*
- *O_p is the set of output parameters which describe instructions for the AV. The output vector $\boldsymbol{o}_p \in O_p$ consists of:*
  - i. *$cd \in \{mv, tk, id, sp\}$: Command to the AV, where $mv \equiv$'Move', $tk \equiv$'Task', $id \equiv$'Idle', and $sp \equiv$'Stop'.*
  - ii. *$\boldsymbol{wp} \subset \{1,\ldots|\mathscr{T}|\}$: Candidate set of navigation waypoints for the AV trajectory on tiling $\mathscr{T}$.*
- *δ is the control function that is a partial mapping from $I_p \times Q \times \Pi_{\mathscr{N}^\ell} \rightarrow Q \times \Pi \times O_p$, where Π is the set of all possible configurations of potentials on MAPS generated by the sets Ξ^ℓ, while $\Pi_{\mathscr{N}^\ell}$ is the above set restricted to a local neighborhood $\mathscr{N}^\ell$ at Level ℓ of the MAPS.*
- *q_0=ST is the initial state, and*
- *F=FN is the final state implying complete coverage.*

Remark 3.1. *An advantage of Turing Machine over Finite State Automaton (FSA) is that it has the capacity of containing memory which is a necessary feature for coverage problems.*

Before delving into the operational details of the ETM, we describe the process of dynamic construction of the MAPS.

A. *Description of MAPS*

In order to build MAPS, first a hierarchical multiscale tiling (MST) is constructed on the area $\mathscr{A}$ by recursive decomposition. As shown in Fig. 1, the ε-cell tiling $\mathscr{T}$ of the search area forms the finest level of MST and is referred as $\mathscr{T}^0$ from now on. Let $n \in \mathbb{N}$ be the maximum number of ε-cells along x-axis over all rows. If n is even, then the axis is divided into two regions of $\frac{n}{2}$ elements each. If n is odd, then the axis is divided into two regions with n' and $n'-1$ elements, such that $n' \in \mathbb{N}$ and $2n'-1 = n$. This procedure is repeated along the y-axis over all columns to generate 4 coarse cells in total, which form the coarsest tiling, i.e. $\mathscr{T}^L$, $L \in \mathbb{N}$. Now, again let $n \in \mathbb{N}$ be the maximum number of ε-cells along the x-axis in a coarse cell. Then, using the above procedure, each of these four coarse cells are further divided into two regions along each axis to generate 16 cells in tiling $\mathscr{T}^{L-1}$. This procedure is repeated until $n/2 < 2$ or $n'-1 < 2$ to generate a MST with tilings $\mathscr{T}^0, \mathscr{T}^1, \ldots \mathscr{T}^L$ such that $\mathscr{T}^\ell = \{\tau_{\alpha^\ell} : \alpha^\ell = 1, \ldots |\mathscr{T}^\ell|\}$, $\forall \ell \in \{0, \ldots L\}$, where α^ℓ, $\forall \ell \geq 1$, indexes coarse cells at Level ℓ of the MST, while α^0 indexes ε-cells.

i.) **Modeling of the Potential Surface at the Lowest Level**: For level $\ell = 0$, the potential surface is constructed using a simple process. First, the environmental information is encoded on $\mathscr{T}^0$ by assigning a symbolic state [22] to each ε-cell $\tau_{\alpha^0} \in \mathscr{T}^0$ from the alphabet set $S = \{\text{O, F, E, U}\}$, where O $\equiv$ *obstacle*, F $\equiv$ *forbidden*, E $\equiv$ *explored*, and U $\equiv$ *unexplored*. Then, the potential surface $\mathscr{E}^0(k)$=$\{\mathscr{E}_{\alpha^0}(k) \in \Xi^0 : \alpha^0 = 1, \ldots |\mathscr{T}^0|\}$, is constructed by assigning a discrete potential to each τ_{α^0}, such that

$$\mathscr{E}_{\alpha^0}(k) = \begin{cases} -1, & \text{if } s_{\alpha^0}(k) = \text{O or F} \\ 0, & \text{if } s_{\alpha^0}(k) = \text{E} \\ B_{\alpha^0}, & \text{if } s_{\alpha^0}(k) = \text{U} \end{cases} \tag{2}$$

where $s_{\alpha^0}(k) \in S$ is the state of τ_{α^0} at time k. The first condition in Eq. (2) assigns a potential of -1 to τ_{α^0}, if it contains an obstacle or if it is forbidden, i.e. it lies in an obstacle neighborhood. The latter creates a *forbidden zone* around the obstacles to prevent the AV from colliding with the obstacles due to inertia, skidding, large turning radius, or localization errors. The second condition in Eq. (2) assigns a potential of 0 to τ_{α^0}, if it has been explored by the tasking sensor. The third condition assigns a potential of B_{α^0} to τ_{α^0}, if it is yet unexplored, where B=$\{B_{\alpha^0} \in \{1, \ldots B_{max}\}, \alpha^0 = 1, \ldots |\mathscr{T}^0|\}$ is a time-invariant exogenous potential field. It is designed offline to have plateaus of equipotential surfaces along each column of the tiling. As shown in Fig. 3, the plateaus monotonically increase in height by one unit from 1 on the rightmost column to B_{max} on the leftmost column. This field facilitates back and forth motion in an obstacle-free region by following the highest equipotential surface from left to right. The sweep direction could be adapted by modifying B according to the users' needs. Clearly, $\Xi^0_{min} = -1$ and $\Xi^0_{max} = B_{max}$. The symbolic encoding is updated by the ETM using sensor information and results in a dynamically changing potential surface $\mathscr{E}^0(k)$ as shown in Fig. 3.

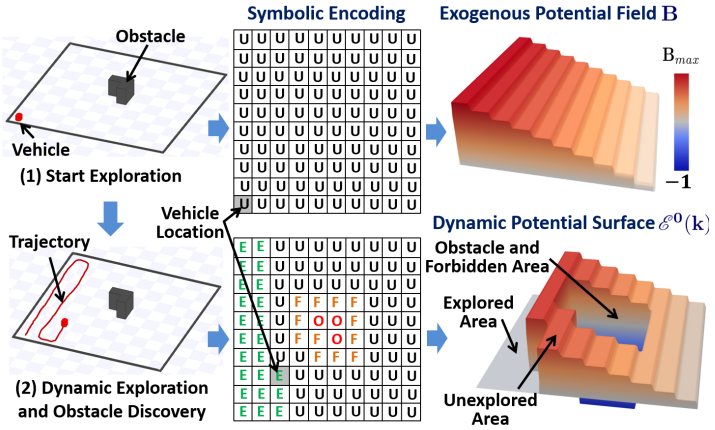


Figure 3: Dynamic construction of the potential surface $\mathscr{E}^0$.

ii.) **Modeling of the Potential Surfaces at Higher Levels**: For $1 \leq \ell \leq L$, the potential surface $\mathscr{E}^\ell(k)$=$\{\mathscr{E}_{\alpha^\ell}(k) \in \Xi^\ell : \alpha^\ell = 1, \ldots |\mathscr{T}^\ell|\}$, is constructed by assigning a potential to each coarse cell $\tau_{\alpha^\ell} \in \mathscr{T}^\ell$. This is done by assigning τ_{α^ℓ} the average potential generated by all the unexplored ε-cells within τ_{α^ℓ}, such that

$$\mathscr{E}_{\alpha^\ell}(k) = p^U_{\alpha^\ell}(k) \overline{B}_{\alpha^\ell} \tag{3}$$

where $\overline{B}_{\alpha^\ell}$ is the mean exogenous potential of τ_{α^ℓ} and $p^U_{\alpha^\ell}(k)$ is the probability of unexplored ε-cells in τ_{α^ℓ}. The probability could be computed using a simple counting process. With little inspection, it could be seen that $\Xi^\ell_{min} = 0$ and $\Xi^\ell_{max} = \max_{\tau_{\alpha^\ell} \in \mathscr{T}^\ell} \overline{B}_{\alpha^\ell}$.

B. *Operation of the ETM as a Supervisor*

The ETM functions as follows. Its head has a state $q \in Q$ and it operates on one level of the MAPS at a time; by default Level 0. Fig. 4 shows the state transition graph of the ETM, which realizes the control function δ. The input vectors $\boldsymbol{i}_{p_i} \in I_p, i = 1, 2$, the output vectors $\boldsymbol{o}_{p_i} \in O_p, i = 1, \ldots 4$ and the state transition conditions are defined therein. While the operational details in each state are presented later, a summary is provided here.

In state ST, the ETM initializes the MAPS. Since the whole area is initially unexplored, all ε-cells are assigned the state U, thus MAPS are constructed using only the potential field B. Then, the ETM cycles on and between the states CP^0 and WT, as follows. In each iteration of state CP^0, the ETM takes input from the AV on the newly discovered obstacle locations and the current position (λ) of the AV. Then, it moves the head on the tape to λ and updates the MAPS in accordance with the discovered obstacles, and performs the following operations: i) reads the potentials from the local neighborhood $\mathscr{N}^0(\lambda)$ of λ to compute the new waypoint, ii) changes the head state to WT if waypoint is reached otherwise stays in CP^0, and iii) generates an output vector for the AV containing the operational command and the new waypoint. In each iteration of state WT, it receives the task status from the AV, and continues to send tasking command until it is complete. Once the current cell is tasked, it updates the MAPS and returns to the state CP^0.

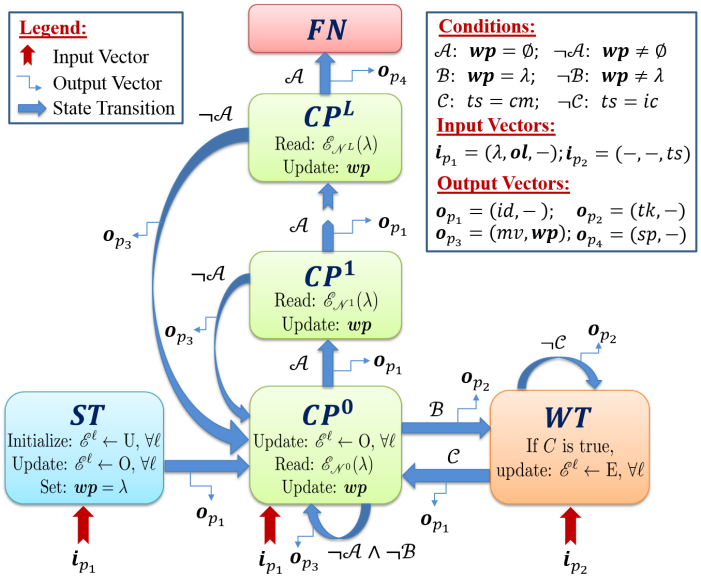


Figure 4: State transition graph of the ETM.

If the head gets stuck in a *local extremum* in state CP^0, i.e. no waypoint could be found in the local neighborhood at Level 0 of the MAPS, then it switches to CP^1 and operates on Level 1. Here it searches for the coarse cell with the highest positive potential in a local neighborhood $\mathcal{N}^1(\lambda)$ to find a waypoint. If no waypoint is found even at Level 1, then it switches to state CP^2 and so on until it finds one, then it comes down to state CP^0 and continues. If no waypoint is found even at the highest level then the ETM halts in state FN and the coverage is complete. The details of operations in each state are explained below.

i.) **Operation in the ST State:** The ETM starts in state $q = ST$ at $k = 0$ when the AV is turned on. Since no *a priori* information is available, all ε-cells in $\mathscr{T}^0$ are initialized with the state U, i.e. *unexplored*. Then all ε-cells are assigned potentials according to field B as per Eq. (2). Subsequently, all higher level cells are assigned potentials using Eq. (3), by substituting $p^U_{\alpha^\ell}(0) = 1$. This MAPS initialization process is denoted as $\mathscr{E}^\ell \leftarrow$ U, $\forall\ell \in \{0, \ldots L\}$.

Map updating process: Next, the AV detects its current location λ and obstacle locations $\boldsymbol{ol}$ using its onboard sensing systems, and sends this information to the ETM via the input vector $\boldsymbol{i}_{p_1}$. The ETM moves its tape head to λ and initializes the waypoint $\boldsymbol{wp} = \lambda$. Then it updates the symbolic encoding by flipping the states at all newly discovered obstacle indices $\boldsymbol{ol}$ to O, and their associated neighborhood cells to F; subsequently, their potentials are updated to -1 using Eq. (2). As a next step, the probabilities $p^U_{\alpha^\ell}(0)$ are updated at all higher levels, for the corresponding coarse cells containing the newly discovered obstacles, by counting the remaining unexplored ε-cells inside those coarse cells. Then, using Eq. (3), the potential surfaces are updated for all levels $\ell \in \{1, \ldots L\}$. In short, this entire MAPS updating process is denoted as $\mathscr{E}^\ell \leftarrow$ O, $\forall\ell \in \{0, \ldots L\}$.

Then, the ETM transitions to the computing state CP^0 and sends the output vector $\boldsymbol{o}_{p_1}$ to command the AV to go 'idle'.

ii.) **Operation in the CP^0 State:** CP^0 is the default state to compute waypoints. Every time the ETM reaches CP^0, it first

Algorithm 1: Update $\boldsymbol{wp}(k)$

input : $\boldsymbol{wp}(k-1)$, λ, $\mathscr{E}_{\mathcal{N}^\ell}$, $q \in \{CP^\ell, \ell = 0, \ldots L\}$
output: $\boldsymbol{wp}(k)$
1 **if** $q = CP^0$ **then**
2 compute $\mathscr{D}^0$ // form the computing set $\mathscr{D}^0$ using Eq. (4)
3 **if** $\lambda \in \mathscr{D}^0$ **then** // current cell λ is unexplored
4 **if** $\{\lambda^{up}, \lambda^{down}\} \subset \mathscr{D}^0$ **then** // λ^{up} & λ^{down} unexplored
5 $\boldsymbol{wp}(k) = \{\lambda^{up}, \lambda^{down}\}$ //AV will pick one per Eq. (5)
6 **else** $\boldsymbol{wp}(k) = \lambda$ // set λ as waypoint and start tasking
7 **else if** $\mathscr{D}^0 \neq \emptyset$ **then** // other eligible ε-cells exist
8 $\boldsymbol{wp}(k) = \arg\max_{\alpha^0 \in \mathscr{D}^0} \mathscr{E}_{\alpha^0}$ // pick the ones with max potential
9 **else if** $\mathscr{E}_{wp(k-1)} > 0$ **then** // pre-computed $\boldsymbol{wp}$ still available
10 $\boldsymbol{wp}(k) = \boldsymbol{wp}(k-1)$
11 **else** $\boldsymbol{wp}(k) = \emptyset$ // local extremum detected at Level 0
12 **end**
13 **if** $q = CP^\ell, 1 \leq \ell \leq L$ **then**
14 compute $\mathscr{D}^\ell$ // form the computing set $\mathscr{D}^\ell$ using Eq. (4)
15 **if** $\mathscr{D}^\ell \neq \emptyset$ **then** // coarse cells with +ve potentials exist
16 $\boldsymbol{wp}(k) = I(\arg\max_{\alpha^\ell \in \mathscr{D}^\ell} \mathscr{E}_{\alpha^\ell})$
17 **else** $\boldsymbol{wp}(k) = \emptyset$ // no waypoint found at Level ℓ
18 **end**

receives the input vector $\boldsymbol{i}_{p_1}$ from the AV containing its current position λ and the newly discovered obstacle locations $\boldsymbol{ol}$, if any. Then, it moves its head to λ and updates the MAPS at all levels, i.e. $\mathscr{E}^\ell \leftarrow$ O, $\forall\ell \in \{0, \ldots L\}$, as described previously. Next, it reads the potentials $\mathscr{E}_{\mathcal{N}^0(\lambda)}$ in the local neighborhood $\mathcal{N}^0(\lambda)$ including λ. Based on these potentials, it computes the next waypoint for the AV by following Algorithm 1 (**Lines 1-12**) as follows.

Waypoint computation: First, it forms a computing set $\mathscr{D}^0 \subseteq \mathcal{N}^0(\lambda)$ (**Line 2**) that consists of *eligible* ε-cells for the next waypoint. An ε-cell is considered eligible if it is: i) *directly reachable*, i.e. it is not behind an obstacle, and ii) *unexplored*, i.e. it has positive potential.

Definition 3.2 (Directly Reachable Set). *An ε-cell is called directly reachable from λ if the line segment joining the centroids of λ and that cell is not obstructed by any obstacle cell. The set of all directly reachable cells in $\mathcal{N}^0(\lambda)$ is defined as the directly reachable set $DR(\lambda)$.*

In general, the computing set $\mathscr{D}^\ell$, $\ell \in \{0, \ldots L\}$, is defined as

$$\mathscr{D}^\ell = \begin{cases} \{\alpha^0 \in \mathcal{N}^0(\lambda) : \mathscr{E}_{\alpha^0} > 0, \alpha^0 \in DR(\lambda)\} & \text{if } \ell = 0 \\ \{\alpha^\ell \in \mathcal{N}^\ell(\lambda) : \mathscr{E}_{\alpha^\ell} > 0\} & \text{if } \ell \geq 1 \end{cases} \quad (4)$$

which means that $\mathscr{D}^0$ contains eligible ε-cells, while $\mathscr{D}^\ell$, $1 \leq \ell \leq L$, contains eligible coarse cells with positive potentials, implying that they contain unexplored ε-cells. The sets $\mathscr{D}^\ell, 1 \leq \ell \leq L$, are used in the CP^ℓ states later. Note that the direct reachability condition is only enforced at Level 0 to prevent unnecessary distortions in the back and forth trajectory. At higher levels, the algorithm uses Bug2 [23] to reach the cells behind obstacles.

Next, if the current cell λ is unexplored, i.e. $\lambda \in \mathscr{D}^0$ (**Line 3**), then it further checks if the cell above (λ^{up}) and the cell below (λ^{down}) both belong to $\mathscr{D}^0$ (**Line 4**). This condition means

that the AV is in the middle of unexplored cells both above and below λ. If this is true, then the AV should rather first move to a cell that is adjacent to a forbidden or explored cell. This step is imposed such that the trajectory is not distorted by tasking in the middle of unexplored cells, and allows for maintaining a nice back and forth motion. Thus, the waypoint candidate set is chosen as $\boldsymbol{wp} = \{\lambda^{up}, \lambda^{down}\}$ (**Line 5**). The AV picks one of these based on its turn and travel cost as shown later. After computing $\boldsymbol{wp}$, the ETM loops in state $q = CP^0$ and sends the output vector $\boldsymbol{o}_{p_3}$ commanding the AV to move to the next waypoint.

If λ is unexplored and the cells above and below are not both unexplored (**Line 6**), then the AV is well positioned for tasking. Then, $\boldsymbol{wp}$ is set equal to λ and the ETM transitions to the state $q = WT$ while sending an output vector $\boldsymbol{o}_{p_2}$ commanding the AV to task at λ. The operation of WT is described later.

If the current cell λ is not unexplored but there exist other eligible ε-cells in $\mathscr{D}^0$ (**Line 7**), then the candidate set $\boldsymbol{wp}$ is selected to consist of the ε-cells that have the highest potential in $\mathscr{D}^0$ (**Line 8**). Note that there could be more than one cell with the highest potential, if they belong to an equipotential surface. Finally, if $\mathscr{D}^0 = \emptyset$, but a pre-computed $\boldsymbol{wp}(k-1)$ is still accessible, s.t. $\mathscr{E}_{wp(k-1)} > 0$ (**Line 9**), then $\boldsymbol{wp}$ remains the same (**Line 10**). If a $\boldsymbol{wp}$ is obtained from the above steps, then the ETM stays in state CP^0 and commands the AV to move to the next waypoint, as seen in Fig. 4.

It is possible that $\boldsymbol{wp}$ contains more than one elements. In that case, the AV selects the cell that requires the least total travel and turn cost to reach it from λ. This cost is defined as follows. Let the current position of the AV be $(\lambda_x, \lambda_y) \in \tau_\lambda$. Then, for each $\mu \in \boldsymbol{wp}$, a cost $C_{\mu,\lambda}$ is defined as the total travel and turn cost needed to reach the centroid (μ_{x_c}, μ_{y_c}) of the ε-cell τ_μ as

$$C_{\mu,\lambda} \triangleq d_{\mu,\lambda} C_{Tr} + \theta_{\mu,\lambda} C_{Tu} \tag{5}$$

where C_{Tr} is the cost of traveling per unit distance; C_{Tu} is the cost of turning per degree from the heading angle θ_h; and $d_{\mu,\lambda} = ||(\mu_{x_c}, \mu_{y_c}) - (\lambda_x, \lambda_y)||_2$ and $\theta_{\mu,\lambda} = |\theta_{(\mu_{x_c}, \mu_{y_c})} - \theta_h|$ are the distance and the turning angle, respectively.

iii.) **Operation in the CP^ℓ States, $1 \le \ell \le L$:** Although the ETM usually cycles between CP^0 and WT states, it may sometimes happen that the computing set $\mathscr{D}^0 = \emptyset$ and the pre-computed waypoint is also not available since $\mathscr{E}_{wp(k-1)} \le 0$. Then $\boldsymbol{wp} = \emptyset$ (**Line 11**) and the ETM is said to be in a *local extremum*.

Escaping from the Local Extremum: As shown in Fig. 4, when $\boldsymbol{wp} = \emptyset$, the ETM transitions to the computing state CP^1, while its head moves to Level 1 on the MAPS and points at the coarse cell containing the current ε-cell λ. Here it reads the potential surface $\mathscr{E}_{\mathscr{N}^1(\lambda)}$ in the local neighborhood $\mathscr{N}^1(\lambda)$ including the coarse cell where λ falls in, and forms the computing set $\mathscr{D}^1 \subseteq \mathscr{N}^1(\lambda)$ (**Line 14**). If there exist coarse cells with positive potentials (**Line 15**), then it first picks the coarse cell with the highest potential in $\mathscr{D}^1$. Subsequently, the function $I(\cdot)$ randomly selects an unexplored ε-cell in this coarse cell and assigns it to $\boldsymbol{wp}$ (**Line 16**). However, it may happen that even at $\ell = 1$, $\nexists\ \alpha^1 \in \mathscr{N}^1(\lambda)$ with positive potential, then $\boldsymbol{wp} = \emptyset$ (**Line 17**). In that case, the ETM switches to the state CP^2 and its head moves up to Level 2 on the MAPS. This process continues until it finds the lowest level $\ell \in \{1, ...L\}$ where $\mathscr{D}^\ell \neq \emptyset$. Once the ETM finds a waypoint it switches back to the state $q = CP^0$ and sends the output vector $\boldsymbol{o}_{p_3}$ commanding the AV to move to the waypoint. If it is unable to find a coarse cell with positive potential even at the highest Level L, it implies that no coarsest cell contains any unexplored ε-cell. In that case, it switches to the finish state FN and sends the output vector $\boldsymbol{o}_{p_4}$ commanding the AV to stop its machinery since the coverage is complete.

iv.) **Operation in the WT State:** The ETM comes to the state $q = WT$ from the state CP^0 if $\boldsymbol{wp} = \lambda$. Here the ETM waits while the AV performs task at λ and reports back the task status via input vector $\boldsymbol{i}_{p_2}$. If it is '*Complete*', then the ETM updates the state of the current cell to E, i.e. *explored*, which is assigned with 0 potential according to Eq. (2). Subsequently, the potential surfaces $\mathscr{E}^\ell$, $\forall \ell \in \{1, ...L\}$, are updated according to Eq. (3). This MAPS update process is represented as $\mathscr{E}^\ell \leftarrow$ E, $\forall \ell \in \{0, ...L\}$. Then, the ETM transitions back to the computing state $q = CP^0$ and resumes searching for a new waypoint. If the task is not completed yet then the ETM loops in the state WT, while sending the output vector $\boldsymbol{o}_{p_2}$ commanding the AV to continue tasking.

In states CP^0 and WT, the MAPS are updated by assigning -1 and 0 potentials to obstacle and explored regions, respectively, while in unexplored regions, the MAPS maintain the +ve potentials defined by B, as shown in Fig. 3. Since the waypoint is mainly chosen as the ε-cell in the neighborhood with the highest +ve potential, this enables tracking the highest equipotential surfaces of B and produces the desired back and forth motion.

Remark 3.2. *The ETM uses the $flood fill$ algorithm to fill the unexplored cells inside a closed obstacle, whose boundary has been detected but interior is not detected, with the obstacle symbol O. This prevents the ETM from trying to pick undetected cells inside large obstacles.*

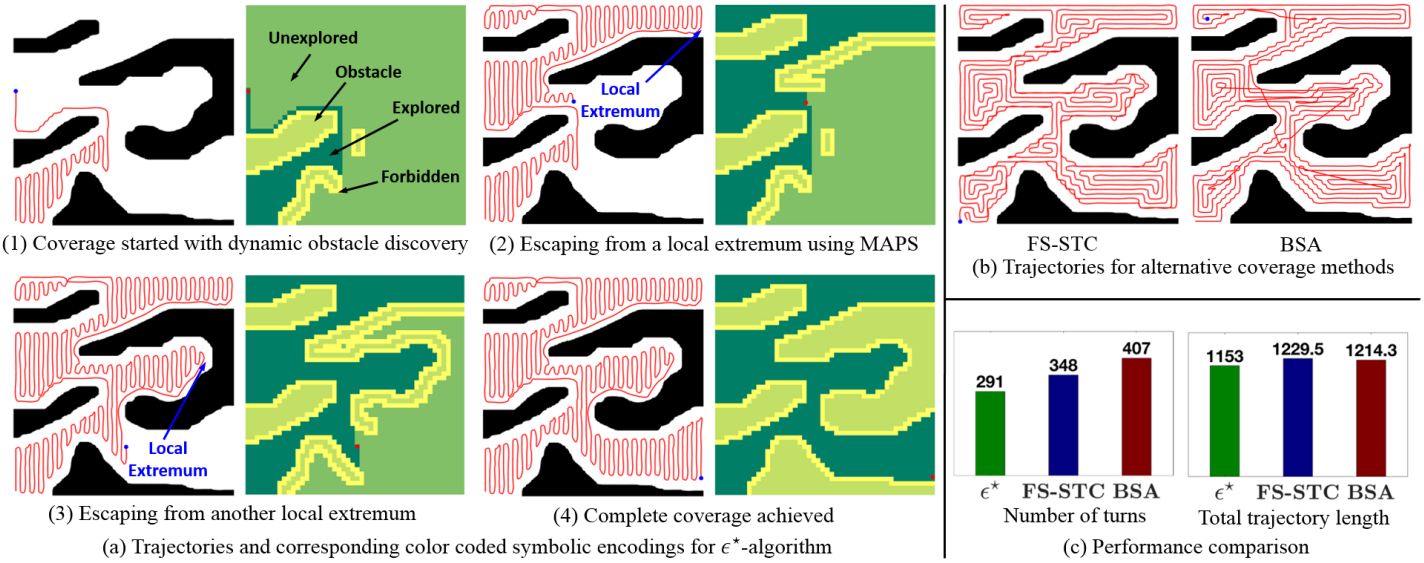
Remark 3.3. *The AV uses the Bug2 algorithm [23] to reach a waypoint which is behind obstacles. While there exist many algorithms for this purpose but Bug2 is used here for simplicity.*

Theorem 1. *The ETM halts in finite time.*

Proof. From the ETM state transition graph in Fig. 4, we see that the ETM halts when $q = FN$. Also, there are two kinds of cycles in the graph: i) between CP^0 and WT states, where the primary operation in these states is to compute $\boldsymbol{wp}$ and check task status ts, respectively; and ii) between CP^0 and $CP^\ell, \ell = 1, ...L$, states, where the primary operation in any of these states is to compute $\boldsymbol{wp}$. Since the computing set $\mathscr{D}^\ell$ used to compute $\boldsymbol{wp}$ in any CP^ℓ state, and the tasking time spent in WT state, are both finite; therefore, the total time spent in each cycle is finite.

Furthermore, during the execution of these cycles, the unexplored ε-cells with state U are constantly flipped to states O, F or E accordingly. This implies that $p^U_{\alpha^\ell}(k)$, $\forall \alpha^\ell \in \{1, ...|\mathscr{T}^\ell|\}$, $\forall \ell = 1, ...L$, decreases monotonically. Thus, $\exists$ a finite $K \in \mathbb{Z}^+$, s.t. $p^U_{\alpha^\ell}(K) = 0$. This in turn implies that $\mathscr{E}_{\alpha^\ell}(K) = 0$, as per Eq. (3). Therefore, at time K, $\mathscr{D}^\ell = \emptyset$, $\forall \ell = 1, ...L$, as per Eq. (4); hence, $\boldsymbol{wp}(K) = \emptyset$. Thus, at time K the control will exit from all the cycles and transition to the FN state where it halts. □

Remark 3.4. *Since $\mathscr{D}^L = \emptyset$ upon halting, ε-coverage is achieved.*



(1) Coverage started with dynamic obstacle discovery (2) Escaping from a local extremum using MAPS

(b) Trajectories for alternative coverage methods

(3) Escaping from another local extremum (4) Complete coverage achieved

(a) Trajectories and corresponding color coded symbolic encodings for $\epsilon^\star$-algorithm

(c) Performance comparison

Figure 5: Scenario 1: An archipelago of islands (full video available in the multimedia material).

Corollary 1. *Each allowed ε-cell is tasked only once.*

Proof. From Theorem 1, the ETM achieves ε-coverage in finite time, thus each allowed cell is tasked, its state is set to E and its potential is updated to 0. Therefore, according to Algorithm 1, this cell will never be assigned to ***wp*** and hence cannot be tasked again. Thus, every allowed cell is tasked only once. □

C. *Computational Complexity*

The $\varepsilon^\star$-algorithm has fairly low computational complexity and is real-time implementable. Suppose the local neighborhood $\mathcal{N}^0(\lambda)$ contains N ε-cells. Similarly, suppose the local neighborhood $\mathcal{N}^\ell(\lambda)$, $\forall \ell \geq 1$, contains M coarse cells. The ETM first searches in the local neighborhood at Level 0 to find navigation waypoints and only if it is stuck into an extremum, it switches to higher levels as needed. Thus, for Level 0 decisions the algorithm has a complexity of $\sim O(N)$; and even in the worst case, when the ETM has to go to the highest Level L to make a decision, the complexity is $\sim O(N+L\cdot M)$. Since the coarse cells at higher levels contain the mean potentials of all unexplored ε-cells within them, this bottom-up hierarchical approach to escape from a local extremum avoids searching for an exponentially increasing number of ε-cells; thus significantly reducing the complexity.

4. Results and Discussion

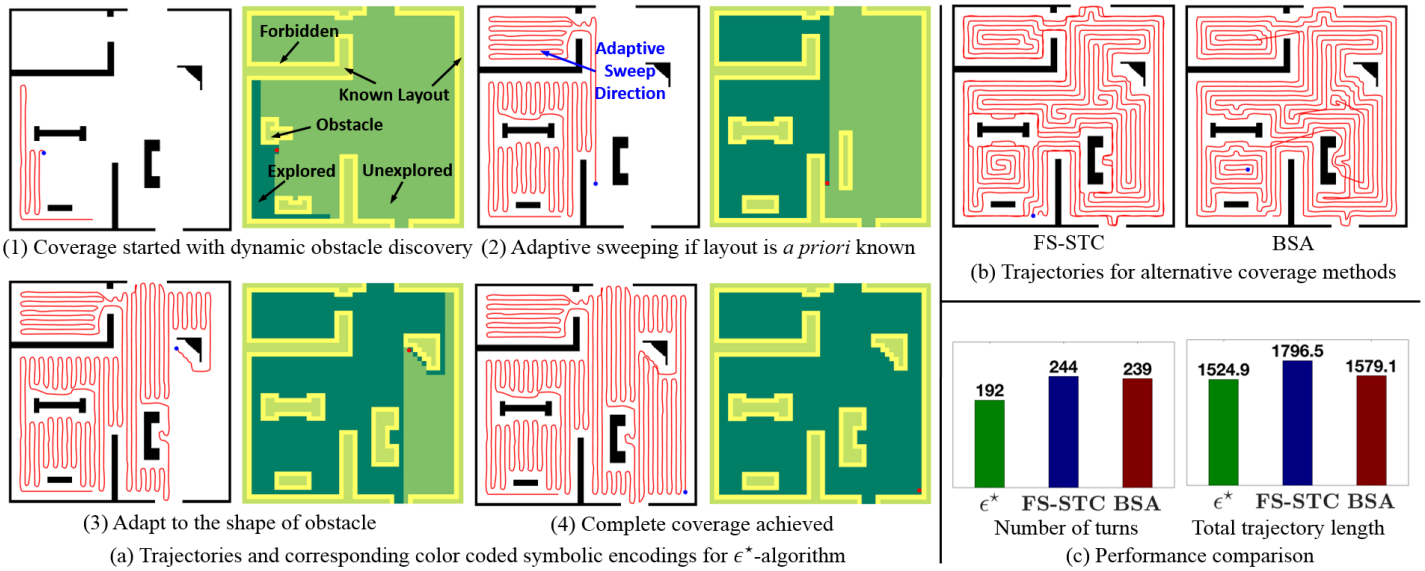
The $\varepsilon^\star$-algorithm is validated by simulations as well as experiments and its performance is compared with other algorithms.

A. *Validation on a Simulation Platform*

The first level of validation was done via simulation runs on a high-fidelity robotic platform called Player/Stage [24]. The robot server Player provides a software base whose libraries contain models of different types of robots, sensors, and actuators. On the other hand, Stage is a highly configurable robot simulator. In this paper, a Pioneer 2AT robot of dimensions $0.44m \times 0.38m \times 0.22m$ was simulated with kinematic constraints such as the top speed $0.5m/s$, maximum acceleration $0.5m/s^2$, and the minimum turn radius 0.04m. It was equipped with a laser sensor with a detection range of 4m, having 16 beams located around the robot to detect obstacles. A 3.40 GHZ CPU computer with 16GB RAM was used for simulations. Several complex scenarios of $50m \times 50m$ search areas with different obstacle layouts were drawn and partitioned into a 50×50 tiling structure consisting of $1m \times 1m$ ε-cells. This resulted in an MST with $L = 5$. For computation, the neighborhood was chosen to contain 7×7 cells at the lowest level and 3×3 cells at higher levels. The simulations were run 8 times faster than the real-time speed.

Figs. 5 and 6 show the results of $\varepsilon^\star$-algorithm for two different scenarios, respectively. The results are compared with two other online algorithms: FS-STC and BSA. Three metrics are used for performance evaluation as follows: i) *coverage ratio* $r_c = \frac{\bigcup_k \tau(k)}{\mathcal{R}(\mathcal{T}^a)}$, ii) *number of turns*, and iii) *trajectory length*. First, in Figs. 5a and 6a we show snapshots of the trajectory generated by $\varepsilon^\star$ at four different instants and the corresponding symbolic encodings discovered *in situ*. These snapshots show several instances when the AV gets stuck into a local extremum, surrounded by either obstacle or explored cells in the local neighborhood. Each time the AV successfully comes out of the local extremum using higher levels of MAPS and finally achieves complete coverage. Since the AV sees only the periphery of large obstacles, the *floodfill* algorithm is called by the ETM at regular intervals to fill the interiors of all closed obstacles.

Fig. 6 tested another condition, if *a priori* knowledge was available, it could be used to adapt the sweep direction of the AV. In this scenario it was assumed that the layouts of all rooms are known but the inside obstacles are unknown. Then, the field B was designed in a manner such that the AV sweeps the top left room horizontally while the other two rooms vertically, as seen in Fig. 6a.2. This enables reducing the total number of turns. Finally, Figs. 5b and 6b show the trajectories of FS-STC and BSA algorithms, which are spiral in contrast to the back and forth trajectories generated by $\varepsilon^\star$. Figs. 5c and 6c compare the number of turns and total trajectory length. Clearly, the $\varepsilon^\star$-algorithm produces significantly less number of turns and shorter trajectory lengths. Furthermore, Table I provides the qualitative comparison



(1) Coverage started with dynamic obstacle discovery (2) Adaptive sweeping if layout is *a priori* known

(b) Trajectories for alternative coverage methods

(3) Adapt to the shape of obstacle (4) Complete coverage achieved

(a) Trajectories and corresponding color coded symbolic encodings for $\epsilon^\star$-algorithm

(c) Performance comparison

Figure 6: Scenario 2: A house with several rooms and structures (full video available in the multimedia material).

Table I: Comparison of Key Features of $\varepsilon^\star$ with Other Algorithms

	$\varepsilon^\star$	FS-STC [16]	BSA [17]	Cellular Decomposition [13]
Environment	any	any	any	non-rectilinear
Path Pattern	back and forth with adjustable sweep direction in known areas	spiral	spiral	back and forth
Approach	uses ETM as a Supervisor	circumvents the spanning tree constructed	uses spiral paths to fill areas and backtracking to escape spiral endings	relies on critical point detection for Morse decomposition. Then uses Reeb graph and cycle algorithm

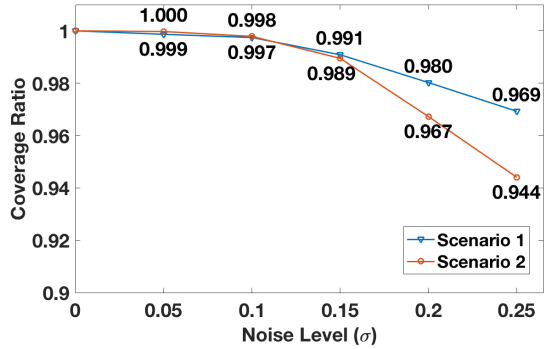


Figure 7: Coverage ratio vs. noise.

between the features of $\varepsilon^\star$ and other online algorithms.

The average computation time to update ***wp*** in state CP^0 was ~ 0.577 milliseconds, while it was ~ 0.437 milliseconds in any $CP^\ell, 1 \le \ell \le L$ state; hence it is suitable for real-time applications.

• **Performance in Presence of Uncertainties:** For uncertainty analysis, noise was injected into the measurements of range detector (laser), the heading angle (compass), and the localization system. A laser sensor typically admits an error of 1% of its operation range. Similarly, a modestly priced compass can provide heading information as accurate as 1^o [25]. The above errors

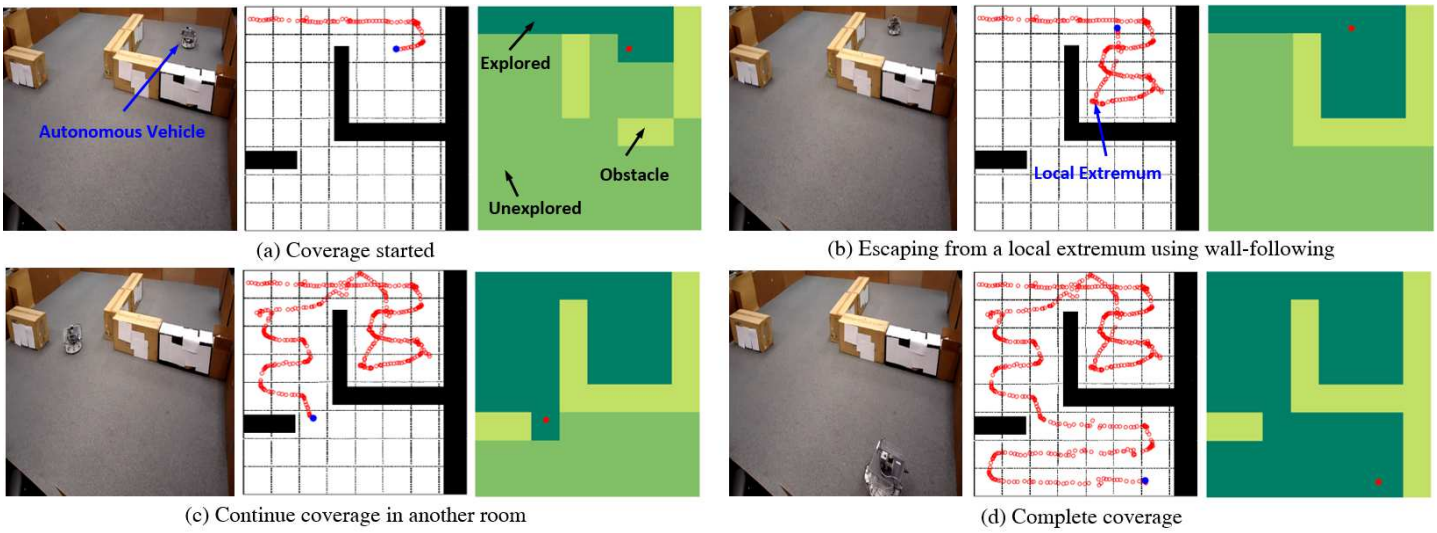
Table II: Specifications of On-board Sensing Systems

	Localization	Laser	Ultrasonic
Model	StarGazer	URG-04LX	XL-MaxSonar-EZ
Range	–	$0.02m \sim 5.6m, 240^o$	$0.2m \sim 7.65m$
Resolution	$1cm, 1^o$	$1mm, 0.36^o$	$1cm$
Accuracy	$2cm, 1^o$	$\pm 1\%$ of Measurement	–

were simulated with Additive White Gaussian Noise (AWGN) with standard deviations of $\sigma_{laser} = 1.5cm$ and $\sigma_{compass} = 0.5^o$, respectively. The Hagisonic StarGaze indoor localization system provides a precision of $2cm$ [26], while the GPS system using Real-Time Kinematic (RTK) can achieve an accuracy of $0.05 \sim 0.5m$ [25]. Thus, the uncertainty due to localization system is studied using AWGN with standard deviation ranging from $\sigma = 0.05m$ to $0.25m$. Fig. 7 shows the average coverage ratio vs. noise over ten Monte Carlo runs for the two scenarios.

B. *Validation by Real Experiments*

The $\varepsilon^\star$-algorithm was further validated by real-experiments in a laboratory setting. The laboratory area was partitioned into a 8×8 tiling structure with each ε-cell of dimension 0.61m. This resulted in an MST with $L = 2$. For computation, a neighborhood of size 3×3 was chosen at all levels. An iRobot Create was utilized that was equipped with the Hagisonic StarGazer system [26] for indoor localization, the HOKUYO URG-04LX scanning laser with $\sim 2m$ detection range, and 10 ultrasonic sensors evenly placed around the robot for collision avoidance. Table II provides the specifications of these sensing systems. A hardware-in-the-loop setup was established, where the robot carries an on-board laptop that runs the Player, which acts as the server to collect the real-time sensor measurements. The robot stops every few seconds to collect data. The client computer runs the ETM which incrementally builds the map by real-time obstacle discovery. The server and the client communicate through a wireless connection for real-time control and navigation. Fig. 8 shows the results of a real experiment, where the robot successfully evacuated from a local extremum and explored different rooms to achieve ε-coverage, thus revealing the effectiveness of the $\varepsilon^\star$-algorithm.



(a) Coverage started (b) Escaping from a local extremum using wall-following

(c) Continue coverage in another room (d) Complete coverage

Figure 8: Real experiment in a laboratory environment (full video available in the multimedia material).

5. Conclusions and Future Work

The paper presents an algorithm, called $\varepsilon^\star$, for online coverage of unknown environments. The algorithm utilizes the concept of an *Exploratory Turing Machine* (ETM) which supervises the AV with adaptive navigation decisions. It is shown that the $\varepsilon^\star$-algorithm is computationally efficient, produces the desired back and forth motion with adjustable sweep direction, does not rely on the critical point detection concept, and guarantees complete coverage. In comparison with other online algorithms, $\varepsilon^\star$ produces less number of turns and shorter trajectory lengths. The algorithm is validated via: i) high-fidelity simulations including sensor uncertainties, and ii) real experiments in laboratory.

Future research areas include: i) extension to multi-robot coverage, ii) integration of SLAM with coverage control.

References

[1] V. Lumelsky, S. Mukhopadhyay, and K. Sun, "Dynamic path planning in sensor-based terrain acquisition," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 4, pp. 462–472, 1990.

[2] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain-covering algorithm for an AUV," *Journal of Autonomous Robots*, vol. 3, pp. 91–119, 1996.

[3] J. Palacin, J. A. Salse, I. Valganon, and X. Clua, "Building a mobile robot for a floor-cleaning operation in domestic environments," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 5, pp. 1418–1424, 2004.

[4] M. Weiss-Cohen, I. Sirotin, and E. Rave, "Lawn mowing system for known areas," in *Proceedings of International Conference on Computational Intelligence for Modelling Control and Automation*, Vienna, 2008, pp. 539–544.

[5] E. Krotkov and R. Hoffman, "Terrain mapping for a walking planetary rover," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 6, pp. 728 – 739, 1994.

[6] J. Song, S. Gupta, J. Hare, and S. Zhou, "Adaptive cleaning of oil spills by autonomous vehicles under partial information," in *OCEANS'13 MTS/IEEE*, San Diego, CA, September 2013, pp. 1–5.

[7] P. F. Santana, J. Barata, and L. Correia, "Sustainable robots for humanitarian demining," *International Journal of Advanced Robotic Systems*, vol. 4, pp. 207–218, 2007.

[8] K. Mukherjee, S. Gupta, A. Ray, and S. Phoha, "Symbolic analysis of sonar data for underwater target detection," *IEEE Journal of Oceanic Engineering*, vol. 36, no. 2, pp. 219–230, 2011.

[9] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, pp. 90–98, 1986.

[10] X. Jin, S. Gupta, J. M. Luff, and A. Ray, "Multiresolution navigation of mobile robots with complete coverage of unknown and complex environments," in *Proceedings of the American Control Conference*, Montreal, Canada, 2012, pp. 4867–4872.

[11] E. Galceran and M. Carreras, "A survey on coverage path planning for robotics," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1258–1276, 2013.

[12] A. Zelinsky, R. Jarvis, J. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *Proceedings of the International Conferenceon Advanced Robotics*, Tokyo, Japan, 1993, pp. 533–538.

[13] E. Acar and H. Choset, "Sensor-based coverage of unknown environments: Incremental construction of Morse decompositions," *International Journal of Robotics Research*, vol. 21, no. 4, pp. 345–366, 2002.

[14] S. Koenig, B. Szymanski, and Y. Liu, "Efficient and inefficient ant coverage methods," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 41–76, 2001.

[15] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continous areas by a mobile robot," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 77–98, 2001.

[16] ——, "Competitive on-line coverage of grid environments by a mobile robot," *Computational Geometry*, vol. 24, no. 3, pp. 197–224, 2003.

[17] E. Gonzalez, O. Alvarez, Y. Diaz, C. Parra, and C. Bustacara, "Bsa: a complete coverage algorithm," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2005, pp. 2040–2044.

[18] E. Garcia and P. G. de Santos, "Mobile-robot navigation with complete coverage of unstructured environments," *Robotics and Autonomous Systems*, vol. 46, pp. 195–204, 2004.

[19] A. Kim and R. Eustice, "Active visual slam for robotic area coverage: Theory and experiment," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 457–475, 2015.

[20] P. Wegner and E. Eberbach, "New models of computation," *The Computer Journal*, vol. 47, no. 1, pp. 4–9, 2004.

[21] D. Goldin, S. Smolka, P. Attie, and E. Sonderegger, "Turing machines, transition systems, and interaction," *The Computer Journal*, vol. 194, no. 2, pp. 101–128, 2004.

[22] S. Gupta, A. Ray, and S. Phoha, "Generalized ising model for dynamic adaptation in autonomous systems," *European Physics Letters*, vol. 87, p. 10009, 2009.

[23] J. Ng and T. Braunl, "Performance comparison of bug navigation algorithms," *Journal of Intelligent and Robotic Systems*, vol. 50, pp. 73–84, 2007.

[24] B. Gerkey, R. Vaughan, and A. Howard, "The Player/Stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics, Coimbra, Portugal*, 2003, pp. 317–323.

[25] L. Paull, S. Saeedi, M. Seto, and H. Li, "Auv navigation and localization: A review," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 1, pp. 131–149, 2014.

[26] J. L. Fernández, C. Watkins, D. P. Losada, and M. D. Medina, "Evaluating different landmark positioning systems within the ride architecture," *Journal of Physical Agents*, vol. 7, no. 1, pp. 3–11, 2013.