4-5-2019

# Coverage and Time-optimal Motion Planning for Autonomous Vehicles

Junnan Song
*University of Connecticut - Storrs*, junnan.song@uconn.edu

# Coverage and Time-optimal Motion Planning for Autonomous Vehicles

Junnan Song, Ph.D.

University of Connecticut, 2019

## ABSTRACT

Autonomous vehicles are rapidly advancing with a variety of applications, such as area surveillance, environment mapping, and intelligent transportation. These applications require coverage and/or time-optimal motion planning, where the major challenges include uncertainties in the environment, motion constraints of vehicles, limited energy resources and potential failures. While dealing with these challenges in various capacities, this dissertation addresses three fundamental motion planning problems: (1) single-robot complete coverage in unknown environment, (2) multi-robot resilient and efficient coverage in unknown environment, and (3) time-optimal risk-aware motion planning for curvature-constrained vehicles.

First, the $\varepsilon^\star$ algorithm is developed for online coverage path planning in unknown environment using a single autonomous vehicle. It is computationally efficient, and can generate the desired back-and-forth path with less turns and overlappings. Moreover, the algorithm prevents the local extrema problem, thus can guarantee complete coverage. Second, the CARE algorithm is developed which extends $\varepsilon^\star$ for multi-robot resilient and efficient coverage in unknown environment. In case of vehicle failures, it guarantees complete coverage via proactively filling coverage gaps through dynamic task reallocations for other

vehicles, hence provides resilience. Moreover, CARE reallocates idling vehicles to support others in their tasks, hence reduces coverage time and improves team efficiency. Finally, the T$^\star$ algorithm is developed to address the time-optimal risk-aware motion planning problem for curvature-constrained variable-speed vehicles. To the best of our knowledge, this problem has not been solved in the presence of obstacles. We present a novel risk function based on the concept of collision time, and seamlessly integrate it with the time cost for optimization. Further, an adaptive state pruning technique is developed that can significantly reduce the computation time, while maintaining the solution quality and the completeness of this algorithm.

The above-mentioned algorithms have been validated via simulations in complex scenarios and/or real experiments, and the results have shown clear advantages over existing popular approaches.

# Coverage and Time-optimal Motion Planning for Autonomous Vehicles

Junnan Song

M.S., University of Connecticut, USA, 2016

M.E., Northeastern University, China, 2012

B.E., Northeastern University, China, 2010

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2019

## APPROVAL PAGE

Doctor of Philosophy Dissertation

# Coverage and Time-optimal Motion Planning for Autonomous Vehicles

Presented by

Junnan Song

Major Advisor_____
Prof. Shalabh Gupta

Associate Advisor_____
Prof. Peter B. Luh

Associate Advisor_____
Prof. Shengli Zhou

Associate Advisor_____
Dr. Thomas A. Wettergren

University of Connecticut

2019

# ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my major advisor, Prof. Shalabh Gupta for his support and motivation during my Ph.D. study. His knowledge and expertise have helped me develop the research skills that I have today. It was his encouragement during my difficult times that have eventually helped me complete my Ph.D., and I can never thank him enough.

Moreover, I would like to thank Dr. Thomas A. Wettergren, who was kind enough to give timely feedback on my research. His valuable comments are greatly appreciated. I would also like to thank my advisory committee, Prof. Peter B. Luh and Prof. Shengli Zhou, for their thoughtful comments during my proposal and dissertation defense which have helped to enhance the presentation of my works.

I thank my lab mates for the inspiring discussions, my friends for the continuous support, and my girlfriend for the care and encouragement. Without their help, I may not become who I am today.

Finally, I would like to thank my family for all of their support during these years. My parents, Hui Zhang and Bo Song, have given me unreserved love and tremendous help, without which I may not go abroad to pursue my Ph.D. degree at the University of Connecticut. Also, I want to express my sincere gratitude to my departed grandma who has played a very important role in my youth.

# Contents

# List of Figures

# List of Tables

# Nomenclature

**List of Symbols**

| | |
|---|---|
| $\bar{\mathcal{P}}$ | Set of non-players |
| $\bar{B}_{\alpha^\ell}$ | Mean time-invariant exogenous potential of cell $\tau_{\alpha^\ell}$ |
| $\chi$ | Learning parameter in the Max-Logit algorithm |
| $\Delta d$ | Sampling interval of distance |
| $\delta$ | State transition function in the ETM |
| $\ell$ | Index number |
| $\emptyset$ | Empty set |
| $\varepsilon$ | Side length of each $\varepsilon$-cell |
| $\eta$ | Threshold of time to identify robots that are near finishing their tasks |
| $\Gamma$ | Set of all collision-free paths between the start state and the goal state |
| $\gamma$ | A collision-free path between the start state and the goal state |
| $\gamma^\star$ | The time-optimal risk-aware path from the start state to the goal state |
| $\Gamma^c$ | Set of candidate paths between a pair of states |
| $\Gamma^c_{\mathbf{BB}}$ | Subset of $\Gamma^c$ containing candidate paths that start and end both with bang arcs |
| $\Gamma^c_{\mathbf{BC}}$ | Subset of $\Gamma^c$ containing candidate paths that start with a bang arc and end with a cornering arc |
| $\Gamma^c_{\mathbf{CB}}$ | Subset of $\Gamma^c$ containing candidate paths that start with a cornering arc and end with a bang arc |

| | |
|---|---|
| $\Gamma^c_{\mathbf{CC}}$ | Subset of $\Gamma^c$ containing candidate paths that start and end both with cornering arcs |
| $\gamma_{i,i+1}$ | A collision-free path between states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$ |
| $\hat{\mathbf{p}}_\ell$ | Sample state with index $\ell$ |
| $\kappa$ | Curvature |
| $\kappa_1$ | Neighborhood size of no-idling games |
| $\kappa_2$ | Neighborhood size of resilience games |
| $\lambda$ | Index of $\varepsilon$-cell containing the autonomous vehicle |
| $\lambda^{down}$ | Index of $\varepsilon$-cell below current cell |
| $\lambda^{up}$ | Index of $\varepsilon$-cell above current cell |
| $\lambda_r$ | Expected number of targets in task $r$ |
| $\mathbb{N}$ | Set of natural numbers |
| $\mathbb{R}$ | Set of real numbers |
| $\mathbf{c}(s)$ | Control at point $s$ on path $\gamma$ |
| $\mathbf{x_r}$ | Random variable denoting the total number of targets in task $r$ |
| $\mathscr{D}^\ell$ | Computing set at Level $\ell$ |
| $\mathscr{E}^\ell$ | Potential surface on Level $\ell$ of MAPS |
| $\mathscr{E}_{\alpha^\ell}$ | Potential assigned to cell $\alpha^\ell$ on Level $\ell$ of MAPS |
| $\mathscr{N}^\ell(\lambda)$ | Set of cells in the neighborhood of cell $\tau_\lambda$ on Level $\ell$ |
| $\mathscr{P}$ | Set of all valid state sequences connecting the start state and the goal state |
| $\mathscr{R}$ | Target search area |
| $\mathscr{R}^a$ | Total area spanned by the allowed cells |
| $\mathscr{R}_r$ | Task with index $r$ |
| $\mathscr{T}$ | Tiling of the search area |
| $\mathscr{T}^\ell$ | Tiling of the search area at Level $\ell$ |

| | |
|---|---|
| $\mathscr{T}^a$ | Set of allowed cells in tiling $\mathscr{T}$ |
| $\mathscr{T}^f$ | Set of forbidden cells in tiling $\mathscr{T}$ |
| $\mathscr{T}^o$ | Set of obstacle cells in tiling $\mathscr{T}$ |
| $\mathcal{A}_i$ | Action set of player $\mathcal{P}_i$ |
| $\mathcal{A}_{-i}$ | Set of joint actions of players other than $\mathcal{P}_i$ |
| $\mathcal{A}_{\mathcal{P}}$ | Set of joint actions of all players |
| $\mathcal{E}$ | Set of discrete events in the DES |
| $\mathcal{L}$ | Number of heading angles associated with each cell in the configuration space $\mathcal{Q}$ |
| $\mathcal{N}_{\kappa}^{v_\ell}$ | $\kappa$ nearest neighbors of robot $v_\ell$ |
| $\mathcal{O}$ | Set of center positions of all obstacle-free cells in the configuration space $\mathcal{Q}$ |
| $\mathcal{P}$ | Set of game players |
| $\mathcal{P}_i$ | Player with index $i$ |
| $\mathcal{Q}$ | The configuration space in $\mathrm{T}^\star$ |
| $\mathcal{R}$ | Risk cost |
| $\mathcal{S}$ | Set of symbolic states that can be encoded to $\varepsilon$-cells |
| $\mathcal{T}$ | Time cost |
| $\mathcal{U}_i$ | Utility function of player $\mathcal{P}_i$ |
| $\mathcal{V}$ | Set of vehicle speeds in the configuration space $\mathcal{Q}$ |
| $\omega$ | Robot tasking speed |
| $\Phi$ | Potential function for the whole team |
| $\phi$ | Potential function for all players |
| $\Pi$ | Set of all possible configurations of potentials on MAPS |
| $\psi$ | Threshold of time to identify incomplete tasks with sufficient work left |
| $\rho_0$ | Curvature of the growth part in robot battery model |

| | |
|---|---|
| $\rho_1$ | Inflection point in robot battery model |
| $\tau_\alpha$ | Cell with index $\alpha$ in the tiling |
| $\tau_{\alpha^\ell}$ | Cell with index $\alpha^\ell$ at Level $\ell$ of MST |
| $\tau_\ell(k)$ | The $\varepsilon$-cell visited and explored by robot $v_\ell$ at time $k$ |
| $\mathbf{B}$ | Bang arc |
| $\mathbf{C}$ | Cornering arc |
| $\mathbf{k}$ | Risk weight |
| $\mathbf{p}$ | Vehicle state |
| $\mathbf{P}^\star$ | Optimal state sequence connecting the start state and the goal state |
| $\mathbf{P}^m$ | The $m$-th state sequence that connects the start state and the goal state |
| $\mathbf{p}_i^m$ | The $i$-th state in state sequence $\mathbf{P}^m$ |
| $\mathbf{p}_{goal}$ | The goal state |
| $\mathbf{p}_{start}$ | The start state |
| $\mathbf{R}$ | Turning radius using maximum turn rate $u_{\max}$ and maximum speed $v_{\max}$ |
| $\mathbf{r}$ | Turning radius using maximum turn rate $u_{\max}$ and minimum speed $v_{\min}$ |
| $\mathbf{S}$ | Straight line segment |
| $\boldsymbol{i}_p$ | Input parameter vector from $I_p$ in the ETM |
| $\boldsymbol{ol}$ | Set of indexes of obstacle cells |
| $\boldsymbol{o}_p$ | Output parameter vector from $O_p$ in the ETM |
| $\boldsymbol{wp}$ | Set of waypoints for the autonomous vehicle |
| $\Theta$ | Set of heading angles in the configuration space $\mathcal{Q}$ |
| $\theta$ | Vehicle heading angle |
| $\theta_{\mu,\lambda}$ | Turning angle from vehicle heading to the centroid of cell $\tau_\mu$ |
| $\tilde{\mathcal{A}}$ | Action set of all players |

| | |
|---|---|
| $\tilde{J}$ | Step-wise cost between a pair of states |
| $\tilde{w}_r$ | Total worth of task $r$ |
| $\Xi$ | Set of all potential values available on each cell in MAPS |
| $\Xi^{\ell}$ | Set of potential values available on each cell at Level $\ell$ of MAPS |
| $\Xi^{\ell}_{\max}$ | Maximum potential value available on each cell at Level $\ell$ of MAPS |
| $\Xi^{\ell}_{\min}$ | Minimum potential value available on each cell at Level $\ell$ of MAPS |
| $\zeta$ | Threshold of angle for heading-based pruning |
| $\{\bar{\mathcal{P}}\}_r$ | Set of non-players assigned to task $r$ |
| $\{\mathcal{P}\}_r$ | Set of players assigned to task $r$ |
| $\{V\}_r$ | Set of robots assigned to task $r$ |
| $a_i$ | Action of player $\mathcal{P}_i$ |
| $a_{-i}$ | Joint action of players other than $\mathcal{P}_i$ |
| $a_{\mathcal{P}}$ | Joint action of all players |
| $a^{\star}_{\mathcal{P}}$ | Pure Nash Equilibrium for all players |
| $B$ | Time-invariant exogenous potential field |
| $B_{\alpha^0}$ | Time-invariant exogenous potential associated to cell $\tau_{\alpha^0}$ |
| $B_{\max}$ | Maximum value of time-invariant exogenous potential in $B$ |
| $C_{\mu,\lambda}$ | Total cost for the vehicle to move to cell $\tau_{\mu}$ |
| $C_{Tr}$ | Cost of traveling per unit distance |
| $C_{Tu}$ | Cost of turning per degree |
| $cd$ | Operational command to the autonomous vehicle |
| $cm$ | Tasking status to indicate task is complete |
| $CP^0$ | Compute state associated with Level 0 in the ETM |
| $CP^{\ell}$ | Compute state associated with Level $\ell$ in the ETM |

| | |
|---|---|
| $CP^L$ | Compute state associated with Level $L$ in the ETM |
| $CR$ | Coverage ratio |
| $CT$ | Coverage time |
| $d_\ell$ | Collision distance of sample state $\hat{\mathbf{p}}_\ell$ |
| $d_{\mu,\lambda}$ | Distance between the vehicle and the centroid of cell $\tau_\mu$ |
| $DR(\lambda)$ | Directly reachable set from current cell $\tau_\lambda$ |
| $f$ | Total cost for an intermediate state during searching for $\mathbf{P}^\star$ |
| $FL$ | Failed state in the DES |
| $FN$ | Finish state in the ETM |
| $g$ | Step-wise cost between two consecutive states during searching for $\mathbf{P}^\star$ |
| $G_P$ | Gain of players |
| $G_T$ | Gain of team |
| $H$ | The discrete event supervisor (DES) |
| $h$ | Heuristic cost for an intermediate state during searching for $\mathbf{P}^\star$ |
| $I_p$ | Set of input parameters in the ETM |
| $ic$ | Tasking status to indicate task is incomplete |
| $ID$ | Idle state in the DES |
| $id$ | Operational command to set the autonomous vehicle to idle |
| $J(\gamma)$ | Total cost of path $\gamma$ |
| $J(\mathbf{P}^m)$ | Total cost of the state sequence $\mathbf{P}^m$ |
| $k$ | Time stamp |
| $L$ | Highest level number of MAPS |
| $M$ | Total number of tasks for coverage |
| $mv$ | Operational command to set the autonomous vehicle to move |

| | |
|---|---|
| $N$ | Total number of robots in the team |
| $n_U$ | Number of unexplored $\varepsilon$-cells in a task |
| $n_{\max}$ | Maximum number of robots allowed to simultaneously work on the same task |
| $NG$ | No-idling Game state in the DES |
| $NoTF$ | Number of targets found |
| $O_p$ | Set of output parameters in the ETM |
| $p_r(\mathcal{P}_i)$ | Success probability for player $\mathcal{P}_i$ to finish task $r$ |
| $p^U_{\alpha^\ell}$ | Probability of unexplored $\varepsilon$-cells in coarse cell $\tau_{\alpha^\ell}$ |
| $Q$ | Set of states in the ETM |
| $q$ | Head state in the ETM |
| $q_0$ | Initial head state in the ETM |
| $r$ | Task index |
| $r_c$ | Allocation function to determine the current tasks of robots |
| $R_s$ | Sensing radius of the range detector |
| $r_t$ | Tasking radius of the tasking sensor |
| $R_{v_\ell}$ | Battery reliability of robot $v_\ell$ |
| $RG$ | Resilience Game state in the DES |
| $risk$ | Risk function |
| $RR$ | Average remaining reliability of healthy robots |
| $s_\alpha$ | Symbolic state of cell $\tau_\alpha$ |
| $SP$ | Stop state in the DES |
| $sp$ | Operational command to set the autonomous vehicle to stop |
| $ST$ | Start state in the ETM and the DES |
| $t^\star$ | Threshold of collision time for a state to be safe for the vehicle |

| | |
|---|---|
| $t_\ell$ | Collision time of sample state $\hat{\mathbf{p}}_\ell$ |
| $t_c$ | Remaining time to complete a task by its assigned robots |
| $t_k$ | Total tasking time of a player since the start of operation till a game is initiated |
| $t_r$ | Estimated time to finish remaining work of task $r$ |
| $t_{tr}$ | Traveling time to reach a task |
| $tk$ | Operational command to set the autonomous vehicle to task |
| $ToTD$ | Time of target discovery |
| $ts$ | Tasking status |
| $u$ | Vehicle turn rate |
| $u_{\max}$ | Vehicle maximum turn rate |
| $V$ | The set of robots |
| $v$ | Vehicle speed |
| $v_\ell$ | Robot with index $\ell$ in the robot team $V$ |
| $v_f$ | Failed robot in the robot team $V$ |
| $v_{\max}$ | Vehicle maximum speed |
| $v_{\min}$ | Vehicle minimum speed |
| $v_{id}$ | Idling robot in the robot team $V$ |
| $w_r$ | Available worth of task $r$ to the players |
| $WK$ | Working state in the DES |
| $WT$ | Wait state in the ETM |
| $X$ | Set of states in the DES |
| $x$ | Vehicle position on $x$-axis |
| $x_0$ | Initial state in the DES |
| $X_m$ | Marked state in the DES |

| | |
|---|---|
| $y$ | Vehicle position on $y$-axis |
| $Z$ | Total number of computation cycles in the Max-Logit Algorithm |
| A | Symbolic state for accessible (i.e., obstacle-free) cells |
| E | Symbolic state for explored cells |
| F | Symbolic state for forbidden cells |
| O | Symbolic state for obstacle cells |
| U | Symbolic state for unexplored cells |
| L | Left turn |
| R | Right turn |
| S | Moving straight |

# Chapter 1

# Introduction

## 1.1 Background and Motivation

With recent advancements in sensor technologies, mobile vehicles and communication systems, vehicle autonomy has become an area of rising interest. Autonomous vehicles have been widely used in area exploration [1], floor cleaning [2], lawn mowing [3][4], terrain-map generation [5][6][7][8], oil spill cleaning [9][10], humanitarian de-mining [11][12], and intelligent transportation [13]. These applications either involve the *Coverage Path Planning* (CPP) problem, which requires to plan collision-free paths that can guide one or multiple autonomous vehicles to completely cover a target search area; or the *Point-to-Point* (PTP) planning problem, which requires to plan a collision-free, feasible path for an autonomous vehicle to move from a start state to a goal state, while optimizing certain metrics such as minimum time, shortest length, minimum energy, and/or maximum safety.

The major difficulties in the above-mentioned two problems include unknown environment, vehicle motion constraints, limited energy resources and potential failures. A brief

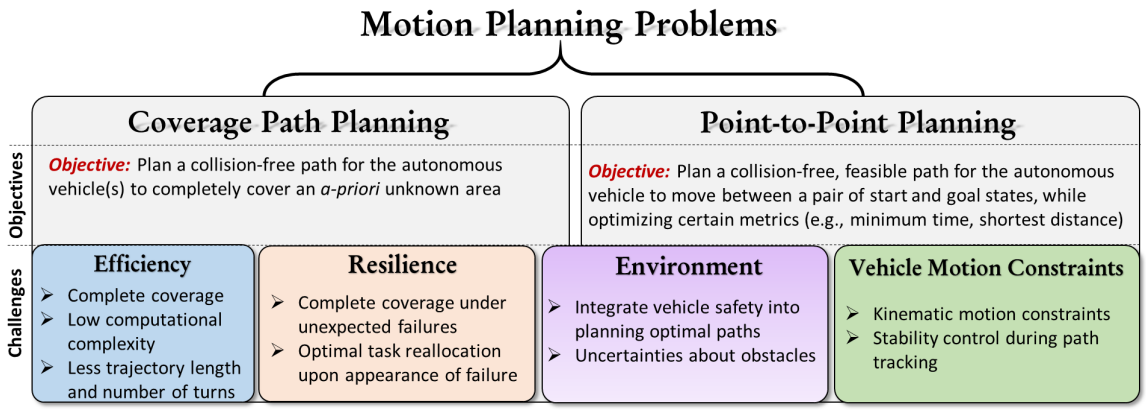



**Figure 1.1:** The objectives and challenges in motion planning problems

overview of these motion planning problems and their challenges is summarized in Fig. 1.1. While dealing with these challenges in various capacities, this dissertation addresses three fundamental motion planning problems as follows.

The first addressed problem is the single-robot CPP problem in unknown environment. Typically, the target area for coverage is either partially known or completely unknown, that is, the information about the exact geometrical shapes and locations of obstacles and area boundaries, may be incomplete or unavailable. Thus, it is imperative to rely on real-time sensor measurements to dynamically unfold the environment via discovering obstacles along the path, and then generate the coverage path *in situ*.

Existing online single-robot CPP approaches either rely on the cellular decomposition of the search area [14][15][16] to generate back-and-forth coverage paths, or they produce spiral paths to fill areas [17][18]. The major drawsbacks in these methods are that, spiral paths are usually embedded with a large number of turns which is undesirable. On the other hand, it must identify the so-called *critical points* on obstacles for accurate cellular decomposition of the search area, and this can be difficult in complex environment; also, it may not work for rectilinear environment [19] where critical points do not exist.

The desired online CPP algorithm is expected to operate in the presence of obstacles of arbitrary shapes, and it should generate the back-and-forth coverage paths, while not relying on critical point detection on obstacles. Also, it should be computationally efficient hence suitable for online applications. Moreover, in order to guarantee complete coverage, it should prevent the local extrema problem. In particular, in the context of CPP, a local extremum describes a situation where the autonomous vehicle cannot find the next waypoint hence gets stuck, even though there still exist unexplored regions.

The second problem addressed in this dissertation is the *Multi-robot Coverage Path Planning* (MCPP) problem with a focus on system resilience and efficiency. Although many works have been presented for coverage using a single autonomous vehicle, only a limited body has focused on MCPP. Since autonomous vehicles typically operate in uncertain environment, they are prone to different failures, such as sensor or actuator malfunctions, mechanical defects and loss of power [20]. The consequences of these failures include coverage gaps, loss of critical data, performance degradation (e.g., missed detection of targets), prolonged operation time, and in extreme cases overall mission failure. It is therefore critical that the whole team is resilient to individual failures, in the sense that it can still ensure complete coverage even in the presence of a few robot failures [21].

Existing online MCPP methods typically partition the search area into sub-regions, and then extend certain single-robot CPP algorithm to search in parallel [22][23][24]. However, if some robot fails, the resulting coverage gaps can only be passively passed on to the rest team members. This means that the critical gaps cannot be immediately filled, but have to wait until some other robots finish their tasks.

Moreover, due to unknown environment, it is very likely that all robots may not finish at the same time. This results in the idling problem, where the robots that finish earlier will become idle while others are suffering from longer operation time with constantly depleting

batteries. Therefore, in order to improve the team efficiency, it should dynamically identify and then proactively reallocate these idling robots.

In addition, regarding the control architecture, *Multi-robot Systems* (MRS) are controlled either in a centralized [25] or decentralized [26] manner. In a centralized architecture, either an extra host or a leader robot is assigned to maintain the global information about the whole team. Although such structure can provide the global optimal decision for the whole team, they may suffer from the scalability problem due to limited computational resources and communication restrictions. In contrast, a decentralized MRS treats each robot as an independent unit, which makes decisions based on its local information. This promotes system scalabilty, thus suitable for the MCPP problem. However, due to lack of complete information, it is critical that the local decisions of each robot can also benefit the whole team in terms of resilience and efficiency.

The third problem addressed in this dissertation involves the PTP planning for curvature-constrained vehicles, and the objective is to find the optimal trajectory with jointly minimized time and risk costs. The PTP planning problem has been researched for decades, and literature has abundant methods to find the shortest path in known environment [27][28][29], or in unknown environment [30][31]. However, due to kinematic constraints like bounded curvature and bounded turn rate [32], many vehicles such as cars and air-planes, are subject to a non-zero minimum turning radius. This means that they cannot make sharp turns without following a circular arc. Such motion constraints can seriously restrict the manuverbility of non-holonomic vehicles [33], and if they are ignored during the planning phase, it can lead to intractable paths.

As shown in [34], the problem of deciding whether a curvature-constrained collision-free path exists between two given configurations amid polygonal obstacles is NP-hard [35]. This implies that no exact algorithms exist for curvature-constrained time-optimal motion

planning in arbitrary environment [34]. For a constant-speed curvature-constrained vehicle moving in the absence of obstacles, the pioneering work by Dubins [36][32] utilized a geometrical approach and showed that the shortest path between a pair of poses belongs to a set of 6 path types. Such results were later verified in [37] using Pontryagin's maximum principle. Thus, one has to solve for all 6 candidate path types, and then picks the one with the minimum length. However in practice, autonomous vehicles can travel at variable speeds, which range from a lower bound (e.g., idle speed) to an upper bound (e.g., maximum speed). Recent works [38] showed that for a variable-speed curvature-constrained vehicle moving in the absence of obstacles, the time-optimal path belongs to a sufficient set of 34 path types. However, to the best of our knowledge, when obstacles are present, the time-optimal motion planning problem for curvature-constrained variable-speed vehicles still has not been solved.

Moreover, due to complexities of the environment, it is also important that the time-optimal path is safe for the autonomous vehicle. Existing literature on risk-aware path planning either builds risk maps about the environment [39][40], or constructs risk zones [41] or inevitable collision states [42] for the vehicle to avoid obstacles. The risk measures presented in these approaches have considered vehicle location with respect to nearby obstacles or threats, while ignoring its speed and/or heading angle. Clearly, when a vehicle operates near obstacles, both its heading and speed can be critical factors to its safety. Therefore, the full information about the vehicle state should be incorporated into planning the time-optimal risk-aware path.

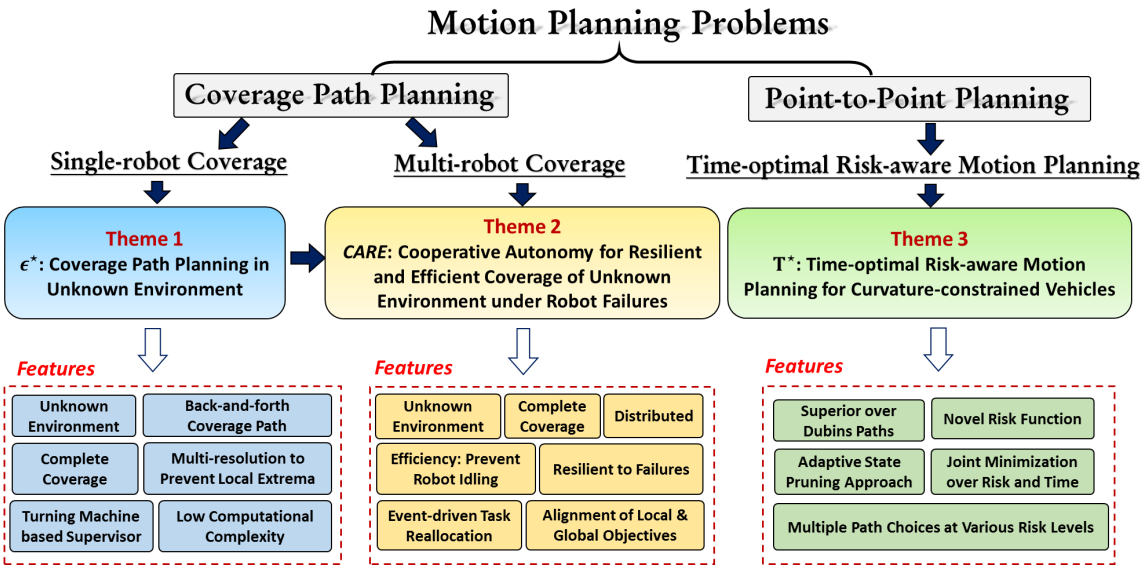



**Figure 1.2:** Thesis outline and the features of each theme

## 1.2 Outline and Contributions

This dissertation consists of three themes as presented below. The outline along with the features for each theme are summarized in Fig. 1.2.

### 1.2.1 Theme 1: Single-robot Coverage Path Planning in Unknown Environment

The first theme focuses on the online single-robot CPP problem. We present a novel approach, named $\varepsilon^\star$, for complete coverage in unknown environment. The $\varepsilon^\star$ algorithm utilizes a supervisory control structure, where the *Exploratory Turing Machine* (ETM) is developed and acts as a supervisor to guide the autonomous vehicle with navigation commands. The ETM consists of a two-dimensional multilevel tape formed by *Multiscale Adaptive Potential Surfaces* (MAPS), and a tape head. Based on real-time sensor measurements, the MAPS store and update the information corresponding to unexplored, explored

and obstacle-occupied regions. On the other hand, the tape head possesses a finite set of states, while it can only be in one state at a time. The state of tape head indicates the operation status of the ETM. By default, the ETM uses the lowest level of MAPS to compute for the new waypoints, while it switches to higher levels as needed to prevent the local extrema problem. It is shown that the ETM halts in finite time, and upon halting, complete coverage is achieved [43]. The $\varepsilon^\star$ algorithm has been validated in various complex scenarios, and the coverage performances in terms of total trajectory length and total number of turns have shown clear advantages over existing popular online coverage methods.

This theme makes the following contributions:

- developed a novel online CPP algorithm, which generates the desired back-and-forth coverage path, but does not rely on critical point detection on obstacles;
- it prevents the local extrema problem, guarantees complete coverage, and it is computationally efficient; and
- validated in high-fidelity simulations and also on a physical autonomous ground vehicle equipped with heterogeneous sensing systems.

### 1.2.2 Theme 2: Multi-robot Resilient and Efficient Coverage in Unknown Environment

The second theme builds on the first theme, and addresses the multi-robot resilient and efficient CPP in unknown environment. We present a distributed yet cooperative algorithm, named *C*ooperative *A*utonomy for *R*esilience and *E*fficiency (CARE), that provides system-level resilience and efficiency for mutli-robot coverage operations under failures. For coverage control, the entire target area is partitioned into sub-areas that are referred as tasks. Each robot is assigned with one task at a time, and it adopts the $\varepsilon^\star$ algorithm to cover

its assigned task. Periodically, the robots exchange their local information in a pair-wise manner, hence each robot also maintains a copy of the global knowledge about the team.

The decision-making for each robot is controlled by a *Discrete Event Supervisor* (DES). In particular, when an event of vehicle failure or idling appears, the DES triggers potential games [44][45] between a set of feasible players to make collaborative decisions for task reallocations. These games are carefully designed such that the local task reallocation decisions are always aligned with the global objective of the whole team. In case of no failures, CARE reallocates idling vehicles to support other vehicles in their tasks, hence reduces coverage time and improves team efficiency. In case of vehicle failures, it guarantees complete coverage via filling coverage gaps by optimal reallocation of other vehicles, hence providing resilience, albeit with a possibly small degradation in coverage time. The CARE algorithm has been validated in complex scenarios, and the results showed that the team achieves complete coverage even under failures. Also, it enables faster target discovery as compared to alternative MCPP methods.

This theme makes the following contributions:

- developed a distributed multi-robot resilient and efficient coverage method, which guarantees complete coverage under some robot failures, as well as prevents robots idling;
- developed a game-theoretic dynamic task reallocation method, where the local task reallocation decision for a subset of robots can always benefit the global objective of the whole team; and
- considered various time-varying optimization factors for task reallocation, including estimated task worths, robot remaining energy and their relative locations.

### 1.2.3 Theme 3: Time-optimal Risk-aware Motion Planning for Curvature-constrained Vehicles

The third theme focuses on the point-to-point motion planning problem for variable-speed vehicles with curvature constraints. We present a grid-based method, named T$^\star$, to address the time-optimal risk-aware motion planning problem for such vehicles in *a priori* known environment. To the best of our knowledge, this problem has not been solved in the presence of obstacles. Due to the NP-hardness of this problem [35], we compute the optimal path in a piece-wise manner in the discrete domain. The configuration space is constructed by populating each grid cell with a finite set of vehicles states. Then, T$^\star$ computes for the optimal path by identifying the optimal state sequence, along which the total cost of time and risk are jointly minimized. The time cost between two consecutive states located in neighboring cells is determined by the time-optimal collision-free path between them subject to curvature constraints [38]. Moreover, we proposed a risk function based on the concept of collision time, which relies on the full information of the vehicle state, including its location with respect to nearby obstacles, heading angle as well as speed. The risk function has been seamlessly integrated into the time-optimal cost for optimization, and the framework of A$^\star$ is used to search for the optimal state sequence. The T$^\star$ algorithm can provide multiple path choices to the planner with decreasing risk costs but at the expense of increasing time costs.

Besides, in order to reduce the computational complexity of T$^\star$ in the high-dimensional configuration space, we also developed an adaptive state pruning technique which can maintain the solution quality while significantly reducing the computation time. The algorithm has been validated in complex obstacle-rich scenarios, and the resulting paths are shown to be superior than the Dubins paths.

This theme makes the following contributions:

- presented a solution to the time-optimal risk-aware motion planning problem in the presence of obstacles;
- presented a risk function based on the concept of collision time and integrated it with the time cost; and
- developed an adaptive state pruning technique that can significantly reduce the computation time while maintaining the path quality and ensuring completeness.

The rest of this dissertation is organized as follows. Chapter 2 presents the $\varepsilon^\star$ algorithm for single-agent coverage path planning in unknown environment. Chapter 3 presents the CARE algorithm which extends $\varepsilon^\star$ for multi-robot resilient and efficient coverage in unknown environment. Chapter 4 presents the T$^\star$ algorithm to address the time-optimal risk-aware motion planning problem for curvature-constrained vehicles, and we summarize the research impacts of these proposed methods and their future works in Chapter 5.

## 1.3 List of Publications

### Journal Papers

1. **J. Song**, K. Mittal, S. Gupta, and T.A. Wettergren, "Real-time motion planning for Dubins vehicles under steady external drifts," In preparation, 2019.
2. J. Hare, **J. Song**, S. Gupta and T.A. Wettergren, "POSE.R: Prediction-based opportunistic sensing for resilient and efficient sensor networks," In preparation, 2019.

3. **J. Song** and S. Gupta, "CARE: Cooperative autonomy for resilience and efficiency of robot teams for complete coverage of unknown environment under robot failures," *Autonomous Robots*, Under review, 2019.

4. Z. Shen, **J. Song**, K. Mittal and S. Gupta, "3-D coverage path planning for terrain reconstruction using autonomous vehicles," *IEEE Robotics and Automation Letters*, Under review, 2019.

5. **J. Song**, S. Gupta and T.A. Wettergren "T$^\star$: Time-optimal risk-aware motion planning for curvature-constrained vehicles," *IEEE Robotics and Automation Letters*, vol. 4, issue 1, pp. 33-40, 2019.

6. **J. Song** and S. Gupta, "$\varepsilon^\star$: An online coverage path planning algorithm," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 526-533, 2018.

## Conference Papers

1. **J. Song**, S. Gupta and T.A. Wettergren, "Time-optimal path planning for underwater vehicles in obstacle constrained environments," in *Proceedings of the MTS/IEEE OCEANS'17*, Anchorage, AK, pp. 1–6, 2017.

2. J. Hare, S. Gupta, **J. Song**, and T.A. Wettergren, "Classification induced distributed sensor scheduling for energy-efficiency in underwater target tracking sensor networks," in *Proceedings of the MTS/IEEE OCEANS'17*, Anchorage, AK, pp. 1–7, 2017.

3. Z. Shen, **J. Song**, K. Mittal and S. Gupta, "Autonomous 3-D mapping and safe-path planning for underwater terrain reconstruction using multi-level coverage trees," in *Proceedings of the MTS/IEEE OCEANS'17*, Monterey, CA, pp. 1–6, 2017.

4. Z. Shen, **J. Song**, K. Mittal and S. Gupta, "An autonomous integrated system for 3-D underwater terrain reconstruction," in *Proceedings of the MTS/IEEE OCEANS'16*, Monterey, CA, pp. 1–6, 2016.

5. **J. Song** and S. Gupta, "SLAM based shape adaptive coverage control using autonomous vehicles," in *Proceedings of the IEEE International Conference on System of Systems Engineering*, San Antonio, TX, pp. 268–273, 2015.

6. **J. Song**, S. Gupta and J. Hare "Game-theoretic cooperative coverage using autonomous vehicles," in *Proceedings of the MTS/IEEE OCEANS'14*, St. John's, Newfoundland, Canada, pp. 1–6, 2014.

7. **J. Song**, K. Qiu, S. Gupta and J. Hare "SLAM based adaptive navigation of AUVs for oil spill cleaning," in *Proceedings of the MTS/IEEE OCEANS'14*, St. John's, Newfoundland, Canada, pp. 1–6, 2014.

8. J. Hare, S. Gupta and **J. Song**, "Distributed smart sensor scheduling for underwater target tracking," in *Proceedings of the MTS/IEEE OCEANS'14*, St. John's, Newfoundland, Canada, pp. 1–6, 2014.

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

## Patents

1. S. Gupta and **J. Song**, "System and method for complete coverage of unknown environments", *U.S. Patent*, Application No. 62/673,090, May 17, 2018.

# Chapter 2

# Single-robot Coverage Path Planning in Unknown Environment

## 2.1 Introduction

This chapter presents the $\varepsilon^\star$ algorithm (that stands for $\varepsilon$-STAR or "$\varepsilon$-*coverage via Structural Transitions to Abstract Resolutions*") for single-robot coverage path planning in unknown environment. The algorithm utilizes an *Exploratory Turing Machine* (ETM) that acts as a supervisor to the autonomous vehicle and guides it with adaptive navigation commands. As shown in Fig. 2.1, the ETM 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

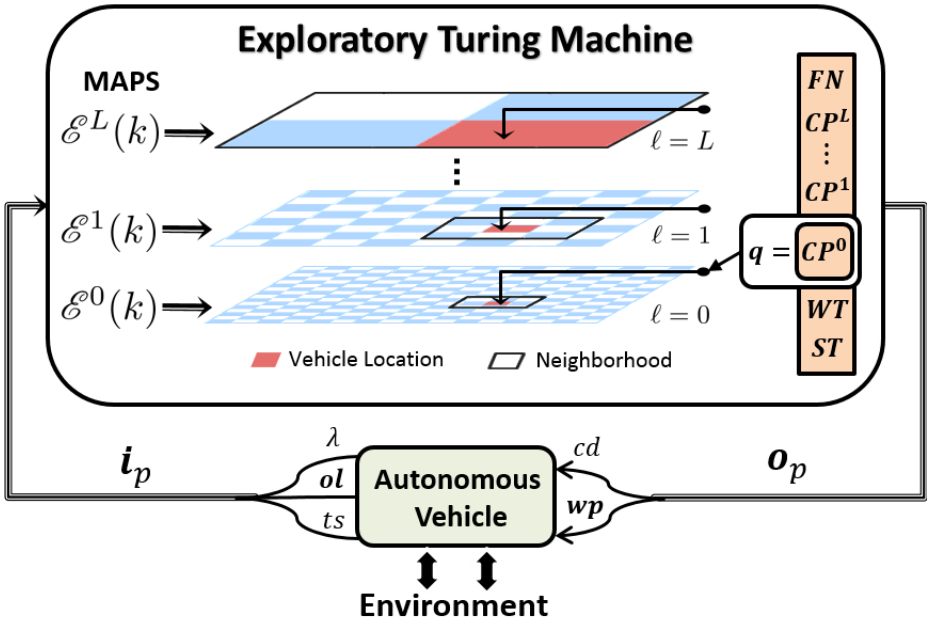



**Figure 2.1:** ETM as a supervisor of the autonomous vehicle

generating the coverage path online, it switches to higher levels as needed to escape from a local extremum.

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 both high-fidelity simulations on Player/Stage and real experiments in a laboratory setting.

## 2.2 Related Work

A variety of coverage algorithms exist in literature; a review of such algorithms is presented in [19]. The CPP methods are categorized into two types: offline or online (i.e.

sensor-based). While offline approaches [46] assume the environment to be *a priori* known, online approaches [16] 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 [16] of the search area into cells of varying shapes. Lumelsky et al. [47] 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. [48] by reducing the upper bound on path length; however, these algorithms are limited to a small set of obstacle geometries.

Zelinsky et al. [46] 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. [49] used the so-called 'ant robots' with limited sensing and computational capabilities to scan unknown areas. Gabriely and Rimon [50] used the *Spanning Tree Covering* (STC) algorithm for online coverage, which was later improved to *Full Spiral STC* (FS-STC) algorithm [17]. Gonzalez et.al. [18] 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 [16] developed a sensor-based coverage method that is based on detection of 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 [51], which could be difficult in complex environment. Furthermore, this method cannot function in rectilinear environment. Ferranti et. al. [52] presented the

**Table 2.1:** A comparison of key features of $\varepsilon^\star$ with other algorithms

| | $\varepsilon^\star$ | FS-STC [17] | BSA [18] | Cellular Decomposition [16] |
|---|---|---|---|---|
| *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 path to fill areas and backtracking to escape spiral endings | relies on critical point detection for Morse decomposition; then uses Reeb graph and cycle algorithm |

Brick and Mortar (B&M) algorithm for 'ant robots', which selects unexplored waypoints while maintaining the connectivity of the res explored and unexplored cells; however, it may suffer from the looping problem. This method was later improved by Andries et. al [53] for multi-agent coverage. Table 2.1 summarizes the comparison between the features of $\varepsilon^\star$ and other popular online coverage algorithms.

The rest of this chapter is organized as follows. Section 2.3 describes and formulates the CPP problem. Section 2.4 presents the details of the $\varepsilon^\star$ algorithm. The results and discussions are presented in Section 2.5. At last, this chapter is concluded in Section 2.6.

## 2.3 Problem Description

This section presents the concept of $\varepsilon$-coverage of an environment that is populated with unknown obstacles of arbitrary shapes.

The autonomous vehicle as shown in Fig. 2.2 contains:

- a localization device (e.g., GPS [54] or SLAM [55]) to obtain vehicle location;

- a range detector (e.g. a laser scanner) to detect obstacles within a circular region of radius $R_s \in \mathbb{R}^+$; and

- a task-specific sensor for performing its main task (e.g., cleaning) with a circular area of radius $r_t \leq R_s$.

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

**Definition 2.3.1 (Tiling).** *A set $\mathscr{T} = \{\tau_\alpha \subset \mathbb{R}^2, \alpha = 1, \dots |\mathscr{T}|\}$ is called a tiling of $\mathscr{R}$ if its elements: i) have mutually exclusive interiors, i.e. $\tau_\alpha^\circ \bigcap \tau_\beta^\circ = \emptyset$, $\forall \alpha, \beta \in \{1, \dots |\mathscr{T}|\}$, $\alpha \neq \beta$, where $^\circ$ denotes an interior, and ii) form an exact cover of $\mathscr{R}$, i.e. $\mathscr{R} = \bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha$. If an exact cover is not possible (e.g., square tiles cannot exactly cover a circular region), condition ii) can be relaxed to $\mathscr{R} \subseteq \bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha$, to form a minimal tiling of $\mathscr{R}$, s.t. removal of any single tile destroys the covering property.*

The tiling formed by square tiles of side $\varepsilon$ is called an $\varepsilon$-*cell tiling*. It is recommended that an $\varepsilon$-cell should be atleast big enough to contain the autonomous vehicle and small enough for the tasking sensor to be able to cover it when the vehicle passes through it. Within these two bounds, the choice of $\varepsilon$ depends on the following factors. A smaller $\varepsilon$ provides a better approximation of the search area and its obstacles. On the other hand, a larger $\varepsilon$ reduces the computational complexity by requiring less number of $\varepsilon$-cells to cover the area and it also improves 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. 2.2. 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 vehicle. The remaining cells are allowed which are desired to

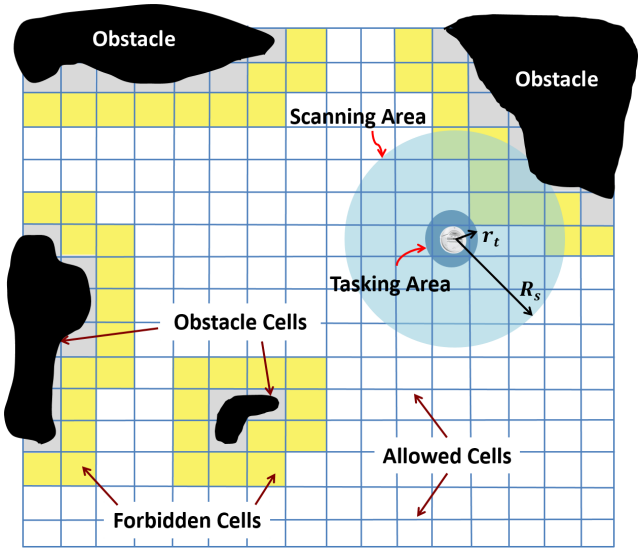



**Figure 2.2:** An autonomous vehicle working in its environment

be covered. The autonomous vehicle discovers the obstacles online, updates the obstacle and forbidden cells, and performs tasks in the allowed cells. Now, we present the concept of $\varepsilon$-coverage.

**Definition 2.3.2** ($\varepsilon$-**Coverage**). *Let $\mathscr{R}^a$ denote the total area of the allowed cells in $\mathscr{T}^a \subseteq \mathscr{T}$. Let $\tau(k) \in \mathscr{T}$ be the $\varepsilon$-cell visited by the autonomous vehicle at time $k$ and explored by its tasking sensor. Then, $\mathscr{R}$ is said to achieve $\varepsilon$-coverage if $\exists\, K \in \mathbb{N}^+$ s.t. the sequence $\{\tau(k),\, k = 1, \ldots K\}$ covers $\mathscr{R}^a$, i.e.*

$$\mathscr{R}^a \subseteq \bigcup_{k=1}^{K} \tau(k). \tag{2.1}$$

**Remark 2.3.1.** *$\varepsilon$-coverage achieves complete coverage if the tasking sensor completely covers every explored $\varepsilon$-cell.*

## 2.4 The $\varepsilon^\star$ algorithm

The $\varepsilon^\star$ algorithm utilizes the concept of ETM for $\varepsilon$-coverage of unknown environment. As shown in Fig. 2.1, the ETM constantly takes feedback from the autonomous vehicle 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* [56][57]. The ETM consists of a single tape head and a two-dimensional multilevel tape formed by MAPS (see Section 2.4.1), which act as guidance surfaces for decision-making. Formally, the ETM is defined as follows.

**Definition 2.4.1** (**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 vehicle to finish tasking 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* $\ell$ *of the MAPS.*

- $I_p$ *is the set of input parameters containing the feedback information received from the autonomous vehicle. An input vector* $\boldsymbol{i}_p \in I_p$ *includes:*

  *i.* $\lambda \in \{1, \ldots |\mathscr{T}|\}$*: Index of the* $\varepsilon$*-cell where the vehicle 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 indexes of all* $\varepsilon$*-cells where obstacles are detected using the range detectors.*

*iii.* $ts \in \{cm, ic\}$*: Task status of the vehicle in its current* $\varepsilon$*-cell, where* $cm \equiv$*'Complete' and* $ic \equiv$*'Incomplete'.*

- $O_p$ *is the set of output parameters containing instructions for the autonomous vehicle. An output vector* $\boldsymbol{o}_p \in O_p$ *includes:*

  *i.* $cd \in \{mv, tk, id, sp\}$*: Operational command, 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 way-points for the vehicle trajectory on tiling* $\mathscr{T}$*.*

- $\delta$ *is the control function that is a partial mapping from* $I_p \times Q \times \Pi_{\mathscr{N}^\ell} \to Q \times \Pi \times O_p$*, where* $\Pi$ *is the set of all possible configurations of potentials on MAPS generated by the sets* $\Xi^\ell$*, while* $\Pi_{\mathscr{N}^\ell}$ *is the above set restricted to a local neighborhood* $\mathscr{N}^\ell$ *at Level* $\ell$ *of the MAPS.*

- $q_0$*=ST is the initial state, and*

- *F=FN is the final state implying complete coverage.*

**Remark 2.4.1.** *An advantage of Turing Machine (TM) over Finite State Automaton (FSA) [58] is that TM 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.

### 2.4.1 Construction of MAPS

To build MAPS, first a hierarchical multiscale tiling (MST) is constructed on the area $\mathscr{R}$ by recursive decomposition [9][59]. As shown in Fig. 2.1, the $\varepsilon$-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 $\varepsilon$-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 $\varepsilon$-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 $\mathscr{T}^0, \mathscr{T}^1, ...\mathscr{T}^L$ such that $\mathscr{T}^\ell = \{\tau_{\alpha^\ell} : \alpha^\ell = 1, \dots |\mathscr{T}^\ell|\}$, $\forall \ell \in \{0, \dots L\}$, where $\alpha^\ell$, $\forall \ell \geq 1$, indexes coarse cells at Level $\ell$ of the MST, while $\alpha^0$ indexes $\varepsilon$-cells.

#### 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 [60] to each $\varepsilon$-cell $\tau_{\alpha^0} \in \mathscr{T}^0$ from the alphabet set $\mathcal{S} = \{\text{O, F, E, U}\}$, where O $\equiv$ *obstacle*, F $\equiv$ *forbidden*, E $\equiv$ *explored*, and U $\equiv$ *unexplored*. Then, the potential surface $\mathscr{E}^0(k)$=$\{\mathscr{E}_{\alpha^0}(k) \in \Xi^0 : \alpha^0 = 1, ...|\mathscr{T}^0|\}$, is constructed by assigning a discrete potential to each $\tau_{\alpha^0}$, such that

$$\mathscr{E}_{\alpha^0}(k) = \begin{cases} -1, & \text{if } s_{\alpha^0}(k) = \text{O or F} \\ 0, & \text{if } s_{\alpha^0}(k) = \text{E} \\ B_{\alpha^0}, & \text{if } s_{\alpha^0}(k) = \text{U} \end{cases}, \tag{2.2}$$

where $s_{\alpha^0}(k) \in \mathcal{S}$ is the state of $\tau_{\alpha^0}$ at time $k$. The first condition in Eq. (2.2) assigns a potential of $-1$ to $\tau_{\alpha^0}$, 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 vehicle from colliding with the obstacles due to inertia, skidding, large turning radius, or localization errors. The second condition in Eq. (2.2) assigns a potential of 0 to $\tau_{\alpha^0}$, if it has been explored by the tasking sensor. The third condition assigns a potential of $B_{\alpha^0}$ to $\tau_{\alpha^0}$, if it is yet unexplored, where $B = \{B_{\alpha^0} \in \{1, ...B_{\max}\}, \alpha^0 = 1, ...|\mathscr{T}^0|\}$ 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. 2.3, 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. 2.3.

**Modeling of the Potential Surfaces at Higher Levels**

For $1 \leq \ell \leq L$, the potential surface $\mathscr{E}^\ell(k)$=$\{\mathscr{E}_{\alpha^\ell}(k) \in \Xi^\ell : \alpha^\ell = 1, ...|\mathscr{T}^\ell|\}$, is constructed by assigning a potential to each coarse cell $\tau_{\alpha^\ell} \in \mathscr{T}^\ell$. This is done by assigning $\tau_{\alpha^\ell}$ the

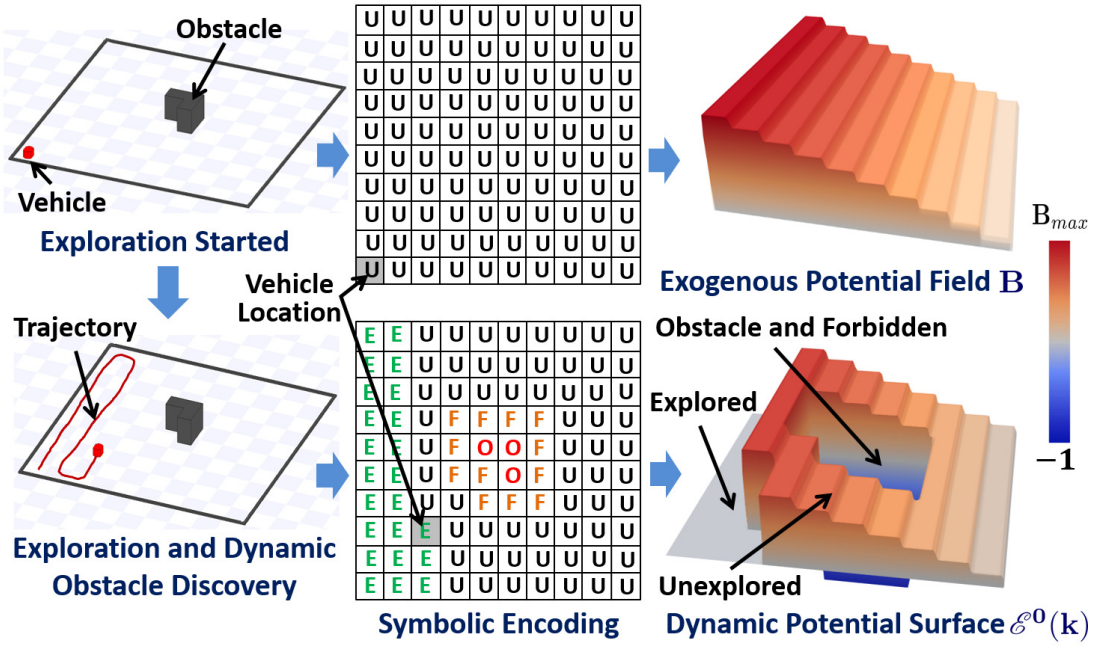



**Figure 2.3:** Dynamic construction of the potential surface $\mathscr{E}^0$

average potential generated by all the unexplored $\varepsilon$-cells within $\tau_{\alpha^\ell}$, such that

$$\mathscr{E}_{\alpha^\ell}(k) = p^U_{\alpha^\ell}(k)\overline{B}_{\alpha^\ell}, \tag{2.3}$$

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

### 2.4.2 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. 2.4 shows the state transition graph of the

ETM, which realizes the control function $\delta$. The input vectors $\boldsymbol{i}_{p_j} \in I_p, j = 1,2$, the output vectors $\boldsymbol{o}_{p_j} \in O_p, j = 1,\ldots 4$ 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 $\varepsilon$-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 autonomous vehicle about the newly discovered obstacle locations and its current position ($\lambda$). Then, it moves the head on the tape to $\lambda$ 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 $\lambda$ 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 vehicle containing the operational command and the new waypoint. In each iteration of state $WT$, it receives the task status from the vehicle, 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 $CP^0$.

If the head gets stuck in a *local extremum* in state $CP^0$, i.e. no waypoint with positive potential 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 $\mathscr{N}^1(\lambda)$ to find a waypoint. If no such coarse cell exists, hence 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.

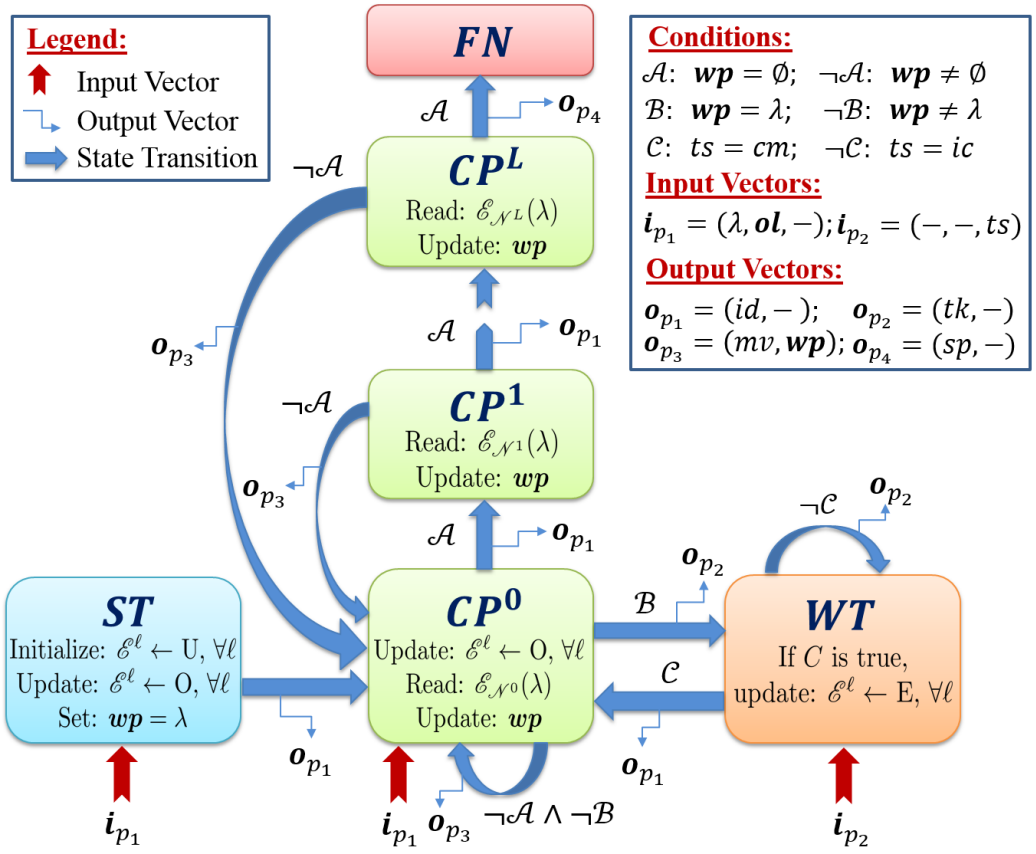



**Figure 2.4:** State transition graph of the ETM

### Operation in the ST State

The ETM starts in state $q = ST$ at $k = 0$ when vehicle is turned on. Since no *a priori* information is available, all $\varepsilon$-cells in $\mathscr{T}^0$ are initialized with the state U, i.e. *unexplored*. Then all $\varepsilon$-cells are assigned potentials according to field $B$ as per Eq. (2.2). Subsequently, all higher level cells are assigned potentials using Eq. (2.3), by substituting $p^U_{\alpha^\ell}(0) = 1$. This MAPS initialization process is denoted as $\mathscr{E}^\ell \leftarrow$ U, $\forall\ell \in \{0, ...L\}$.

Next, the autonomous vehicle detects its current location $\lambda$ and obstacle locations $\boldsymbol{ol}$ using its onboard sensors, and sends this information to the ETM via the input vector $\boldsymbol{i}_{p_1}$. The ETM moves its tape head to $\lambda$ and initializes the waypoint $\boldsymbol{wp} = \lambda$. Then it updates the

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

**input :** $\boldsymbol{wp}(k-1)$, $\lambda$, $\mathscr{E}_{\mathscr{N}^\ell}$, $q \in \{CP^\ell, \ell = 0, \ldots L\}$
**output:** $\boldsymbol{wp}(k)$

**if** $q = CP^0$ **then**
  compute $\mathscr{D}^0$ // form the computing set $\mathscr{D}^0$ using Eq. (2.4)
  **if** $\lambda \in \mathscr{D}^0$ **then** // current cell $\lambda$ is unexplored
    **if** $\{\lambda^{up}, \lambda^{down}\} \subset \mathscr{D}^0$ **then** // $\lambda^{up}$ & $\lambda^{down}$ unexplored
      $\boldsymbol{wp}(k) = \{\lambda^{up}, \lambda^{down}\}$ // Vehicle picks one per Eq. (2.5)
    **else** $\boldsymbol{wp}(k) = \lambda$ // set $\lambda$ as waypoint and start tasking
  **else if** $\mathscr{D}^0 \neq \emptyset$ **then** // other eligible $\varepsilon$-cells exist
    $\boldsymbol{wp}(k) = \arg\max_{\alpha^0 \in \mathscr{D}^0} \mathscr{E}_{\alpha^0}$ // pick the ones with max potential
  **else if** $\mathscr{E}_{wp(k-1)} > 0$ **then** // pre-computed *wp* still available
    $\boldsymbol{wp}(k) = \boldsymbol{wp}(k-1)$
  **else** $\boldsymbol{wp}(k) = \emptyset$ // local extremum detected at Level 0
**end**
**if** $q = CP^\ell, 1 \leq \ell \leq L$ **then**
  compute $\mathscr{D}^\ell$ // form the computing set $\mathscr{D}^\ell$ using Eq. (2.4)
  **if** $\mathscr{D}^\ell \neq \emptyset$ **then** // coarse cells with positive potentials exist
    $\boldsymbol{wp}(k) = I(\arg\max_{\alpha^\ell \in \mathscr{D}^\ell} \mathscr{E}_{\alpha^\ell})$
  **else** $\boldsymbol{wp}(k) = \emptyset$ // no waypoint found at Level $\ell$
**end**

symbolic encoding by flipping the states at all newly discovered obstacle indexes ***ol*** to O, and their associated neighborhood cells to F; subsequently, their potentials are updated to $-1$ using Eq. (2.2). 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 $\varepsilon$-cells inside those coarse cells. Then, using Eq. (2.3), the potential surfaces are updated $\forall \ell \in \{1, \ldots L\}$. In short, this entire MAPS updating process is denoted as $\mathscr{E}^\ell \leftarrow \mathrm{O}, \forall \ell \in \{0, \ldots L\}$.

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

### Operation in the $CP^0$ State

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

First, it forms a computing set $\mathscr{D}^0 \subseteq \mathscr{N}^0(\lambda)$ (**Line 2**) that consists of *eligible* $\varepsilon$-cells for the next waypoint. An $\varepsilon$-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 2.4.2** (**Directly Reachable Set**). *An $\varepsilon$-cell is called directly reachable from $\lambda$ if the line segment joining the centroids of $\lambda$ and that cell is not obstructed by any obstacle cell. The set of all directly reachable cells in $\mathscr{N}^0(\lambda)$ is defined as the directly reachable set $DR(\lambda)$.*

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

$$\mathscr{D}^\ell = \begin{cases} \{\alpha^0 \in \mathscr{N}^0(\lambda) : \mathscr{E}_{\alpha^0} > 0, \alpha^0 \in DR(\lambda)\} & \text{if } \ell = 0 \\ \{\alpha^\ell \in \mathscr{N}^\ell(\lambda) : \mathscr{E}_{\alpha^\ell} > 0\} & \text{if } \ell \geq 1 \end{cases}, \tag{2.4}$$

which means that $\mathscr{D}^0$ contains eligible $\varepsilon$-cells, while $\mathscr{D}^\ell$, $1 \leq \ell \leq L$, contains eligible coarse cells with positive potentials, implying that they contain unexplored $\varepsilon$-cells. The sets $\mathscr{D}^\ell, 1 \leq \ell \leq L$, are used in the $CP^\ell$ 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, Bug2 [61] is used to reach the cells behind obstacles.

Next, if the current cell $\lambda$ is unexplored, i.e. $\lambda \in \mathscr{D}^0$ (**Line 3**), then it further checks if the cell above ($\lambda^{up}$) and the cell below ($\lambda^{down}$) both belong to $\mathscr{D}^0$ (**Line 4**). This condition means that the vehicle is in the middle of unexplored cells both above and below $\lambda$. If this is true, then it 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 vehicle picks one of these based on its turn and travel cost as per Eq. (2.5) below. After computing $\boldsymbol{wp}$, the ETM loops in state $q = CP^0$ and sends the output vector $\boldsymbol{o}_{p_3}$ to move the vehicle to the next waypoint.

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

If the current cell $\lambda$ is not unexplored but there exist other eligible $\varepsilon$-cells in $\mathscr{D}^0$ (**Line 7**), then the candidate set $\boldsymbol{wp}$ is selected to consist of the $\varepsilon$-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 $q = CP^0$ and sends the output vector $\boldsymbol{o}_{p_3}$ to move the vehicle to the next waypoint, as seen in Fig. 2.4.

It is possible that $\boldsymbol{wp}$ contains more than one elements. In that case, the autonomous vehicle selects the cell with the least total travel and turn cost to reach it from $\lambda$. Let the current position of the vehicle 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 $(\mu_{x_c}, \mu_{y_c})$ of the $\varepsilon$-cell $\tau_\mu$ such

that

$$C_{\mu,\lambda} \triangleq d_{\mu,\lambda} C_{Tr} + \theta_{\mu,\lambda} C_{Tu}, \tag{2.5}$$

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

**Operation in the $CP^\ell$ States, $1 \leq \ell \leq L$**

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 $\mathscr{E}_{wp(k-1)} \leq 0$. 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. 2.4, when $\boldsymbol{wp} = \emptyset$, the ETM transitions to the computing state $q = CP^1$, while its head moves to Level 1 on the MAPS and points at the coarse cell containing the current $\varepsilon$-cell $\lambda$. 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 $\lambda$ 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 $\varepsilon$-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 $q = CP^2$ 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 $\boldsymbol{o}_{p_3}$ to move the vehicle 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 $\varepsilon$-cell. In that case, it switches to the state $q = FN$ and sends the output vector $\boldsymbol{o}_{p_4}$ commanding the vehicle to stop its machinery since the coverage is complete.

**Operation in the WT State**

The ETM comes to the state $q = WT$ from the state $q = CP^0$ if $\boldsymbol{wp} = \lambda$. Here the ETM waits while the vehicle performs task at $\lambda$ 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. (2.2). Subsequently, the potential surfaces $\mathscr{E}^\ell$, $\forall \ell \in \{1, ...L\}$, are updated according to Eq. (2.3). This MAPS update process is represented as $\mathscr{E}^\ell \leftarrow \text{E}, \forall \ell \in \{0, ...L\}$. 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 $q = WT$, while sending the output vector $\boldsymbol{o}_{p_2}$ 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 positive potentials defined by $B$, as shown in Fig. 2.3. Since the waypoint is mainly chosen as the $\varepsilon$-cell in the neighborhood with the highest positive potential, this enables tracking the highest equipotential surfaces of $B$ and produces the desired back-and-forth motion.

**Remark 2.4.2.** *The ETM uses the $floodfill$ algorithm to fill the unexplored cells inside a closed obstacle with the obstacle symbol O, when the boundary of the obstacle is detected but interior is not detected. This prevents the ETM from trying to pick undetected cells inside large obstacles.*

**Remark 2.4.3.** *At higher levels, if the computed waypoint is behind an obstacle, then the*

*vehicle could use any existing shortest path algorithm to reach it. Here we use Bug2 [61] for simplicity.*

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

*Proof.* From the ETM state transition graph in Fig. 2.4, 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 $CP^\ell, 1 \leq \ell \leq L$, 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 $CP^\ell$ state, and the tasking time spent in $WT$ state, are both finite; therefore, the total time spent in each cycle is finite.

Further, during the execution of these cycles, the unexplored $\varepsilon$-cells with state U are constantly flipped to states O, F or E accordingly. This implies that $p^U_{\alpha^\ell}(k), \forall \alpha^\ell \in \{1,...|\mathscr{T}^\ell|\}$, $\forall \ell \in \{1,...L\}$, decreases monotonically. Thus, $\exists$ a finite $K \in \mathbb{N}^+$, s.t. $p^U_{\alpha^\ell}(K) = 0$. This in turn implies that $\mathscr{E}_{\alpha^\ell}(K) = 0$, as per Eq. (2.3). Therefore, at time $K$, $\mathscr{D}^\ell = \emptyset$, $\forall \ell \in \{1,...L\}$, as per Eq. (2.4); 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 2.4.4.** *Since $\mathscr{D}^L = \emptyset$ upon halting, $\varepsilon$-coverage is achieved.*

**Corollary 2.4.1.** *Each allowed $\varepsilon$-cell is tasked only once.*

*Proof.* From Theorem 1, the ETM achieves $\varepsilon$-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 $\boldsymbol{wp}$ and hence cannot be tasked again. Thus, every allowed cell is tasked only once. □

### 2.4.3 Computational Complexity

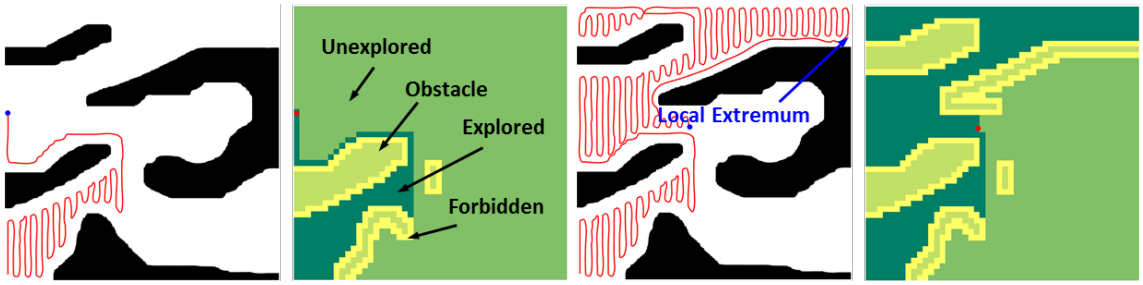
The $\varepsilon^\star$ algorithm has fairly low computational complexity and is real-time implementable. Suppose the local neighborhood $\mathscr{N}^0(\lambda)$ contains $N$ $\varepsilon$-cells. Similarly, suppose the local neighborhood $\mathscr{N}^\ell(\lambda)$, $\forall \ell \geq 1$, contains $M$ coarse cells. The ETM first searches in the local neighborhood at Level 0 to find navigation waypoints and only if it is stuck into a local 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 $\varepsilon$-cells within them, this bottom-up hierarchical approach to escape from a local extremum avoids searching for an exponentially increasing number of $\varepsilon$-cells; thus significantly reducing the computational complexity.

## 2.5 Results and Discussion

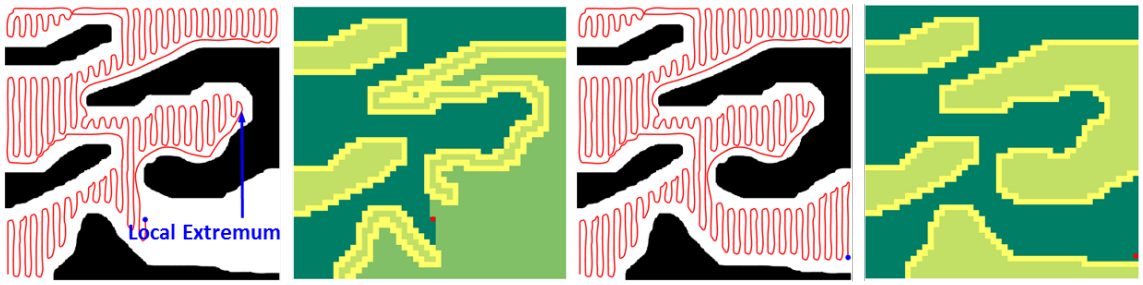
The $\varepsilon^\star$ algorithm is validated by simulations as well as experiments and its performance is compared with other algorithms.

### 2.5.1 Validation on a Simulation Platform

The first level of validation was done via simulation runs on a high-fidelity robotic platform called Player/Stage [62]. 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 chapter, a Pioneer 2AT robot of dimensions $0.44\text{m} \times 0.38\text{m} \times 0.22\text{m}$ was simulated with kinematic constraints such as the




**(a)** Coverage started with dynamic obstacle discovery **(b)** Escaping from a local extremum using MAPS

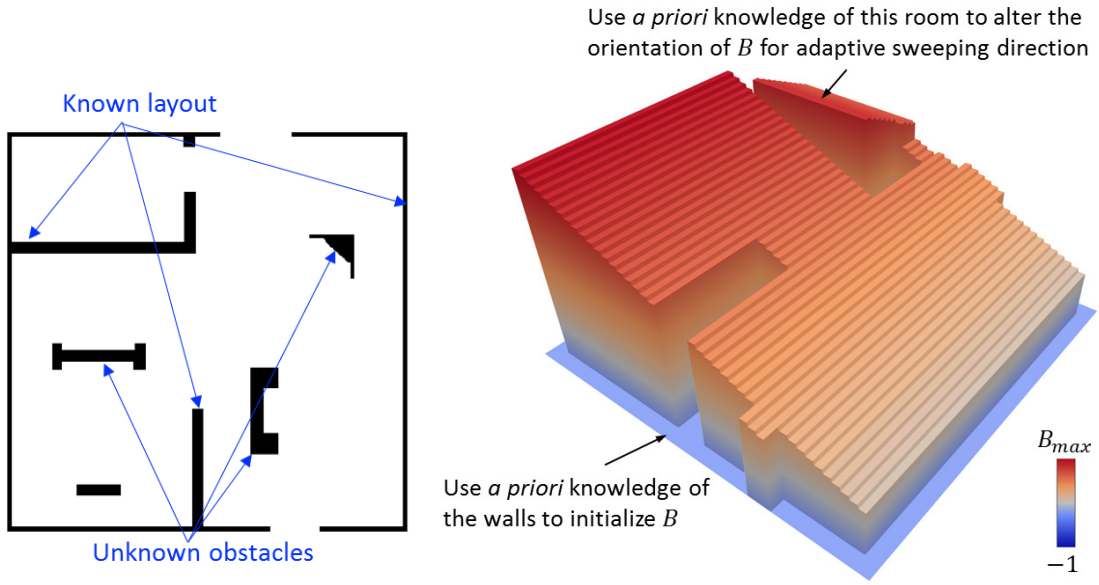



**(c)** Escaping from another local extremum **(d)** Complete coverage achieved

**Figure 2.5:** Scenario 1: Trajectories (left) and corresponding color-coded symbolic encodings (right) for the $\varepsilon^\star$ algorithm in a complex environment with arbitrary obstacles

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 computer with 3.40 GHZ CPU and 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 $50 \times 50$ tiling structure consisting of 1m $\times$ 1m $\varepsilon$-cells. This resulted in an MST with $L = 5$. For computation, the neighborhood was chosen to contain $7 \times 7$ cells at the lowest level and $3 \times 3$ cells at higher levels. The simulations were run 8 times faster than the real-time speed.

Fig. 2.5 shows the results of $\varepsilon^\star$ algorithm in Scenario 1, which has a complex environment with arbitrary obstacles. We show the snapshots of the trajectory generated by $\varepsilon^\star$ and the corresponding symbolic encodings discovered *in situ* at four different instants, as



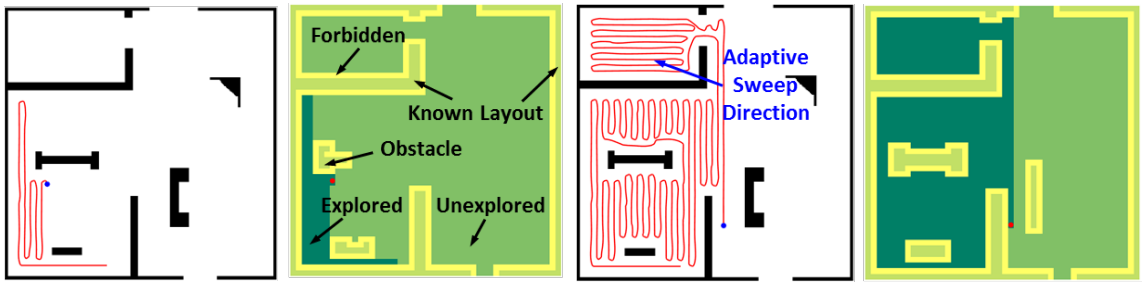

**(a)** A house environment

**(b)** Exogenous potential field $B$ for Scenario 2

**Figure 2.6:** Scenario 2: Validation for adaptive sweeping direction if provided *a priori* knowledge

shown in Fig. 2.5a∼Fig. 2.5d. These snapshots show several instances when the vehicle gets stuck into a local extremum, or surrounded by either obstacle or explored cells in the local neighborhood. Specifically, as shown in Fig. 2.5b and Fig. 2.5c, the vehicle successfully comes out of two local extrema using higher levels of MAPS, respectively. Finally, as seen in Fig. 2.5d, $\mathcal{E}^\star$ achieves complete coverage. Since the vehicle sees only the periphery of large obstacles, the *floodfill* algorithm is used by the ETM at regular intervals to fill the interiors of all closed obstacles.

Further, we illustrate that if *a priori* knowledge of the environment is available, it could be used to adapt the sweep direction of the vehicle. Fig. 2.6a presents the environment layout for Scenario 2, which is a house with several rooms and structures. It was assumed that the layouts of all rooms are *a priori* known but the inside obstacles are unknown. Since the shape of the upper left room is a rectangle with longer width, we constructed the




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

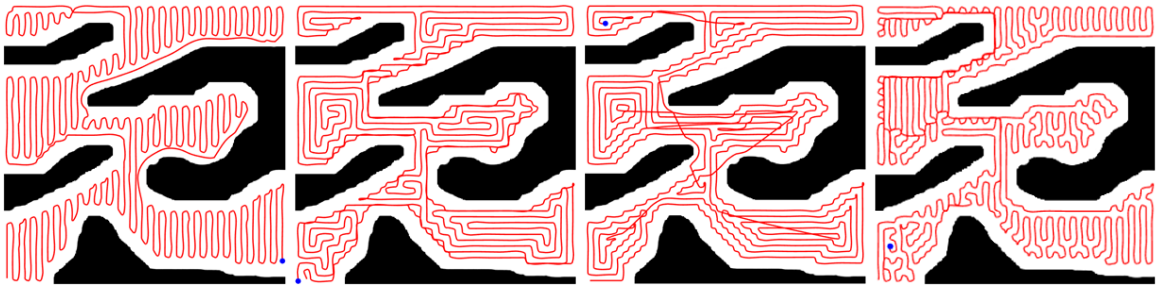
**(c)** Adapt to the shape of obstacle **(d)** Complete coverage achieved

**Figure 2.7:** Scenario 2: Trajectories (left) and corresponding color-coded symbolic encodings (right) for the $\varepsilon^\star$ algorithm in a house with several rooms and structures

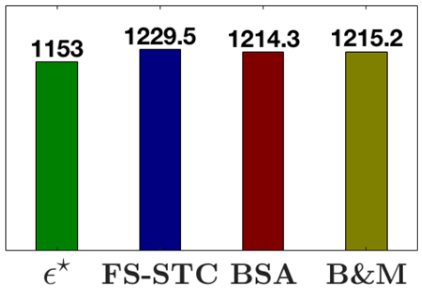
exogenous potential field $B$ in that room such that it has plateaus of equipotential surfaces along each row of the tiling, while monotonically increasing in height by one unit starting from the uppermost row. For the rest of the search area, $B$ follows the regular design. Therefore, the vehicle was able to sweep the upper left room horizontally and the other rooms vertically. A visualization for the potential field $B$ is shown in Fig. 2.6b.

Fig. 2.7 presents the coverage trajectories of $\varepsilon^\star$ and the corresponding color-coded symbolic encodings at four time instants for Scenario 2. As seen in Fig. 2.7b, the autonomous vehicle cleans the upper left room horizontally, which helps to further reduce the total number of turns. Moreover, as shown in Fig. 2.7d, $\varepsilon^\star$ again achieves complete coverage.

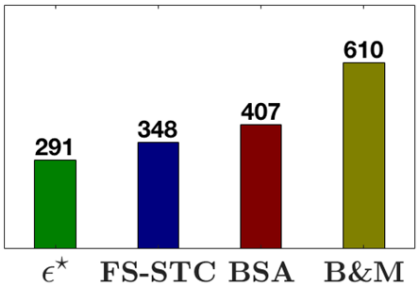
On average, the computation time to update $\boldsymbol{wp}$ in state $CP^0$ was $\sim 0.577$ milliseconds, while it was $\sim 0.437$ milliseconds in any $CP^\ell, 1 \leq \ell \leq L$ state; hence it is suitable for



**(a)** $\varepsilon^\star$ **(b)** FS-STC **(c)** BSA **(d)** Brick & Mortar




**(e)** Comparison of trajectory length




**(f)** Comparison of number of turns

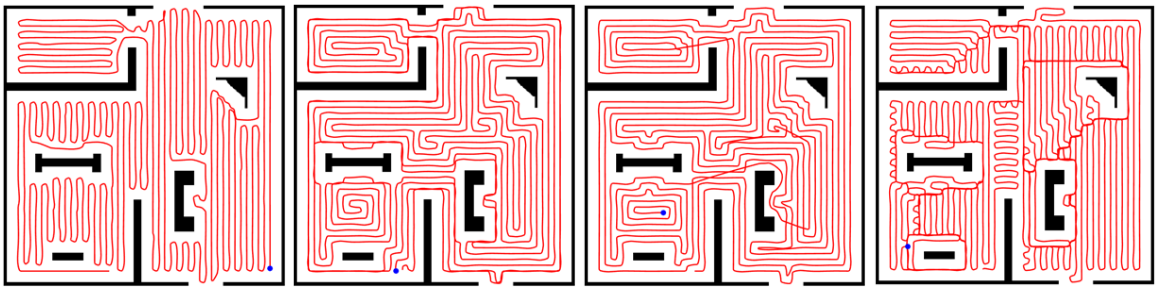
**Figure 2.8:** Scenario 1: A comparison of trajectories and coverage performances with alternative methods

real-time applications.

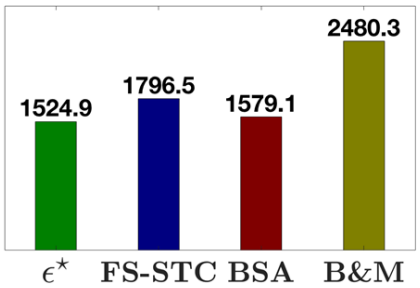
### 2.5.2 Comparison with Alternative Coverage Methods

The results of $\varepsilon^\star$ are compared with three other online coverage algorithms: FS-STC [17], BSA [18] and Brick & Mortar [52]. Three metrics are used for performance evaluation, including: i) *coverage ratio* $CR = \frac{\cup_k \tau(k) \cap \mathscr{R}^a}{\mathscr{R}^a}$, ii) *number of turns*, and iii) *trajectory length*.

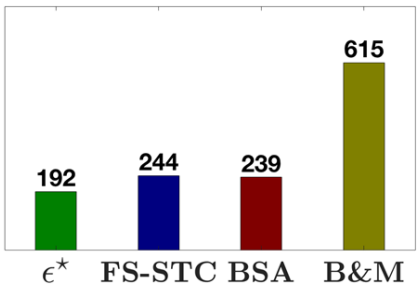
Fig. 2.8a$\sim$Fig. 2.8d present the coverage trajectories for Scenario 1 using $\varepsilon^\star$, FS-STC, BSA and Brick & Mortar, respectively. It is seen that the trajectories of FS-STC, BSA and Brick & Mortar algorithms are spiral in contrast to the back-and-forth trajectories generated by $\varepsilon^\star$. Fig. 2.8e and Fig. 2.8f compare the number of turns and total trajectory length for these methods in Scenario 1, respectively. Clearly, all four methods have achieved complete



**(a)** $\varepsilon^\star$ **(b)** FS-STC **(c)** BSA **(d)** Brick & Mortar




**(e)** Comparison of trajectory length




**(f)** Comparison of number of turns

**Figure 2.9:** Scenario 2: A comparison of trajectories and coverage performances with alternative methods

coverage, i.e., $CR = 1$, while $\varepsilon^\star$ produces significantly less number of turns and shorter trajectory length than others.

Moreover, Fig. 2.9a∼Fig. 2.9d show the coverage trajectories for Scenario 2 using $\varepsilon^\star$, FS-STC, BSA and Brick & Mortar, respectively. Fig. 2.9e and Fig. 2.9f compare the number of turns and total trajectory length for these methods in Scenario 2, respectively. Again, it is seen that all four methods have achieved complete coverage, while $\varepsilon^\star$ again generates much less number of turns and shorter trajectory length as compared to others.

### 2.5.3 Performance in the Presence of Uncertainties

For uncertainty analysis, noise was injected into the measurements of range detector (laser), the heading angle (compass), and the localization system.

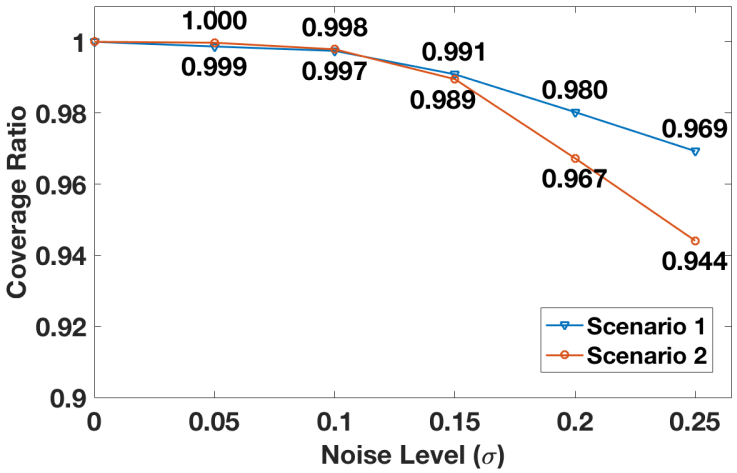


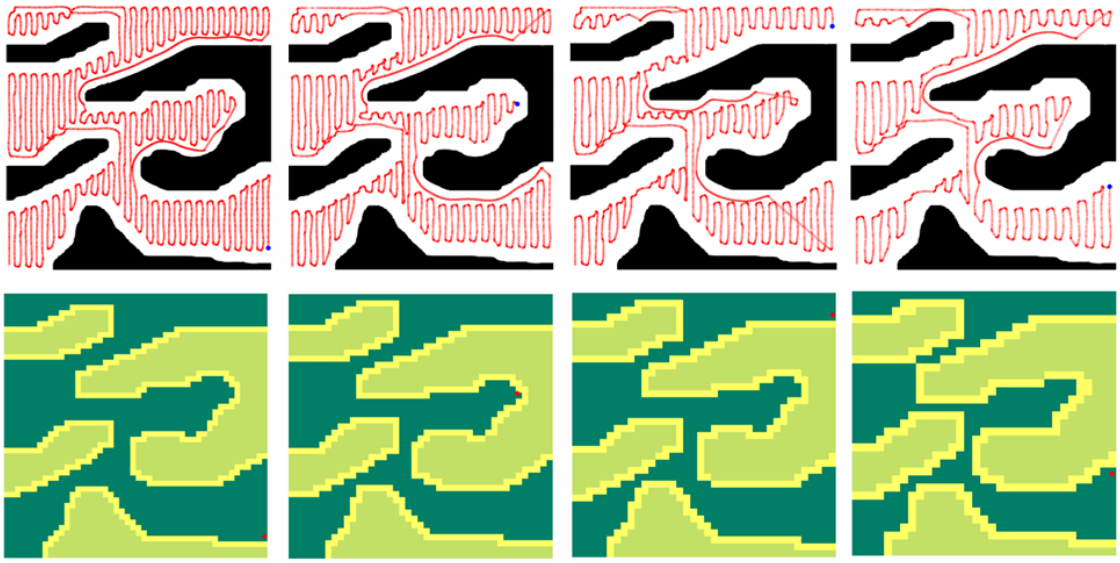

**Figure 2.10:** Coverage ratio vs. noise

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 $1^o$ [63]. The above errors were simulated with Additive White Gaussian Noise (AWGN) with standard deviations of $\sigma_{laser} = 1.5$cm and $\sigma_{compass} = 0.5^o$, respectively. The Hagisonic StarGaze indoor localization system provides a precision of 2cm [64], while the GPS system using Real-Time Kinematic (RTK) can achieve an accuracy of $0.05 \sim 0.5$m [63]. Thus, the uncertainty due to localization system is studied using AWGN with standard deviation ranging from $\sigma = 0.05$m to 0.25m. Fig. 2.10 shows the average coverage ratio vs. noise over ten Monte Carlo runs for the two scenarios.

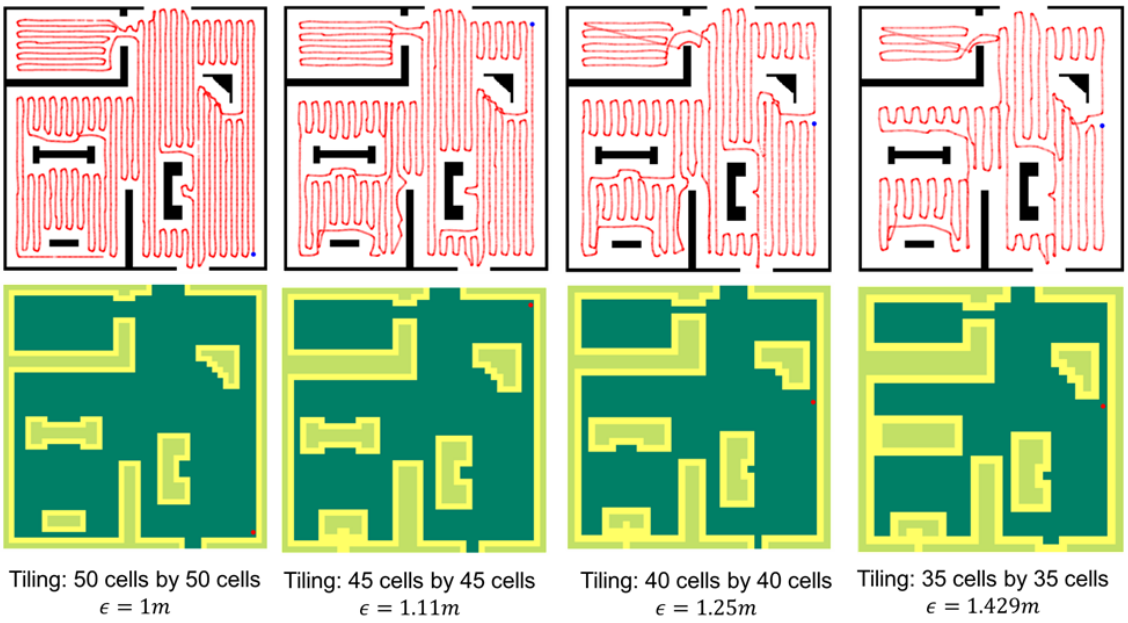
### 2.5.4 Performance using Different Sizes of $\varepsilon$

This section examines the effects of using different sizes of $\varepsilon$ for coverage. A general guide to select a proper size of $\varepsilon$ is provided in Section 2.3.

Fig. 2.11a and Fig. 2.11b present the coverage trajectories and the corresponding color-




**(a)** Scenario 1




**(b)** Scenario 2

**Figure 2.11:** Coverage trajectories and the corresponding color-coded symbolic encodings for different $\epsilon$

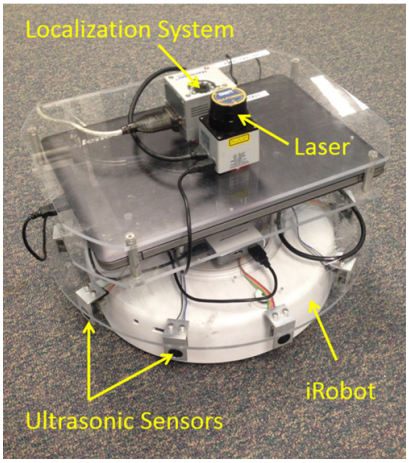



**Figure 2.12:** The autonomous ground vehicle for real experiments

coded symbolic encodings using different sizes of $\varepsilon$ for Scenario 1 and Scenario 2, respectively. It is seen that, with a larger $\varepsilon$, the tiling of the search area consists of less number of $\varepsilon$-cells. However, at the same time it results in a coarser approximation of the search area with a lower resolution. For both scenarios, $\varepsilon^\star$ is able to achieve $\varepsilon$-coverage in all cases.

### 2.5.5 Validation by Real Experiments

The $\varepsilon^\star$ algorithm was further validated by real experiments in a laboratory setting. An iRobot Create, as shown in Fig. 2.12, was utilized that was equipped with the Hagisonic StarGazer system [64] for indoor localization, the Hokuyo URG-04LX scanning laser [65] with $\sim$ 2m detection range, and 10 ultrasonic sensors evenly placed around the vehicle body for collision avoidance. Table 2.2 provides the specifications of these sensing systems.

The laboratory area was partitioned into a $8 \times 8$ tiling structure with each $\varepsilon$-cell of dimension 0.61m. This resulted in an MST with $L = 2$. The forbidden region was not defined due to limited laboratory space. For computation, a neighborhood of size $3 \times 3$ was chosen at all levels. A hardware-in-the-loop setup was established, where the vehicle

**Table 2.2:** Specifications of onboard sensing systems

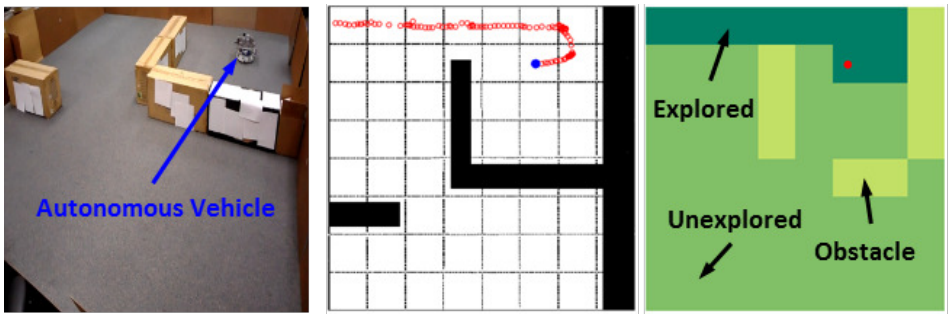
| | Localization | Laser | Ultrasonic |
|---|---|---|---|
| Model | StarGazer | URG-04LX | XL-MaxSonar-EZ |
| Range | – | $0.02\text{m} \sim 5.6\text{m}, 240^o$ | $0.2\text{m} \sim 7.65\text{m}$ |
| Resolution | $1\text{cm}, 1^o$ | $1\text{mm}, 0.36^o$ | 1cm |
| Accuracy | $2\text{cm}, 1^o$ | $\pm 1\%$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

carries an onboard laptop that runs the Player, which acts as the server to collect the real-time sensor measurements. The autonomous vehicle 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. 2.13 shows the results of a real experiment, where at each time instant, it shows the snapshot of the autonomous vehicle in the environment (left), its trajectory (middle) and the corresponding symbolic encodings (right), respectively. The vehicle successfully evacuated from a local extremum and explored different rooms to achieve $\varepsilon$-coverage, thus revealing the effectiveness of the $\varepsilon^\star$ algorithm.

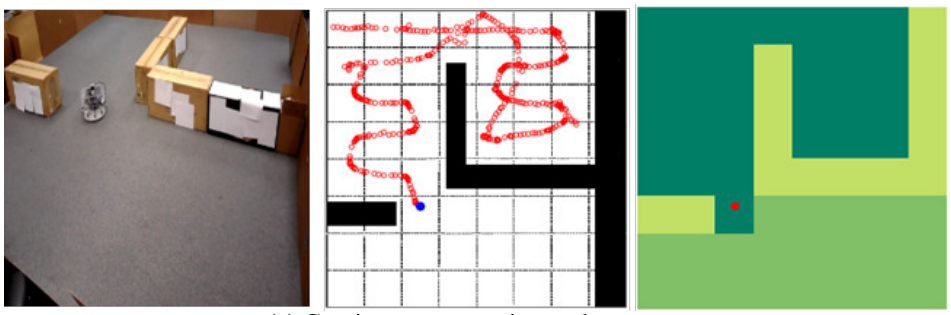
Below, we use the local extremum in Fig. 2.13b as an example, and dive deep into the mechanism that $\varepsilon^\star$ utilizes to prevent local extrema.

Since the search area was constructed into a tiling of $8 \times 8$ $\varepsilon$-cells, the MST has $L = 2$, i.e. it contains Levels 0, 1 and 2. Fig. 2.14a shows the time instant when the autonomous vehicle got stuck in the local extremum. At that moment, the ETM was operating in state $CP^0$ at Level 0, while all $\varepsilon$-cells in the local neighborhood were either explored or obstacles. Accordingly, as shown in Fig. 2.14b, there was no $\varepsilon$-cell with positive potential in its local neighborhood at Level 0 of MAPS. Thus, the ETM switched to state $CP^1$ and scanned in the local neighborhood at Level 1 of MAPS to look for coarse cells with positive potentials.

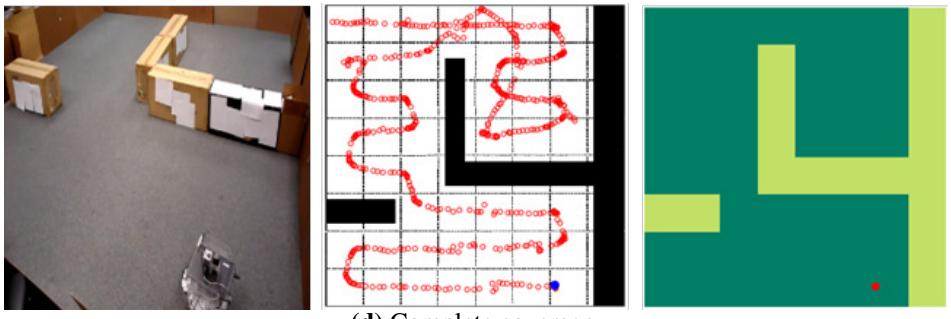



**(a)** Coverage started


**(b)** Escaping from a local extremum using wall following

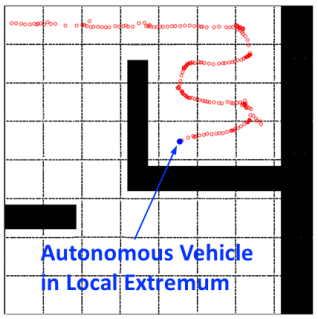


**(c)** Continue coverage in another room

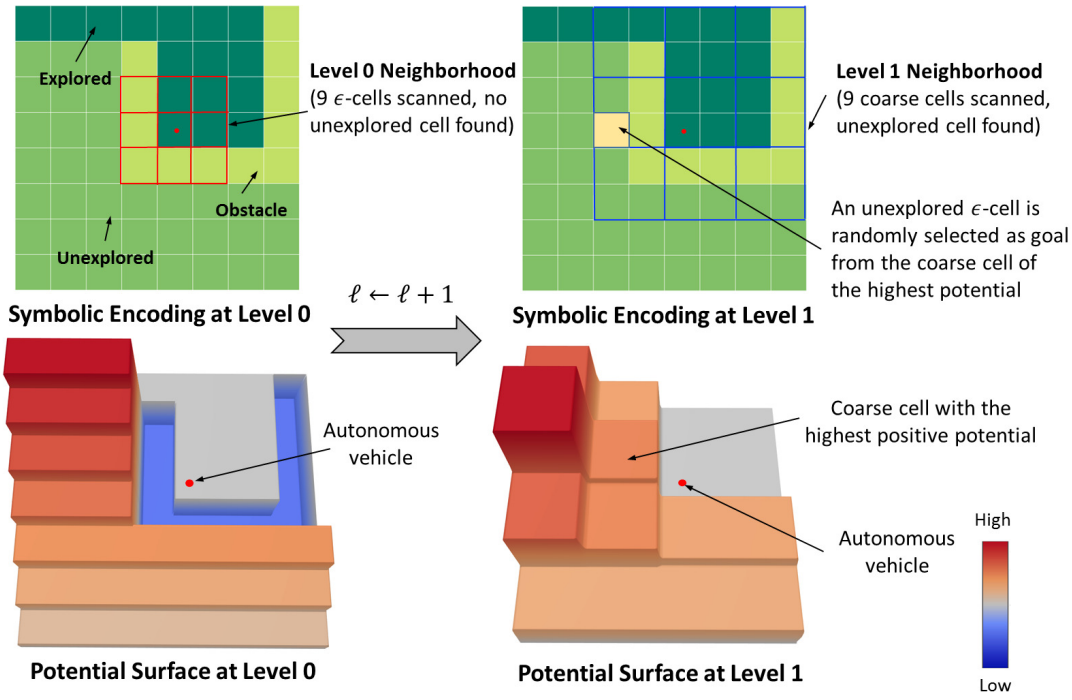


**(d)** Complete coverage

**Figure 2.13:** Real experiment in a laboratory environment




**(a)** Local extremum situation




**(b)** Use higher levels of MAPS to find the next waypoint

**Figure 2.14:** Use higher levels of MAPS to evacuate from a local extremum

Since each coarse cell on Level 1 contains four $\varepsilon$-cells, it was able to discover unexplored cells in a larger local region. In this scenario, there were unexplored cells spotted in its local neighborhood at Level 1. Then, it selected the coarse cell with the highest positive potential and picked an unexplored $\varepsilon$-cell from it as the next waypoint. Thereafter, the ETM switched back to state $CP^0$ and resumed its regular operations using Level 0 of MAPS.

## 2.6 Conclusions

This chapter presents an algorithm, called $\varepsilon^\star$, for online coverage of unknown environment. The algorithm utilizes the concept of an ETM which supervises the vehicle 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 has been validated via: i) high-fidelity simulations including sensor uncertainties, and ii) real experiments in a laboratory setting.

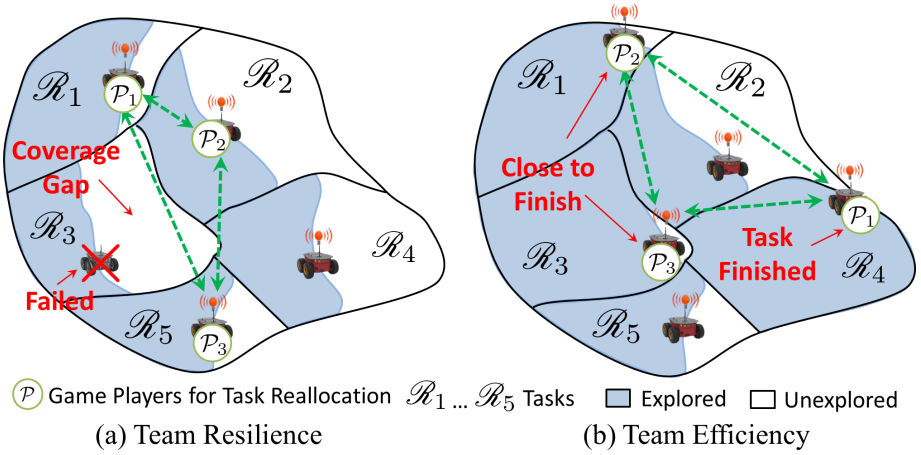
# Chapter 3

# Multi-robot Resilient and Efficient Coverage in Unknown Environment

## 3.1 Introduction

In Chapter 2, we presented the $\varepsilon^\star$ algorithm to address the single-robot CPP problem in unknown environment. This algorithm is computationally efficient, and can generate back-and-forth coverage path with guaranteed complete coverage. In this chapter, we extend the $\varepsilon^\star$ algorithm and address the *Multi-robot Coverage Path Planning* (MCPP) problem in unknown environment.

The typical control architecture for MCPP is to first partition the target area into multiple sub-areas, which constitute the set of coverage tasks. Then, one can coordinate the team of robots by using some single-robot CPP method for coverage operations in each task [24][66]. However, since the autonomous vehicles typically operate in uncertain environment, they are prone to different failures such as sensor or actuator malfunctions, me-




(a) Team Resilience (b) Team Efficiency

**Figure 3.1:** Concepts of resilience and efficiency of a robot team

chanical defects, loss of power [20]. The consequences of these failures include coverage gaps, loss of critical data, performance degradation (e.g., missed detections of targets), prolonged operation time, and in extreme cases overall mission failure. For example, coverage gaps in mine countermeasure operations can leave undetected underwater mines which are serious threats to traversing vessels. It is therefore critical that the robot team is resilient to failures, in the sense that it can sustain the overall team operation and protect the mission goals (e.g., complete coverage) even in presence of a few robot failures [21]. The role of resilience is to assure system-level survivability and fast recovery to normalcy from unanticipated emergency situations (e.g., robot failures). In the context of the MCPP problem, a resilient robot team is expected to autonomously re-organize the active robots in an optimal manner to complete the unfinished tasks of failed robots.

Secondly, it is also important that the robot team operates efficiently. Typically, due to incorrect, incomplete or lack of *a priori* knowledge of the environment, the initial task allocation may be sub-optimal. As a result, some robots may finish their tasks earlier and become idle, which is a waste of their resources. Thus, it is critical that the robot team

autonomously reallocates these idling robots in an optimal manner to assist other robots to reduce the total coverage time.

Fig. 3.1 illustrates the above concepts of resilience and efficiency. Fig. 3.1a shows an example of resilience where the neighbors of a failed robot proactively negotiate to decide whether any of them should leave its current task to fill the coverage gap. Fig. 3.1b shows an example of efficiency where a group of robots that have finished (or are close to finish) their current tasks negotiate to optimally reallocate to new tasks or to help other robots in their existing tasks.

The challenges associated with the problem of resilient and efficient MCPP include:

- *Scalability*: The MCPP algorithm should be scalable for a growing number of tasks and/or robots, thus making a distributed control structure appropriate.

- *Optimization factors*: The optimization for task reallocation must consider the following factors:

    1. Task worths, which can be quantified by the expected number of undiscovered targets (e.g., crops to cut or mines to discover) in the tasks.

    2. Probabilities of success of the available robots in finishing the contested tasks, which depend on various factors including their current energy levels, the costs of traveling to the contested tasks, and the costs of finishing those tasks.

- *Dynamically changing conditions*: The conditions of robots as well as tasks change dynamically during coverage. The task worths decrease as targets are discovered. On the other hand, the robots drain their batteries during exploration, hence decreasing their probabilities of success. Therefore, the optimization process must accommodate these dynamic factors.

- *Computation time*: First of all, the optimization must be event-driven, i.e., triggered only in case of failures and/or idling. Secondly, once the optimization is triggered, the task reallocation decision must be made in a timely manner to avoid prolonged coverage time, thus motivating a local distributed event-focused optimization over only a subset of available robots and tasks.
- *Connection between local and global objectives*: Although the local optimization decision can be sub-optimal for the whole team, it is important that it is still aligned with the global objectives of the team. In other words, the local optimization must not only benefit the involved robots but also the whole team, in terms of early detection of remaining targets, reduction in the total coverage time, and complete coverage.
- *Complete coverage*: The MCPP algorithm must guarantee complete coverage even under some robot failures.

To the best of our knowledge, the concept of resilient coverage has not been adequately addressed in the existing MCPP methods. In this regard, we present a novel online MCPP algorithm for resilient and efficient coverage in unknown environment, which addresses the challenges discussed above. The algorithm is called *Cooperative Autonomy for Resilience and Efficiency* (CARE). For coverage control in each task, CARE utilizes the $\varepsilon^\star$ algorithm as described in Chapter 2.

The CARE algorithm operates in a distributed yet cooperative fashion. Each robot is controlled using a *Discrete Event Supervisor* (DES), which triggers games between a set of feasible players in the event of a robot failure or idling, to make collaborative decisions for task reallocations. The game-theoretic structure is modeled using *Potential Games* [67], where the utility of each player is connected with a shared objective function for all players.

In case of no failures, CARE reallocates idling robots to support other robots in their

tasks, hence reduces coverage time and improves team efficiency. In case of robot failures, it guarantees complete coverage via filling coverage gaps by optimal reallocation of other robots, hence providing resilience, albeit with a possibly small degradation in coverage time. The CARE algorithm has been validated on Player/Stage in various complex obstacle-rich scenarios. The results demonstrate that the team achieves complete coverage under failures and enables faster target discovery as compared to three alternative methods.

The rest of this chapter is organized as follows. Section 3.2 presents a brief review of the existing MCPP algorithms. Section 3.3 formulates the MCPP problem and Section 3.4 presents the details of the CARE algorithm. The results are discussed in Section 3.5 and this chapter is concluded in Section 3.6.

## 3.2 Related Work

Regarding MCPP, Batalin and Sukhatme [68] proposed two local approaches for unknown environment, based on mutually dispersive interaction between robots. Latimer et al. [69] presented a boustrophedon cellular decomposition-based approach using a team of circular robots. The robots operate together, but can split up into smaller teams when cells are created or completed. Rekleitis et al. [70] presented a distributed auction-based cooperative coverage algorithm, where the whole space is partitioned into tasks of fixed height and width, and robots utilized the Morse decomposition based single-robot CPP algorithm to search within each task. Sheng et al. [71] proposed a multi-robot area exploration method with limited communication range, where the waypoint of each robot is computed using a distributed bidding mechanism based on frontier cells. The bids rely on the information gain, communication limitation, and traveling costs to frontier cells. Rutishauser et al. [72]

presented a distributed coverage method using miniature robots that are subject to sensor and actuator noise. Xu and Stentz [73] presented the $k$-Rural Postman Problem ($k$-RPP) algorithm to achieve environmental coverage with incomplete prior information using $k$ robots, that seeks to equalize the lengths of $k$ paths. Bhattacharya et al. [74] generalized the control law towards minimizing the coverage functional to non-Euclidean spaces, and presented a discrete implementation using graph search-based algorithms for MCPP. Karapetyan et al. [75] presented two approximation heuristics for MCPP in known environment, where the search area is divided into equal regions and exact cellular decomposition based coverage was used to search each region. Later, these methods were improved to consider vehicle kinematic constraints [76]. Yang et al. [77] proposed an online neural network based MCPP approach. In their method, the discovered environment was represented according to the dynamic activity landscape of the neural network, which is used to compute robot waypoints; and robots treat each other as moving obstacles during operation.

However, the above-mentioned algorithms have not addressed the problem of resilience in MCPP. In this regard, Agmon et al. [22] presented a family of Multi-robot Spanning Tree Coverage (MSTC) algorithms, where the Online Robust MSTC (ORMSTC) algorithm enables each robot to incrementally construct a local spanning tree to cover a portion of the whole space. If some robot fails, its local tree is released and taken over by its neighbors, but the already explored region of the failed robot must be scanned again. Also, the tree grows on the scale of $2 \times 2$ cells, while if any cell within such larger cell is occupied by obstacles, the whole larger cell would not be covered, thus leading to incomplete coverage. Zheng et al. [23] presented a polynomial-time Multi-Robot Forest Coverage (MFC) algorithm that computes tree covers for each robot with trees of balanced weights, and they showed the superiority of MFC in time to MSTC via simulations; however, their algorithm does not consider failures. Song et. al. [24] presented the First-Responder (FR) cooperative

coverage strategy, where early completed robots are reassigned to available new tasks that can maximize their own utility. However, this algorithm is not proactive, i.e., the coverage gaps caused by robot failures will not be filled until some other robots complete their tasks. Ferranti et. al. [52] presented the Brick and Mortar (B&M) algorithm, where the waypoint of each robot is computed locally based on the states of cells in the neighborhood. The idea behind B&M is to gradually thicken the blocks of inaccessible cells (i.e., visited or wall cells), while maintaining the connectivity of accessible cells (i.e., explored or unexplored cells). An unexplored cell can be marked as explored or visited, where the latter is allowed if it does not block the path between any two accessible cells in the neighborhood. The waypoint gives priority to the unexplored cell in the neighborhood, which has the most inaccessible cells around it. When some robot fails, the remaining robots continue regularly and the coverage gap becomes an extra workload; however, their method may produce redundant coverage due to the looping problem.

Although resilience concepts have been discussed in robot design [78], robot damage detection and recovery [79], flocking of robot teams [80] and networked control systems security under attacks [81], there is a scarcity of efforts that deal with the resilient coverage using multiple robots. Some of above-mentioned papers considered robot failures during coverage, however, their remedy was to simply release the coverage gaps to the remaining team, without optimization over the criticality (i.e., available worth) of such coverage gaps and the reliability of remaining robots. Thus, they are not proactive in filling the coverage gaps immediately if they satisfy optimization criteria, they wait until some other robots finish their tasks. In this regard, this chapter presents the CARE algorithm for resilient and efficient coverage, which incorporates these optimization factors while making event-driven proactive task reallocations. Table 3.1 and Table 3.2 presents a comparison of the key features of CARE with the other relevant online MCPP algorithms.

**Table 3.1:** A comparison of key features with other online MCPP algorithms

| | CARE | First-responder [24] |
|---|---|---|
| *Path Pattern* | Back and forth | Back and forth |
| *Resilience Strategy* | Neighbors jointly optimize to reorganize themselves to immediately fill the coverage gap caused by the failed robot if the optimization criteria are satisfied | Wait until some robot finishes its task and is reassigned to fill the coverage gap |
| *No-idling Strategy* | The idling robot and its near-finishing neighbors jointly optimize to help other robots to reduce coverage time and collect more worth early | Idling robots are reallocated to new tasks that maximize their own utility |
| *Optimization Factors* | Estimated worths of contested tasks, remaining reliability and traveling time of live robots | Unexplored portion of tasks and traveling time of robots |

**Table 3.2:** A comparison of key features with other online MCPP algorithms (Cont.)

| | Brick & Mortar [52] | ORMSTC [22] |
|---|---|---|
| *Path Pattern* | No obvious pattern observed | Spiral |
| *Resilience Strategy* | Remaining robots continue regularly. The coverage gaps become extra workloads. May produce strongly overlapped paths due to the looping problem | Neighbors extend their trees to fill the coverage gap of the failed robot, but the approach is not proactive and the already explored area by the failed robot is scanned again |
| *No-idling Strategy* | None | None |
| *Optimization Factors* | None | None |

## 3.3 Problem Description

This section presents the description of the robots, the MCPP problem and the performance metrics.

### 3.3.1 Description of the Robots

Let $V = \{v_\ell, \ell = 1, \ldots N\}$ be the team of $N \in \mathbb{N}^+$ robots, which are unmanned autonomous vehicles, as shown in Fig. 3.2. It is assumed that each robot is equipped with:

- a localization system (e.g., GPS [54] or SLAM [55]) to access vehicle location;
- a range detector (e.g., laser scanner) to detect obstacles within a radius $R_s \in \mathbb{R}^+$;
- a task specific sensor for performing the desired task (e.g., target detection); and
- a wireless communication device for (periodic or event-driven) information exchange between all pairs of robots. The communication is assumed to be perfect.

Since the robots continuously deplete their energies from the batteries during operation, their reliability is assessed based on the remaining energy as presented below.

**Battery Reliability**: Each robot $v_\ell \in V$, is assumed to carry a battery whose reliability [82], denoted as $R_{v_\ell}(t)$, can be computed as $R_{v_\ell}(t) = 1 - F(t)$, where $F(t)$ is the probability of battery being drained up to time $t$. Typically, the state-of-charge of a battery can be model using the realistic *Kinetic Battery Model* (KiBaM), which takes into account many important non-linear properties of batteries such as the rate-capacity effect and the recovery effect [83]. It is shown in [84] that with KiBaM, $F(t)$ follows a S-shaped curve when operating under different stochastic workload models (e.g., the on/off model and the

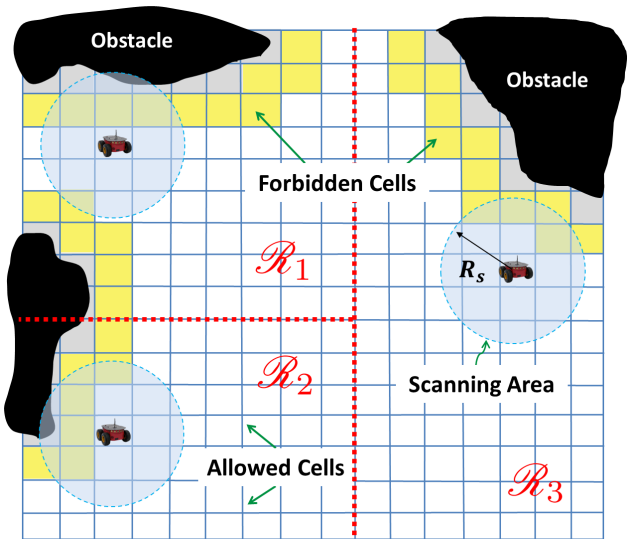



**Figure 3.2:** Example of a search area and its tiling. A team of 3 robots are scanning in three different tasks $\mathscr{R}_1$, $\mathscr{R}_2$ and $\mathscr{R}_3$. Robots are equipped with lasers for obstacle mapping

burst model). The S-shaped curve can be approximated using a sigmoid function [82]. As such, the reliability of a robot $v_\ell$ is given as:

$$R_{v_\ell}(t) = \frac{1}{1 + e^{\rho_0(t-\rho_1)}}, \tag{3.1}$$

where $\rho_0$ and $\rho_1$ indicate the curvature of the growth part and the inflection point, respectively. Their exact values depend on the choice of batteries. More details on the selection of these parameters are presented in Section 3.5.

### 3.3.2 The MCPP Problem

The search area $\mathscr{R} \subset \mathbb{R}^2$ is assumed to be a planar field whose borderline is defined either by a hard barrier (e.g., walls or obstacles) or by a soft boundary (e.g., sub-area of a large field). A finite but unknown number of obstacles with arbitrary shapes are assumed to populate this area, but their exact locations and shapes are *a priori* unknown.

For the purpose of coverage path planning, a tiling $\mathscr{T} = \{\tau_\alpha \subset \mathbb{R}^2, \alpha = 1, \ldots |\mathscr{T}|\}$ is constructed to cover $\mathscr{R}$ using the same procedure as explained in Chapter 2, i.e. $\mathscr{R} \subseteq \bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha$, as shown in Fig. 3.2. The tiling $\mathscr{T}$ is partitioned into three sets: i) *obstacle* ($\mathscr{T}^o$), ii) *forbidden* ($\mathscr{T}^f$), and iii) *allowed* ($\mathscr{T}^a$). While the cells in $\mathscr{T}^o$ are occupied by obstacles, the cells in $\mathscr{T}^f$ create a buffer around the obstacles to prevent collisions due to inertia or large turning radius of the robots. Due to lack of *a priori* knowledge of the environment, the obstacle cells and forbidden cells are discovered online using sensor measurements. The remaining cells are allowed, which form the free space $\mathscr{R}^a = \bigcup_{\tau_\alpha \in \mathscr{T}^a} \tau_\alpha$ that is desired to be covered.

For distribution of multiple robots, an initial task allocation is required. Thus, the tiling $\mathscr{T}$ is grouped into $M$ disjoint regions $\{\mathscr{R}_r \subset \mathscr{T}, r = 1, \ldots M\}$, s.t. $\mathscr{R} = \bigcup_{r=1}^{M} \mathscr{R}_r$. Each region $\mathscr{R}_r$ is regarded as one task and is referred as task $r$. Fig. 3.2 shows an example of the area with $M = 3$ tasks. Each robot can work on one task at a time, but one task can be assigned to multiple robots. Note that $M$ may not be equal to $N$.

**Remark 3.3.1.** *The problem of optimal space partitioning into disjoint tasks and optimal initial robot allocations may require consideration of several factors (e.g., obstacle distribution, robot capabilities, bathymetry) and is beyond the scope of this chapter. Here, we assume that no a priori knowledge of the environment is available, thus the tasks are made of equal sizes. However, as more information is obtained during exploration, event-driven task reallocations are performed for performance improvement.*

**Definition 3.3.1** (**Complete Coverage**). *Let $\tau_\ell(k) \in \mathscr{T}$ be the $\varepsilon$-cell that is visited and explored by the robot $v_\ell$ at time $k$. Then the robot team $V$ is said to achieve complete coverage, if $\exists K \in \mathbb{N}^+$, s.t. the sequences $\{\tau_\ell(k), k = 1, \ldots K\}, \forall \ell = 1, \ldots N$, jointly cover the free space $\mathscr{R}^a$, i.e.,*

$$\mathscr{R}^a \subseteq \bigcup_{\ell=1}^{N} \bigcup_{k=1}^{K} \tau_\ell(k). \tag{3.2}$$

In other words, the coverage is said to be complete if every cell in $\mathscr{R}^a$ is explored by at least one robot.

Next, it is assumed that each task contains randomly distributed targets, and their exact numbers and locations are unknown (see Section 3.4.2). However, it is assumed that the expected number of targets in each task is known, which in practice could be obtained by various means such as field surveys, aerial views or prior knowledge from other sources.

**Remark 3.3.2.** *If the total number and spatial distribution of targets is a priori known, then complete coverage may not be necessary and an optimal traversing strategy could be constructed to find all the targets. However, in this chapter, we assume that the planner neither knows the exact number of these targets, nor their exact locations, thus complete coverage becomes mandatory to guarantee finding all the targets.*

Due to non-uniform spatial distribution of targets and obstacles within each task, the targets are discovered at unequal rates by all robots. Thus, at any point of time all tasks could contain significantly different numbers of undiscovered targets. It is therefore critical that the regions with the maximum number of targets are scanned earlier and are given priority. Early detection of targets helps when the mission is terminated prematurely due to emergencies, failures or other reasons. For example, once the highly utilized areas of a building floor are cleaned then other areas could be cleaned gradually at ease.

Furthermore, the robots may suffer from unexpected failures during the coverage operation due to several reasons (e.g., actuator malfunctions or mechanical defects) which lead to coverage gaps. Thus, it is important to fill these coverage gaps by task reallocations of healthy robots. It is also important that the criticality of the task of the failed robot, as

measured by its expected number of remaining targets, is evaluated for task reallocations in comparison with the existing tasks of healthy robots.

### 3.3.3 Performance Metrics

The quality of multi-robot coverage can be evaluated based on the following performance metrics:

- *Coverage ratio* ($CR$): The ratio of the explored free space to the total free space, i.e.,

$$CR = \frac{\left(\cup_{\ell=1}^{N} \cup_{k=1}^{K} \tau_\ell(k)\right) \cap \mathscr{R}^a}{\mathscr{R}^a} \in [0,1]. \tag{3.3}$$

  Note that $CR < 1$ if the coverage gaps caused by robot failures are left unattended.

- *Coverage time* ($CT$): The total operation time of the team. This is measured by the last robot that finishes its task.

- *Remaining reliability* ($RR$): The remaining reliability of all live robots at the end of the operation.

- *Number of Targets Found* ($NoTF$): The total number of targets discovered by the whole team.

- *Time of Target Discovery* ($ToTD$): The time for the whole team to discover a certain percentage of all targets. Note that the time of discovering all targets is less than or equal to the coverage time. Only in the limiting case, when the last target is discovered in the last visited cell by the robot that stops last, the coverage time will be equal to the $ToTD$ for all targets.

The objective of MCPP is to achieve $CR = 1$ even under a few robot failures, hence maximizing $NoTF$, while minimizing $CT$, minimizing $ToTD$ and maximizing $RR$.

## 3.4 The CARE Algorithm

The CARE algorithm addresses the above-mentioned MCPP problem via facilitating distributed event-driven task reallocations. In CARE, a set of local robots jointly re-plan their task assignments in two situations: (1) when a robot has finished its current task, or (2) when a robot has failed and is detected as non-responsive. The re-planning strategy relies on a game-theoretic formulation, which computes the task worths and the success probabilities for each participating robot-task pair as optimization factors for optimal task reallocations. The task worths are measured by their expected number of undiscovered targets, while the success probabilities of robot-task pairs are computed based on the battery reliabilities, travel times, and predicted times to finish the contested tasks for the robots.

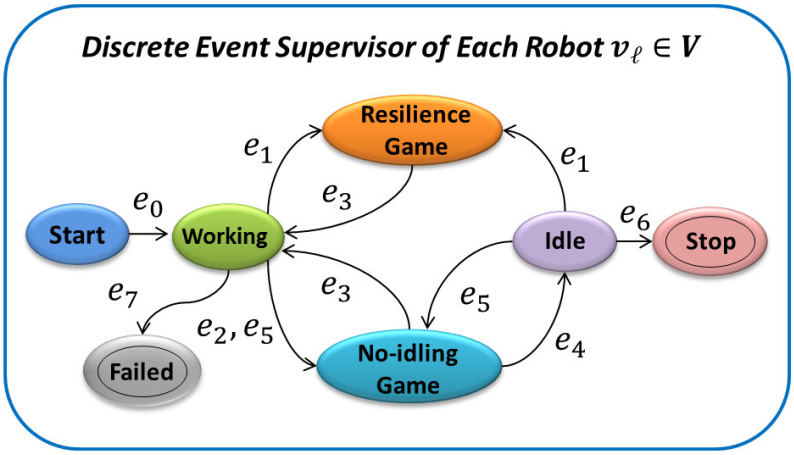
The CARE algorithm adopts a distributed yet cooperative control architecture, where each robot $v_\ell \in V$ is controlled by a Discrete Event Supervisor (DES) that is modeled as a finite state automaton.

### 3.4.1 Discrete Event Supervisor

The DES as shown in Fig. 3.3 is defined below.

**Definition 3.4.1 (DES).** *The DES, denoted as H, is a deterministic finite state automaton represented by a* 5*-tuple as follows*

$$H = (X, \mathcal{E}, \delta, x_0, X_m),$$




| *Events* | *Description* | *Generation Condition* |
|---|---|---|
| $e_0$ | Start exploration | Robot $v_\ell$ is turned on |
| $e_1$ | Neighbor failed | Confirmation of neighbor failure |
| $e_2$ | Own task completed | $n_U(r_c(v_\ell)) = 0$ |
| $e_3$ | Task assigned | Task assigned by the Optimizer |
| $e_4$ | No task assigned | No task assigned by the Optimizer |
| $e_5$ | Neighbor task completed | Some neighbor completes its task and $t_c(r_c(v_\ell)) \leq \eta$ |
| $e_6$ | All tasks completed | $\sum n_U(r) = 0$ |
| $e_7$ | Robot failed | Robot $v_\ell$ is diagnosed as failed |

**Figure 3.3:** The discrete event supervisor in the CARE algorithm

*where:*

- $X = \{ST, WK, NG, RG, ID, FL, SP\}$ *is the set of states, where* $ST \equiv$ *'Start',* $WK \equiv$ *'Working',* $NG \equiv$ *'No-idling Game',* $RG \equiv$ *'Resilience Game',* $ID \equiv$ *'Idle',* $FL \equiv$ *'Failed' and* $SP \equiv$ *'Stop'.*

- $\mathcal{E} = \{e_0, e_1, \ldots e_7\}$ *is the finite set of events.*

- $\delta : X \times \mathcal{E} \rightarrow X$ *is the partial state transition function. It is defined from one state to another if and only if there exists an arrow connecting them carrying an event.*

- $x_0 = ST$ *is the initial state.*

- $X_m = \{SP, FL\}$ *is the set of marked states, which means a robot can either stop after finishing all the tasks or it may fail unexpectedly.*

While the states $ST$, $ID$, $FL$ and $SP$ are self-explanatory, the operations in states $WK$, $NG$ and $RG$ are described as follows. In state $WK$, the supervisor $H$ of robot $v_\ell$ adopts the $\varepsilon^\star$ algorithm [85] for online coverage within its own task. Since no *a priori* information is available, all cells are initialized as unexplored. As the robot explores its task, it updates these cells as explored, obstacles and forbidden as suitable to track the progress of exploration. This information is then periodically shared and synchronized with other robots such that each robot maintains a symbolic map of the entire region.

In states $RG$ and $NG$, $H$ triggers the *Optimizer* to play resilience games and no-idling games, respectively. The objective of resilience games is to optimally re-organize the neighbors of the failed robot to immediately fill the coverage gap, if it contains higher worth; while for no-idling games, the objective is to optimally reallocate the idling robot and its near-finishing neighbors to help other robots to reduce coverage time and collect more worth early. Details of *Optimizer* functionality are explained later in Section 3.4.2.

**Events and State Transitions**: The events in $\mathcal{E}$ enable state transitions in $H$, which are explained below. First, we define:

1. $r_c : V \to \{1, \ldots M\}$ to be the allocation function that indicates the current task allocations of robots;

2. $t_c : \{1, \ldots M\} \to [0, \infty)$ to be the remaining time required to complete a given task by its assigned robots;

3. $n_U : \{1, \ldots M\} \to \mathbb{N}$ to be the number of unexplored cells in a given task.

Now, consider a robot $v_\ell \in V$ that is currently working in task $r_c(v_\ell)$. Event $e_0$ is generated when $v_\ell$ is turned on, and $H$ moves to the state $WK$ to start searching in task $r_c(v_\ell)$ using the $\varepsilon^\star$ algorithm.

Event $e_1$ is produced if any of its neighbor robot fails. This transitions $H$ to the state *RG*, that in turn invokes the *Optimizer* to play the resilience game to generate a task reallocation decision for $v_\ell$. Failure of a robot is detected using a standard mechanism based on heartbeat signals [86]. Each robot periodically broadcasts heartbeat signals, and also listening from others. Then a neighbor robot is detected as failed if its message is not received by $v_\ell$ constantly for a certain period of time. To ensure robustness to false alarms, its failure is further confirmed if the majority of $\kappa_2 \in \mathbb{N}^+$ neighbors detect its failure. Event $e_2$ occurs as soon as task $r_c(v_\ell)$ is completed, i.e., the number of unexplored cells in task $r_c(v_\ell)$, denoted as $n_U(r_c(v_\ell))$, becomes 0. Event $e_2$ moves $H$ to the state *NG*, where the *Optimizer* is called to play the no-idling game for finding a new task for $v_\ell$.

Event $e_3$ appears if the *Optimizer* assigns a new (or current) task to robot $v_\ell$, which drives $H$ back to the state $WK$ to search in the assigned task; otherwise if no task is assigned, event $e_4$ is generated that moves $H$ to the state *ID* and the robot becomes idle. Event $e_5$ is produced if some neighbor robot just completed its task and triggered the no-idling game, while $v_\ell$ is close to finish task $r_c(v_\ell)$, i.e., $t_c(r_c(v_\ell)) \leq \eta \in \mathbb{R}^+$, hence ready to reallocate after finishing the current task. Specifically, $t_c(r_c(v_\ell)) = \frac{n_U(r_c(v_\ell))}{\omega}$, where $\omega \in \mathbb{R}^+$ is the speed of tasking a cell by the assigned robots. Then again, $H$ comes to the state *NG* and the *Optimizer* is invoked to compute for a new task.

Event $e_6$ occurs when the entire area $\mathscr{R}$ is covered, by satisfying Eq. (3.2). This happens when no unexplored cells are left in the whole region, i.e., $\sum_{r=1}^{M} n_U(r) = 0$. This moves $H$ to the terminal state *SP* and the coverage is complete. At last, event $e_7$ is generated if $v_\ell$ itself is diagnosed as failed by its own diagnosis, and $H$ moves to the state $FL$. An advanced

failure diagnostic tool is beyond the scope of this chapter.

### 3.4.2 Distributed Optimizer

The *Optimizer* is invoked by the supervisor $H$ to compute reallocation decisions under two conditions: (i) $H$ reaches $RG$ state upon detection of a neighbor failure (i.e., event $e_1$); or (ii) $H$ reaches $NG$ state upon completion of its own task (i.e., event $e_2$), or completion of a neighbor's task (i.e., event $e_5$).

Specifically, the *Optimizer* is built based on the concept of *Potential Games* [67], which have the following advantages: (i) at least one Nash Equilibrium is guaranteed to exist, (ii) several learning algorithms are available (e.g., the Max-Logit algorithm [87][88]) that can converge fast to the optimal equilibrium, and (iii) the utility of each player is perfectly aligned with a globally shared potential function, thus as each player seeks to increase its own utility, the potential function is simultaneously improved and maximized upon reaching the optimal equilibrium.

Before presenting the details of the *Optimizer*, we list the various useful parameters in Table 3.3. Below, we present some mathematical preliminaries.

**Preliminaries**: A game $G$ in strategic form [89] consists of:

- A finite set of players $\mathcal{P} = \{\mathcal{P}_i \in V, i = 1, \ldots |\mathcal{P}|\}$, which includes all available robots that could be reallocated.

- A non-empty set of actions $\mathcal{A}_i$ associated to each player $\mathcal{P}_i$. In this chapter, each action $a_i \in \mathcal{A}_i$ corresponds to the index of an available task, and the action set is assumed identical for all players, i.e., $\mathcal{A}_i = \mathcal{A}_j = \tilde{\mathcal{A}}$, $\forall i, j \in \{1, \ldots |\mathcal{P}|\}$.

- The *utility function* associated with each player $\mathcal{P}_i$, defined as $\mathcal{U}_i : \mathcal{A}_{\mathcal{P}} \rightarrow \mathbb{R}$, where

**Table 3.3:** List of key parameters in CARE

| Parameter | Description |
|---|---|
| $N$ | Total number of robots |
| $M$ | Total number of tasks |
| $\rho_0$ | Curvature of the growth part in battery model |
| $\rho_1$ | Inflection point in battery model |
| $v$ | Robot traveling speed |
| $\omega$ | Robot tasking speed |
| $\lambda_r$ | Expected number of targets in task $r$ |
| $\eta$ | Threshold of time to identify robots that are close to finishing their tasks |
| $\psi$ | Threshold to identify incomplete tasks with sufficient work left |
| $\kappa_1$ | Neighborhood size in no-idling games |
| $\kappa_2$ | Neighborhood size in resilient games |

$\mathcal{A}_\mathcal{P} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_{|\mathcal{P}|}$ denotes the set of joint actions for all players.

The utility function computes the payoff that $\mathcal{P}_i$ can receive by taking an action $a_i \in \mathcal{A}_i$, given that the rest of the players jointly select $a_{-i} \in \mathcal{A}_{-i}$, where $\mathcal{A}_{-i} := \mathcal{A}_1 \times \ldots \times \mathcal{A}_{i-1} \times \mathcal{A}_{i+1} \times \ldots \times \mathcal{A}_{|\mathcal{P}|}$. A joint action of all players $a_\mathcal{P} \in \mathcal{A}_\mathcal{P}$ is often written as $a_\mathcal{P} = (a_i, a_{-i})$.

**Definition 3.4.2** (**Nash Equlibrium**). *A joint action $a^\star_\mathcal{P} = (a^\star_i, a^\star_{-i}) \in \mathcal{A}_\mathcal{P}$ is called a pure Nash Equilibrium if*

$$\mathcal{U}_i(a^\star_i, a^\star_{-i}) = \max_{a_i \in \mathcal{A}_i} \mathcal{U}_i(a_i, a^\star_{-i}),\ \forall \mathcal{P}_i \in \mathcal{P}. \tag{3.4}$$

**Definition 3.4.3** (**Potential Games**). *A game $G$ in strategic form with action sets $\{\mathcal{A}_i\}_{i=1}^{|\mathcal{P}|}$ together with utility functions $\{\mathcal{U}_i\}_{i=1}^{|\mathcal{P}|}$ is a potential game if and only if, a potential function*

$\phi : \mathcal{A}_{\mathcal{P}} \to \mathbb{R}$ *exists, s.t.* $\forall\, \mathcal{P}_i \in \mathcal{P}$

$$\mathcal{U}_i(a_i', a_{-i}) - \mathcal{U}_i(a_i'', a_{-i}) = \phi(a_i', a_{-i}) - \phi(a_i'', a_{-i}), \tag{3.5}$$

$\forall\, a_i', a_i'' \in \mathcal{A}_i$ *and* $\forall\, a_{-i} \in \mathcal{A}_{-i}$.

A potential game requires perfect alignment between the utility of an individual player and the globally shared potential function $\phi$ for all players, in the sense that the utility change by unilaterally deviating a player's action is equal to the amount of change in the potential function. In other words, the potential function $\phi$ can track the changes in payoffs as some player unilaterally deviates from its current action. Therefore, if $\phi$ is designed as the global objective, then as players negotiate towards maximizing their individual utilities, the global objective is simultaneously optimized.

Now, we present the resilience games and no-idling games modeled as potential games.

**Specifics of Resilience Games and No-idling Games**: Due to different objectives and triggering conditions, the player set and action set are fundamentally different for resilience games and no-idling games. Let $\mathcal{N}_{\kappa}^{v_\ell}$ be the set of $\kappa \in \mathbb{N}^+$ nearest neighbors of robot $v_\ell$.

• **No-idling Game**: A no-idling game is triggered when some robot $v_{id} \in V$ completes its current task and becomes idle. Then, it calls its $\kappa_1$ nearest neighbors $v_\ell \in \mathcal{N}_{\kappa_1}^{v_{id}}$ that are close to finish their tasks to participate in the game. Thus, a no-idling game comprises of:

- $\mathcal{P} = \{v_{id}\} \cup \{v_\ell \in \mathcal{N}_{\kappa_1}^{v_{id}} : t_c(r_c(v_\ell)) \leq \eta\}$.
- $\tilde{\mathcal{A}} = \{r \in \{1, \ldots M\} : t_c(r) \geq \psi \in \mathbb{R}^+\}$, which contains incomplete tasks that have sufficient work left to be finished by their currently assigned robots. If some players still have some work left in their current tasks, they are assigned such that they finish their current tasks before being reallocated to new tasks.

• **Resilience Game:** A resilience game is triggered when some robot $v_f$ fails. Then, the $\kappa_2$ nearest neighbors of $v_f$ are involved in the game to re-optimize their current task allocations. Thus, a resilience game comprises of:

- $\mathcal{P} = \mathcal{N}_{\kappa_2}^{v_f}$.
- $\tilde{\mathcal{A}} = \{r_c(v_f)\} \cup \{r_c(v_\ell), v_\ell \in \mathcal{N}_{\kappa_2}^{v_f} : t_c(r_c(v_\ell)) > \eta\}$, which contains the current tasks of all players and the failed robot. The condition $t_c(r_c(v_\ell)) > \eta$ ensures that those tasks close to be finished will be completed by their currently assigned robot and hence not needed to be part of the game.

**Remark 3.4.1.** *If there exist other active robots working in the same task of the failed robot, then they will take over this task and no resilience game is triggered.*

**Remark 3.4.2.** *When a game is initiated, the information is exchanged and synchronized between all players, including their locations, discovered environment maps, success probabilities and estimated task worths.*

Although the game specifics are different for the resilience and no-idling games, they follow the same design of the potential function and utility function as follows.

**Design of Potential Function for Task Reallocations**: As explained in Section 3.1, the players must analyze the following optimization factors during task reallocation:

1. Task worths, which can be quantified by the expected number of undiscovered targets in the tasks.

2. Probability of success of each player to finish a certain task, which depends on its current battery reliability, the cost of traveling to the new task, and the cost of finishing the new task.

Thus, the potential function $\phi$ for all players in the game is defined to be the total expected worth [90] obtained by choosing a joint action $a_{\mathcal{P}} \in \mathcal{A}_{\mathcal{P}}$, as follows.

$$\phi(a_{\mathcal{P}}) = \sum_{r \in \tilde{\mathcal{A}}} w_r \left(1 - \prod_{\mathcal{P}_i \in \{\mathcal{P}\}_r} \left[1 - p_r(\mathcal{P}_i)\right]\right), \tag{3.6}$$

where $\{\mathcal{P}\}_r \triangleq \{\mathcal{P}_i \in \mathcal{P} : a_i = r\}$ denotes the subset of players that choose the same task $r \in \{1, \ldots M\}$ in the joint action $a_{\mathcal{P}}$; $w_r$ is the current available worth of task $r$; and $p_r(\mathcal{P}_i)$ is the success probability of player $\mathcal{P}_i$ to finish task $r$. The term $p(r) := 1 - \prod_{\mathcal{P}_i \in \{\mathcal{P}\}_r} \left[1 - p_r(\mathcal{P}_i)\right]$ is the joint success probability for all players to finish task $r$ together.

As exploration continues, the conditions of robots and tasks change dynamically. Thus, the success probability $p_r(\mathcal{P}_i)$ and the task worth $w_r$ in Eq. (3.6) must be updated before a game is played.

**Computation of Success Probability**: The success probability $p_r(\mathcal{P}_i)$ is evaluated online using Eq. (3.1) as follows.

$$p_r(\mathcal{P}_i) = R_{\mathcal{P}_i}(\tilde{t}), \tag{3.7}$$

where $R_{\mathcal{P}_i}(\tilde{t})$ is the reliability of player $\mathcal{P}_i$ at time $\tilde{t}$, which is estimated as

$$\tilde{t} = t_k + t_{tr} + t_r, \tag{3.8}$$

where $t_k$ is the total tasking time of $\mathcal{P}_i$ since the beginning until the game was initiated, $t_{tr}$ is the traveling time to task $r$, and $t_r$ is the estimated time to complete task $r$. Specifically, $t_{tr} = \frac{Dist(\mathcal{P}_i, r)}{v}$, where $Dist(\mathcal{P}_i, r)$ measures the distance between player $\mathcal{P}_i$'s current location and the centroid of task $r$, and $v \in \mathbb{R}^+$ is the traveling speed of robot; and the time $t_r = \frac{n_U(r)}{\omega}$, where $\omega$ is the speed of tasking a cell by the assigned robots.

In addition, if a robot is selected as a player to find a new task but it still has a small

portion left in its current task, then it would like to first finish this task before being reallocated to a new one. Hence, an extra term $t_c$ is included in Eq. (3.8) if the estimated time to complete the unfinished part of its current task $r_c(\mathcal{P}_i)$ satisfies $t_c(r_c(\mathcal{P}_i)) \leq \eta$.

**Computation of Task Worths**: The worth $w_r$ in Eq. (3.6) indicates the expected number of undiscovered targets in task $r$ that are available to the players. Let $\mathbf{x_r}$ be a random variable that denotes the total number of targets in task $r$, which is assumed to follow the Poisson distribution with parameter $\lambda_r$. Its probability mass function is given as:

$$Pr\big(\mathbf{x_r} = x\big) = e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!},\ x = 0, 1, 2 \ldots \tag{3.9}$$

If $\xi$ targets have been already discovered in task $r$, then the estimated remaining number of targets, $\tilde{w_r}$, is computed as:

$$\begin{aligned} \tilde{w_r} &= \sum_{x=\xi+1}^{\infty} (x-\xi) \cdot e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} \\ &= \sum_{x=0}^{\infty} x \cdot e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} - \xi \cdot \sum_{x=0}^{\infty} e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} - \sum_{x=0}^{\xi} (x-\xi) \cdot e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} \end{aligned}$$

By definition, Poisson distribution has mean $\lambda_r$, i.e., $\sum_{x=0}^{\infty} x \cdot e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} = \lambda_r$. Also, one has $\sum_{x=0}^{\infty} e^{-\lambda_r} \cdot \frac{{\lambda_r}^x}{x!} = 1$. Thus, $\tilde{w_r}$ is computed as:

$$\tilde{w_r} = (\lambda_r - \xi) + e^{-\lambda_r} \cdot \sum_{x=0}^{\xi} (\xi - x) \cdot \frac{{\lambda_r}^x}{x!} \tag{3.10}$$

Next, we decide the portion of $\tilde{w_r}$ available to the players, i.e., $w_r$. Since task $r$ may contain some robots currently working there but are not participating in the task reallocation, i.e., they are not players, then if a player selects task $r$, it must work together with these existing robots. In turn, the maximum payoff a player could expect from task $r$ be-

comes less due to sharing with the existing robots. Let $\bar{\mathcal{P}} \triangleq V \setminus \mathcal{P}$ denote the subset of robots that are not players. Similarly, let $\{\bar{\mathcal{P}}\}_r$ be the set of non-player robots that are currently working in task $r$, which have a joint success probability $q(r)$ for task $r$, i.e., $q(r) = 1 - \prod_{v_\ell \in \{\bar{\mathcal{P}}\}_r} \big(1 - p_r(v_\ell)\big)$. Then $w_r$ is computed as:

$$w_r = \tilde{w}_r \cdot \big(1 - q(r)\big). \tag{3.11}$$

**Utility Function of Each Player**: In order to form a potential game, the utility function, together with the potential function defined in Eq. (3.6), must satisfy Eq. (3.5). Since the utility of a player also depends on the actions taken by the rest of the players, thus a rule is needed to distribute the total produced payoff among contributing players. In this regard, this chapter adopts the concept of *Marginal Contribution* due to its low computation burden thus feasible for online decision-making [90].

**Definition 3.4.4** (**Marginal Contribution**). *The marginal contribution of player $\mathcal{P}_i$ in a joint action $a_{\mathcal{P}} = (a_i, a_{-i})$ is*

$$\mathcal{MC}_i = \phi(a_i, a_{-i}) - \phi(\emptyset, a_{-i}), \tag{3.12}$$

*where $\emptyset$ represents player $\mathcal{P}_i$'s null action, indicating no task is assigned to it.*

The utility function is derived as follows. First, substitute Eq. (3.6) into Eq. (3.12), one has:

$$
\begin{aligned}
\mathcal{U}_i(a_i, a_{-i}) &= \mathcal{MC}_i \\
&= \sum_{r \in \tilde{\mathcal{A}}} w_r \left(1 - \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_r} \left[1 - p_r(\mathcal{P}_j)\right]\right) - \sum_{r \in \tilde{\mathcal{A}}} w_r \left(1 - \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_r \setminus \mathcal{P}_i} \left[1 - p_r(\mathcal{P}_j)\right]\right) \\
&= \sum_{r \in \tilde{\mathcal{A}}} w_r \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_r \setminus \mathcal{P}_i} \left[1 - p_r(\mathcal{P}_j)\right] - \sum_{r \in \tilde{\mathcal{A}}} w_r \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_r} \left[1 - p_r(\mathcal{P}_j)\right]
\end{aligned}
$$

Note that for any task $r$ not selected by player $\mathcal{P}_i$, i.e., $r \neq a_i$, one has $\{\mathcal{P}\}_r = \{\mathcal{P}\}_r \setminus \mathcal{P}_i$. Thus, the produced potentials in these tasks are canceled in the above equation. It can then be further simplified as below, where $w_{a_i}$ is the worth of task $a_i$.

$$
\begin{aligned}
\mathcal{U}_i(a_i, a_{-i}) &= w_{a_i} \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_{a_i} \setminus \mathcal{P}_i} \left[1 - p_{a_i}(\mathcal{P}_j)\right] - w_{a_i} \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_{a_i}} \left[1 - p_{a_i}(\mathcal{P}_j)\right] \\
&= w_{a_i} \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_{a_i} \setminus \mathcal{P}_i} \left[1 - p_{a_i}(\mathcal{P}_j)\right] - w_{a_i} \cdot \left[1 - p_{a_i}(\mathcal{P}_i)\right] \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_{a_i} \setminus \mathcal{P}_i} \left[1 - p_{a_i}(\mathcal{P}_j)\right] \\
&= w_{a_i} \cdot p_{a_i}(\mathcal{P}_i) \cdot \prod_{\mathcal{P}_j \in \{\mathcal{P}\}_{a_i} \setminus \mathcal{P}_i} \left[1 - p_{a_i}(\mathcal{P}_j)\right] \qquad (3.13)
\end{aligned}
$$

**Proposition 3.4.1.** *The game G with potential function $\phi$ of Eq. (3.6) and the utility function $\mathcal{U}_i$ of Eq. (3.13) is a potential game.*

*Proof.* Given a joint action $a_{-i}$, the difference in potential $\phi$ when player $\mathcal{P}_i$ deviates its action from $a_i'$ to $a_i''$ is:

$$
\begin{aligned}
&\phi(a_i', a_{-i}) - \phi(a_i'', a_{-i}) \\
&= (\phi(a_i', a_{-i}) - \phi(\emptyset, a_{-i})) - (\phi(a_i'', a_{-i}) - \phi(\emptyset, a_{-i})) \\
&= \mathcal{U}_i(a_i', a_{-i}) - \mathcal{U}_i(a_i'', a_{-i})
\end{aligned}
$$

Thus game $G$ satisfies Eq. (3.5) and it is a potential game. □

In this chapter, the optimal equilibrium $a^\star_\mathcal{P}$ is acquired using the Max-Logit algorithm [87]. Before any game starts, each player computes its success probability $p_r(\mathcal{P}_i), \forall r \in \tilde{\mathcal{A}}$ using Eq. (3.7), and updates the estimated task worth $w_r, \forall r \in \tilde{\mathcal{A}}$ using Eq. (3.10). Then, necessary information are communicated and synchronized as mentioned in Remark 3.4.2.

Algorithm 2 presents details to acquire $a^\star_\mathcal{P}$ in a distributed manner using the Max-Logit algorithm. In particular, the initial joint action $a_\mathcal{P}(1)$ (**Line 1**) is initialized as follows: for resilience games, $a_i(1)$ is set as the current task $r_c(\mathcal{P}_i)$ of player $\mathcal{P}_i$, while for no-idling games, $a_i(1)$ is randomly picked from $\tilde{\mathcal{A}}$; then, $a_\mathcal{P}(1)$ is determined via synchronization with all other players.

Once $a^\star_\mathcal{P}$ is obtained using Algorithm 2, the new task $r$ for player $\mathcal{P}_i$ is set as its action $a^\star_i$ in the equilibrium $a^\star_\mathcal{P}$.

**Post-game Coordination**: If multiple robots (including both existing robots and incoming players) are assigned to the same task $r$, it becomes imperative to utilize some strategy to ensure their safety and efficiency when searching together. Let $n_{\max} \in \mathbb{N}^+$ be the maximum number of robots allowed to work in the same task at the same time. In this regard, task $r$ is evenly partitioned into $n_{\max}$ sub-regions, where each sub-region is only allowed one robot at a time.

Consider some non-player robot $v_\ell \in \{\bar{\mathcal{P}}\}_r$ that is currently working in task $r$. It continues as usual but its task is restricted to the sub-region determined by its current location. This produces $n_0 \in \mathbb{N}$ incomplete sub-regions that are instantly available to the incoming players. Now consider a player $\mathcal{P}_i \in \{\mathcal{P}\}^\star_r \triangleq \{\mathcal{P}_i \in \mathcal{P} : a^\star_i = r\}$ that is also assigned to task $r$. It selects the sub-region by following the steps below. First, it computes its rank in $\{\mathcal{P}\}^\star_r$ based on its success probability. If it ranks in the top $n_0$ and all other players ranked above it have selected their new sub-region, then it selects the new available sub-region for itself

**Algorithm 2:** The Optimizer for $\mathcal{P}_i$ using Max-Logit [87]

**input :** $w_r$, $p_r(\mathcal{P}_i), \forall r \in \tilde{\mathcal{A}}$, and $\mathcal{P}_i \in \mathcal{P}$
**output:** $a^{\star}_{\mathcal{P}}$

Initialize: set initial joint action $a_{\mathcal{P}}(1) \in \mathcal{A}_{\mathcal{P}}$ using a fixed rule, set the learning parameter $\chi$ and the number of computation cycles $Z$
**for** $k \leftarrow 1$ **to** $Z$ **do**
    Determine randomly if $\mathcal{P}_i$ is the single player among others that may alter its action $a_i(k)$;
    **if** $\mathcal{P}_i$ *is not selected* **then**
        Repeat $a_i(k+1) = a_i(k)$;
        Continue;
    **else**
        Select an alternative action $\hat{a}_i(k) \in \mathcal{A}_i$ with equal probability;
        Compute alternative utility $\mathcal{U}_i(\hat{a}_i(k), a_{-i}(k))$ using Eq. (3.13);
        Compute $\mu = e^{\mathcal{U}_i(\hat{a}_i, a_{-i})/\chi} / \max\{e^{\mathcal{U}_i(a_i, a_{-i})/\chi}, e^{\mathcal{U}_i(\hat{a}_i, a_{-i})/\chi}\}$;
        Update $a_i(k+1)$ as follows:

$$a_i(k+1) = \begin{cases} \hat{a}_i(m), & \text{with probability } \mu \\ a_i(m), & \text{with probability } 1-\mu. \end{cases}$$

        Inform $a_i(k+1)$ to other players $\mathcal{P}_j, j \in \mathcal{P} \setminus \mathcal{P}_i$;
    **end**
**end**
Return $a^{\star}_{\mathcal{P}} = a(k+1)$.

that minimizes its traveling distance. However, if it ranks after $n_0$, it stays temporarily idle but can later be reactivated to replace any robot in task $r$ should it fail. Once $\mathcal{P}_i$ finds a new sub-region, its centroid is set as the movement goal. As described previously, $\mathcal{P}_i$ resumes to search its new sub-region using the $\varepsilon^{\star}$ algorithm upon its arrival, and its supervisor $H$ transitions to the state $WK$ accordingly.

### 3.4.3 Computational Complexity of the Optimizer

As described above, once the *Optimizer* triggers a game involving the player set $\mathcal{P}$ and the action set $\tilde{\mathcal{A}}$, the joint action $a_{\mathcal{P}}$ is first initialized locally and then synchronized with other players. This process takes $O(|\mathcal{P}|)$ complexity.

Thereafter, the game follows Algorithm 2 in a distributed manner, which operates in a loop for a user-defined $Z \in \mathbb{N}^+$ computation cycles. At each cycle, one player is randomly selected and is allowed to probabilistically alter its action, which takes $O(|\mathcal{P}|)$ to find out if $\mathcal{P}_i$ is selected. If not, its action is repeated, which takes $O(1)$ complexity; otherwise, $\mathcal{P}_i$ first randomly chooses an alternative action $\hat{a}_i \in \tilde{\mathcal{A}}$ with equal probability, which is $O(|\tilde{\mathcal{A}}|)$. Then the associated utility $\mathcal{U}_i(\hat{a}_i, a_{-i})$ is computed using Eq. (3.13), which takes $O(|\mathcal{P}|)$ complexity. Thereafter, $\mathcal{P}_i$ uses $\hat{a}_i$ to update its action $a_i$ in a probabilistic manner, which has $O(1)$. At the end of each cycle, the updated action $a_i$ is transmitted to other players, which requires $O(|\mathcal{P}|)$ complexity.

Therefore, in the worst case where $\mathcal{P}_i$ is selected in every cycle, the total complexity becomes $O(|\mathcal{P}| + Z \cdot (3|\mathcal{P}| + |\tilde{\mathcal{A}}|))$. In comparison, for a centralized optimization algorithm, it must search over $|\tilde{\mathcal{A}}|^{|\mathcal{P}|}$ possible joint actions, which grows significantly faster as $|\mathcal{A}|$ and $|\mathcal{P}|$ increase.

### 3.4.4 Connection between Local Games and Team Potential

As discussed earlier, in both resilience games and no-idling games, the potential function $\phi$ is optimized for the set of players, which form a subset of the robot team. Now, we show that the increase in $\phi$ will directly improve the performance of the whole team.

To illustrate this, let $\Phi(a)$ denote the total team potential that defines the total expected worth achievable by the team, where $a = (a_{\mathcal{P}}, a_{\bar{\mathcal{P}}})$ is the joint action of the team including

players $\mathcal{P}$ and non-players $\bar{\mathcal{P}}$. Note for the non-players, the action $a_{\bar{\mathcal{P}}}$ simply represent their current tasks. Since the players and non-players are mixed and distributed over different tasks, the total team potential $\Phi(a)$ is defined as:

$$\Phi(a) = \sum_{r=1}^{M} \tilde{w}_r \Big(1 - \prod_{v_\ell \in \{V\}_r} [1 - p_r(v_\ell)]\Big), \tag{3.14}$$

where $\{V\}_r = \{\mathcal{P}\}_r \cup \{\bar{\mathcal{P}}\}_r$ is the set of all robots that are assigned to task $r$ in the joint action $a$, and the term within the parentheses on the right hand side computes the joint success probability to complete task $r$ by all of its assigned robots.

As the players reach the optimal equilibrium, the joint action becomes $a^\star = (a^\star_{\mathcal{P}}, a_{\bar{\mathcal{P}}})$ and the team potential becomes $\Phi(a^\star)$.

**Theorem 3.4.1.** *The optimal equilibrium $a^\star$ increases the total team potential $\Phi(a)$, i.e., $\Phi(a^\star) \geq \Phi(a)$.*

*Proof.* First, let us show that the team potential $\Phi(a)$ is separable by the worth created by the players (i.e., $\mathcal{P}$) and the rest of the robots (i.e., $\bar{\mathcal{P}}$). Then we will investigate the change of $\Phi$ due to task reallocation. Now $\Phi(a)$ can be decomposed as follows.

$$\begin{aligned}
\Phi(a) &= \sum_{r=1}^{M} \tilde{w}_r \Big(1 - \prod_{\mathcal{P}_i \in \{\mathcal{P}\}_r} [1 - p_r(\mathcal{P}_i)] \cdot \prod_{v_\ell \in \{\bar{\mathcal{P}}\}_r} [1 - p_r(v_\ell)]\Big) \\
&= \sum_{r=1}^{M} \tilde{w}_r \Big(1 - [1 - p(r)] \cdot [1 - q(r)]\Big)
\end{aligned} \tag{3.15}$$

where $p(r) := 1 - \prod_{\mathcal{P}_i \in \{\mathcal{P}\}_r} [1 - p_r(\mathcal{P}_i)]$ and $q(r) := 1 - \prod_{v_\ell \in \{\bar{\mathcal{P}}\}_r} [1 - p_r(v_\ell)]$ are used to denote the joint success probability of the players and the rest of the robots that are assigned to task $r$ in the joint action $a$, respectively.

Then we can further break down $\Phi(a)$ as follows.

$$
\begin{aligned}
\Phi(a) &= \sum_{r=1}^{M} \tilde{w}_r \Big( 1 - [1 - q(r)] + p(r)[1 - q(r)] \Big) \\
&= \sum_{r=1}^{M} \tilde{w}_r \cdot [1 - q(r)] \cdot p(r) + \sum_{r=1}^{M} \tilde{w}_r \cdot q(r) \\
&= \sum_{r=1}^{M} w_r \cdot p(r) + \sum_{r=1}^{M} \tilde{w}_r \cdot q(r) \\
&= \Big( \sum_{r \in \tilde{\mathcal{A}}} w_r \cdot p(r) + \sum_{r \notin \tilde{\mathcal{A}}} w_r \cdot p(r) \Big) + \sum_{r=1}^{M} \tilde{w}_r \cdot q(r) \\
&= \phi(a_{\mathcal{P}}) + \sum_{r \notin \tilde{\mathcal{A}}} w_r \cdot p(r) + \sum_{r=1}^{M} \tilde{w}_r \cdot q(r)
\end{aligned} \tag{3.16}
$$

where the second term in the last step is the worth generated by the players (if any) that would like to finish the small unfinished part in their current tasks before being reallocated to new tasks, while the third term indicates the worth generated by the non-player robots $\bar{\mathcal{P}}$. The values of both these terms do not change by games. Since $\phi(a^{\star}_{\mathcal{P}}) \geq \phi(a_{\mathcal{P}})$, $\forall a_{\mathcal{P}} \in \mathcal{A}_{\mathcal{P}}$, we have $\Phi(a^{\star}) \geq \Phi(a)$. □

**Game Performance Metrics:** The quality of the task reallocation decision (i.e., $a^{\star}_{\mathcal{P}}$ for the players and $a^{\star}$ for the team) can be evaluated by the worth gain. Note that in any task reallocation, there is a tradeoff between whether the robot should continue with its current task or reallocate to a new task. Thus, where a higher gain implies early detection of targets. Specifically, at player-level, the *Gain of Players* ($G_P$) is defined as

$$
G_P = \frac{\phi(a^{\star}_{\mathcal{P}}) - \phi(a_{\mathcal{P}})}{\sum_{r \in \tilde{\mathcal{A}}} w_r} \in [0, 1]. \tag{3.17}
$$

Similarly, at team-level, the *Gain of Team* ($G_T$) is

$$
G_T = \frac{\Phi(a^{\star}) - \Phi(a)}{\sum_{r=1}^{M} \tilde{w}_r} \in [0, 1]. \tag{3.18}
$$

Note that since $\phi(a^{\star}_{\mathcal{P}}) \geq \phi(a_{\mathcal{P}})$ and $\Phi(a^{\star}) \geq \Phi(a)$, both $G_p$ and $G_T$ are non-negative, which implies that the outcome of a game results in the gain of worth not only for the players but also for the whole team. Both $G_P$ and $G_T$ will be quantitatively examined in Section 3.5.

### 3.4.5 Complete Coverage under Failures

The success of finding all hidden targets relies on the complete coverage of the whole area $\mathscr{R}$. Due to the completeness of the $\varepsilon^{\star}$ algorithm, each task can be fully covered by the assigned robot in finite time if it stays alive. Now, let us examine coverage under failures.

**Theorem 3.4.2.** *The CARE algorithm guarantees complete coverage in finite time as long as one robot is alive.*

*Proof.* Consider a robot $v_{\ell}$ that is alive during the whole operation, whose supervisor $H$ starts with the state $WK$ upon robot being turned on. We show below that $v_{\ell}$ must reach the terminal state $SP$ in finite time, which happens if and only if $\sum_{r=1}^{M} n_U(r) = 0$, i.e., complete coverage.

First, as shown in Fig. 3.3, any cycle between states in $H$ involves either state $NG$ or $RG$. Also, a robot can reach the states $NG$ or $RG$ due to completion of some task or failure of some robot, respectively. Now, since there are only a finite number of robots (i.e., $N$) and a finite number of tasks (i.e., $M$), each robot can visit these states only a finite number of times. Thus, $H$ cannot have any live lock. Moreover, in states $NG$ or $RG$, it takes a finite amount of time to reach an equilibrium solution using the Max-Logit algorithm. Thus, $H$ will always switch to either state $WK$ or $ID$ after games. In state $WK$, the underlying $\varepsilon^{\star}$ algorithm is used to explore in the current assigned task $r_c(v_{\ell})$ of robot $v_{\ell}$. Since $\varepsilon^{\star}$

constantly reduces $n_U(r_c(v_\ell))$ until task $r_c(v_\ell)$ is completed in finite time, so $H$ can only stay in state $WK$ for finite time.

Further, in state $ID$, $H$ can either be invoked to play new games and hence move to states $NG$ or $RG$, or it can move to the state $SP$ upon complete coverage, i.e., $\sum_{r=1}^{M} n_U(r) = 0$. Since the former case can only happen for a finite number of times, $H$ will come back to the state $ID$ when no incomplete task is available to $v_\ell$ anymore. The same logic applies to all other active robots. Thus, all active robots including $v_\ell$ will reach state $ID$ in finite time, which implies that no incomplete tasks exist, i.e., $\sum_{r=1}^{M} n_U(r) = 0$. Then, they all transition to the terminal state $SP$ and the complete coverage is achieved. □

## 3.5 Results and Discussion

The CARE algorithm was validated on the high-fidelity robotic simulator Player/Stage [62], using a computer with 3.40 GHZ CPU and 16 GB RAM.

In this section, we present the performance of the CARE algorithm in three complex obstacle-rich scenarios. The search area $\mathscr{R}$ of size 50m $\times$ 50m, was partitioned into a $50 \times 50$ tiling consisting of $\varepsilon$-cells of size 1m $\times$ 1m. The $\mathscr{R}$ was partitioned into $M = 10$ tasks $\{\mathscr{R}_r, r = 1, \ldots 10\}$, each of size 10m $\times$ 25m. Each task $r \in \{1, \ldots 10\}$ was initially assigned with one robot, and a maximum number of $n_{\max} = 4$ robots were allowed to search together in one task. Each task $r$ contained an unknown number of targets distributed randomly according to the Poisson distribution with mean $\lambda_r \in \{1, \ldots 32\}$.

A team of $N = 10$ Pioneer 2AT robots was simulated, where each robot has dimensions of 0.44m $\times$ 0.38m $\times$ 0.22m, and was equipped with 16-beam laser scanners with a detection range of $R_s = 5$m. The kinematic constraints of the robot, such as the top speed of 0.4m/s

and the minimum turn radius of 0.04m, were included in the simulations. The tasking speed was $\omega = 0.32$ cell/s. The parameters $\rho_0$ and $\rho_1$ in the battery reliability model were chosen such that each robot can finish one and two tasks with more than 0.9 and 0.4 remaining reliability, respectively. Specifically, based on the size of each task ($\sim 250$ cells) and the robot tasking speed, it takes $\sim 780$s to finish an obstacle-free task. Then, using Eq. (3.1):

$$\begin{cases} \frac{1}{1+e^{\rho_0(780-\rho_1)}} & = 0.9 \\ \frac{1}{1+e^{\rho_0(2\times 780-\rho_1)}} & = 0.4 \end{cases}, \tag{3.19}$$

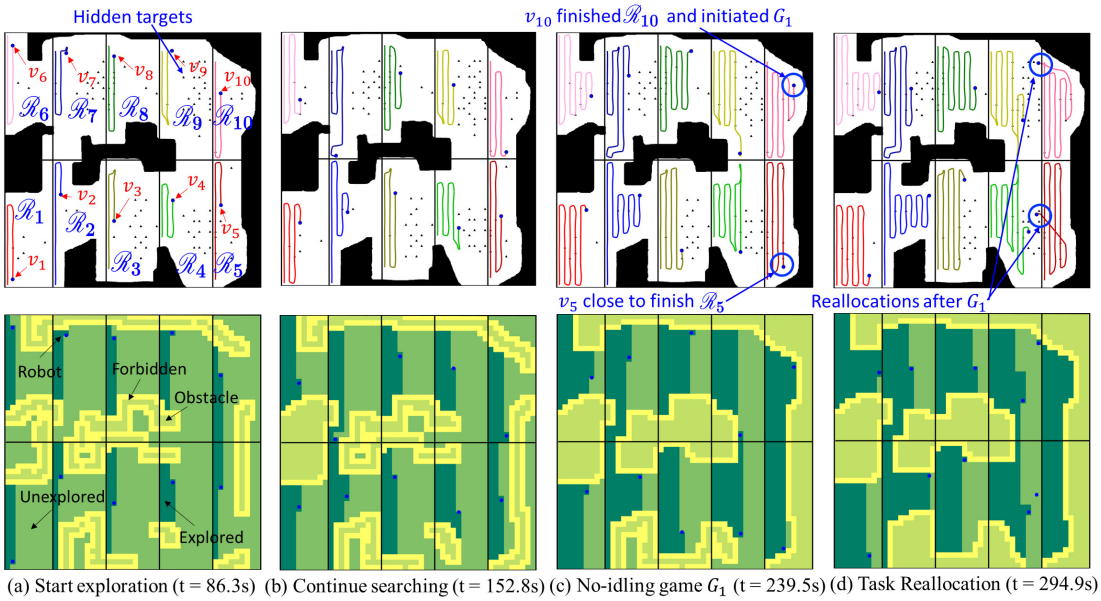
which lead to $\rho_0 \sim 3.0 \times 10^{-3}$ and $\rho_1 \sim 1400$s. Then, considering stochastic uncertainties in the initial battery charging conditions, these parameters are generated on different robots using Gaussian distributions, s.t., $\rho_0 \sim N(3 \times 10^{-3}, 7.5 \times 10^{-5})$ and $\rho_1 \sim N(1400, 35)$, where the standard deviation is chosen as 2.5% of the corresponding mean value.

Initially, due to lack of *a priori* knowledge of the environment, all $\varepsilon$-cells are initialized as unexplored, and as the robot explores the environment, the obstacle and forbidden cells are discovered and updated accordingly. The game parameters are chosen as: $\kappa_1 = 6$ and $\kappa_2 = 3$, and in Max-Logit, the number of computation cycles is set as $Z = 50$ and the learning parameter is $\chi = 0.05$. The other parameters $\eta$ and $\psi$ are chosen as follows. We set $\eta$ such that it corresponds to less than 4% of the time to finish one task, i.e., $780 \times 4\% = 31.2$s. Thus, robots which have only 4% of the task left will participate as players for no-idling games. Similarly, we set $\psi$ such that it corresponds to over 25% of the time to finish one task, i.e., $780 \times 25\% = 195$s. Thus, tasks which have still more than 25% unexplored area become contested tasks. Hence, further rounding up we used $\eta = 30$s and $\psi = 200$s.

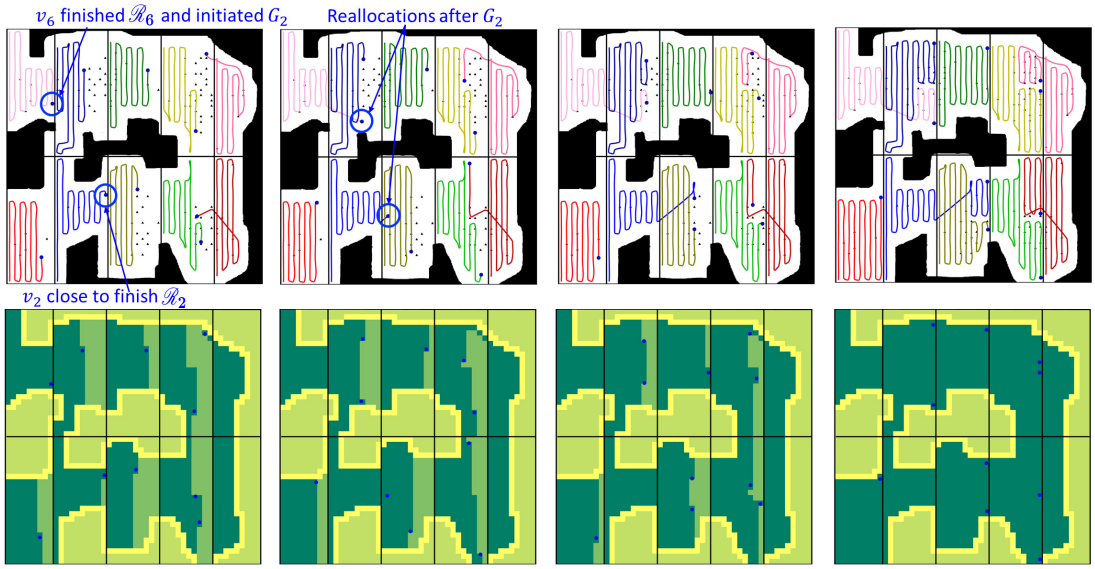
### 3.5.1 Scenario 1: No Failures but Some Robots Idle

Fig. 3.4 presents the cooperative coverage of a complex islands scenario. A total number of 107 targets were distributed randomly in the field. No failure appeared throughout the whole search, while two no-idling games were triggered to reallocate early completed robots to reduce the coverage time. Each subfigure in Fig. 3.4, i.e., Fig. 3.4a$\sim$Fig. 3.4h, is comprised of a top figure showing the trajectories of robots by different colors, and a bottom figure showing the corresponding overall symbolic map of the entire search area $\mathscr{R}$, which is periodically synchronized and merged by all live robots. The different colors in the symbolic map represent the following regions: i) light green for obstacles, ii) medium green for unexplored, iii) dark green for explored with no obstacles, and iv) yellow for the forbidden region around the obstacles.

Fig. 3.4a shows that the robots started exploration and used their on-board sensing systems to explore the *a priori* unknown environment. Fig. 3.4b shows that the robots continue searching within their assigned tasks. Fig. 3.4c shows the instance when robot $v_{10}$ finished task 10 and triggered a no-idling game $G_1$. The player set was formed as $\mathcal{P} = \{v_5, v_{10}\}$, where $v_5$ was near finishing its task. At that moment, tasks 3, 4, 7 and 9 still had a lot of area unexplored and required significant time to finish by their currently assigned robots, thus they formed the action set $\tilde{\mathcal{A}} = \{3,4,7,9\}$. The optimal equilibrium of $G_1$ reassigned $v_5$ and $v_{10}$ to task 4 and task 9, respectively. Because task 10 was already completed, $v_{10}$ immediately traveled to its new task 9, while $v_5$ had to first finish the remainder of its current task 5 before moving to task 4. Since there was a robot $v_9$ currently working in task 9, the post-game coordination strategy was used to further partition task 9 into $n_{\max} = 4$ sub-regions. As observed in Fig. 3.4d, $v_{10}$ selected the closest sub-region in the upper right corner and searched in parallel with $v_9$. Similar post-game coordination was performed




(a) Start exploration (t = 86.3s) (b) Continue searching (t = 152.8s) (c) No-idling game $G_1$ (t = 239.5s) (d) Task Reallocation (t = 294.9s)




(e) No-idling game $G_2$ (t = 303.3s) (f) Task Reallocation (t = 345.5s) (g) Continue searching (t = 400.2s) (h) Complete coverage (t = 557.9s)

**Figure 3.4:** Scenario 1: Coverage trajectories and the corresponding symbolic encodings using CARE

when $v_5$ joined to search with $v_4$ in task 4.

Later, another no-idling game $G_2$ was triggered when $v_6$ finished task 6, as shown in Fig. 3.4e. The player set was formed as $\mathcal{P} = \{v_2, v_6\}$, where $v_2$ was near finishing its task. Since tasks 4 and 9 have been assigned with extra robots after game $G_1$, the estimated time

| Games | Triggering Condition | Players $\mathcal{P}$ | Actions $\tilde{\mathcal{A}}$ | Task Worth $w_r$ | Success Probabilities $p_r(v_\ell)$ | Equilibrium $a^*_{\mathcal{P}}$ | $G_P$ | $G_T$ |
|---|---|---|---|---|---|---|---|---|
| $G_1$ (No-idling) | Task Done of $v_{10}$ | $\{v_5, v_{10}\}$ | $\{3, 4, 7, 9\}$ | $w_3 = 0.852$<br>$w_4 = 1.22$<br>$w_7 = 0.82$<br>$w_9 = 1.31$ | $\mathcal{R}_3$ $\mathcal{R}_4$ $\mathcal{R}_7$ $\mathcal{R}_9$<br>$v_5$ 0.908 0.896 0.890 0.889<br>$v_{10}$ 0.900 0.883 0.899 0.905 | $(4, 9)$ | 54.23 % | 3.29 % |
| $G_2$ (No-idling) | Task Done of $v_6$ | $\{v_2, v_6\}$ | $\{3, 7\}$ | $w_3 = 0.79$<br>$w_7 = 0.714$ | $\mathcal{R}_3$ $\mathcal{R}_7$<br>$v_2$ 0.925 0.916<br>$v_6$ 0.914 0.924 | $(3, 7)$ | 92.45 % | 2.42 % |

**Figure 3.5:** Scenario 1: Summary of game specifics and performances

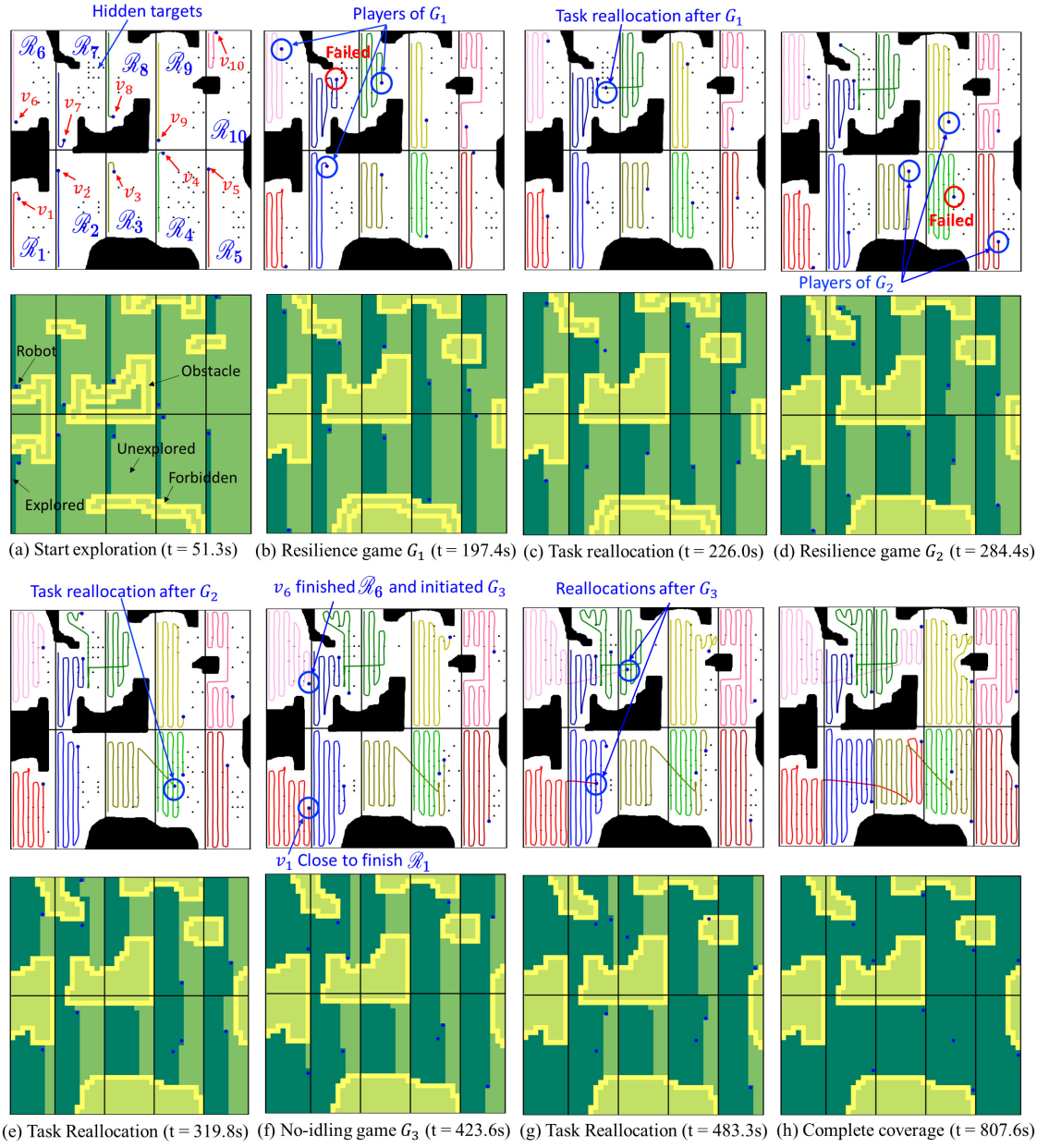
to finish these tasks dropped significantly, hence they were excluded from game $G_2$. The tasks 3 and 7, however, still required significant time to finish, thus they formed the action set $\tilde{\mathcal{A}} = \{3, 7\}$. The optimal equilibrium of $G_2$ in turn reassigned $v_2$ and $v_6$ to task 3 and task 7, respectively, as shown in Fig. 3.4f. It is observed in Fig. 3.4g that, robot $v_2$ selected the upper right sub-region of task 3 and continued searching in parallel with robot $v_3$, while robot $v_6$ joined robot $v_7$ to search task 7 in a similar fashion. Finally, complete coverage was achieved with all targets discovered, as shown in Fig. 3.4h.

Fig. 3.5 summarizes the specifics and performance of the two games. As observed, the player-level worth gain $G_P$ reached 54.23% and 92.45% in games $G_1$ and $G_2$, respectively, which means that after task reallocations, the idling robots can expect a higher number of targets from the remaining tasks. At the team-level, $G_T$ is 3.29% for $G_1$ and 2.42% for $G_2$, thus the whole team also benefits from the task reallocations.

### 3.5.2 Scenario 2: Some Robots Fail and Some Idle

Fig. 3.6 presents a more complex scenario where two robots failed unexpectedly during operation. A total number of 106 targets were randomly distributed in the field.

Fig. 3.6a shows that the robots start exploration while using their on-board sensing systems to discover the environment. Fig. 3.6b shows that robot $v_7$ failed unexpectedly and a resilience game $G_1$ was triggered involving $\kappa_2 = 3$ of its closest neighbors. The player set




(a) Start exploration (t = 51.3s) (b) Resilience game $G_1$ (t = 197.4s) (c) Task reallocation (t = 226.0s) (d) Resilience game $G_2$ (t = 284.4s)

(e) Task Reallocation (t = 319.8s) (f) No-idling game $G_3$ (t = 423.6s) (g) Task Reallocation (t = 483.3s) (h) Complete coverage (t = 807.6s)

**Figure 3.6:** Scenario 2: Coverage trajectories and the corresponding symbolic encodings using CARE

was formed as $\mathcal{P} = \{v_2, v_6, v_8\}$. The action set consisted of the current tasks of all players, as well as the task belonging to the failed robot, i.e., $\tilde{\mathcal{A}} = \{2, 6, 7, 8\}$. As seen in Fig. 3.6c, the optimal equilibrium of $G_1$ immediately reallocated $v_8$ to drop its current task and help

| Games | Triggering Condition | Players $\mathcal{P}$ | Actions $\tilde{\mathcal{A}}$ | Task Worth $w_r$ | Success Probabilities $p_r(v_\ell)$ | | | | | Equilibrium $a^\star_{\mathcal{P}}$ | $G_P$ | $G_T$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $G_1$ (Resilience) | Failure of $v_7$ | $\{v_2, v_6, v_8\}$ | $\{2, 6, 7, 8\}$ | $w_2 = 10$<br>$w_6 = 8$<br>$w_7 = 16$<br>$w_8 = 1.35$ | | $\mathcal{R}_2$ | $\mathcal{R}_6$ | $\mathcal{R}_7$ | $\mathcal{R}_8$ | (2,6,7) | 37.64 % | 15.22 % |
| | | | | | $v_2$ | 0.911 | 0.934 | 0.907 | 0.919 | | | |
| | | | | | $v_6$ | 0.888 | 0.941 | 0.909 | 0.918 | | | |
| | | | | | $v_8$ | 0.893 | 0.933 | 0.910 | 0.929 | | | |
| $G_2$ (Resilience) | Failure of $v_4$ | $\{v_3, v_5, v_9\}$ | $\{3, 4, 5, 9\}$ | $w_3 = 0.78$<br>$w_4 = 12$<br>$w_5 = 9$<br>$w_9 = 6.01$ | | $\mathcal{R}_3$ | $\mathcal{R}_4$ | $\mathcal{R}_5$ | $\mathcal{R}_9$ | (4, 5, 9) | 36.98 % | 16.19 % |
| | | | | | $v_3$ | 0.949 | 0.918 | 0.915 | 0.922 | | | |
| | | | | | $v_5$ | 0.941 | 0.917 | 0.926 | 0.914 | | | |
| | | | | | $v_9$ | 0.941 | 0.914 | 0.914 | 0.931 | | | |
| $G_3$ (No-idling) | Task Done of $v_6$ | $\{v_1, v_6\}$ | $\{3, 8\}$ | $w_3 = 0.78$<br>$w_8 = 1.35$ | | $\mathcal{R}_3$ | $\mathcal{R}_8$ | | | (3, 8) | 91.16 % | 5.25 % |
| | | | | | $v_1$ | 0.911 | 0.902 | | | | | |
| | | | | | $v_6$ | 0.904 | 0.912 | | | | | |

**Figure 3.7:** Scenario 2: Summary of game specifics and performances

$v_7$, because task 7 has a much higher expected worth even at the expense of traveling. Later, as shown in Fig. 3.6d, robot $v_4$ failed too, which initiated the second resilience game $G_2$. Similarly, robots $v_3$, $v_5$ and $v_9$ were the closest neighbors, hence they formed the player set $\mathcal{P} = \{v_3, v_5, v_9\}$. The action set was $\tilde{\mathcal{A}} = \{3, 4, 5, 9\}$. As observed in Fig. 3.6e, the optimal equilibrium of $G_2$ lead $v_3$ to drop its task 3 and immediately transition to help $v_4$, in pursuit of a higher worth task. Since tasks 3 and 8 were dropped by their initially assigned robots after games $G_1$ and $G_2$, thus they are available to any future no-idling games. Fig. 3.6f shows that robot $v_6$ has just completed its task and triggered the third no-idling game $G_3$. It called the other robot $v_1$ to join $G_3$, which was close to finish task 1. Thus the player set was $\mathcal{P} = \{v_1, v_6\}$. The action set $\tilde{\mathcal{A}} = \{3, 8\}$ included tasks 3 and 8 that were assigned with no robot. No other region had sufficient task left. The optimal equilibrium of $G_3$ reallocated $v_1$ and $v_6$ to task 3 and task 8, respectively, as seen in Fig. 3.6g. Finally, complete coverage was achieved with all targets found, as shown in Fig. 3.6h.

Fig. 3.7 presents the details of all games. It is observed that $G_P$ is 37.64% for $G_1$ and 36.98% for $G_2$, thus via event-driven task reallocations, the neighbors of the failed robot can re-organize to compensate for the loss of expected worth due to robot failures. Also, $G_P$ is 91.16% for game $G_3$, hence the idling robots can expect to discover more targets

from the remaining tasks after task reallocation. Accordingly, $G_T$ is 15.22%, 16.19% and 5.25% for games $G_1$, $G_2$ and $G_3$, respectively. Thus, the whole team also benefits from each task reallocation.

### 3.5.3 Performance Comparison with Alternative Methods

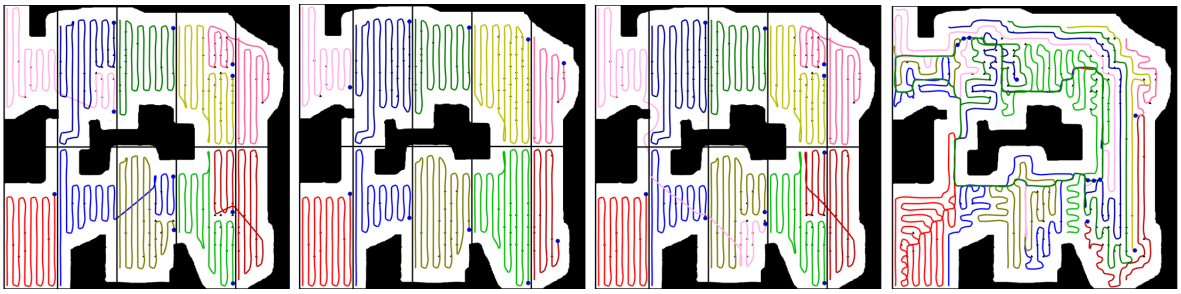
Now, we examine the performance of CARE as compared to three other online multi-robot coverage methods, including: (1) Non-cooperative (Non-Co.) strategy, where each robot covers its own task using the $\varepsilon^\star$ algorithm without cooperation upon task completion or robot failures; (2) modified First-responder (FR) strategy [24], where robots that finish early would selfishly seek for new tasks that can maximize their own utility using Eq. (3.13); and (3) Brick and Mortar (B&M) algorithm [52]. The performance metrics of the alternative methods have been examined in both scenarios using the same initial conditions, and the time was measured in seconds.

#### Performance Comparison in Scenario 1

Figs. 3.8a∼Fig. 3.8d show the robot trajectories for Scenario 1 using CARE and the three alternative methods. Since there was no failure in this scenario, complete coverage was achieved using all methods. The corresponding performance metrics are shown in Fig. 3.9.

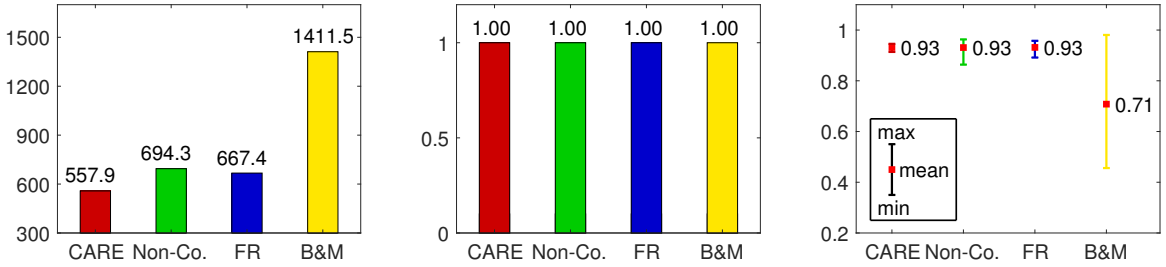
As seen in Fig. 3.9a, CARE requires the least coverage time ($CT$). As compared to using the Non-Co method, it saves $\frac{694.3-557.9}{694.3} \approx 19.65\%$ in time. Similarly, CARE saves about 16.41% and 60.47% in $CT$ as compared to using the FR and B&M methods, respectively. These significant savings in $CT$ are due to the no-idling games that reallocated early completed robots (i.e., $v_2$, $v_5$, $v_6$, and $v_{10}$) in an optimized way.

Moreover, due to lack of cooperation, the Non-Co. method requires a much higher $CT$

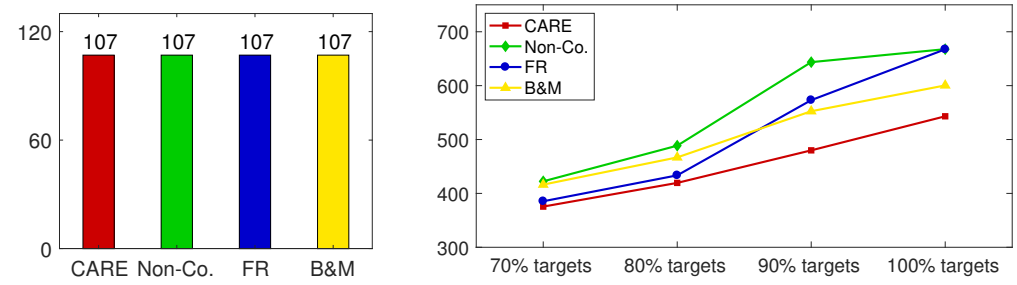


**(a)** CARE **(b)** Non-cooperative **(c)** First-responder **(d)** Brick and Mortar

**Figure 3.8:** Scenario 1: Comparison of coverage trajectories using different online MCPP methods




**(a)** Coverage Time ($CT$) **(b)** Coverage Ratio ($CR$) **(c)** Remaining Reliability ($RR$)




**(d)** Number of Targets Found ($NoTF$) **(e)** Time of Target Discovery $ToTD$

**Figure 3.9:** Scenario 1: Comparison of coverage performance using different online MCPP methods

than CARE and the FR method. In the FR strategy, since early completed robots selfishly selects their new tasks that can maximize their own utility, robot $v_6$ ended up picking task 3 upon finishing task 6, which contains higher worth even at the expense of long traveling time (see Fig. 3.8c). In this regard, the FR method presents higher $CT$ than CARE. Further, due to lack of task partitioning as well as the looping problem, the B&M method generated

strongly overlapped trajectories that leads to the highest $CT$.

Fig. 3.9c shows the minimum, mean and maximum remaining reliability ($RR$) among all live robots as the operation ended. It is seen that, although CARE shares the same mean $RR$ with the Non-Co. and FR methods, it has the smallest difference between minimum and maximum $RR$ of all robots, which implies a more balanced battery depletion for different robots. In contrast, the B&M method presents the smallest mean $RR$ due to the highest $CT$.

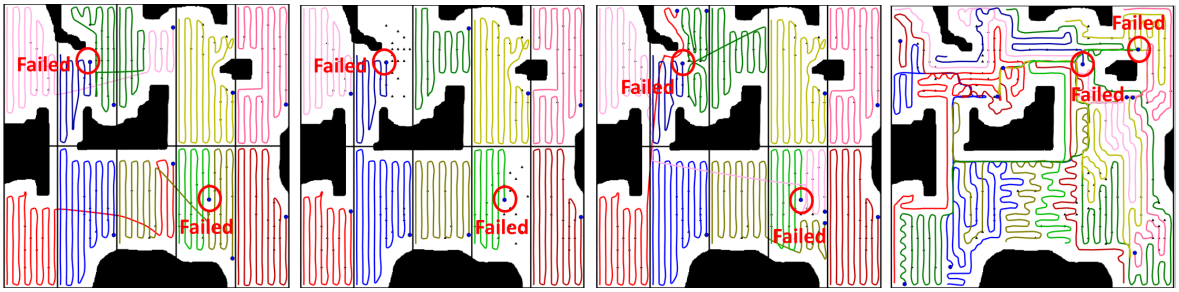
As for the number of targets found ($NoTF$), since $CR = 1$ in this scenario, all 107 hidden targets were found using all methods, as shown in Fig. 3.9b and Fig. 3.9d.

Fig. 3.9e shows the time of target discovery ($ToTD$). It is seen that at each percentage of targets found, CARE always requires the least time, thus leading to the fastest target discovery progress as compared to other methods. This is due to the optimized task reallocations of early completed robots $v_2$, $v_5$, $v_6$ and $v_{10}$ after playing no-idling games $G_1$ and $G_2$. Note that the $ToTD$ when 100% targets are discovered, is different from $CT$, because robots must continue searching in unexplored regions towards complete coverage.

**Performance Comparison in Scenario 2**

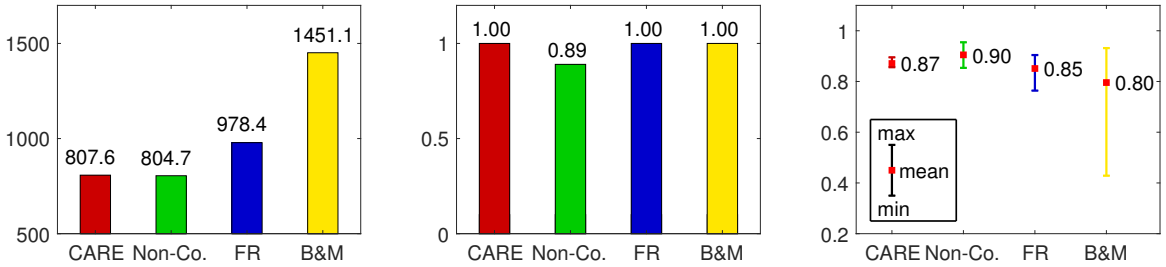
Figs. 3.10a∼Fig. 3.10d show the robot trajectories using CARE and the three alternative methods for Scenario 2. The corresponding performance metrics are presented in Fig. 3.11. In this scenario, two robots of $v_4$ and $v_7$ failed during operation. The alternative methods were evaluated using the same failing condition, i.e., the same robots failed after traveling for the same amount of time.

As shown in Fig. 3.11a, CARE saves about 17.46% and 44.35% in $CT$ as compared to using the FR and B&M methods, respectively. This is due to the no-idling game $G_3$ that reallocated early completed robots $v_1$ and $v_6$ in an optimized manner; while in the FR

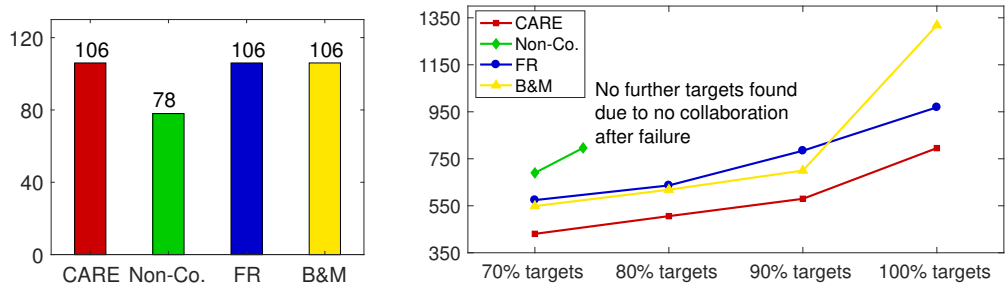



**(a)** CARE **(b)** Non-cooperative **(c)** First-responder **(d)** Brick and Mortar

**Figure 3.10:** Scenario 2: Comparison of coverage trajectories using different online MCPP methods




**(a)** Coverage Time ($CT$) **(b)** Coverage Ratio ($CR$) **(c)** Remaining Reliability ($RR$)




**(d)** Number of Targets Found ($NoTF$) **(e)** Time of Target Discovery ($ToTD$)

**Figure 3.11:** Scenario 2: Comparison of coverage performance using different online MCPP methods

method, the initially assigned tasks of the failed robots were left unattended until some other robot completes its task. Again, the B&M method has the highest $CT$ due to strongly overlapped trajectories.

Fig. 3.11c shows the $RR$ of live robots at the end of the team operation. It is seen that,

CARE has a higher mean *RR*, as well as the smallest difference between minimum and maximum *RR* of all live robots, as compared to the FR and B&M methods. Also, since tasks 4 and 7 were left unattended after robots $v_4$ and $v_7$ failed, the Non-Co. method has the highest mean *RR*.

As shown in Fig. 3.11b, coverage was incomplete ($CR = 0.89$) using the Non-Co. method, while all other methods achieved $CR = 1$. Accordingly, a total number of 28 hidden targets were missed using the Non-Co. method, while all other methods successfully discovered all 106 targets, as shown in Fig. 3.11d.

Fig. 3.11e shows the performance of $ToTD$, where CARE again shows the fastest target discovery progress as compared to other methods. This is mainly because after playing the resilience games $G_1$ and $G_2$, robots $v_3$ and $v_6$ immediately determined to drop their current tasks and search tasks 4 and 7 when robots $v_4$ and $v_7$ failed, respectively, in pursuit of a much higher expected worth than their current tasks. Also, the Non-Co. method only collected around 73.6% of all targets at the end of the team operation due to incomplete coverage.

### 3.5.4 Effects of Different Parameters on Coverage Performance

This section evaluates the effects of different parameters on the coverage performance. Specifically, we vary the values of $N$, $\lambda_r$, $\kappa_1$ and $\kappa_2$, while keeping the values of all other parameters constant. The performance metrics presented in Section 3.3.3 are used for evaluations.

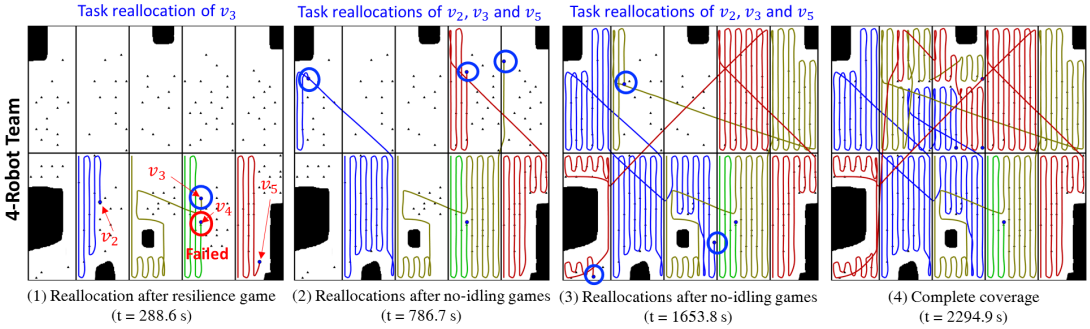
**Table 3.4:** Effects of varying team size $N$

| $N$ | $CT$ | $RR$ | | |
|---|---|---|---|---|
| | | min | mean | max |
| 4 | 2258.6s | 0.071 | 0.086 | 0.109 |
| 6 | 1338.2s | 0.546 | 0.610 | 0.672 |
| 8 | 984.71s | 0.777 | 0.826 | 0.883 |
| 10 | 747.4s | 0.876 | 0.904 | 0.926 |

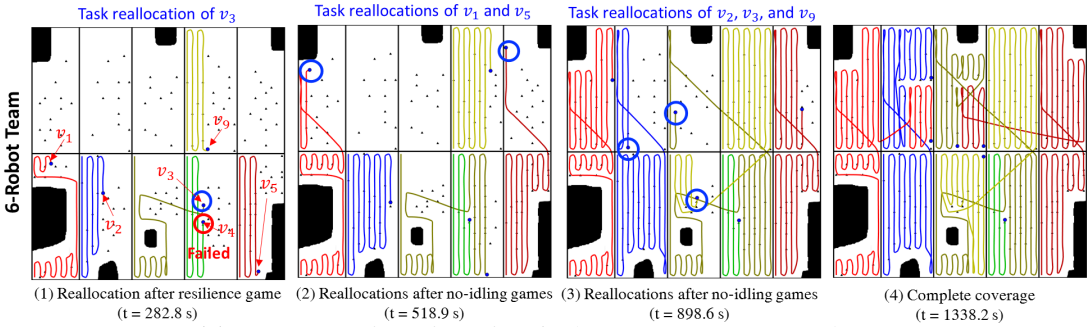
### Effects of Team Size ($N$)

We examine the effectiveness of CARE when different number of robots are deployed to search the same area $\mathscr{R}$. For this purpose, we present Scenario 3, where teams of $N = 4$, 6, 8 and 10 robots were deployed to cover the same 10 tasks, as shown in Fig. 3.12. Fig. 3.12a, Fig. 3.12b, Fig. 3.12c and Fig. 3.12d present the coverage trajectories at different time instants for $N = 4$, 6, 8, and 10, respectively. The scenario setup is kept the same across all simulations, where robot $v_4$ fails after it travels for the same amount of time. As seen in Fig. 3.12a(1), Fig. 3.12b(1), Fig. 3.12c(1) and Fig. 3.12d(1), a resilience game was initiated upon failure of $v_4$, and its neighbor $v_3$ immediately dropped its task 3 to help $v_4$, due to a higher expected worth. Later several no-idling reallocations occurred and in all cases complete coverage was achieved. Moreover, it is seen that with a smaller $N$, task reallocation appears more often. As shown in Fig. 3.12a(4), Fig. 3.12b(4), Fig. 3.12c(4) and Fig. 3.12d(4), the total coverage time clearly decreases when $N$ increases.

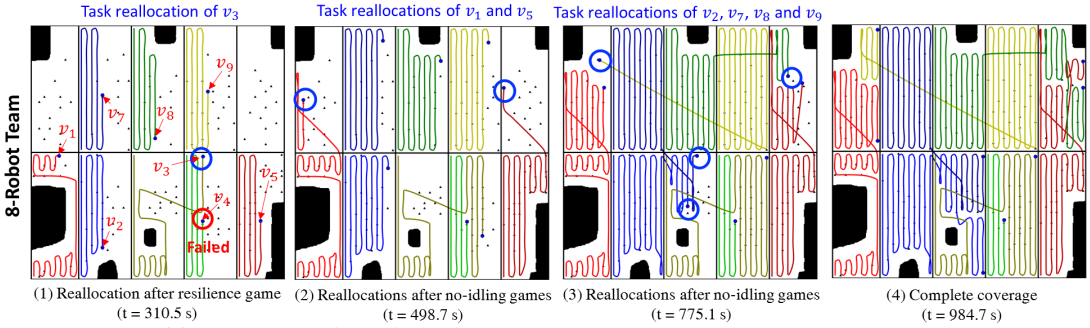
Table 3.4 presents the corresponding coverage performances. For a smaller $N$, since each robot must cover more tasks, the average $RR$ of all live robots become smaller. Also, the minimum and maximum $RR$ are close to the mean, which implies the live robots have been operating for similar amounts of time and robot idling was successfully prevented.



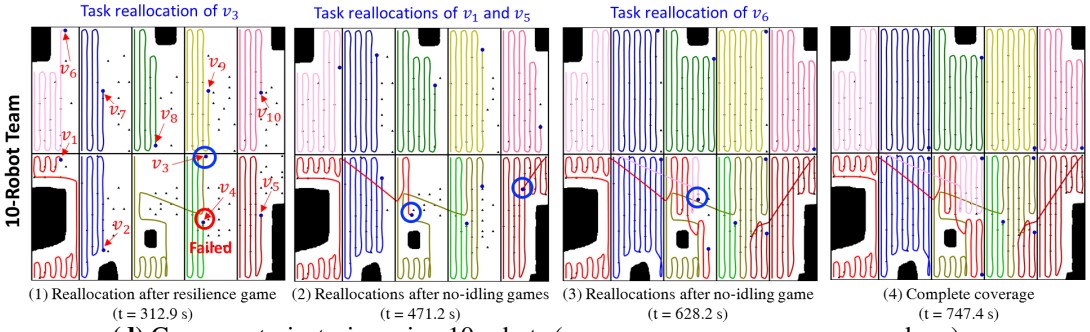

**(a)** Coverage trajectories using 4 robots ($v_2, v_3, v_4$ and $v_5$)




**(b)** Coverage trajectories using 6 robots ($v_1, v_2, v_3, v_4$, $v_5$ and $v_9$)




**(c)** Coverage trajectories using 8 robots ($v_1, v_2, v_3, v_4$, $v_5$, $v_7$, $v_8$ and $v_9$)




**(d)** Coverage trajectories using 10 robots ($v_1, v_2, v_3, v_4$, $v_5$, $v_6$, $v_7$, $v_8$, $v_9$ and $v_{10}$)

**Figure 3.12:** Scenario 3: CARE using a team of 4, 6, 8 and 8 robots. Robot $v_4$ failed during exploration.

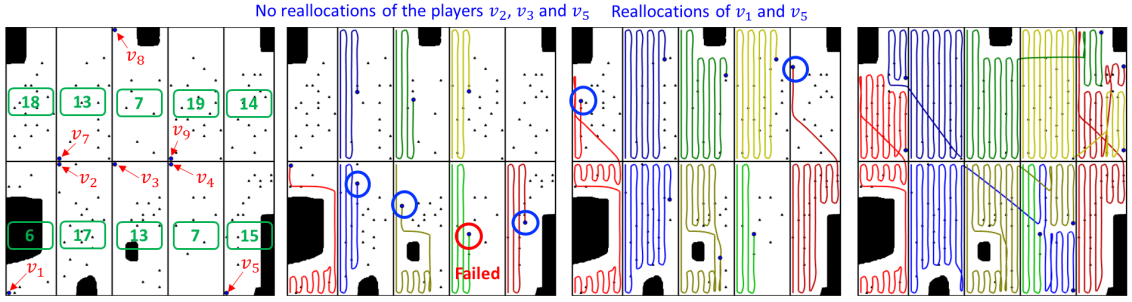
**Effects of Target Distribution ($\lambda_r$)**

Here, we utilize a team of 8 robots to examine the performance of CARE under different target distributions. Fig. 3.13 shows the coverage trajectories at different time instants under three different target distribution examples. The number of targets in each task $r$ is labeled in Fig. 3.13a(1), Fig. 3.13b(1), and Fig. 3.13c(1), and $\lambda_r$ is set as the actual number in each task. While the target distributions are randomly generated, in particular, the task of the failed robot $v_4$ has significantly different $\lambda_r$ in the three examples.

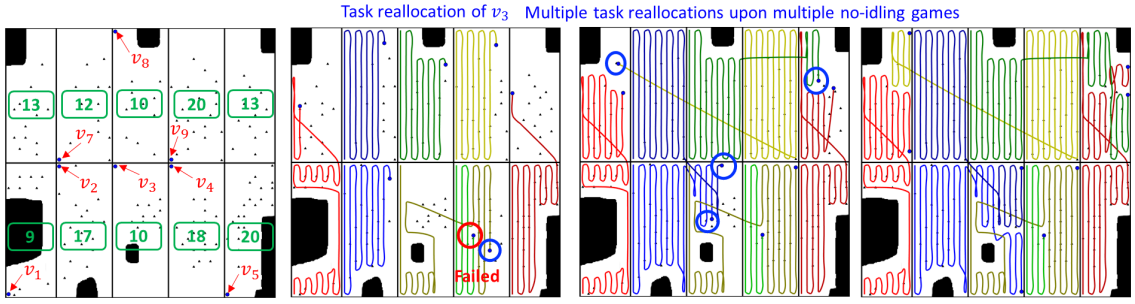
In target distribution example 1, as seen in Fig. 3.13a, task 4 has sparse targets. When $v_4$ failed, its neighbors $v_2$, $v_3$ and $v_5$ played a resilience game, but none of them was reallocated to help $v_4$. This is because they can expect higher utility from their current tasks at the moment. Later, as shown in Fig. 3.13a(3), when $v_1$ and $v_5$ finished, they were reallocated to tasks 6 and 10 after playing no-idling games, respectively. At that moment, due to much higher estimated worths in tasks 6 and 10, again none of them was reallocated to task 4. At last, as seen in Fig. 3.13a(4), when $v_2$ and $v_3$ finished, they moved to task 4 and eventually complete coverage was achieved and all targets were found.

Fig. 3.13b shows the coverage trajectories under target distribution example 2. As compared to the previous example, now tasks 4 and 5 have slightly more targets, but less targets are present in tasks 3. Thus, upon failure of $v_4$, $v_3$ was reallocated to task 4 to pursue a higher utility, as shown in Fig. 3.13b(2). Later, multiple no-idling games were played and the idling robots $v_2$, $v_7$, $v_8$ and $v_9$ were reallocated, as shown in Fig. 3.13b(3). Finally, complete coverage was achieved as shown in Fig. 3.13b(4).

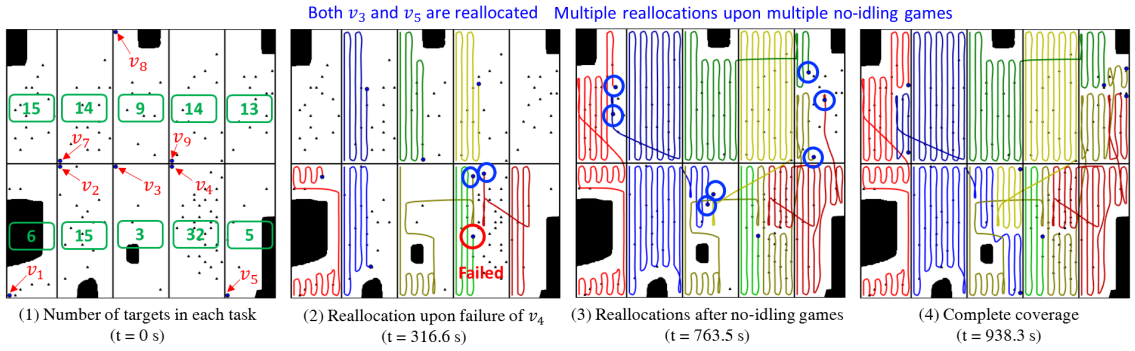
In target distribution example 3, as shown in Fig. 3.13c(1), task 4 has significantly more targets, which makes it prioritized for coverage. In contrast, tasks 3 and 5 have much less targets. Hence, as seen in Fig. 3.13c(2), when $v_4$ failed, a resilience game was initiated




**(a) Target distribution example 1**: sparse number of targets in the task of failed robot $v_4$ in a 8-robot team. Tasking sequence of each robot: $v_1 : \mathscr{R}_1 \rightarrow \mathscr{R}_6, v_2 : \mathscr{R}_2 \rightarrow \mathscr{R}_4, v_3 : \mathscr{R}_3 \rightarrow \mathscr{R}_4, v_4 : \mathscr{R}_4, v_5 : \mathscr{R}_5 \rightarrow \mathscr{R}_{10}, v_7 : \mathscr{R}_7 \rightarrow \mathscr{R}_6, v_8 : \mathscr{R}_8 \rightarrow \mathscr{R}_{10}, v_9 : \mathscr{R}_9 \rightarrow \mathscr{R}_{10}$




**(b) Target distribution example 2**: medium number of targets in the task of failed robot $v_4$ in a 8-robot team. Tasking sequence of each robot: $v_1 : \mathscr{R}_1 \rightarrow \mathscr{R}_6, v_2 : \mathscr{R}_2 \rightarrow \mathscr{R}_3, v_3 : \mathscr{R}_3 \rightarrow \mathscr{R}_4, v_4 : \mathscr{R}_4, v_5 : \mathscr{R}_5 \rightarrow \mathscr{R}_{10}, v_7 : \mathscr{R}_7 \rightarrow \mathscr{R}_3, v_8 : \mathscr{R}_8 \rightarrow \mathscr{R}_{10}, v_9 : \mathscr{R}_9 \rightarrow \mathscr{R}_{10}$




**(c) Target distribution example 3**: dense number of targets in the task of failed robot $v_4$ in a 8-robot team. Tasking sequence of each robot: $v_1 : \mathscr{R}_1 \rightarrow \mathscr{R}_6, v_2 : \mathscr{R}_2 \rightarrow \mathscr{R}_3, v_3 : \mathscr{R}_3 \rightarrow \mathscr{R}_4 \rightarrow \mathscr{R}_{10}, v_4 : \mathscr{R}_4, v_5 : \mathscr{R}_5 \rightarrow \mathscr{R}_4 \rightarrow \mathscr{R}_5 \rightarrow \mathscr{R}_{10}, v_7 : \mathscr{R}_7 \rightarrow \mathscr{R}_6, v_8 : \mathscr{R}_8 \rightarrow \mathscr{R}_{10}, v_9 : \mathscr{R}_9 \rightarrow \mathscr{R}_3$

**Figure 3.13:** Coverage trajectories under different target distributions using a team of 8 robots

involving players $v_2$, $v_3$ and $v_5$; then both $v_3$ and $v_5$ moved to task 4. Upon reaching task 4, each of them picked a sub-area and worked in parallel. Thereafter, as shown in Fig. 3.13c(3), multiple no-idling games appeared and all the live robots were reallocated all around to fill the incomplete tasks. At the end, as shown in Fig. 3.13c(4), complete coverage was achieved with all targets found.

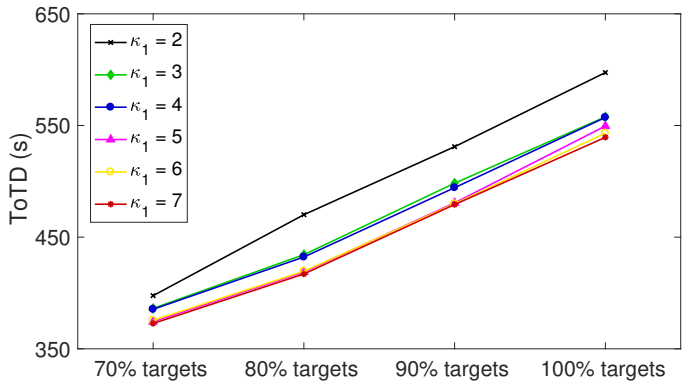
Based on the above analysis, it is seen that the target distribution has a direct impact on the game decisions, which tends to drive the players to pursue prioritized coverage in tasks with higher estimated worth in general.

**Effects of Player Set Sizes ($\kappa_1, \kappa_2$)**

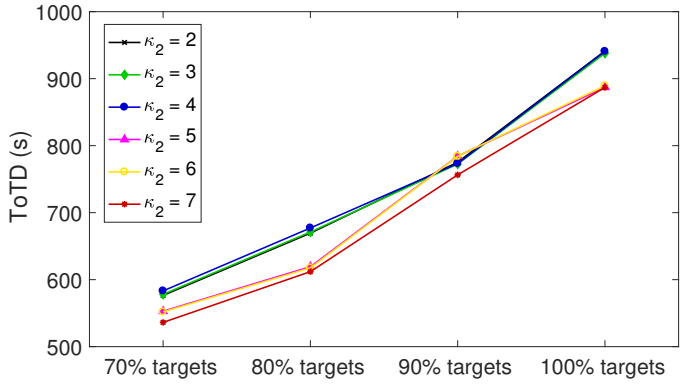
Now, we examine the effects of neighborhood sizes $\kappa_1$ and $\kappa_2$ on the coverage performance for a team of 8 robots in total. Specifically, we focus on two aspects: (1) using a varying $\kappa_1$ with a fixed $\kappa_2$; and (2) using a varying $\kappa_2$ with a fixed $\kappa_1$. The team-level performance metric $ToTD$ is used for evaluation.

First, we fix $\kappa_2 = 3$ and vary $\kappa_1$ in Scenario 1. Note that $\kappa_1$ describes the neighborhood size in no-idling games, which is used to define player set $\mathcal{P}$ in Section 3.4.2. However, the players within the $\kappa_1$ neighborhood of the idling robot $v_{id}$, must also satisfy another condition of being close to finish their current tasks at that moment. Thus, the actual number of players (i.e., $|\mathcal{P}|$) could be smaller than $\kappa_1$.

Fig. 3.14a shows the $ToTD$ when $\kappa_1$ gradually increases from 2 to 7 in Scenario 1. As described in Section 3.5.1, in this scenario, the no-idling games involved $v_2$, $v_5$, $v_6$ and $v_{10}$ that finished earlier than the rest; hence, multiple no-idling games were initiated containing different subset of these players depending on the size of $\kappa_1$. Clearly, it shows that with a larger $\kappa_1$, $ToTD$ is reduced at different target discovery percentages. Moreover,




**(a)** $ToTD$ using different $\kappa_1$ in Scenario 1




**(b)** $ToTD$ using different $\kappa_2$ in Scenario 3

**Figure 3.14:** Time of target discovery ($ToTD$) using different neighborhood sizes $\kappa_1$ and $\kappa_2$

when $\kappa_1 = 3,4$, and when $\kappa_1 = 6,7$, $ToTD$ are overlapping. This is due to the same task reallocation decisions in the corresponding no-idling games.

Next, we fix $\kappa_1 = 6$, as was used in the previous scenarios, and measure $ToTD$ at different target discovery percentages when $\kappa_2$ varies from 2 to 7. Fig. 3.14b shows the results in Scenario 3, where $v_4$ failed during exploration.

As defined in Section 3.4.2, the neighborhood size $\kappa_2$ equals the number of players for resilience games, i.e., $|\mathcal{P}| = \kappa_2$. Thus, as $v_4$ failed, a larger $\kappa_2$ could benefit the team via involving more players in the resilience game for optimization. Note that for $\kappa_2 = 7$, all live

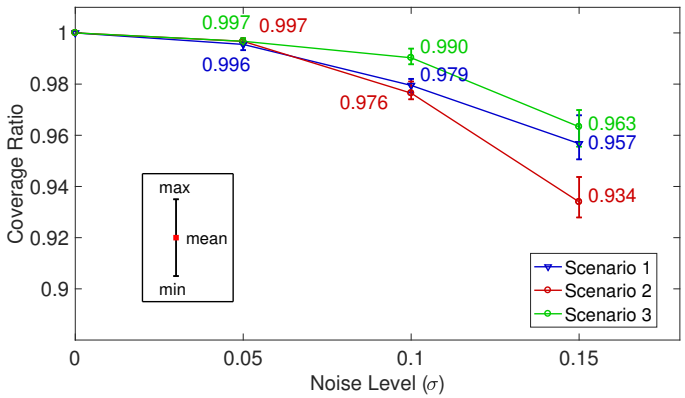



**Figure 3.15:** Coverage ratios at various noise levels over 5 runs per scenario

robots in the team participated in the resilience game. It is seen in Fig. 3.14b that $ToTD$ is reduced when $\kappa_2$ increases. Moreover, when $\kappa_2 = 2,3,4$ and when $\kappa_2 = 5,6$, the $ToTD$ are almost the same. This is because the same task reallocation decisions were made in the resilience games.

### 3.5.5 Performance in the Presence of Uncertainties

In practice, uncertainties in the robot sensing systems could affect the coverage performance. Thus, for uncertainty quantification, noise was injected into the measurements of laser, compass and localization system for each robot. Typically, the uncertainty in laser measurement is 1% of its sensing range, while a modestly priced compass can be as accurate as $1^o$ [63]. These errors were simulated with Additive White Gaussian Noise (AWGN) with standard deviations of $\sigma_{laser} = 1.6$cm and $\sigma_{compass} = 0.5^o$, respectively. On the other hand, indoor localization systems [64] can achieve an accuracy of 0.02m, while Real-Time Kinematic (RTK) based GPS system can be as precise as 0.05m [63]. Therefore, the uncertainty due to localization system is investigated using AWGN at various levels of $\sigma = 0.05$m, 0.10m and 0.15m. Fig. 3.15 shows the minimum, mean and maximum cover-

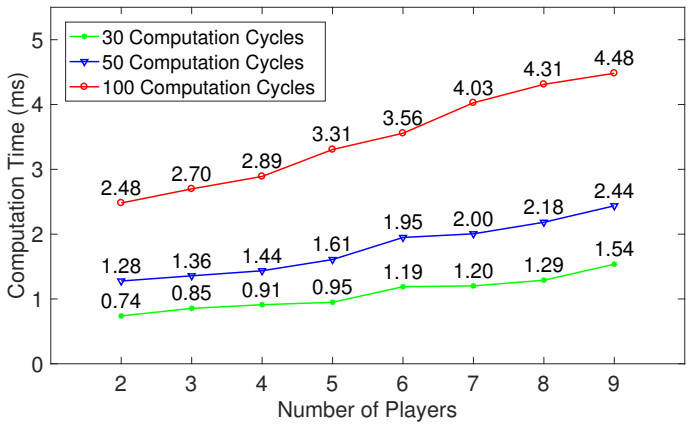



**Figure 3.16:** The computation time for task reallocation

age ratios over five runs under different $\sigma$ for the three scenarios using 10 robots.

### 3.5.6 Computation Time for Task Reallocation

As explained earlier, once a resilience game or no-idling game is triggered, the Max-Logit algorithm was used to rapidly converge to the optimal equilibrium. This section evaluates the computation time using Algorithm 2 for different numbers of players (i.e., $|\mathcal{P}|$) and computation cycles (i.e., $Z$).

As an example, Fig. 3.16 shows the average computation time of game $G_1$ in Scenario 2 under five runs. It is observed that the computation time monotonically increases as more players are involved; however, due to a distributed computation framework, the slope of growth is gentle. On the other hand, for a fixed number of players, the computation time is proportional to the number of computation cycles.

Note that if less players are involved in a resilience game, a larger number of non-player robots will be able to continue exploration during the task reallocation computation, which facilitates a smooth operation under failures; however, a game with less number of players

and tasks may result in a sub-optimal reallocation decision for the whole team. Therefore, the selection of game size must consider these tradeoffs.

### 3.5.7 Practical Applications of CARE

In practice, CARE can be useful in the following applications:

- Cleaning tasks: The floor cleaning task in a large area (e.g., manufacturing factory environment, shopping malls, train stations, airports, and commercial buildings) is one example, where CARE can be implemented to a team of cleaning robots to co-operatively clean up a large space containing unmapped obstacles. With appropriate modifications in the formulation, the spread of dirt on the floor can be treated as targets; also, based on day to day experience, the planner could get a good estimate of the regions with heavy or light dirt. Moreover, it is possible that some robot fails during operation, thus the nearby robots could be reallocated to immediately offer help based on their task priorities.

- Searching tasks: The searching task in a hazardous environment is another example of time-critical coverage application, where a robot team is expected to efficiently find all hidden targets (e.g., underwater or field mines) in a dangerous area. In this case, since the environment is unknown and dangerous for the robots, CARE can be implemented for resilient and efficient operation to secure mission success.

- Agricultural tasks: The agricultural tasks can use a team of robots for seeding and crops cutting in a large farm land. With appropriate modifications in the formulation, the crops or seeds can be regarded as targets which contribute to the task priorities. CARE can be implemented to achieve efficient operations.

## 3.6 Conclusions

This chapter presents a multi-robot coverage algorithm for resilient and efficient coverage of *a priori* unknown environment. The resilience and efficiency of the system are addressed via event-driven task reallocations, using game theoretic solutions. The reallocation decisions are determined by the optimal equilibrium, which is analytically shown to increase the team potential gain. Further, the efficacy of this algorithm has been validated in complex obstacle-rich scenarios on a high-fidelity robotic simulator. The results show that CARE guarantees complete coverage even in presence of failures of some robots. Also, it shows superior coverage performances as compared to three alternative methods in terms of less coverage time and faster target discovery progress.

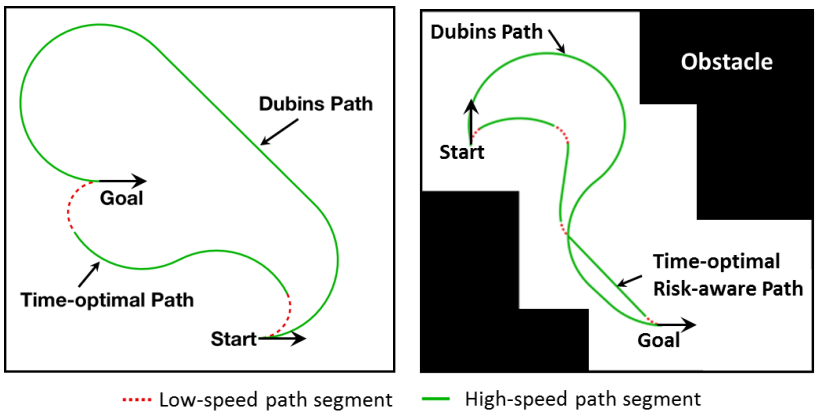
# Chapter 4

# Time-optimal Risk-aware Motion Planning for Curvature-constrained Vehicles

## 4.1 Introduction

The previous chapters addressed the coverage problem using one a single robot or a team of robots. In this chapter, we investigate another important category of the Point-to-Point (PTP) motion planning, where the general objective is to plan a collision-free, feasible path for an autonomous vehicle to safely reach a goal state while optimizing certain metrics, such as shortest distance and minimum time.

Typically, autonomous vehicles are subject to kinematic constraints. As an example, a vehicle with bounded curvature indicates that its turning motion is subject to a non-zero minimum turning radius, which could seriously limit its manoeuvrability. As shown in [34], the problem of deciding whether a curvature-constrained collision-free path exists between




**(a)** Time-optimal path vs. Dubins path

**(b)** Time-optimal risk-aware path vs. Dubins path

**Figure 4.1:** Time-optimal and time-optimal risk-aware paths vs. the Dubins paths in different environment

two given poses amid polygonal obstacles is NP-hard [35]. This implies that no efficient exact algorithms exist for curvature-constrained time-optimal motion planning in arbitrary environment [34].

Besides, autonomous vehicles usually can travel at variable speeds, which range from a lower bound (e.g., idle speed) to an upper bound (e.g., maximum speed). Since the turning radius becomes smaller if turning at a lower speed and a higher turn rate, the time-optimal path could be different from the shortest path. Fig. 4.1a shows an example of time-optimal path versus Dubins path, where the slow speed segments in the time-optimal path enable smaller turning radii which help in reducing the total path length.

Furthermore, due to complexities of the environment, it is also critical that the time-optimal path is safe for the vehicle. Existing risk-aware motion planning approaches evaluate vehicle safety based on its locations to nearby obstacles, while ignoring its heading and/or speed (see Section 4.2 for details). In this regard, we proposed a continuous risk function based on the concept of *collision time*, which is the time in which the vehicle can hit the obstacle along its heading direction, if it loses control. Thus, a vehicle is deemed

safe if its collision time is greater than the time it takes to stop, maneuver around, or regain its control. The concept of collision time considers the complete information about the vehicle state, including its location with respect to the obstacles, heading angle, as well as speed. Fig. 4.1b presents an example where the time-optimal risk-aware path stays away from the obstacles as compared to the Dubins path.

To the best of our knowledge, the time-optimal motion planning problem for curvature-constrained variable-speed vehicles has not been solved in the presence of obstacles. In this context, this chapter presents the T$^\star$ algorithm [91] that provides an approximate solution to solve this problem in the discrete domain [92]. Specifically, we construct a discrete configuration space, and compute the optimal path by finding the optimal sequence of vehicle states that can lead the vehicle to reach the goal state with minimized total cost of time and risk. The risk function is seamlessly integrated with the time cost in the joint optimization function. At last, we used the A$^\star$ framework to search for the optimal state sequence, which is then used to reconstruct the optimal path in a piece-wise manner.

The T$^\star$ algorithm has been validated in complex obstacle-rich scenarios, where the results showed superiority in time savings over the Dubins paths. The algorithm can also produce multiple path choices to the planner with decreasing risk costs but at the expense of increasing time costs. As compared to other grid-based methods under motion constraints such as Hybrid-A$^\star$[93], the motion primitives between two consecutive states in T$^\star$ are optimized over both risk and time.

The remainder of this chapter is organized as follows. Section 4.2 briefly reviews existing approaches related to motion planning for curvature-constrained vehicles. Section 4.3 formulates the time-optimal risk-aware motion planning problem, and Section 4.4 explains the details of the T$^\star$ algorithm. Section 4.5 presents the results on simulated scenarios, and this chapter is concluded in Section 4.6.

## 4.2 Related Work

Thus far, plenty of planning algorithms have been proposed for shortest path planning [36]. To name a few, for known environment, there exist Dijkstra's algorithm [94], A$^\star$ algorithm [95], Fast Marching algorithm [29], etc.; while for unknown environment, there exist Lifelong Planning A$^\star$ (LPA$^\star$) algorithm [96], D$^\star$-Lite algorithm [97], etc. While these methods can generate collision-free paths for the vehicle, the constraints in vehicle motion were not considered, thus the resulting paths could lead to difficulties during path tracking.

Along this line, Dubins [32] addressed the problem of finding the shortest path for a vehicle with curvature constraints, that moves at a constant speed in an obstacle-free space. He used a geometric approach and showed that the shortest path between a pair of vehicle poses must be one of the following 6 types: RSR, RSL, LSR, LSL, RLR and LRL, where L(R) refer to a left (right) turn with maximum curvature, and S indicates a straight line segment. These paths are known as the *Dubins Curves*, which were later verified in [98] using Pontryagin's Minimum Principle. Dubins-like path planning is popular because the resulting path consists of optimized parametric curves that can be expressed analytically and computed quickly, based on which many path-following methods could be designed. Further research on this problem considered bounded acceleration [99], field-of-view constraint [100], the orienteering problem [101][102], polygonal obstacles [34][35], and external disturbances [103][104].

The above methods following the Dubins' approach focused on shortest path planning. However, autonomous vehicles can travel at variable speeds, thus the time-optimal path can be different from the shortest path. In a recent study, Wolek et al. [38] derived the solution to find the time-optimal path for curvature-constrained variable-speed vehicles in an obstacle-free space. They identified a sufficient set of 34 candidate paths, where each

candidate path contains circular arcs or straight line segments on which the vehicle travels at extremal (i.e., maximum or minimum) speeds. Then, for a given pair of start and goal poses, the time-optimal path is the least-cost candidate path. However, to the best of our knowledge, the time-optimal motion planning problem for curvature-constrained variable-speed vehicles has not been solved in the presence of obstacles.

On the other hand, regarding risk-aware motion planning, Pereira et al. [39] proposed a minimum-risk planning strategy using the risk map for autonomous underwater vehicles (AUVs), where the risk map is constructed offline using Automatic Identification System (AIS) data. Hernández et al. [41] presented a safe path planning algorithm for AUVs, where safety is ensured by creating a risk zone around the vehicle for obstacle avoidance. Liu et al. [105] proposed a episodic memory-based planning method, where the risk in the local behavioral planning strategy is measured by the minimum distance to danger areas. De Filippis et al. [106] addressed the minimum-risk path planning problem for Unmanned Air Vehicles (UAVs) based on orography, where the risk is evaluated using the altitude information. Pfeiffer et al. [40] assumed *a priori* known threat zones to UAVs and presented an approach to find paths with minimized probability of being exposed to threats. Davoodi [107] proposed a bi-objective optimization approach to find the Pareto-optimal paths with minimized path length and maximized distance to the obstacles. Wang et al. [108] presented a multi-objective Particle Swarm Optimization (PSO) approach for car-like robots, where the objectives are to minimize the path length and the total risk defined by terrain roughness. Huang and Savkin [109] presented the Shortest Viable Path Planning (SVPP) algorithm to address a variant of Dubins Travelling Salesman Problem in the presence of obstacles to visit a set of sensor nodes in a sensing field.

The above-mentioned methods addressed safe-path planning, but the proposed risk measures were using partial information of the vehicle state, i.e., its location with respect

to the obstacles or threats, while ignoring its speed and heading angle. Fraichard and Asama [42] introduced the concept of *inevitable collision states* into safe-path planning, where the states that cannot avoid future collision must be prohibited. Clearly, when the vehicle operates near obstacles, its speed and heading angle are also critical to assess its safety, thus the full information about vehicle state must be incorporated into planning.

## 4.3 Problem Formulation

Let $\mathscr{R} \subset \mathbb{R}^2$ be the search area which is populated with a number of *a priori* known obstacles. The vehicle motion is described as:

$$\begin{cases} \dot{x}(t) &= v(t)\cos\theta(t) \\ \dot{y}(t) &= v(t)\sin\theta(t) \\ \dot{\theta}(t) &= u(t), \end{cases} \tag{4.1}$$

where $(x, y, \theta) \in SE(2)$, $u$ is the turn rate, and $v$ is the speed.

It is assumed that the autonomous vehicle is capable of traveling at a variable speed, s.t., $v \in [v_{\min}, 1]$, where $v_{\min} \in \mathbb{R}^+$ is the minimum speed, and with a modified distance unit the maximum speed $v_{\max}$ is normalized to 1 [38]. Note that the Dubins path considered the special case where $v$ is constant.

Also, the turn rate $u$ is symmetric and bounded, s.t., $u \in [-u_{\max}, u_{\max}]$, where $u_{\max} \in \mathbb{R}^+$ is the maximum turn rate and the $+/-$ sign refers to a left/right turn. The turn rate and speed are connected by the curvature $\kappa = u/v$, which is the inverse of the turning radius. The curvature $|\kappa| = 0$ means that the vehicle is moving in a straight line, this happens when $u = 0$. On the other hand, when the vehicle turns at the maximum turn rate (i.e., $\pm u_{\max}$),

it can do so at the maximum or minimum speed, which result in the curvature $|\kappa| = u_{\max}$ or $|\kappa| = u_{\max}/v_{\min}$, respectively. These correspond to the maximum and minimum turning radii of the vehicle as $\mathbf{R} = 1/u_{\max}$ and $\mathbf{r} = v_{\min}/u_{\max}$, respectively. Thus, the curvature is bounded as $|\kappa| \in [0, u_{\max}/v_{\min}]$.

Now, let us denote the state of the vehicle as $\mathbf{p} = (x, y, \theta, v)$. Let $\Gamma$ denote the set of all collision-free paths between the start state $\mathbf{p}_{start}$ and the goal state $\mathbf{p}_{goal}$. Then, for each path $\gamma \in \Gamma$, the control $\mathbf{c}(s) = (\kappa, v)$ at any point $s$ on $\gamma$ belongs to the following constraint set [38]:

$$\Omega = \left\{ (\kappa, v) \mid v_{\min} \leq v \leq 1 \text{ and } |\kappa| \leq \frac{u_{\max}}{v} \right\}. \tag{4.2}$$

The admissible control must be piece-wise continuous and should satisfy the boundary conditions, i.e., $\mathbf{c}$ must drive the vehicle from $\mathbf{p}_{start}$ to $\mathbf{p}_{goal}$ along the path $\gamma$ while avoiding obstacles. Further, let $\mathcal{R}(s)$ denote the risk cost at a point $s$ on $\gamma$. Then, the total cost of time and risk is defined as

$$J(\gamma) = \int_{\gamma} \mathcal{R}(s) \cdot \frac{1}{v(s)} \, ds, \tag{4.3}$$

where the term $\frac{1}{v(s)}$ evaluates the time cost when the vehicle moves along a small path segment $ds$.

Therefore, the objective is to find the control $\mathbf{c}^\star \in \Omega$, which generates a collision-free path $\gamma^\star$ with the minimal cost $J(\gamma^\star)$, s.t. $J(\gamma^\star) \leq J(\gamma), \forall \gamma \in \Gamma$. Note that the outcome of optimization is a trajectory; however, we use the term path for trajectory in this chapter with slight abuse of terminology.

## 4.4 The T$^\star$ Algorithm

Since there is no efficient exact algorithm to minimize Eq. (4.3), even when the risk is ignored, in the presence of obstacles [34], this section develops a novel grid-based algorithm, called T$^\star$, to obtain an approximate solution in the discrete domain.

### 4.4.1 Configuration Space

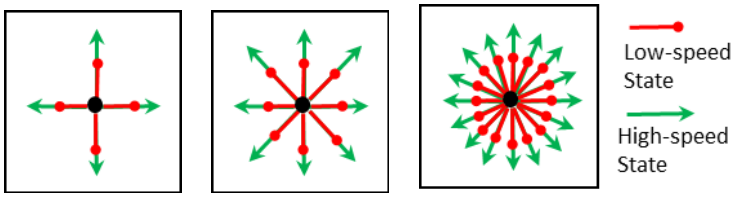
Since any feasible path $\gamma$ between $\mathbf{p}_{start}$ and $\mathbf{p}_{goal}$ consists of multiple intermediate states, the time-optimal risk-aware motion planning problem can be solved by identifying the optimal sequence of states. This in turn motivates to partition the search area into grid cells and then assign each cell with a sufficient number of possible discrete states, which forms a configuration space described as below.

First, the search area $\mathscr{R}$ is discretized into a set of grid cells, $\mathscr{T} = \{\tau_\alpha \subset \mathbb{R}^2, \alpha = 1, \dots |\mathscr{T}|\}$, where $\bigcup_{\alpha=1}^{|\mathscr{T}|} \tau_\alpha = \mathscr{R}$, $\tau_\alpha^\circ \bigcap \tau_\beta^\circ = \emptyset$, $\forall \alpha, \beta \in \{1, \dots |\mathscr{T}|\}$, $\alpha \neq \beta$, and $^\circ$ denotes the interior.

Then, each cell $\tau_\alpha \in \mathscr{T}$ is encoded with a symbolic state $s_\alpha \in \{\mathrm{O}, \mathrm{A}\}$, where $\mathrm{O} \equiv$ *obstacle* and $\mathrm{A} \equiv$ *accessible*. Specifically, $s_\alpha = \mathrm{O}$ if $\tau_\alpha$ is (partially) occupied by an obstacle; otherwise, $s_\alpha = \mathrm{A}$. Then the configuration space $\mathcal{Q}$ is constructed as follows.

**Definition 4.4.1** (**Configuration Space**). *Let $\mathcal{O} = \{(x_\alpha, y_\alpha) \in \tau_\alpha : s_\alpha = A\}$ be the set of center positions of all obstacle-free cells. Let $\Theta = \{\frac{2\pi\ell}{\mathcal{L}} : \ell = 0, \dots \mathcal{L} - 1\}$ be the set of $\mathcal{L} \in \mathbb{N}^+$ heading angles. Let $\mathcal{V} = \{v_{\min}, 1\}$ be the set of speeds. Then, the configuration space is defined as:*

$$\mathcal{Q} = \mathcal{O} \times \Theta \times \mathcal{V}$$




**(a)** $\mathcal{L} = 4$ **(b)** $\mathcal{L} = 8$ **(c)** $\mathcal{L} = 16$

**Figure 4.2:** State expansion in each cell when $\mathcal{L} = 4$, 8 and 16, respectively

Fig. 4.2 shows the state expansion for different $\mathcal{L}$. Clearly, the size of $\mathcal{Q}$ relies on the size of obstacle-free space and the value of $\mathcal{L}$. Although a larger $\mathcal{L}$ could potentially produce better results, it will also increase the size of $\mathcal{Q}$ that leads to higher computational complexity for motion planning. This chapter adopts the *8-orientation* state expansion with $\mathcal{L} = 8$.

### 4.4.2 Approximate Optimization Function

Now, we present an approximation to the optimization problem in Section 4.3. Let $\mathscr{P} = \{\mathbf{P}^m, m = 1, \dots |\mathscr{P}|\}$ be the set of all valid state sequences, where $\mathbf{P}^m = [\mathbf{p}_1^m, \mathbf{p}_2^m, \dots \mathbf{p}_n^m]$ is a sequence of states that connects $\mathbf{p}_{start}$ and $\mathbf{p}_{goal}$. Here $\mathbf{p}_i^m = (x_i, y_i, \theta_i, v_i) \in \mathcal{Q}, \forall i \in \{1, \dots n\}$, and $n \in \mathbb{N}^+$ denotes the length of the sequence. To satisfy the boundary conditions, it should have $\mathbf{p}_1^m = \mathbf{p}_{start}$, and $\mathbf{p}_n^m = \mathbf{p}_{goal}$. Note that any two consecutive states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m, \forall i \in \{1, \dots n-1\}$, must belong to two neighboring cells. Also, a valid state sequence cannot contain duplicate states, i.e., $\mathbf{p}_i^m \neq \mathbf{p}_j^m, \forall i, j \in \{1, \dots n\}, i \neq j$; otherwise a loop will be present, thus it will not be optimal since its cost can be further reduced by removing such loop.

Then, we compute the total cost of a state sequence $\mathbf{P}^m$ in a piece-wise manner as follows:

$$J(\mathbf{P}^m) \triangleq \sum_{i=1}^{n-1} \widetilde{J}(\mathbf{p}_i^m, \mathbf{p}_{i+1}^m), \tag{4.4}$$

where $\widetilde{J}$ is the step-wise cost to move from $\mathbf{p}_i^m$ to $\mathbf{p}_{i+1}^m$, subject to the vehicle motion constraints and the obstacle layout. Let $\Gamma^c$ be the set of all feasible paths between $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, then $\widetilde{J}$ is computed as the minimal cost among all paths $\gamma_{i,i+1} \in \Gamma^c$:

$$\widetilde{J}(\mathbf{p}_i^m, \mathbf{p}_{i+1}^m) = \min_{\gamma_{i,i+1} \in \Gamma^c} J(\gamma_{i,i+1}), \tag{4.5}$$

where according to Eq. (4.3),

$$J(\gamma_{i,i+1}) = \int_{\gamma_{i,i+1}} \mathcal{R}(s) \cdot \frac{1}{v(s)}\, ds. \tag{4.6}$$

For simplicity, the risk is assumed constant along $\gamma_{i,i+1}$ and is measured at the most dangerous state on $\gamma_{i,i+1}$ which has the least collision time (see details in Section 4.4.4). Thus, the risk cost and the time cost can be separated, as follows:

$$J(\gamma_{i,i+1}) = \underbrace{\mathcal{R}(\gamma_{i,i+1})}_{\text{risk cost}} \cdot \underbrace{\int_{\gamma_{i,i+1}} \frac{1}{v(s)}\, ds}_{\text{time cost } \mathcal{T}(\gamma_{i,i+1})}. \tag{4.7}$$

We aim to find the optimal state sequence $\mathbf{P}^\star \in \mathscr{P}$, s.t. $J(\mathbf{P}^\star) \leq J(\mathbf{P}^m), \forall \mathbf{P}^m \in \mathscr{P}$. Then, the optimal path $\gamma^\star$ can be reconstructed in a piece-wise manner using the states of $\mathbf{P}^\star$ and the optimal paths between adjacent states. While the solution to the optimization problem is presented in Section 4.4.6, we first present the details on computation of the time cost $\mathcal{T}(\gamma_{i,i+1})$ and the risk cost $\mathcal{R}(\gamma_{i,i+1})$.

### 4.4.3 Time Cost $\mathcal{T}(\gamma_{i,i+1})$

Now, we define the time cost $\mathcal{T}(\gamma_{i,i+1})$ in Eq. (4.7). As compared to the Dubins curves, since the vehicle can travel at variable speeds, the time-optimal path that minimizes $\mathcal{T}(\gamma_{i,i+1})$ can contain arcs of different turning radii (see Fig. 4.1), which implies more choices of candidate paths, or in other words, a larger candidate set than the Dubins curves.

#### The Sufficient Set of Candidate Paths

For any pair of states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, the authors of [38] showed that the sufficient set, which guarantees to contain the time-optimal path in the absence of obstacles, consists of 34 candidate paths. These candidate paths form the set $\Gamma^c$, as shown in Table 4.1. Each path $\gamma_{i,i+1} \in \Gamma^c$ consists of up to five segments, where each segment could be one of the following [38]:

1. Bang arcs (**B**), where the vehicle turns at maximum speed $v_{\max}$ with maximum turn rate $u_{\max}$ (i.e., with radius **R**);

2. Cornering arcs (**C**), where the vehicle turns at minimum speed $v_{\min}$ with maximum turn rate $u_{\max}$ (i.e., with radius **r**);

3. Straight line segments (**S**), where the vehicle moves straight at maximum speed $v_{\max}$.

Any circular arc (i.e., **B** or **C**) can be either left (L) or right (R), as shown under the direction column. Each candidate path is read from left to right, and the consecutive turns within parentheses are of the same direction. For example, the No.18 path of (**B**)**S**(**BC**) with direction LSR is read as: a left bang arc, followed by a straight line segment, a right bang arc, and then a right cornering arc.

**Table 4.1:** The set of candidate paths ($\Gamma^c$) between any pair of states

| No. | Path Type[1] | Direction[2] | No. | Path Type | Direction |
|---|---|---|---|---|---|
| 1 | (**B**)**S**(**B**) | LSL | 18 | (**B**)**S**(**BC**) | LSR |
| 2 | (**B**)**S**(**B**) | LSR | 19 | (**B**)**S**(**BC**) | RSL |
| 3 | (**B**)**S**(**B**) | RSL | 20 | (**B**)**S**(**BC**) | RSR |
| 4 | (**B**)**S**(**B**) | RSR | 21 | (**CB**)(**BCB**) | LL |
| 5 | (**BCB**)(**B**) | LL | 22 | (**CB**)(**BCB**) | LR |
| 6 | (**BCB**)(**B**) | LR | 23 | (**CB**)(**BCB**) | RL |
| 7 | (**BCB**)(**B**) | RL | 24 | (**CB**)(**BCB**) | RR |
| 8 | (**BCB**)(**B**) | RR | 25 | (**CB**)**S**(**B**) | LSL |
| 9 | (**B**)(**BCB**) | LL | 26 | (**CB**)**S**(**B**) | LSR |
| 10 | (**B**)(**BCB**) | LR | 27 | (**CB**)**S**(**B**) | RSL |
| 11 | (**B**)(**BCB**) | RL | 28 | (**CB**)**S**(**B**) | RSR |
| 12 | (**B**)(**BCB**) | RR | 29 | (**C**)(**C**)(**C**) | LRL |
| 13 | (**BCB**)(**BC**) | LL | 30 | (**C**)(**C**)(**C**) | RLR |
| 14 | (**BCB**)(**BC**) | LR | 31 | (**CB**)**S**(**BC**) | LSL |
| 15 | (**BCB**)(**BC**) | RL | 32 | (**CB**)**S**(**BC**) | LSR |
| 16 | (**BCB**)(**BC**) | RR | 33 | (**CB**)**S**(**BC**) | RSL |
| 17 | (**B**)**S**(**BC**) | LSL | 34 | (**CB**)**S**(**BC**) | RSR |

[1] **B** is a bang arc, **C** is a cornering arc and **S** is a straight line segment. The parentheses are used to indicate consecutive turns of the same direction.

[2] L is a left turn, R is a right turn and S means moving straight.

Each candidate path has to be optimized for its parameters (i.e., the angles for arc segments and the lengths of straight line segments) to achieve time optimality, thus one must solve a nonlinear constrained optimization problem [38]. Specifically, the total time cost for each candidate path between a pair of states is the summation of cost for each segment. An arc segment (**B** or **C**) with angle $\Delta\theta$ contributes to the time cost by $\frac{|\Delta\theta|}{u_{\max}}$; while a displacement of $d \in \mathbb{R}^+$ for the straight line segment contributes to the cost by $d$ while moving at $v_{\max} = 1$.

Since each candidate path is required to exactly reach $\mathbf{p}_{i+1}^m$ from $\mathbf{p}_i^m$, there are five constraints including the total displacement along each axis, the total change in the heading

angle, and the speeds specified by the first and last arc types. These boundary conditions ensure the continuity in position, heading and speed when the vehicle reaches every state $\mathbf{p}_i^m \in \mathbf{P}^m$, including $\mathbf{p}_{start}$ and $\mathbf{p}_{goal}$.

In this chapter, the path parameters for each candidate path are optimized using the nonlinear solver *IPOPT* (*I*nterior *P*oint *OPT*imizer) [38]. The time-optimal path between a given pair of states is the candidate path with the least time cost.

Next, we construct the Optimized Candidate Paths for State-pairs (OCPS) table, that contains the optimized candidate paths for all possible pairs of states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$.

### The OCPS Table

To avoid computational burden during motion planning, the parameters of the candidate paths are optimized offline to construct the OCPS Table.

Specifically, consider a state $\mathbf{p}_i^m$ located in a center cell and all the possible states $\mathbf{p}_{i+1}^m$ located in its $3 \times 3$ neighborhood. Since $\mathbf{p}_i^m$ has $8 \times 2 = 16$ choices corresponding to 8 directions and 2 speeds, and $\mathbf{p}_{i+1}^m$ has $8 \times 8 \times 2 = 128$ choices corresponding to 8 positions in the neighborhood, 8 directions, and 2 speeds, the total number of 2048 pairs are considered. However, by exploring symmetry, one can easily figure out that only 272 pairs must be optimized for, while the rest can be derived accordingly.

For each pair of $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, depending on the speed information of $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, the associated candidate paths belong to one of the following four subsets of $\Gamma^c$:

- $\Gamma^c_{\mathbf{BB}}$, which contains No.1-12 paths that start and end with $v_{\max}$, i.e., they start and end with bang arcs **B**;
- $\Gamma^c_{\mathbf{BC}}$, which contains No.13-20 paths that start with $v_{\max}$ and end with $v_{\min}$, i.e., they start with a bang arc **B** and end with a cornering arc **C**;

- $\Gamma^c_{\mathbf{CB}}$, which contains No.21-28 paths that start with $v_{\min}$ and end with $v_{\max}$, i.e., they start with a cornering arc **C** and end with a bang arc **B**; and

- $\Gamma^c_{\mathbf{CC}}$, which contains No.29-34 paths that start and end with $v_{\min}$, i.e., they start and end with cornering arcs **C**.

Thus, for each pair of $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, the path parameters of candidate paths within the corresponding subset are optimized offline using *IPOPT*. The resulting optimized candidate paths and their time costs are stored in the OCPS table.

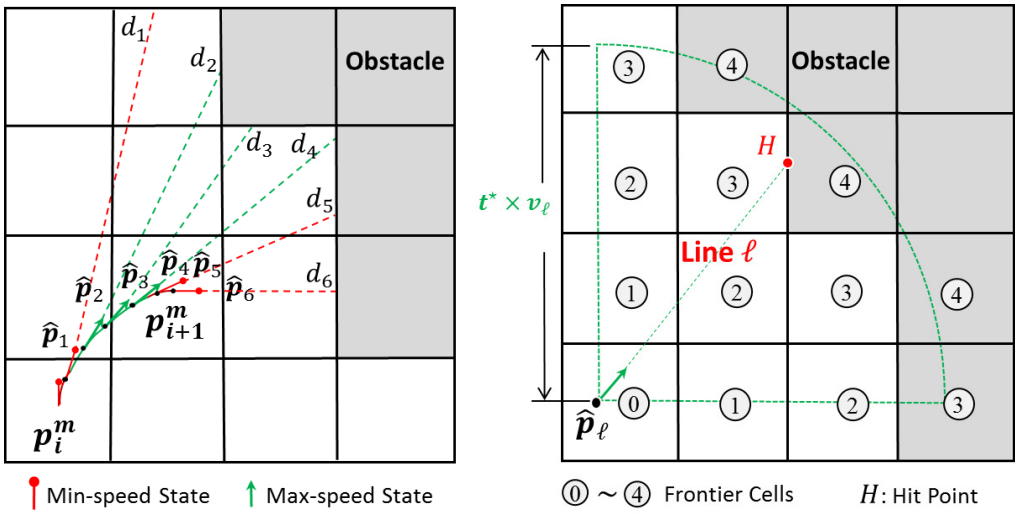
**Computation of $\mathcal{T}(\gamma_{\mathbf{i,i+1}})$**

During the planning process, when computing for the time cost $\mathcal{T}$ between a certain state pair $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$, the planner can initiate a query to the OCPS table to obtain a set of optimized candidate paths within the corresponding subset determined by the speed information of $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$. Then for each obtained optimized candidate path $\gamma_{i,i+1}$, $\mathcal{T}(\gamma_{i,i+1})$ is assigned with the associated time cost if it is collision-free; otherwise, $\mathcal{T}(\gamma_{i,i+1}) = +\infty$.

### 4.4.4 Risk Cost $\mathcal{R}(\gamma_{i,i+1})$

This section presents a state-based risk function to evaluate the risk cost of each candidate path $\gamma_{i,i+1} \in \Gamma^c$.

As mentioned in Section 4.4.2, the risk cost $\mathcal{R}(\gamma_{i,i+1})$ of a candidate path $\gamma_{i,i+1}$ between a pair of states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$ is determined by the most dangerous state along $\gamma_{i,i+1}$ that results in the least collision time to an obstacle or the space boundary in its heading direction, if the vehicle loses control. For any state, the vehicle is considered safe if the corresponding collision time is greater than a threshold $t^\star \in \mathbb{R}^+$, which indicates the time for the vehicle




**(a)** Sampling of $\gamma_{i,i+1}$ **(b)** Computation of the collision distance $d_\ell$

**Figure 4.3:** The collision distance

to fully stop, maneuver around, or re-gain its control. Thus, the risk of the vehicle relies on its state, including its location, heading and speed.

Now, we present the computation of $\mathcal{R}(\gamma_{i,i+1})$. Consider a candidate path $\gamma_{i,i+1} \in \Gamma^c$. First, a set of states, $\{\hat{\mathbf{p}}_\ell, \ell = 1,2,\ldots\}$, are evenly sampled along $\gamma_{i,i+1}$ with a sampling interval $\Delta d \in \mathbb{R}^+$, and the state $\mathbf{p}^m_{i+1}$ is included in this sample set. Specifically, $\hat{\mathbf{p}}_\ell = (x_\ell, y_\ell, \theta_\ell, v_\ell)$, where $(x_\ell, y_\ell, \theta_\ell) \in SE(2)$ and $v_\ell \in \{v_{\min}, 1\}$ indicates the speed of the sampled state. Fig. 4.3a shows an example of six sampled states between $\mathbf{p}^m_i$ and $\mathbf{p}^m_{i+1}$.

Then, for each $\hat{\mathbf{p}}_\ell$, one can geometrically compute the collision distance $d_\ell$, as shown in Fig. 4.3b. Denote by *line* $\ell$ the extended line of state $\hat{\mathbf{p}}_\ell$ along angle $\theta_\ell$, which hits the obstacle at the hit point $(x_h, y_h) \in \mathbb{R}^2$. Then, $d_\ell = \|(x_\ell - x_h, y_\ell - y_h)\|$. Once $d_\ell$ is determined, the corresponding collision time $t_\ell$ is computed as:

$$t_\ell = \frac{d_\ell}{v_\ell}. \tag{4.8}$$

**Algorithm 3:** Computation of Collision Time $t_\ell$

**input :** $\hat{\mathbf{p}}_\ell = (x_\ell, y_\ell, \theta_\ell, v_\ell)$, $t^\star$
**output:** $t_\ell$

1 *Initialize: queue* $q \leftarrow$ *cell that contains* $\hat{\mathbf{p}}_\ell$
2 **while** *q is not empty* **do**
3  Update $len \leftarrow$ current size of $q$;
4  **for** $i \leftarrow 1$ **to** $len$ **do**
5  Remove the front cell from $q$ and make it $\tau_\alpha$;
6  **if** $s_\alpha = O$ *or* $\tau_\alpha$ *contains the space boundary* **then**
7  **if** *line* $\ell$ *intersects* $\tau_\alpha$ *at* $(x_h, y_h) \in \mathbb{R}^2$ **then**
8  Compute $d_\ell \leftarrow \|(x_\ell - x_h, y_\ell - y_h)\|$;
9  Return $t_\ell \leftarrow \frac{d_\ell}{v_\ell}$;
10  **end**
11  **end**
 `// Enqueue the new frontier cells`
12  Push into $q$ the neighbor cells of $\tau_\alpha$ that are located within a distance of $d_\ell^\star = t^\star \times v_\ell$ from $\hat{\mathbf{p}}_\ell$;
13  **end**
14 **end**
15 Return $t_\ell \leftarrow t^\star$ `// If` $t_\ell$ `cannot be found from the above steps`

Algorithm 3 computes $t_\ell$, by sequentially exploring the frontier cells in the quadrant determined by $\theta_\ell$, until *line* $\ell$ intersects a frontier cell that either contains an obstacle or lies on the space boundary. The frontier cells are labeled with incrementing numbers during exploration, i.e., the numbers ⓪ $\sim$ ④ in Fig. 4.3b. Note that *line* $\ell$ cannot intersect a higher numbered frontier cell before intersecting a lower numbered frontier cell. Also, for all frontier cells that are labeled with the same number, *line* $\ell$ can only intersect one of them.

Algorithm 3 uses a queue $q$ to record the frontier cells discovered during exploration. The queue is initialized with the cell that contains the state $\hat{\mathbf{p}}_\ell$ and it is labeled with ⓪ (**Line 1**). As long as the queue is non-empty (**Line 2**), the variable *len* is updated with the size of the queue (**Line 3**). Then, within the for loop, these frontier cells are sequentially extracted

from the front of the queue (**Line 4**), and recorded in a variable $\tau_\alpha$ at each iteration (**Line 5**). Thereafter, it checks whether: (i) $\tau_\alpha$ is occupied by any obstacle (i.e., $s_\alpha = \mathrm{O}$) and *line* $\ell$ intersects with any edge of $\tau_\alpha$; or (ii) $\tau_\alpha$ lies on the space boundary and *line* $\ell$ intersects with its boundary edge (**Line 6-7**). If either is true, then the collision distance $d_\ell$ and the collision time $t_\ell$ are computed based on the hit point $(x_h, y_h) \in \mathbb{R}^2$(**Line 8-9**).

If $t_\ell$ is not found from the above steps, then new frontier cells (i.e., the neighbor cells of $\tau_\alpha$) are enqueued for further computations and they are labeled with a number incremented by 1 (**Line 12**). Note that the new frontier cells are determined by $\theta_\ell$. For example, as shown in Fig. 4.3b, $\theta_\ell$ resides in the first quadrant, hence the new frontier cells are the neighbor cells located in the north and east directions of $\tau_\alpha$.

Since the vehicle is safe at $\hat{\mathbf{p}}_\ell$ when $t_\ell \geq t^\star$, one only needs to search the frontier cells located within a distance of $d_\ell^\star = t^\star \times v_\ell$ from $\hat{\mathbf{p}}_\ell$. A frontier cell is said to be within a distance of $d_\ell^\star$ from $\hat{\mathbf{p}}_\ell$, if the distance between any of its four vertices and the point $(x_\ell, y_\ell)$ is smaller than $d_\ell^\star$.

If no $t_\ell$ is returned and the queue becomes empty, then it implies that *line* $\ell$ does not intersect any obstacle cell or the space boundary up to distance $d_\ell^\star$. Then, $t_\ell = t^\star$, i.e., state $\hat{\mathbf{p}}_\ell$ is risk-free. (**Line 15**).

Based on the collision time $t_\ell$, the risk at $\hat{\mathbf{p}}_\ell$ is defined as

$$risk(\hat{\mathbf{p}}_\ell) = \begin{cases} 1 + \log\left(\frac{t^\star}{t_\ell}\right) & \text{if } t_\ell < t^\star \\ 1 & \text{if } t_\ell \geq t^\star. \end{cases} \tag{4.9}$$

As $t_\ell \to 0$, the risk approaches $+\infty$; while for $t_\ell \geq t^\star$, the risk reduces to 1, i.e., no risk penalty is added to the time cost.

The risk cost for a candidate path $\gamma_{i,i+1}$ is computed as

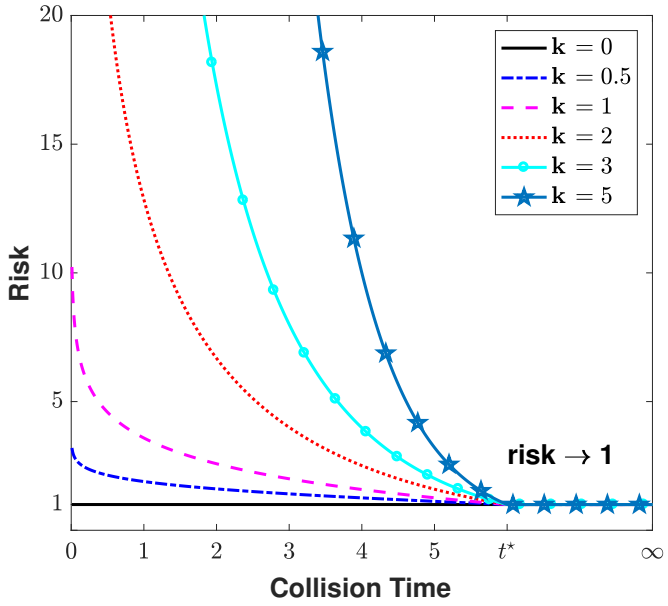



**Figure 4.4:** The risk function $(risk(\hat{\mathbf{p}}_\ell))^{\mathbf{k}}$

$$\mathcal{R}(\gamma_{i,i+1}) = \max_{\ell=1,2,\ldots} \Big(risk(\hat{\mathbf{p}}_\ell)\Big)^{\mathbf{k}}, \tag{4.10}$$

where $\mathbf{k} \geq 0$ is the weight parameter. Fig. 4.4 shows the curves of $(risk(\hat{\mathbf{p}}_\ell))^{\mathbf{k}}$ for different $\mathbf{k}$ when $t^\star = 6$. It is seen that for a fixed $t_\ell \in (0, t^\star)$, a larger $\mathbf{k}$ produces a higher risk. In particular, when $\mathbf{k} = 0$, $\mathcal{R}(\gamma_{i,i+1}) = 1$, thus the total cost $\widetilde{J}(\mathbf{p}_i^m, \mathbf{p}_{i+1}^m) = \mathcal{T}(\gamma_{i,i+1})$, hence the resulting path is time-optimal. Note that $\mathcal{R}(\gamma_{i,i+1}) \in [1, \infty)$, because $risk(\hat{\mathbf{p}}_\ell) \in [1, \infty)$, $\forall \ell$.

### Planning under Uncertainties

During plan execution, the vehicle may have uncertainties in its heading and position. Thus, an uncertainty of $\Delta\theta$ is added to $\theta_\ell$ during planning, and the collision distances $d_\ell^+$ and $d_\ell^-$ corresponding to headings of $\theta + \Delta\theta$ and $\theta - \Delta\theta$ are computed, as shown in Fig. 4.5. Then $t_\ell$ is computed based on $\min\{d_\ell, d_\ell^+, d_\ell^-\}$. Since a modestly priced compass has an accuracy

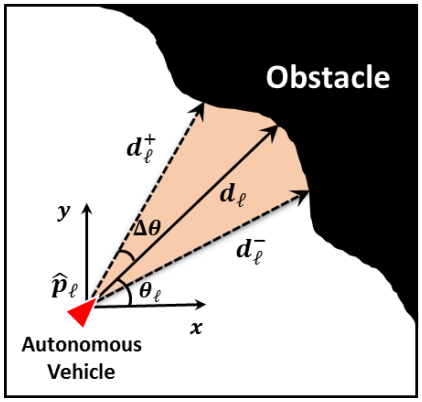



**Figure 4.5:** Uncertainties in the heading angle $\theta_\ell$

of $1^o$ [63], $\Delta\theta$ is chosen as $1.5^o$. Also, a buffer of size 0.1m is added around all obstacles to compensate position uncertainties, such that any path $\gamma_{i,i+1}$ which intersects with the buffer has $\mathcal{R}(\gamma_{i,i+1}) = +\infty$.

### 4.4.5 Adaptive State Pruning for Complexity Reduction

This section presents a three-step adaptive state pruning technique for complexity reduction of $\mathcal{Q}$, as shown in Fig. 4.6. Consider a state $\mathbf{p}_i^m$ and the baseline 8-*orientation* 2-*speed* state expansion in its neighbor cells, as shown in Fig. 4.6a.

1. *Obstacle-based Pruning*: The states close to and facing obstacles or boundaries are considered as inevitable collision states, thus they are pruned, as shown in Fig. 4.6b.

2. *Speed-based Pruning*: In open regions away from obstacles, the vehicle is expected to travel at the highest speed to minimize the time cost; while the low-speed states are typically useful near obstacles to allow turning with a smaller turning radius for better controllability. Therefore, the low-speed states in the cells located far from obstacles can be pruned, as shown in Fig. 4.6c.

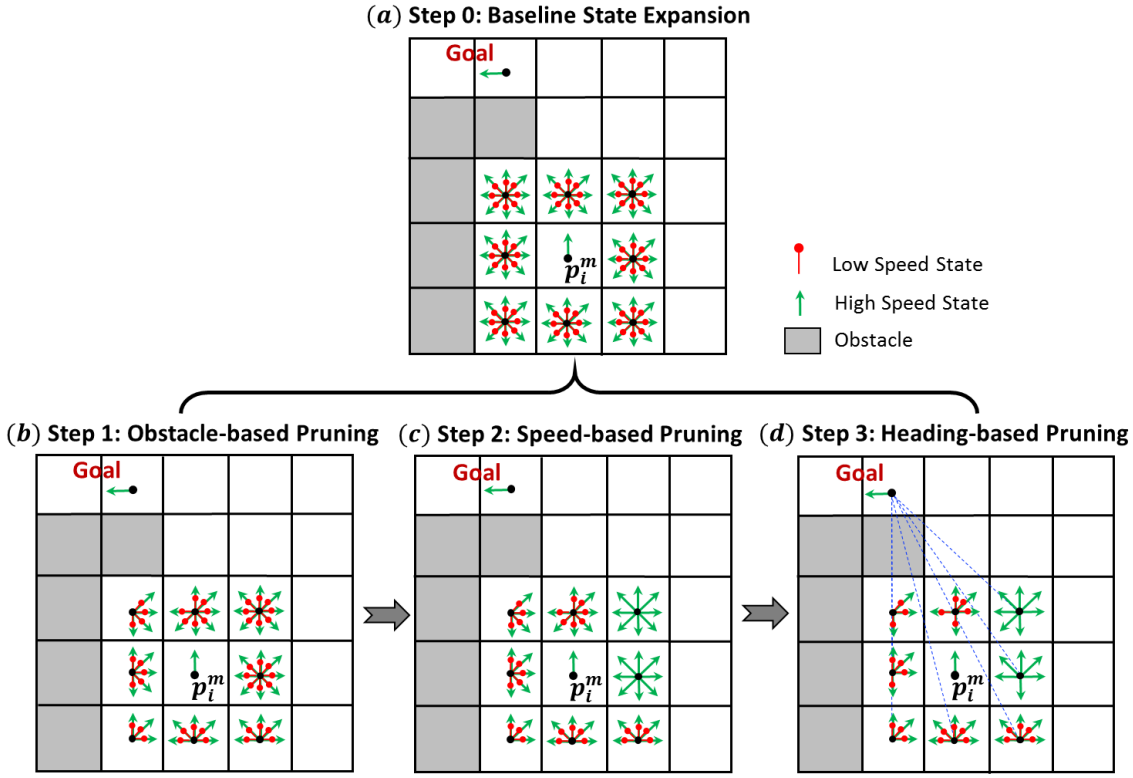



**Figure 4.6:** Illustration of the adaptive state pruning within each cell

3. *Heading-based Pruning*: Since the states with a diagonal heading and in an opposite direction to the goal will very likely produce higher costs, they are less likely to be part of the optimal state sequence. Thus, they can be dynamically identified and removed from $\mathcal{Q}$. Note that the states with non-diagonal heading angles of 0, $\pi/2$, $\pi$ and $3\pi/2$ must be retained to ensure the completeness of the algorithm. In this regard, first connect the centers of each cell and $\mathbf{p}_{goal}$ using a straight line as shown in Fig. 4.6d. Then, for each state with a diagonal heading, compute the angle $\xi \in (-\pi, \pi]$ formed with the corresponding line. If $|\xi| > \zeta$, where $\zeta \in (0, \pi]$ is a pre-defined threshold, then such state is pruned. Fig. 4.6d shows an example when $\zeta = \pi/2$.

### 4.4.6 Searching for the Time-optimal Risk-aware Path

The step-wise cost $\widetilde{J}(\mathbf{p}_i^m, \mathbf{p}_{i+1}^m)$ between any two consecutive states $\mathbf{p}_i^m$ and $\mathbf{p}_{i+1}^m$ is determined by the least product cost of $\mathcal{T}(\gamma_{i,i+1})$ and $\mathcal{R}(\gamma_{i,i+1})$ among all corresponding optimized candidate paths in the OCPS table. Now, for an intermediate state $\mathbf{p}_i^m$, we define

$$f(\mathbf{p}_i^m) = g(\mathbf{p}_{start}, \mathbf{p}_i^m) + h(\mathbf{p}_i^m, \mathbf{p}_{goal}), \tag{4.11}$$

where the cumulative cost function $g$ is

$$g(\mathbf{p}_{start}, \mathbf{p}_i^m) = \sum_{j=1}^{i-1} \widetilde{J}(\mathbf{p}_j^m, \mathbf{p}_{j+1}^m), \tag{4.12}$$

and the heuristic cost $h(\mathbf{p}_i^m, \mathbf{p}_{goal})$ is determined by the length of the shortest Dubins path using turning radius $\mathbf{r}$ divided by the maximum speed $v_{\max}$. Such heuristic is admissible thus it guarantees the optimality of $\mathbf{P}^\star$. Thereafter, we adopt the framework of A$^\star$ algorithm to search for $\mathbf{P}^\star$, where the states are gradually explored and assigned with the cost using Eq. (4.11). The search process repeats until the goal state is found. Then, the time-optimal risk-aware path $\gamma^\star$ can be reconstructed in a piece-wise manner using the states of $\mathbf{P}^\star$ and the optimal paths between the adjacent states of $\mathbf{P}^\star$.

## 4.5 Results and Discussion

The T$^\star$ algorithm has been validated in complex obstacle-rich scenarios. The results were compared with Dubins paths, the effect of risk weight $\mathbf{k}$ in Eq. (4.10) was investigated, and the efficiency of the adaptive state pruning technique in reducing the computational complexity was quantitatively examined.

The autonomous vehicle is subject to the following kinematic constraints: maximum

turn rate $u_{\max} = 0.5$rad/s, and speeds $v_{\min} = 0.5$m/s and $v_{\max} = 1$m/s. Hence, its turning radii are $\mathbf{R} = 2$m and $\mathbf{r} = 1$m. Also, $t^\star$ is chosen as 6s.

The search area $\mathscr{R}$ of size 30m $\times$ 30m is partitioned into a set $\mathscr{T}$ consisting of $15 \times 15$ cells, where each cell is of size 2m $\times$ 2m. The adaptive state pruning method was applied with the threshold $\zeta = \pi/2$.

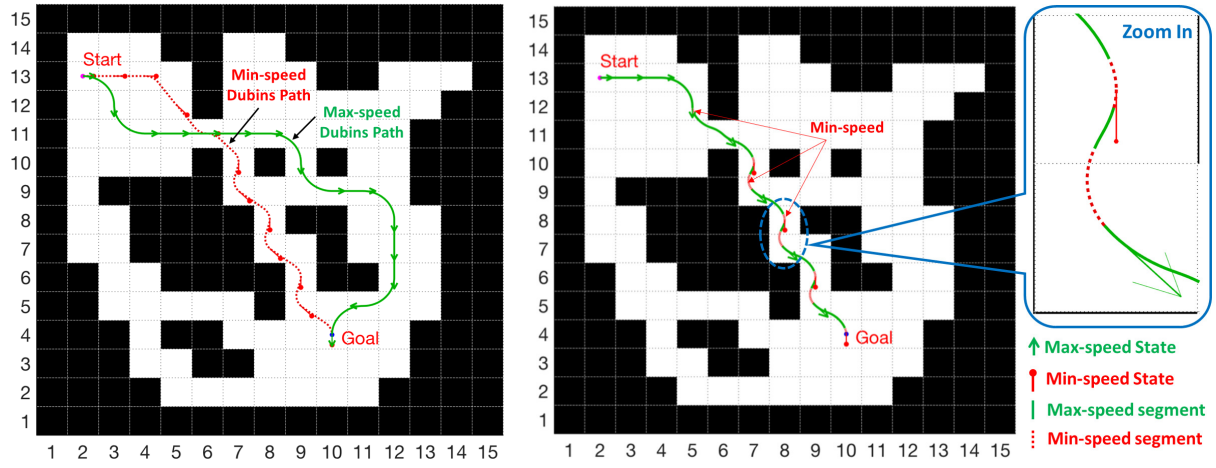
### 4.5.1 Comparison of Time-optimal and Dubins Approaches

First, we compare the time-optimal path with the Dubins paths. The time-optimal path was obtained by using $\mathbf{k} = 0$ in Eq. (4.10), i.e., the effect of risk was not considered. On the other hand, since a Dubins vehicle must move at a constant speed, we generated two such paths one at the maximum and the other at the minimum speed. In this section, for the purpose of illustration, we present the time-optimal paths and the Dubins paths in two obstacle-rich scenarios as follows.

**Scenario 1**: Fig. 4.7a shows the two Dubins paths, while Fig. 4.7b shows the time-optimal path for Scenario 1. The start state was chosen as $\mathbf{p}_{start} = (4\text{m}, 26\text{m}, 0, v_{\max})$, and the goal state was set as $\mathbf{p}_{goal} = (20\text{m}, 8\text{m}, 3\pi/2, v_{\min})$.

Since the Dubins path with $v_{\max}$ results in large turning radius $\mathbf{R}$, its movement through the shortcut taken by the time-optimal path is restricted. Thus, it takes a longer path with a total time cost of 35.99s. On the other hand, the Dubins path with $v_{\min}$ has better controllability with the turning radius $\mathbf{r}$; thus, it produces the minimum-length path through the shortcut. However, it takes 55.95s which is much higher due to the minimum speed.

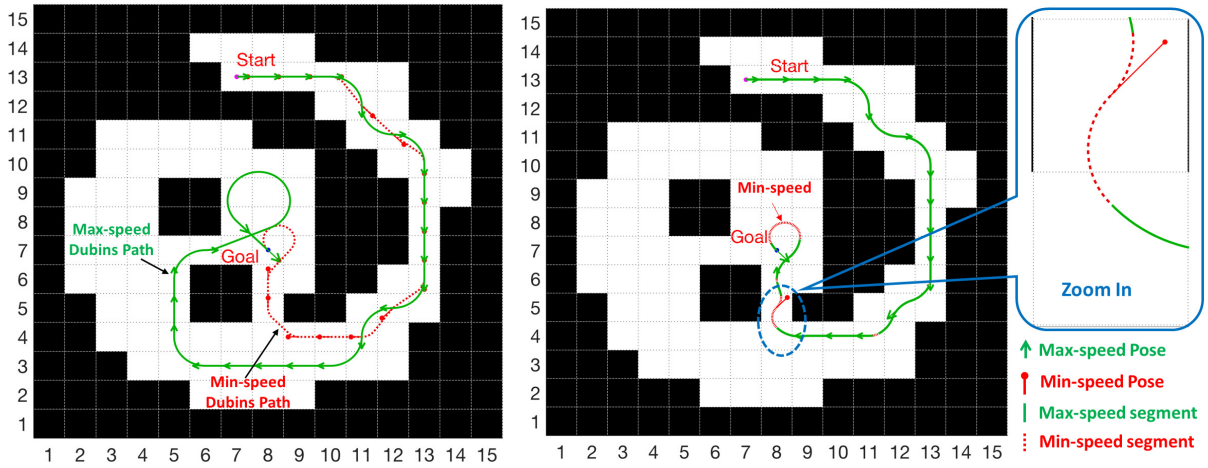
In comparison, the time-optimal path shown in Fig. 4.7b is composed of segments with different speeds. This enables the vehicle to travel at $v_{\max}$ in relatively open regions to reduce the total time cost, while subject to a larger turning radius $\mathbf{R}$. In congested regions,




**(a)** Dubins paths. Total time cost using $v_{\max}$: **35.99s**. Total time cost using $v_{\min}$: **55.95s**

**(b)** Time-optimal path (for $\mathbf{k} = 0$). Total time cost: **34.51s**

**Figure 4.7:** Scenario 1: The Dubins paths for min and max speeds vs. the time-optimal path with variable speed in an obstacle-rich environment




**(a)** Dubins paths. Total time cost using $v_{\max}$: **68.65s**. Total time cost using $v_{\min}$: **89.47s**

**(b)** Time-optimal path (for $\mathbf{k} = 0$). Total time cost: **54.24s**

**Figure 4.8:** Scenario 2: The Dubins paths for min and max speeds vs. the time-optimal path with variable speed in another obstacle-rich environment

it tends to decrease its speed to $v_{\min}$ to gain better maneuverability with a smaller turning radius $\mathbf{r}$. The total time cost of the time-optimal path is 34.51s, which is lower than both the Dubins paths described above.

**Scenario 2**: Fig. 4.8a and Fig. 4.8b show the Dubins paths and the time-optimal path for Scenario 2. The start state was chosen as $\mathbf{p}_{start} = (14\text{m}, 26\text{m}, 0, v_{\max})$, and the goal state was set as $\mathbf{p}_{goal} = (16\text{m}, 14\text{m}, 7\pi/4, v_{\max})$.

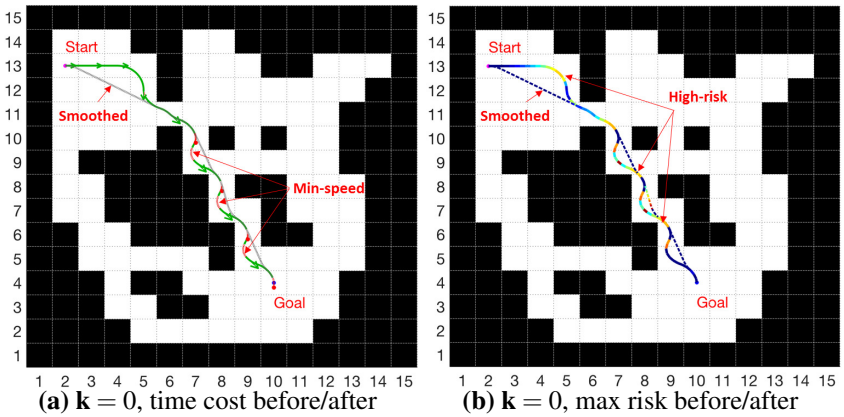
In this example, the obstacles are more congested, and the vehicle must travel through one of the three slim tunnels near cells $(8,5)$, $(6,7)$ and $(5,10)$ before reaching the goal. Due to the large turning radius $\mathbf{R}$ at speed $v_{\max}$, it is seen from Fig. 4.8a that the Dubins path took the route around cell $(6,7)$; however, when the vehicle travels at $v_{\min}$, it was able to travel through the shortcut near cell $(8,5)$ to reach the goal with the minimum length. The time costs associated with the Dubins paths at $v_{\max}$ and $v_{\min}$ are 68.65s and 89.47s, respectively.

On the other hand, the time-optimal path shown in Fig. 4.8b comprises of segments with different speeds. The vehicle tends to utilize the maximum speed $v_{\max}$ at the beginning segments to minimize the time cost; while it reduces its speed to $v_{\min}$ and turns with $\mathbf{r}$ to travel through the shortcut near cell $(8,5)$. Accordingly, the time cost is 54.24s which is much less than both of the Dubins paths mentioned above.

### 4.5.2 Time-optimal Risk-aware Paths for Different k

This section examines the effect of $\mathbf{k}$ in Eq. (4.10) on the motion planning. It is expected that a higher $\mathbf{k}$ should produce a safer path, however, its time cost would be potentially higher.

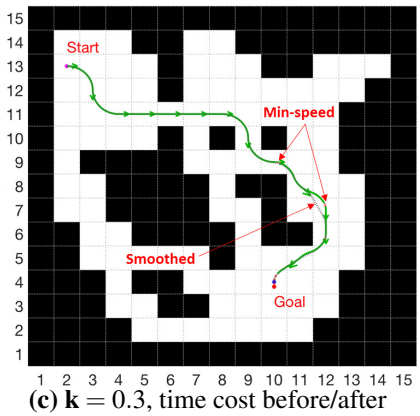
**Scenario 1**: Fig. 4.9a, Fig. 4.9c and Fig. 4.9e show the time-optimal risk-aware paths for $\mathbf{k} = 0$, 0.3 and 3, respectively. These paths are color-coded based on the speed information. Fig. 4.9b, Fig. 4.9d and Fig. 4.9f show the same paths but they are color-coded based on the risk information. The risk of a state along the path was evaluated using Eq. (4.9),



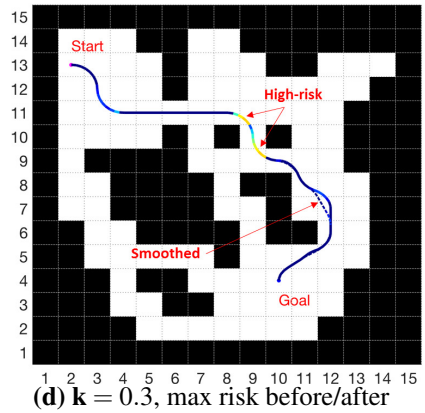

**(a)** $\mathbf{k} = 0$, time cost before/after smoothing: **34.51s/25.51s**



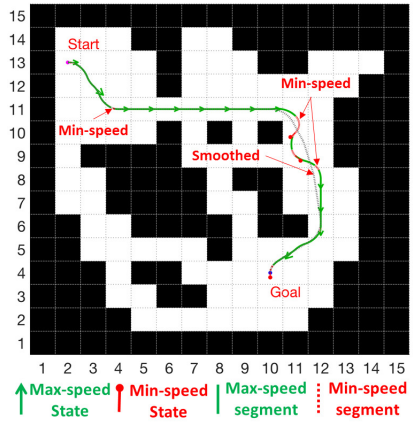

**(b)** $\mathbf{k} = 0$, max risk before/after smoothing: **2.45/2.40**



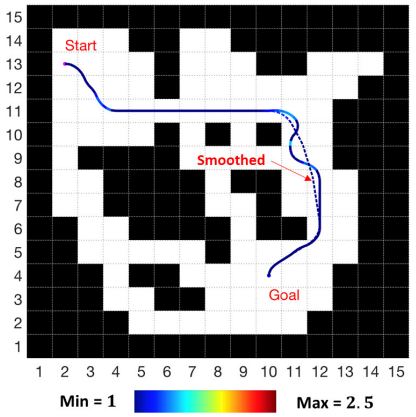

**(c)** $\mathbf{k} = 0.3$, time cost before/after smoothing: **36.30s/33.61s**




**(d)** $\mathbf{k} = 0.3$, max risk before/after smoothing: **2.01/2.01**




**(e)** $\mathbf{k} = 3$, time cost before/after smoothing: **41.89s/35.68s**




**(f)** $\mathbf{k} = 3$, max risk before/after smoothing: **1.48/1.33**

**Figure 4.9:** Scenario 1: Speed (left) and risk (right) encodings of the optimal paths for different **k**

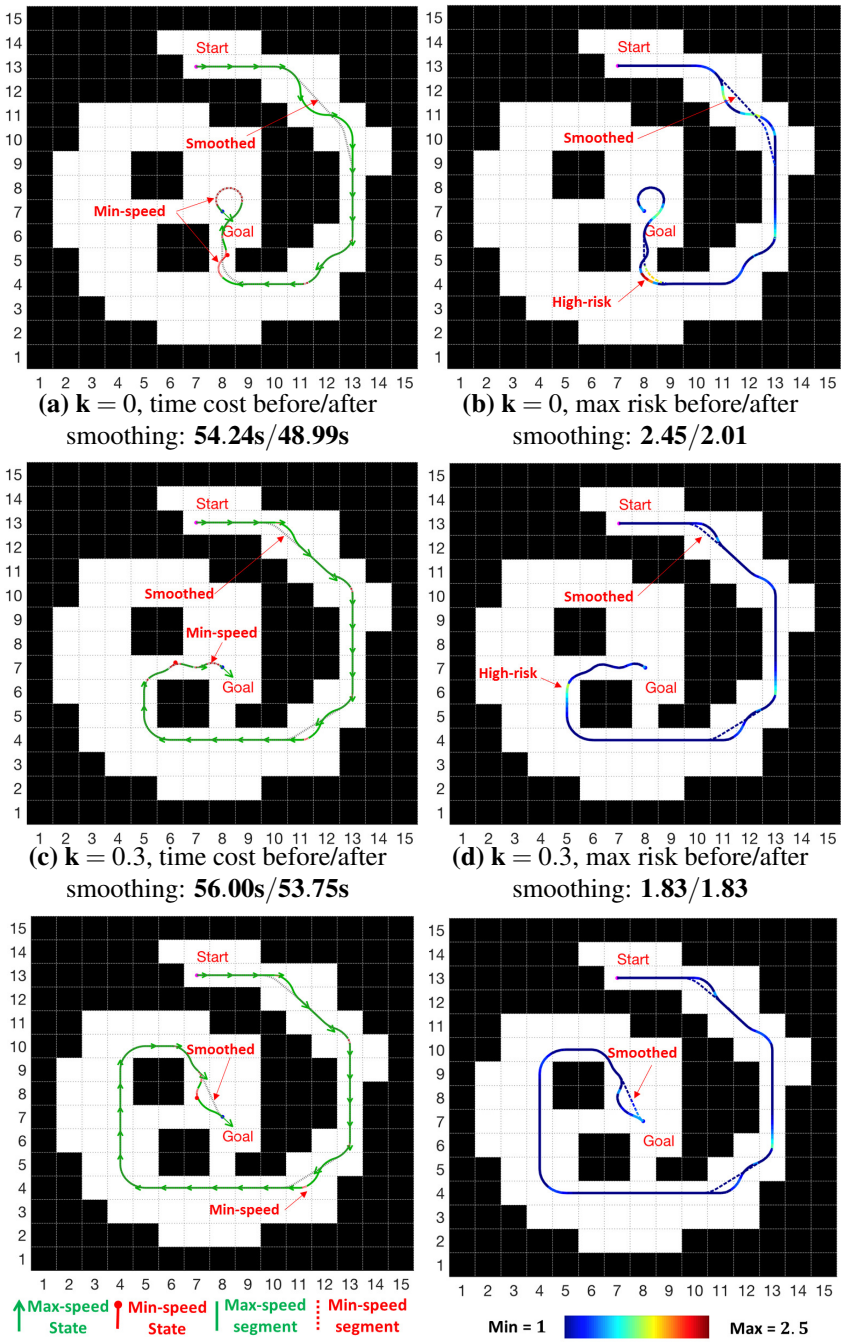
where the sample states were generated using the sampling interval of $\Delta d = 0.4$m.

It is seen that, for $\mathbf{k} = 0$, the time-optimal path reaches the goal in 34.51s through the shortcut in congested regions. However, it has multiple dangerous segments as shown in Fig. 4.9b by the risk color-coding. The max risk of the whole path is 2.45. For $\mathbf{k} = 0.3$, the time-optimal risk-aware path selects a less-risky route to avoid the congested region, but requires a higher time cost of 36.30s. Accordingly, the number of risky segments is significantly reduced; however, the segments in cells $(9,9)$ and $(9,11)$ still present high risk. Further, for $\mathbf{k} = 3$, the resulting path picks the safest route, where the max risk reduces to 1.48, but the corresponding time cost reaches 41.89s.

In addition, a smoothing operation can help remove zigzag-shaped segments, e.g., near cell $(8,7)$ in Fig. 4.9a; and further reduce $J(\mathbf{P}^\star)$. Here, we re-optimize over randomly sampled state pairs along the optimal path and replace with lower-cost collision-free segments if they exist. This operation was repeated for four iterations, and the smoothed paths and their risk color-codings are shown in Fig. 4.9a$\sim$Fig. 4.9f, respectively.

**Scenario 2**: Fig. 4.10 evaluates the T$^\star$ algorithm in another obstacle-rich scenario. The simulations followed the same setup as in Scenario 1.

Fig. 4.10a, Fig. 4.10c and Fig. 4.10e show the time-optimal risk-aware paths for $\mathbf{k} = 0$, 0.3 and 3.5, respectively. As seen in Fig. 4.10a, for $\mathbf{k} = 0$, the time-optimal path can lead the vehicle to the goal in 54.24s via traversing through the shortcut near cell $(8,5)$. Although the vehicle tends to slow down to $v_{\min}$ before entering the such shortcut region, it still consists of multiple dangerous path segments, as shown by the risk color-coding in Fig. 4.10b. Accordingly, the max risk of the whole path is 2.45. When $\mathbf{k}$ increases to 0.3, the optimal path selects a less-risky path, but at the expense of a higher time cost of 56.00s. As shown in Fig. 4.10d, the number of high-risk segments is significantly reduced,




**(a)** $\mathbf{k} = 0$, time cost before/after smoothing: **54.24s/48.99s**

**(b)** $\mathbf{k} = 0$, max risk before/after smoothing: **2.45/2.01**

**(c)** $\mathbf{k} = 0.3$, time cost before/after smoothing: **56.00s/53.75s**

**(d)** $\mathbf{k} = 0.3$, max risk before/after smoothing: **1.83/1.83**

**(e)** $\mathbf{k} = 3.5$, time cost before/after smoothing: **68.92s/63.10s**

**(f)** $\mathbf{k} = 3.5$, max risk before/after smoothing: **1.69/1.66**

**Figure 4.10:** Scenario 2: Speed (left) and risk (right) encodings of the optimal paths for different **k**

**Table 4.2:** Scenario 1: Total costs and computation times using adaptive state pruning

| *Risk Weight* | *State Pruning Threshold* | *Total Cost* $J(\mathbf{P}^\star)$ | *Computation Time* | *Saving in Computation Time* |
|---|---|---|---|---|
| $\mathbf{k}=0$ | None | 33.31 | 259.92s | – |
| | $\zeta=\pi$ | 34.51 | 73.31s | 71.80% |
| | $\zeta=\pi/2$ | 34.51 | 54.29s | 79.11% |
| | $\zeta=\pi/4$ | 34.51 | 38.06s | 85.36% |
| $\mathbf{k}=0.3$ | None | 38.46 | 709.56s | – |
| | $\zeta=\pi$ | 38.87 | 227.90s | 67.88% |
| | $\zeta=\pi/2$ | 38.87 | 169.97s | 76.05% |
| | $\zeta=\pi/4$ | 39.52 | 124.78s | 82.41% |
| $\mathbf{k}=3$ | None | 62.98 | 569.43s | – |
| | $\zeta=\pi$ | 62.98 | 157.95s | 72.26% |
| | $\zeta=\pi/2$ | 62.98 | 116.08s | 79.61% |
| | $\zeta=\pi/4$ | 71.94 | 96.56s | 83.04% |

but there still exist high risks at the path segments near cell $(5,6)$. When $\mathbf{k}$ further increases to 3.5, the optimal path selects the safest route, and the max risk of the whole path reduces to 1.69, but it has the highest time cost of 68.92s.

Moreover, one can apply the same smoothing operation as described in the previous scenario to remove zigzag-shaped segments, e.g., near cell $(11,11)$ in Fig. 4.10a and near cell $(12,5)$ in Fig. 4.10c. The smoothed paths and their risk color-codings in this scenario are presented in Fig. 4.10a$\sim$Fig. 4.10f, respectively.

### 4.5.3 Complexity Reduction by Adaptive State Pruning

This section examines the efficiency of the adaptive state pruning technique in reducing the computational complexity. Table 4.2 and Table 4.3 summarize the total cost $J(\mathbf{P}^\star)$, the average computation time over 5 runs, and the savings in computation time as compared to

**Table 4.3:** Scenario 2: Total costs and computation times using adaptive state pruning

| *Risk Weight* | *State Pruning Threshold* | *Total Cost* $J(\mathbf{P}^\star)$ | *Computation Time* | *Saving in Computation Time* |
|---|---|---|---|---|
| $\mathbf{k} = 0$ | None | 54.24 | 97.20s | – |
| | $\zeta = \pi$ | 54.24 | 69.53s | 28.47% |
| | $\zeta = \pi/2$ | 54.24 | 58.47s | 39.85% |
| | $\zeta = \pi/4$ | 54.76 | 34.82s | 64.18% |
| $\mathbf{k} = 0.3$ | None | 59.50 | 264.51s | – |
| | $\zeta = \pi$ | 59.50 | 204.72s | 22.60% |
| | $\zeta = \pi/2$ | 59.50 | 151.03s | 42.90% |
| | $\zeta = \pi/4$ | 62.87 | 94.59s | 64.24% |
| $\mathbf{k} = 3.5$ | None | 138.56 | 306.10s | – |
| | $\zeta = \pi$ | 138.56 | 218.03s | 28.77% |
| | $\zeta = \pi/2$ | 144.18 | 128.52s | 58.01% |
| | $\zeta = \pi/4$ | 255.52 | 94.74s | 69.05% |

no pruning, for varying $\zeta$, in Scenario 1 and Scenario 2, respectively. For each scenario, the results are presented for three values of the risk weight $\mathbf{k}$. The results are generated using MATLAB on a computer with 3.4GHz CPU and 16GB RAM.

Note that a higher threshold of $\zeta$ would retain more diagonally facing states during the heading-based pruning. The results in Table 4.2 and Table 4.3 show that the adaptive state pruning technique can significantly reduce the computation time. When $\zeta$ is reduced from $\pi$ to $\pi/4$, the computational cost reduces; however, the total optimization cost remains more or less the same, thus revealing the effectiveness of the pruning approach.

## 4.6 Conclusions

This chapter presents the T$^\star$ algorithm to address the time-optimal risk-aware motion planning problem for curvature-constrained variable-speed vehicles. The results show superi-

ority in time savings over the Dubins paths. This algorithm allows users to generate various path choices of decreasing risk at the expense of increasing time cost. The effectiveness of the proposed adaptive state pruning technique was thoroughly tested to reduce the computation time while maintaining the path quality.

# Chapter 5

# Conclusions and Future Work

In the past three decades, we have witnessed fast advancements in motion planning research. Motion planning plays a key role in vehicle autonomy, and it has significantly helped the development and applications for autonomous vehicles. In this context, this dissertation developed three algorithms that have addressed three widely concerned problems in coverage and time-optimal motion planning for autonomous vehicles.

## 5.1 The $\varepsilon^\star$ Algorithm

In Chapter 2, we presented a single-robot coverage path planning algorithm, called $\varepsilon^\star$. The objective is to compute a collision-free path that can guide the autonomous vehicle to completely cover an *a priori* unknown search area.

The $\varepsilon^\star$ algorithm is a grid-based algorithm, where $\varepsilon$ specifies the cell size. For the purpose of navigation, a tiling is first constructed over the search area; however, due to unknown environment, the obstacle cells must be dynamically detected based on real-time

sensor measurements. The most recent environment information about discovered obstacles and explored regions are translated into symbols, and then encoded onto the corresponding $\varepsilon$-cells in the tiling. Thereafter, such symbolic encodings are utilized to construct the MAPS, which are essentially a set of hierarchical two dimensional potential surfaces that can assist the computation of navigational waypoints. By default, the lowest level of the MAPS is used for navigation control, while the higher levels can be sequentially activated only in the occasions of local extrema.

For online coverage control, we developed the concept of the ETM, which uses the MAPS and works as a supervisor to generate navigational commands to the autonomous vehicle. The ETM consists of multiple machine states, and it operates in one state at a time. It is shown in [43] that the ETM always halts in finite time; and upon halting, complete coverage is achieved.

An important advantage of using the $\varepsilon^\star$ algorithm is that, it generates the desired back-and-forth path, while does not need to rely on critical point detection on obstacles. The back-and-forth pattern of path is favorable because it has less number of turns and overlappings, while in general, turning operations can be costly for autonomous vehicles. Moreover, $\varepsilon^\star$ computes the new waypoint based on the information in the local neighborhood, thus requiring low computation time and suitable for real-time applications.

For the purpose of validation, the $\varepsilon^\star$ algorithm has been examined both in simulations and real experiments. Simulations were conducted on the high-fidelity Player/Stage simulator, and the coverage performance in terms of trajectory length and number of turns have been compared with three alternative online coverage methods (i.e., FS-STC, BSA and Brick & Mortar). It is seen that $\varepsilon^\star$ achieved complete coverage in all simulations. Further, the coverage ratio of $\varepsilon^\star$ was evaluated in the presence of uncertainties, where different levels of noises in localization, compass and laser measurements were considered. In addi-

tion, we studied the effects of the size of $\varepsilon$, and a general guide on choosing proper $\varepsilon$ was provided. On the other hand, $\varepsilon^\star$ has also been successfully implemented in a laboratory setting, using a physical autonomous ground vehicle equipped with heterogeneous sensing systems, including an indoor localization system, ten ultrasonic sensors and a laser scanner.

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

## 5.2 The CARE Algorithm

In Chapter 3, we extended the $\varepsilon^\star$ algorithm and presented a multi-robot resilient and efficient coverage path planning algorithm for unknown environment, called *CARE*. The objective is to utilize a team of robots (i.e., unmanned autonomous vehicles) to efficiently cover the search area and secure complete coverage even under some robot failures.

Due to uncertainties in the environment, the robots may encounter unexpected failures during operations, which are caused by various factors such as mechanical failures or battery depletion. Robot failures will immediately lead to coverage gaps, while the critical information within those coverage gaps would not be retrieved without the helps from the rest of the team. It is therefore critical that the robot team is resilient to failures, in the sense that complete coverage is still achieved with the rest of the team in an optimized manner. Moreover, due to unknown environment, it is very likely that all robots would not finish at the same time. This means that the early completed robots will become idle, which prolongs the total coverage time. Thus, it is also important the robots are prevented from idling and can be reallocated in an optimized manner to assist others for efficient coverage.

In this regard, we developed the CARE algorithm that accounts for the above-mentioned

issues. This algorithm operates in a distributed manner, and controls each robot with an event-driven supervisory controller. The supervisor is implemented as a finite state automaton, where the transitions between states are triggered by discrete events. In particular, upon detection of robot failure or idling, the robot will collaborate with its neighbors and independently re-optimize its task assignment. The task reallocation is modeled as a potential game, which incorporates various optimization factors such as task worth, robot locations and their remaining energy levels. The game is carefully designed such that the local task reallocation decision is perfectly aligned with the global potential function for the whole team. Thus, whenever a local game is played, the outcome would always benefit the team. Besides, CARE is shown to guarantee complete coverage as long as one robot is still alive.

The CARE algorithm has been validated in several complex environments, and the coverage performances have been compared with alternative multi-robot coverage approaches (i.e., FR, Non-Co. and Brick & Mortar). Moreover, the effectiveness of CARE was further evaluated by varying different system parameters, including the neighborhood sizes, numbers of deployed robots, and distributions of hidden targets. Besides, we have also discussed a number of practical applications for CARE towards the end of Chapter 3.

Future research areas for multi-robot coverage path planning include: i) opportunistic scheduling [110] to enhance the speed of target discovery, ii) extension of the CARE algorithm to account for restricted communication, iii) integration of SLAM with multi-robot control in absence of localization devices, and iv) consideration of potential threats in different tasks to compute the success probability.

## 5.3 The T$^\star$ Algorithm

In Chapter 4, we presented the T$^\star$ algorithm for time-optimal risk-aware motion planning for curvature-constrained vehicles in known environment. The autonomous vehicle under discussion can travel at variable speeds, but its motion is subject to bounded curvature and symmetric turn rate. This means that its turning motion is limited by a non-zero minimum turning radius.

To the best of our knowledge, the time-optimal motion planning problem for curvature-constrained variable-speed vehicles has not been solved in the presence of obstacles. In this regard, we developed the T$^\star$ algorithm, which is a grid-based method that constructs the optimal path in a piece-wise manner in the discrete domain. The configuration space is constructed by associating each cell with a set of vehicle states, which are located at the center of the cell but with different heading angles and speed choices. The time cost as the vehicle travels between two neighboring states is determined by the time-optimal path; and as presented in [38], it belongs to a sufficient set containing 34 candidate path types.

In addition, due to the presence of obstacles, it is also important that the time-optimal path is safe for the vehicle. Hence, we introduced the concept of collision time to evaluate the risk associated to each vehicle state, and then seamlessly integrated it into the time cost during optimization in T$^\star$. The risk function relies on the full information about the vehicle state, including its relative location to nearby obstacles, heading angle and speed; and its rate of change is controllable by a risk weight. The framework of A$^\star$ was used to search over the configuration space for the optimal state sequence that drives the vehicle to reach the goal state with minimized total costs of time and risk. The T$^\star$ algorithm can provide multiple path choices with decreasing risk costs but at the expense of increasing time costs by tuning the risk weight. Further, we also presented an adaptive state pruning technique

to reduce the complexity of the configuration space, so as to significantly decrease the total computation time.

This algorithm has been validated in several complex scenarios, and the results showed clear advantages over Dubins paths in terms of time costs. Moreover, we tested the effects of using different risk weights, where the vehicle tends to select safer path and stay away from obstacles with a higher risk weight. Also, the effectiveness of the adaptive state pruning technique has been numerically examined using different pruning thresholds.

Future research areas for this topic include: i) incorporation of other motion constraints into planning, such as continuous curvature [111], to enhance the tractability of the resulting path, ii) consideration of external drifts [112] in the environment such as wind or ocean currents, and iii) extension to operate in unknown environment.

# Bibliography


[1] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, 2005.

[2] 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.

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

[4] J. Song and S. Gupta, "SLAM based shape adaptive coverage control using autonomous vehicles," in *Proceedings of the IEEE International Conference on System of Systems Engineering*, (San Antonio, Texas), pp. 268–273, 2015.

[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. Gutmann, M. Fukuchi, and M. Fujita, "3D perception and environment map generation for humanoid robot navigation," *The International Journal of Robotics Research*, vol. 27, no. 10, pp. 1117–1134, 2008.

[7] Z. Shen, J. Song, K. Mittal, and S. Gupta, "Autonomous 3-D mapping and safe-path planning for underwater terrain reconstruction using multi-level coverage trees," in *Proceedings of the MTS/IEEE OCEANS'17*, (Anchorage, Alaska), pp. 1–6, 2017.

[8] Z. Shen, J. Song, K. Mittal, and S. Gupta, "An autonomous integrated system for 3-D underwater terrain reconstruction," in *Proceedings of the MTS/IEEE OCEANS'16*, (Monterey, California), pp. 1–6, 2016.

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

[10] J. Song, K. Qiu, S. Gupta, and J. Hare, "SLAM based adaptive navigation of AUVs for oil spill cleaning," in *Proceedings of the MTS/IEEE OCEANS'14*, (St. John's, Newfoundland, Canada), pp. 1–6, 2014.

[11] P. F. Santana, J. Barata, and L. Correia, "Sustainable robots for humanitarian demining," *International Journal of Advanced Robotic Systems*, vol. 4, no. 2, pp. 207–218, 2007.

[12] 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.

[13] B. Paden, M. Čáp, S. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[14] H. Choset and P.Pignon, "Coverage path planning: The boustrophedon cellular decomposition," in *Field and Service Robotics*, pp. 203–209, Springer, 1998.

[15] H. Choset, "Coverage of known spaces: the boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, pp. 247–253, 2000.

[16] 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.

[17] Y. Gabriely and E. Rimon, "Competitive on-line coverage of grid environments by a mobile robot," *Computational Geometry*, vol. 24, no. 3, pp. 197–224, 2003.

[18] 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*, (Barcelona, Spain), pp. 2040–2044, 2005.

[19] 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.

[20] J. Carlson and R. Murphy, "How UGVs physically fail in the field," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 423–437, 2005.

[21] C. Rieger, D. Gertman, and M. McQueen, "Resilient control systems: next generation design research," in *Proceedings of the IEEE Conference on Human System Interactions*, (Catania, Italy), pp. 632–636, 2009.

[22] N. Agmon, N. Hazon, G. Kaminka, M. Group, *et al.*, "The giving tree: constructing trees for efficient offline and online multi-robot coverage," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 143–168, 2008.

[23] X. Zheng, S. Koenig, D. Kempe, and S. Jain, "Multirobot forest coverage for weighted and unweighted terrain," *IEEE Transactions on Robotics*, vol. 26, no. 6, pp. 1018–1031, 2010.

[24] J. Song, S. Gupta, and J. Hare, "Game-theoretic cooperative coverage using autonomous vehicles," in *Proceedings of the MTS/IEEE OCEANS'14*, (St. John's, Newfoundland, Canada), pp. 1–6, 2014.

[25] R. Luna and K. Bekris, "Efficient and complete centralized multi-robot path planning," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3268–3275, 2011.

[26] W. Sheng, Q. Yang, J. Tan, and N. Xi, "Distributed multi-robot coordination in area exploration," *Robotics and Autonomous Systems*, vol. 54, pp. 945–955, 2006.

[27] T. Cormen, *Introduction to algorithms*. MIT press, 2009.

[28] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[29] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[30] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Intelligent Unmanned Ground Vehicles*, pp. 203–220, Springer, 1997.

[31] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[32] L. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[33] X. Bui, J. Boissonnat, P. Soueres, and J. Laumond, "Shortest path synthesis for dubins non-holonomic robot," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (San Diego, California), pp. 2–7, 1994.

[34] S. Lazard, J. Reif, and H. Wang, "The complexity of the two dimensional curvature constrained shortest-path problem," in *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, (Houston, Texas), pp. 49–57, 1998.

[35] P. Agarwal, T. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides, "Curvature-constrained shortest paths in a convex polygon," *SIAM Journal on Computing*, vol. 31, no. 6, pp. 1814–1851, 2002.

[36] S. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[37] J. Boissonnat, A. Cérézo, and J. Leblond, "Shortest paths of bounded curvature in the plane," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Nice, France), pp. 2315–2320, 1992.

[38] A. Wolek, E. Cliff, and C. Woolsey, "Time-optimal path planning for a kinematic car with variable speed," *Journal of Guidance, Control, and Dynamics*, pp. 2374–2390, 2016.

[39] A. Pereira, J. Binney, G. Hollinger, and G. Sukhatme, "Risk-aware path planning for autonomous underwater vehicles using predictive ocean models," *Journal of Field Robotics*, vol. 30, no. 5, pp. 741–762, 2013.

[40] B. Pfeiffer, R. Batta, K. Klamroth, and R. Nagi, "Probabilistic modeling for UAV path planning in the presence of threat zones," in *Handbook of Military Industrial Engineering, Edited by A. B. Badiru and M.U. Thomas*, pp. 5–1–5–18, Taylor and Francis Group LLC, 2009.

[41] J. Hernández, M. Moll, E. Vidal, M. Carreras, and L. Kavraki, "Planning feasible and safe paths online for autonomous underwater vehicles in unknown environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Daejeon, South Korea), pp. 1313–1320, 2016.

[42] T. Fraichard and H. Asama, "Inevitable collision states: A step towards safer robots?," *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[43] J. Song and S. Gupta, "$\varepsilon^\star$: An online coverage path planning algorithm," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 526–533, 2018.

[44] J. Marden, G. Arslan, and J. Shamma, "Joint strategy fictitious play with inertia for potential games," *IEEE Transactions on Automatic Control*, vol. 54, no. 2, pp. 208–220, 2009.

[45] J. Marden, G. Arslan, and J. Shamma, "Cooperative control and potential games," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 39, no. 6, pp. 1393–1407, 2009.

[46] 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), pp. 533–538, 1993.

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] E. Ferranti, N. Trigoni, and M. Levene, "Brick & Mortar: an on-line multi-agent exploration algorithm," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Roma, Italy), pp. 761–767, 2007.

[53] M. Andries and F. Charpillet, "Multi-robot taboo-list exploration of unknown structured environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Hamburg, Germany), pp. 5195–5201, 2015.

[54] E. Kaplan and C. Hegarty, *Understanding GPS: Principles and Applications*. Artech House Publishers, Norwood, 2005.

[55] 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.

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

[57] 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.

[58] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[59] 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), pp. 4867–4872, 2012.

[60] S. Gupta, A. Ray, and S. Phoha, "Generalized Ising model for dynamic adaptation in autonomous systems," *Europhysics Letters*, vol. 87, no. 1, p. 10009, 2009.

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

[62] 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*, vol. 1, (Coimbra, Portugal), pp. 317–323, 2003.

[63] 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.

[64] 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.

[65] K. Kneip, F. Tâche, G. Caprari, and R. Siegwart, "Characterization of the compact Hokuyo URG-04LX 2D laser range scanner," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Kobe, Japan), pp. 1447–1454, 2009.

[66] N. Hazon and G. Kaminka, "On redundancy, efficiency, and robustness in coverage for multiple robots," *Robotics and Autonomous Systems*, vol. 56, no. 12, pp. 1102–1114, 2008.

[67] D. Monderer and L. Shapley, "Potential games," *Games and Economic Behavior*, vol. 14, no. 1, pp. 124–143, 1996.

[68] M. Batalin and G. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Distributed Autonomous Robotic Systems 5*, pp. 373–382, Springer, 2002.

[69] D. Latimer, S. Srinivasa, V. Lee-Shue, S. Sonne, H. Choset, and A. Hurst, "Towards sensor based coverage with robot teams," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 1, (Washington D.C.), pp. 961–967, 2002.

[70] I. Rekleitis, A. New, E. Rankin, and H. Choset, "Efficient boustrophedon multi-robot coverage: an algorithmic approach," *Annals of Mathematics and Artificial Intelligence*, vol. 52, no. 2-4, pp. 109–142, 2008.

[71] W. Sheng, Q. Yang, J. Tan, and N. Xi, "Distributed multi-robot coordination in area exploration," *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 945–955, 2006.

[72] S. Rutishauser, N. Correll, and A. Martinoli, "Collaborative coverage using a swarm of networked miniature rob," *Robotics and Autonomous Systems*, vol. 57, pp. 517–525, 2009.

[73] L. Xu and A. Stentz, "An efficient algorithm for environmental coverage with multiple robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 4950–4955, 2011.

[74] S. Bhattacharya, R. Ghrist, and V. Kumar, "Multi-robot coverage and exploration on riemannian manifolds with boundaries," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 113–137, 2014.

[75] N. Karapetyan, K. Benson, C. McKinney, P. Taslakian, and I. Rekleitis, "Efficient multi-robot coverage of a known environment," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vancouver, Canada), pp. 1846–1852, 2017.

[76] N. Karapetyan, J. Moulton, J. Lewis, A. Li, J. O'Kane, and I. Rekleitis, "Multi-robot Dubins coverage with autonomous surface vehicles," in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Brisbane, Australia), pp. 2373–2379, 2018.

[77] S. Yang and C. Luo, "A neural network approach to complete coverage path planning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 1, pp. 718–724, 2004.

[78] M. Tolley, R. Shepherd, B. Mosadegh, K. Galloway, M. Wehner, M. Karpelson, R. Wood, and G. Whitesides, "A resilient, untethered soft robot," *Soft Robotics*, vol. 1, no. 3, pp. 213–223, 2014.

[79] S. Koos, A. Cully, and J. Mouret, "Fast damage recovery in robotics with the t-resilience algorithm," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1700–1723, 2013.

[80] K. Saulnier, D. Saldana, A. Prorok, G. Pappas, and V. Kumar, "Resilient flocking for mobile robot teams," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1039–1046, 2017.

[81] H. Sun, C. Peng, T. Yang, H. Zhang, and W. He, "Resilient control of networked control systems with stochastic denial of service attacks," *Neurocomputing*, vol. 270, pp. 170–177, 2017.

[82] A. Islam, A. Alim, C. Hyder, and K. Zubaer, "Digging the innate reliability of wireless networked systems," in *Proceedings of the International Conference on Networking Systems and Security*, (Dhaka, Bangladesh), pp. 1–10, 2015.

[83] M. Jongerden and B. Haverkort, "Which battery model to use?," *IET Software*, vol. 3, no. 6, pp. 445–457, 2009.

[84] L. Cloth, M. Jongerden, and B. Haverkort, "Computing battery lifetime distributions," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, (Edinburgh, UK), pp. 780–789, 2007.

[85] J. Song and S. Gupta, "CARE: Cooperative autonomy for resilience and efficiency of robot teams for complete coverage of unknown environment under robot failures," *Autonomous Robots*, Under Review, 2019.

[86] W. Chen, S. Toueg, and M. Aguilera, "On the quality of service of failure detectors," *IEEE Transactions on Computers*, vol. 51, no. 5, pp. 561–580, 2002.

[87] Y. Song, S. Wong, and K. Lee, "Optimal gateway selection in multi-domain wireless networks: a potential game perspective," in *Proceedings of the Annual International Conference on Mobile Computing and Networking*, (Las Vegas, Nevada), pp. 325–336, 2011.

[88] H. Dai, Y. Huang, and L. Yang, "Game theoretic max-logit learning approaches for joint base station selection and resource allocation in heterogeneous networks," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 6, pp. 1068–1081, 2015.

[89] R. Myerson, *Game theory*. Harvard university press, 2013.

[90] G. Arslan, J. Marden, and J. Shamma, "Autonomous vehicle-target assignment: A game-theoretical formulation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 584–596, 2007.

[91] J. Song, S. Gupta, and T. Wettergren, "T$^\star$: Time-optimal risk-aware motion planning for curvature-constrained vehicles," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 33–40, 2019.

[92] J. Song, S. Gupta, and T. Wettergren, "Time-optimal path planning for underwater vehicles in obstacle constrained environments," in *Proceedings of the MTS/IEEE OCEANS'17*, (Anchorage, Alaska), pp. 1–6, 2017.

[93] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, *et al.*, "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[94] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[95] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 2, pp. 100–107, 1968.

[96] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A$^\star$," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[97] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[98] J. Boissonnat, A. Cérézo, and J. Leblond, "Shortest paths of bounded curvature in the plane," *Journal of Intelligent and Robotic Systems*, vol. 11, no. 1-2, pp. 5–20, 1994.

[99] J. Faigl and P. Váňa, "Surveillance planning with bézier curves," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 750–757, 2018.

[100] P. Salaris, D. Fontanelli, L. Pallottino, and A. Bicchi, "Shortest paths for a robot with nonholonomic and field-of-view constraints," *IEEE Transactions on Robotics*, vol. 26, no. 2, pp. 269–281, 2010.

[101] R. Pěnička, J. Faigl, P. Váňa, and M. Saska, "Dubins orienteering problem," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1210–1217, 2017.

[102] N. Tsiogkas and D. Lane, "Dcop: Dubins correlated orienteering problem optimizing sensing missions of a nonholonomic vehicle under budget constraints," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2926–2933, 2018.

[103] T. McGee and J. Hedrick, "Optimal path planning with a kinematic airplane model," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 2, pp. 629–633, 2007.

[104] A. Wolek and C. Woolsey, "Feasible Dubins paths in presence of unknown, unsteady velocity disturbances," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 4, pp. 782–787, 2014.

[105] D. Liu, M. Cong, and Y. Du, "Episodic memory-based robotic planning under uncertainty," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 2, pp. 1762–1772, 2017.

[106] L. D. Filippis, G. Guglieri, and F. Quagliotti, "A minimum risk approach for path planning of uavs," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1, pp. 203–219, 2011.

[107] M. Davoodi, "Bi-objective path planning using deterministic algorithms," *Robotics and Autonomous Systems*, vol. 93, pp. 105–115, 2017.

[108] B. Wang, S. Li, J. Guo, and Q. Chen, "Car-like mobile robot path planning in rough terrain using multi-objective particle swarm optimization algorithm," *Neurocomputing*, vol. 282, pp. 42–51, 2018.

[109] H. Huang and A. V. Savkin, "Viable path planning for data collection robots in a sensing field with obstacles," *Computer Communications*, vol. 111, pp. 84–96, 2017.

[110] J. Hare, S. Gupta, and T. Wettergren, "POSE: Prediction-based opportunistic sensing for energy efficiency in sensor networks using distributed supervisors," *IEEE Transactions on Cybernetics*, vol. 48, no. 7, pp. 2114–2127, 2018.

[111] A. Scheuer and T. Fraichard, "Continuous-curvature path planning for car-like vehicles," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2, (Grenoble, France), pp. 997–1003, 1997.

[112] E. Bakolas and P. Tsiotras, "Optimal synthesis of the Zermelo–Markov–Dubins problem in a constant drift field," *Journal of Optimization Theory and Applications*, vol. 156, no. 2, pp. 469–492, 2013.