# Multi-robot path planning for coverage in cluttered environment

Raphaël Dousson

Semester project

Under the supervision of:
Kai Ren
Prof. Maryam Kamgarpour

30.01.2025

sycamore lab
SYSTEMS CONTROL AND MULTIAGENT OPTIMIZATION RESEARCH

École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland

# Abstract

Multi-robot coverage path planning is essential in various industrial and household applications. Traditional solutions, primarily based on Spanning Trees (ST), are effective in large, obstacle-free environments but face limitations in cluttered environments. This project builds on the DARP algorithm, which divides the workspace into subareas for each robot, enabling collision-free single-robot coverage path planning within each subarea. Instead of relying on ST-based coverage, a greedy coverage path planning approach is adopted to address the challenges posed by cluttered environments. The proposed method is evaluated in diverse scenarios and validated on hardware using Nvidia JetBots.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

Many applications require a coverage of an area in a given environment, such as "vacuum cleaning robots, painter robots, autonomous underwater vehicles creating image mosaics, demining robots, lawn mowers, automated harvesters, window cleaners and inspection of complex underwater structures" [1]. In a cluttered environment, specifically, one could envision applications such as surveillance or rescue. Our project focuses on coverage where the robot's action space is larger than its physical footprint. This is particularly relevant when the robot uses a sensor, such as a LiDAR or a camera, to cover an area. In such cases, the robot itself does not need to physically traverse the entire area. Instead, only the sensor's range must fully cover the target area.

For applications requiring rapid coverage, such as surveillance or rescue missions, deploying multiple robots to perform coverage simultaneously can reduce the overall time compared to using a single robot.

In order to obtain a robust and optimal coverage, following objectives are given in [1]:

1. Complete coverage
2. Minimal Path overlapping or repetition
3. Obstacle avoidance
4. Simple motion to simplify the control
5. Fair work division across the agents
6. Collision avoidance with other agents

With the two last objectives being specific to multi-robot coverage.

## 1.2 Previous Work

[2] reviews some of the latest single robot coverage path planning algorithms (CPP) and give following classification:

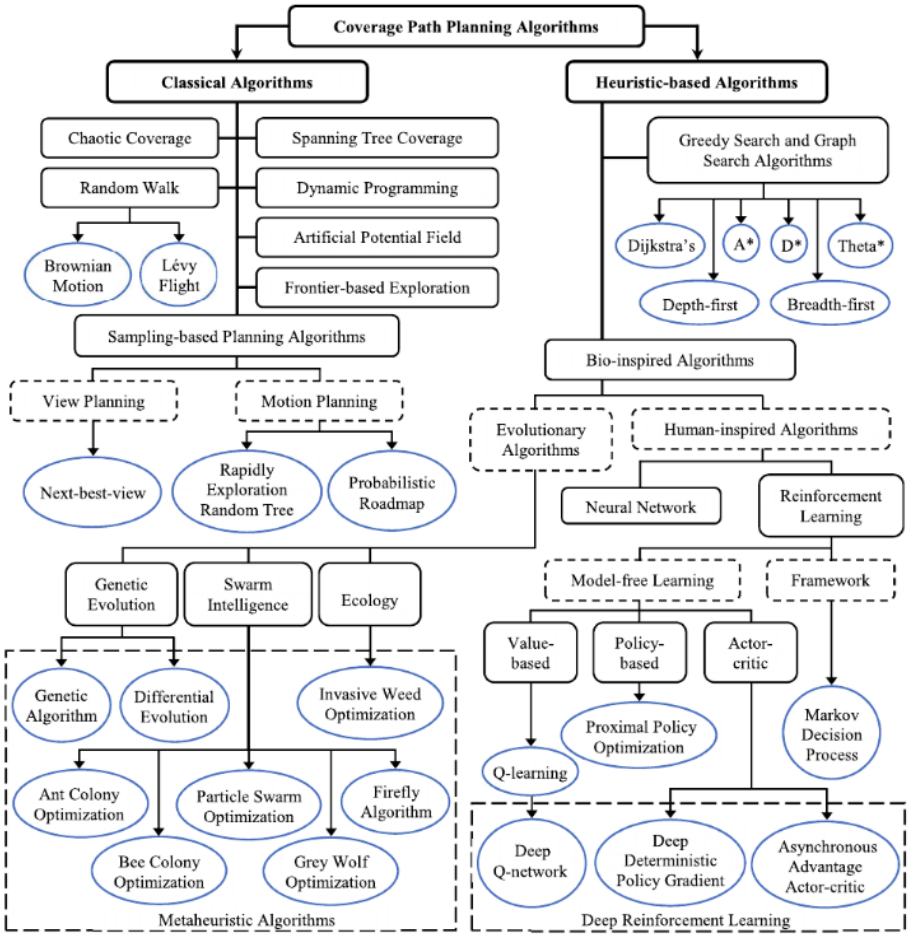



FIGURE 4. The classification of coverage path planning (CPP) algorithms.

Figure 1.1: CPP classification [2]

Some of these algorithms were extended to multi-robot coverage path planning (mCPP). One of the most studied algorithm for this is spanning tree coverage (STC) [3], which was extended multiple times. The idea behind these algorithms is to build a spanning tree to cover the whole area and then split it across each agents. The main challenge is to ensure fair work division across the agents. The spanning tree is therefore built to maximize the work division fairness, which was achieve in the MSTC* [4]. This algorithm was then extended into the TMSTC* [5] to minimize the number of turns.

Another approach is to first divide the work across the agents and then build a spanning tree for each of the agents individually. The DARP (divide areas based on initial positions) algorithm [6] uses such an approach.

STC is inherently complete with no overlapping, which enables collision avoidance among agents. While STC shows great potential for multi-robot coverage, its major drawback is the necessity to divide each cell into four sub-cells to perform path planning. This limitation is negligible in large, open environments but becomes problematic in cluttered environments. Specifi-

cally, passages narrower than twice the desired coverage cell size are treated as obstacles during the area division, resulting in incomplete coverage.

To address this limitation, the DARP approach followed by STC was extended in [7], introducing Up-First STC (UF-STC) to include subcells initially considered as obstacles. However, this solution introduces path overlapping leading to highly suboptimal results in cluttered environments.

## 1.3 Contribution

To perform multi-robot coverage in cluttered and known environments, our pipeline consists of task (area) division, high-level path planning and low-level control. the DARP algorithm is promising, as it allows fair work division. Nevertheless, since the cell subdivision required by STC is not suitable for cluttered environments, it is replaced by another path planner. Our main focus is optimizing the high-level path planning. We explore an $\epsilon^*$ based coverage path planner, which provides a non-iterative greedy solution without cell subdivision. Additionally, the environment representation will also be adapted to account for obstacles significantly smaller than the cell size, with the objectives of reducing path overlapping and minimizing the number of turns.

An extension of DARP, called A* DARP, was developed in [7] and [8] to consider obstacles in the fair work division phase of the algorithm. We will extend this solution to also account for obstacles in the connectivity phase of the algorithm.

Finally, the robustness of the algorithms pipeline will be tested on different scenarios and will be tested on Nvidia Jetbots [9].

# Chapter 2

# Environment Representation

## 2.1 Grid Representation

The environment with obstacles is known. A representation of it must be defined.

For coverage, one suited and widespread used solution is a grid-based representation with square cells. Many others solutions have been developed such as cells with other polygonal shapes such as triangles or hexagons. Another option is to not decompose the area or to do exact cellular decomposition. On one hand this removes the errors in the obstacle representation due to the grid discretization. On the other hand, this implies using a simple path planning such as spirals or back and forth motions, which are not suited in our cluttered environment. We will therefore keep a grid-based representation of the environment. [10]

## 2.2 Environment

The environment on which the project is built is shown in Figure 2.1, even if any other environment being fully connected would work aswell. The environment has a dimension of 3 [m] by 6 [m]. The obstacles are tapes on the ground representing walls with a width of 5 [cm]. In Figure 2.1, a grid with a cell size of 50 [cm] is plotted alongside the obstacles.

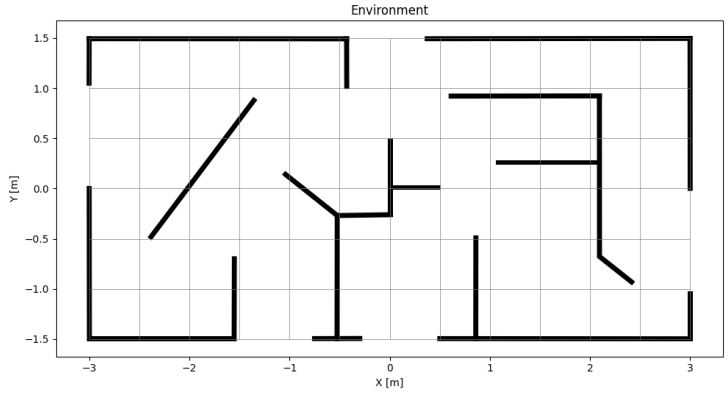



Figure 2.1: Environment

## 2.3 Obstacles Representation

To discretize the environment, a common technique is to mark as an obstacle, each cell which overlaps with any of the obstacle. In our environment, it would lead to the discretization shown in Figure 2.2. This discretization does not suit our application since most of the area is marked as obstacle, so that no correct coverage can be performed.

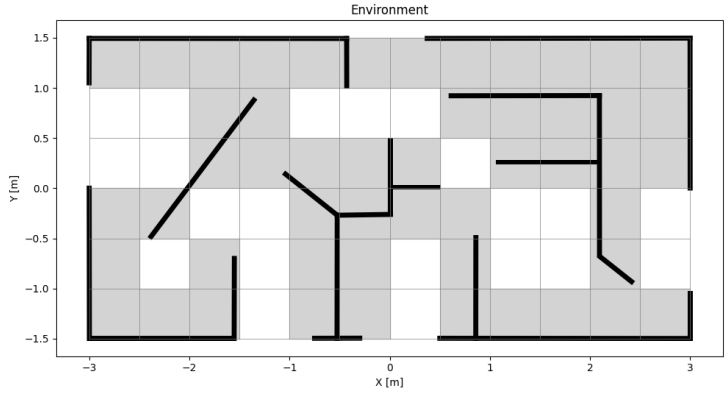



Figure 2.2: Environment obstacle full cells

Another option is to look for intersection between a disk with the robots radius, centered on the cells center and any obstacle instead of the complete cell. This leads to the discretization shown in Figure 2.3 with a robot's radius of 10 [cm]. This environment representation does allow coverage since all cells are connected. Nevertheless, two neighbor cells are not necessary connected, because not all the obstacles are taken into account in this representation.

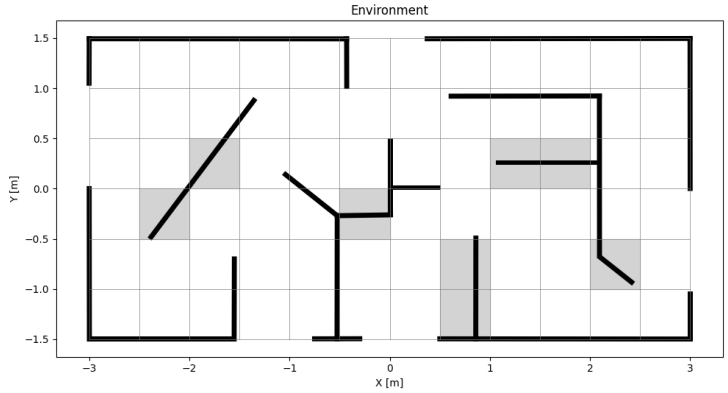



Figure 2.3: Environment obstacle robots radius

To complete the environment representation, we introduce following steps, which are performed for each free cell (white cells in Figure 2.3):

1. The paths to each of the eight neighboring cells are computed, allowing diagonal motions.
2. Each path is enlarged by the robot's radius.
3. The intersections between these paths and all obstacles are calculated.

If there is no intersection, the path is deemed valid and can be followed by a robot without any risk of collision. Otherwise, the path is considered invalid and should be avoided to prevent collisions with obstacles. Examples of paths without intersections are shown in Figure 2.4, while examples with intersections are depicted in Figure 2.5.

The intersection checks are implemented using the intersects function from the Python library Shapely [11]. Paths and obstacles are represented by the coordinates of their vertices, enabling the use of any polygon-shaped obstacle in the implementation.

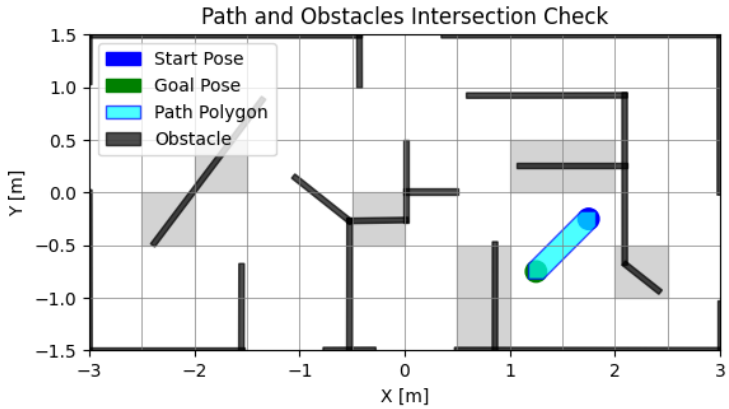



Figure 2.4: Obstacle and path intersection check : valid

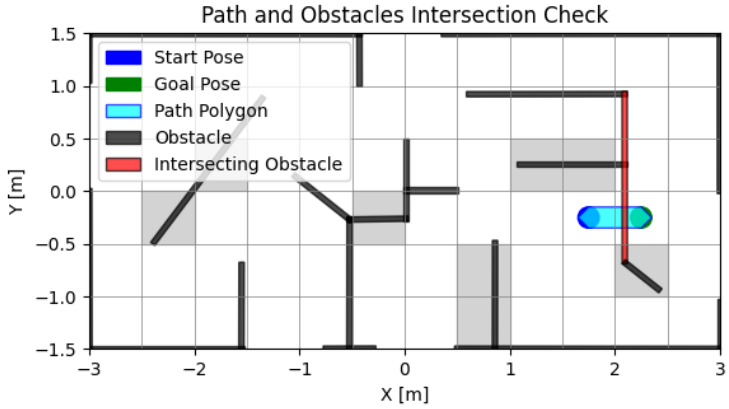



Figure 2.5: Obstacle and path intersection check : not valid

Once all intersections are calculated, the **Movement Map** shown in Figure 2.6 is generated. In this map, blue arrows indicate all valid motions that ensure collision avoidance with obstacles.

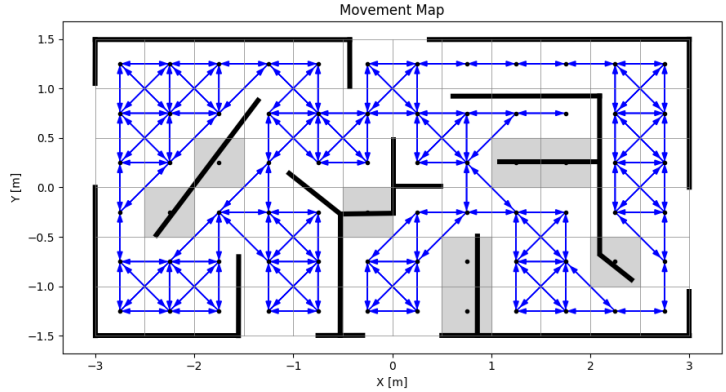
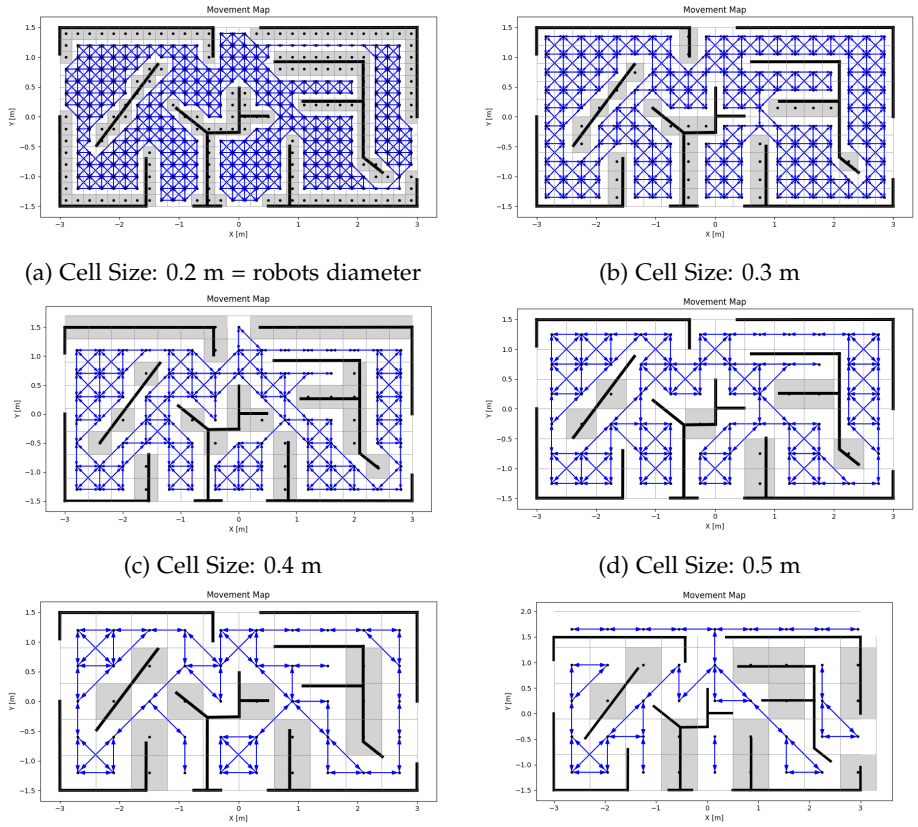



Figure 2.6: Environment Movement Map

In the implementation, the map is stored in an array with dimensions $\dim(x) \times \dim(y) \times 8$, where $\dim(x)$ represents the number of cells along the $x$-axis, and $\dim(y)$ represents the number of cells along the $y$-axis. For each cell $x, y$, an array of dimension 8, stores a boolean containing the information if the corresponding motion is allowed. The motion are stored in following arbitrary order: left, right, up, down, down-left, up-left, down-right, up-right. This allows the computation of the intersection checks to be done for each cell only once at the beginning. The stored information is then used in any algorithm which requires it.

## 2.4 Results

As a result, the following environments can be defined with different cell sizes. These environments will be used to test the algorithm pipeline's robustness against various environment configurations.




(a) Cell Size: 0.2 m = robots diameter

(b) Cell Size: 0.3 m

(c) Cell Size: 0.4 m

(d) Cell Size: 0.5 m

(e) Cell Size: 0.6 m

(f) Cell Size: 0.7 m - not used

Figure 2.7: Environments for different cell sizes

These environments are generated with a robot's radius of 10 [cm]. The first environment has a cell size corresponding to the robot's diameter, making it the smallest grid environment for this robot's radius. Cell sizes larger than 0.6 result in a non-fully connected environment and are therefore not considered (see Figure 2.7f).

# Chapter 3

# Area Division

## 3.1 DARP Algorithm

The ”Divide Areas based on Robot’s Initial Positions” (DARP) algorithm, developed in [6], divides the area into subareas for each robot based on their initial positions. This allows for individual robot coverage path planning within their assigned subarea. The algorithm fully addresses work division and collision avoidance between agents and obstacles, while other objectives are considered, with the path planner still responsible for ensuring them.

### 3.1.1 Objectives

The global objectives given in section 1.1 can be rewritten as follows for the area division [6]:

1. No subarea overlaps another, i.e. one cell can be assigned to only one subarea: No path overlapping or repetition.
2. The union of all subareas is equal to the entire area to cover: Complete coverage.
3. The size of all subareas is as equal as possible: Fair work division.
4. All cells of any subarea are connected: Collision avoidance.
5. The initial pose of a robot is included in its subarea: Collision avoidance.

To solve the area division, the assumption is made that all robots’ initial positions are on distinct grid cells and are not located on obstacle cells.

### 3.1.2 Iterative Cell Assignment

The DARP algorithm is described in [6] and is summarized here. DARP assigns the cells to the subareas in an iterative manner until the objectives are reached. Initially the cells are assigned to the subareas based on their euclidean distance to each initial robot’s position.

Then, at each iteration and for subarea (robot) $i$, the **Evaluation Matrices** $E_i$ are computed, containing for each cell their distance to the robot's initial pose [6]:

$$E_{i|x,y} = ||\mathcal{X}_i(t_0) - [x, y]|| \tag{3.1}$$

for each cell $x, y$ for each subarea $i$, with $\mathcal{X}_i(t_0)$ the initial position of the robot $i$.

These matrices are then multiplied by **Correction Factors** $m_i$ to obtain $E_i = m_i E_i$.

**Cost functions** $J_i$ are defined describing the current fairness of the area division for each subarea [6]:

$$J_i = \frac{1}{2}(k_i - f)^2 \tag{3.2}$$

With $k_i$ the number of cells assigned to subarea $i$ and $f$ the optimal number of cells for each subarea, which is the number of cells to cover divided by the number of robots $n_r$.

The correction factor $m_i$ is updated at each iteration performing cyclic Coordinate Descent [12] on the cost functions $J_i$ [6]:

$$m_i = m_i - \eta \frac{\partial J_i}{\partial m_i} = m_i - 2\eta(k_i - f)\frac{\partial k_i}{\partial m_i}, \quad \eta > 0 \tag{3.3}$$

According to [6], $\frac{\partial k_i}{\partial m_i}$ being almost identical for all subareas $i$, it is omitted to obtain following update for the correction factor [6]:

$$m_i = m_i + c(k_i - f) \tag{3.4}$$

with $c > 0$ a tunable constant, typically set to 0.1 in [13].

The connection of each subarea is added to the Evaluation Matrix with the introduction of **Connectivity Matrices** $C_i$. For each subarea, this matrix can be defined as [6]:

$$C_{i|x,y} = min(|||[x, y] - r||) - min(|||[x, y] - q||), \quad \forall r \in \mathcal{R}_i, \forall q \in \mathcal{Q}_i \tag{3.5}$$

for each cell $x, y$ for each subarea $i$, with $\mathcal{R}_i$ the set of cells assigned to subarea $i$ and connected to the robot $i$'s initial pose and $\mathcal{Q}_i$ the set of other cells also assigned to subarea $i$ but not connected to the robot $i$'s initial pose.

$C_{i|x,y}$ rewards cells close to $\mathcal{R}_i$, while penalizing cells close to $\mathcal{Q}_i$. This allows the penalization of assigned regions, which are not connected to the initial robot position.

In practice, $C_{i|x,y} = 1$ for each cells $x, y$ if $\mathcal{Q}_i$ is empty [13].

The Evaluation Matrices are finally updated using following element-wise matrix multiplication [6]:

$$E_i = C_i \odot (m_i E_i) \tag{3.6}$$

Finally, the cells are assigned using an **Assignment Matrix** $A$ which is defined as [6]:

$$A_{x,y} = \underset{i \in \{1,\dots,n_r\}}{\operatorname{argmin}} E_{i|x,y} \tag{3.7}$$

for each cell $x, y$ for each subarea $i$.

This process is iteratively repeated until the ending condition is reached. The Pseudocode of the A* DARP with BFS Connectivity algorithm is given in the algorithm 1.

### 3.1.3 Ending Conditions

Ideally, the algorithm would converge once it finds a solution that satisfies all objectives. The third condition for fair work division is redefined as follows: all subareas should contain an equal number of cells, if the total number of cells is divisible by the number of robots (see Equation 3.8). Otherwise, if the division does not result in an integer, a difference of one cell in the sizes of the subareas is allowed for convergence. However, this condition can be relaxed if the algorithm has not converged within a specified maximum number of iterations. In such cases, the allowed maximum cell difference between subareas is increased, and the algorithm continues until a solution is found where all subareas are connected and the relaxed fair work distribution condition is met.

$$f = \frac{\text{number of cells}}{\text{number of robots}} \tag{3.8}$$

## 3.2 Improvements

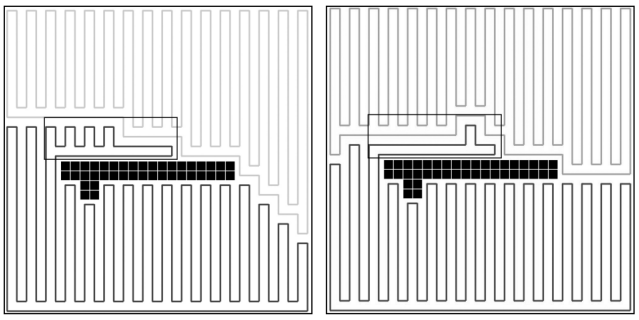
### 3.2.1 A* DARP

A* DARP was developped in [7] and [8]. The improvements of this approach is to replace the euclidean distance in the evaluation matrices $E_i$ by the A* path distance, for each cell to the robot's initial position, to take the obstacles into account. The Equation 3.1 becomes therefore [7]:

$$E_{i|x,y} = A^*(\mathcal{X}_i(t_0) - [x, y]) \tag{3.9}$$

In fact, euclidean distance does not take obstacles into account. If for example two cells have a given euclidean distance, but are separated by an obstacle, then the euclidean distance is not representative of the actual path the robot will do to go from one cell to the other, because it has to go around the obstacle. Both [7] and [8] conclude that A* DARP reduces the number of turns when combining it with STC path planning (see example in Figure 3.1). Nevertheless, the improvements of using A* in DARP requires testing if changing the path planning algorithm.

This approach introduces oscillatory behavior during convergence due to conflicting terms in the Evaluation and Connectivity matrices. Specifically, while the Evaluation matrix accounts for obstacles by incorporating A* path distances, the Connectivity matrix still relies on Euclidean distance, leading to inconsistencies.



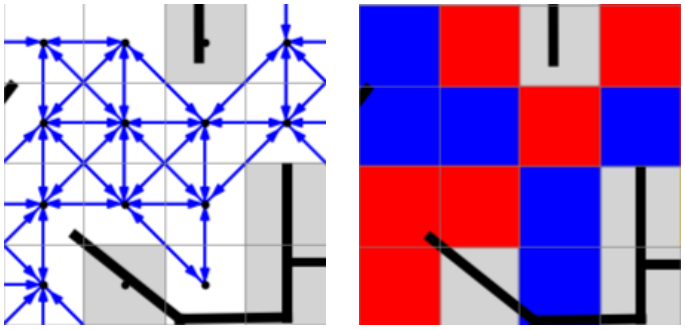
(a) path of DARP algorithm (b) path of A*-DARP algorithm

Figure 3.1: DARP and A* DARP comparison [7]

### 3.2.2 Movement Map

A* DARP not only allows to take into account obstacles in the distances, it also allows to account for the Movement Map described in section 2.3. In fact, for a given current cell and a neighbor cell during the A* search, instead of verifying if the neighbor cell is not an obstacle to consider allow the motion to the neighbor cell, the Movement Map can be consulted to verify if the motion between the current to the neighbor cell is allowed. This Movement Map is also used in the BFS introduced in the next section.

The Movement Map ensures complete obstacle avoidance, while DARP, by assigning each cell to only one robot, provides reliable collision avoidance between robots. Nevertheless, in a given square of four cells, if two diagonal cells are assigned to one subarea and the other two to another subarea (see Figure 3.2), a collision between two robots can occur. To prevent this specific scenario, two diagonal cells are considered not connected if they belong to the same subarea, while the other two cells in the $2 \times 2$ square are assigned to different subareas and are connected according to the Movement Map.



(a) Crossing: Movement Map (b) Crossing: Cell Assignment

Figure 3.2: Crossing Cell Assignment Example

With red cells belonging to a given subarea and blue cells belonging to another one.

### 3.2.3 BFS Connectivity

A* DARP incorporates obstacles into the distances to the initial positions within the evaluation matrices $E_i$, which are updated using correction factors $m_i$ to ensure fair area division. However, the Connectivity Matrices $C_i$ (see Equation 3.5) still rely on Euclidean distance to calculate the distance between unconnected and connected assigned cell sets. As a result, obstacles are not considered in the connectivity matrices, causing oscillatory behaviors during the convergence. Specifically, for a given subarea, two cells can have contradictory distance measures: one from the evaluation matrix (using A* with obstacle consideration) and another from the connectivity matrix (using Euclidean distance without obstacle consideration).

To resolve this issue, we introduce the BFS Connectivity Matrices, where the Euclidean distance is replaced by the shortest path, taking obstacles into account using Breadth-First Search (BFS). Consequently, Equation 3.5 becomes:

$$C_{i|x,y} = BFS([x,y],r) - BFS([x,y],q), \quad \forall r \in \mathcal{R}_i, \forall q \in \mathcal{Q}_i \tag{3.10}$$

To compute $C_{i|x,y}$ following steps are performed. First, $\mathcal{R}_i$ and $\mathcal{Q}_i$ are found using BFS and then $BFS([x,y],r)$ and $BFS([x,y],q)$ are computed using BFS as well.

To compute $\mathcal{R}_i$ and $\mathcal{Q}_i$ for each subarea $i$, BFS is performed for each cell assigned to the subarea $i$. The search is performed only among the cells assigned to the subarea $i$ and the Movement Map (see section 2.3) is consulted for each movement during the search. If a path to the robot $i$'s initial position, the cell is added to $\mathcal{R}_i$. If no path is found, then the assigned cell is added to $\mathcal{Q}_i$.

Once $\mathcal{R}_i$ and $\mathcal{Q}_i$ are computed for all subareas, the Connectivity Matrices $C_i$ are computed. If $\mathcal{Q}_i$ is empty, then $C_i$ is filled with ones. In fact, the assigned cells being all connected, no correction to the Evaluation Matrix with the Connectivity Matrix is needed. If $\mathcal{Q}_i$ is not empty, $C_i$ is computed as described in Equation 3.10. To compute $BFS([x,y],r)$, BFS is performed for each cell, until any cell belonging to $\mathcal{R}_i$ is reached. Once such a cell is found, $BFS([x,y],r)$ is given as the path length to reach it. $BFS([x,y],q)$ is computed in a similar way, with BFS until a cell belonging to $\mathcal{Q}_i$ is reached. In these BFSs, the Movement Map is also used, to take all the obstacles into account in the path length.

### 3.2.4 Implementation

The A* DARP with BFS Connectivity implementation in Python is based on the DARP implementation provided in [13]. This implementation uses OpenCV's `connectedComponents` [14] function to find $\mathcal{R}_i$ and $\mathcal{Q}_i$, and OpenCV's `distanceTransform` [15] function to compute $C_i$, as shown in Equation 3.5. In this approach, the map is treated as an image, with each cell represented as a pixel and obstacles and free space differentiated by color.

However, this approach cannot be used to compute $\mathcal{R}_i$, $\mathcal{Q}_i$, and $C_i$ as defined in Equation 3.10, since the Movement Map must be incorporated. As a result, the BFS function needs to be explicitly implemented. The nested loops within the BFS function result in significantly slower execution compared to the OpenCV-based implementation.

To reduce execution time, the BFS functions are compiled into machine code at the start of the Python script execution using the Numba library with the `@njit(fastmath=True)` option [16]. The execution times with and without this compilation are given below, using the environment described in chapter 2, for three robots with given initial positions.

| **Function** | **Execution Time without Numba** | **Execution Time with Numba** |
|---|---|---|
| BFS to compute $R_i$ and $Q_i$ | 2 ms | 30 $\mu$s |
| BFS to compute $C_i$ | 4.5 ms | 70 $\mu$s |

Table 3.1: Numba Compilation : Execution Time Comparison

This represents a performance improvement by a factor of approximately 65, which is significant, as the BFS function is called for each cell in the environment's grid during each iteration of DARP. Execution times were measured using Python's built-in `time` library.

### 3.2.5 Ending conditions

The implementation in [13] has a maximum number of iterations set to 80,000. The stopping condition is met when a solution is found where all subareas are connected, with size differences of zero cells if possible, or one cell otherwise (see subsection 3.1.3). If no solution is found after the maximum number of iterations, the allowed size difference between subareas is increased by one cell, and the maximum number of iterations is halved. The algorithm continues iteratively until a solution is found or the new maximum number of iterations is reached, progressively relaxing the subarea size difference and halving the maximum number of iterations at each step.

In our cluttered environment, it was observed that either a solution is found in fewer than 1,000 iterations or a solution with a fair cell distribution does not exist. By default, each subarea is initially assigned the robot's starting position cell. However, if a robot's starting cell is located in a narrow corridor with a width of one cell, other robots cannot traverse this corridor, as no cell can be assigned to multiple subareas. This leads to scenarios where some robots become trapped in small zones due to another robot's starting position, making it impossible to achieve an equal and connected area division.

To address this, the initial maximum number of iterations is set to 2,000. If no solution is found within this limit, the subarea size difference condition is relaxed by one cell, while the maximum number of iterations remains unchanged. This subarea size difference condition is relaxed until a maximum difference threshold *dcells_thresh* is reached. This threshold is given by

$$dcells\_thresh = dcells - (1 - initial\_max\_cell\_diff) \tag{3.11}$$

With *dcells* a tunable parameter set to 2 and *initial_max_cell_diff*, which is set to 0 if $f$ (see Equation 3.8) is integer and 1 if not. The result is shown in Table 3.2.

If still no solution is found once this threshold is reached, the subarea size difference condition is entirely removed, and the selected solution is the

one with the fairest work distribution among the connected solutions in a maximum number of iteration (2,000). The fairest solution is the one with the lowest Gini coefficient, see section 3.3.2.

Nevertheless, for some complex initial positions, where multiple robots are trapped in a small subarea by another robot's initial position, no solution is found. If this is the case after a total maximum number of iterations set to 6,000, the DARP algorithm is reinitialized with an initial maximum number of iterations reduced to 1,000, and the BFS Connectivity Matrices are modified to:

$$C_{i|x,y} = -BFS([x,y],q), \quad \forall q \in \mathcal{Q}_i \tag{3.12}$$

This keeps only the penalty term for cells being close to $\mathcal{Q}_i$ (assigned but unconnected regions). It enforces cells not to be assigned if they are disconnected from the robot's initial position, encouraging connected subareas at the expense of a fair work division. Experimentally, this approach has been found effective in finding solutions for complex scenarios where an optimal solution in terms of fair work division is not feasible.

A summary of the maximum number of iterations and their corresponding ending conditions is provided in Table 3.2. A solution is considered complete when both ending conditions (maximum cell difference across subareas and connectivity) are satisfied. If the maximum number of iterations is reached without fulfilling these conditions, the ending conditions are relaxed according to the next row in the table, with `dcells` equal to 2 and $f$ from Equation 3.8.

| **Max number of iterations** | **Max cell difference Across subareas** | | **Connectivity Matrix** |
|---|---|---|---|
| | $f$ integer | $f$ not integer | |
| 2000 | 0 | 1 | Equation 3.10 |
| 2000 | 1 | 2 | |
| 2000 | $\infty$ - Lowest Gini Coeff | | |
| 1000 | 0 | 1 | Equation 3.12 |
| 1000 | 1 | 2 | |
| 12000 | $\infty$ - Lowest Gini Coeff | | |

Table 3.2: Evolution of Ending Conditions and Maximum Number of Iterations

### 3.2.6 Pseudocode

The A* DARP with the BFS Connectivity extension is described in algorithm 1. The entire algorithm Pipeline is given in section 5.3.

**Algorithm 1:** A*-DARP with BFS Connectivity

**Data:** Robots initial positions, Movement Map, `dcells`, $c$ a tunable constant, *first DARP run* a boolean
**Result:** $A$ Assignment Matrix

————————————————Initialization————————————————

**for** *each robot* $i$ **do**
  **for** *all cell* $x$,$y$ **do**
    $E_{i|x,y} \leftarrow$ A* search distance from the robot's initial position to the cell $x, y$; `/* Equation 3.9 */`
$f \leftarrow$ (number of cells) / (number of robots); `/* Equation 3.8 */`
**if** $f$ *is an integer* **then**
  *max cell diff* $\leftarrow 0$; `/* Initial max cell diff */`
**else**
  *max cell diff* $\leftarrow 1$; `/* Initial max cell diff */`

**Algorithm 1:** A*-DARP with BFS Connectivity (continued)

```
-------------------------Optimization loop-------------------------
while not connected or (cell diff) > (max cell diff) do
    while iter < max nb of iter do
        A ← Assign each cell based on E_i s;                    /* Equation 3.7 */
        for each robot i do
            R_i, Q_i ← perform BFS among assigned cells, starting from
              robot's initial position;                         /* subsection 3.2.3 */
            for each cell x,y do
                dist1_{i|x,y} ← BFS for shortest distance to R_i;
                dist2_{i|x,y} ← BFS for shortest distance to Q_i;
                if first DARP run then                          /* subsection 3.2.5 */
                    C_{i|x,y} ← dist1_{i|x,y} − dist2_{i|x,y} ;  /* Equation 3.10 */
                else
                    C_{i|x,y} ← −dist2_{i|x,y} ;                /* Equation 3.12 */
        for each robot i do
            k_i ← Number of cells assigned to robot i;
            m_i ← m_i + c(k_i − f) ;                            /* Equation 3.4 */
            E_i ← C_i ⊙ (m_i E_i);                              /* Equation 3.6 */
        connected ← all(Q_i is empty for all robots);
        cell diff ← max(|k_i − f|);
        if max cell diff equal to ∞ then
            Compute Gini Coefficient and if it is the lowest yet and
              connected is True, store A in A_best as the best solution
              found yet;                                        /* Equation 3.13 */
        else if connected and (cell diff) ≤ (max cell diff) then
            return A;
        iter ← iter + 1;
    if max cell diff equal to ∞ then
        return A_best
    max cell diff ← max cell diff + 1;
    if max cell diff > dcells - (1-initial max cell diff) then
                                                                /* Equation 3.11 */
        max cell diff ← ∞;                                      /* Table 3.2 */
    update max nb of iter;                                      /* Table 3.2 */
return None;                                                    /* No solution found */
```

## 3.3 Metrics

### 3.3.1 Number of iterations for convergence

The number of iterations for convergence (nb iter) is an indicator of the complexity of the environment and the robots' initial position. In fact, if no area division with equally sized subarea is possible, a solution is possible only if the work division fairness condition is relaxed.

### 3.3.2 Work division fairness

#### Gini Coefficient

The Gini coefficient is a metric used to measure inequalities on a scale from 0 to 1, where a lower value indicates higher equality. Although it is primarily used to assess income disparities in a population, here it is applied to evaluate the inequality of the area division. [17, 18]

The Gini coefficient is defined as [17]:

$$G = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \bar{x}} \tag{3.13}$$

where $n$ is the number of subareas, $|x_i - x_j|$ is the difference in the number of cells assigned to subarea $i$ and subarea $j$, and $\bar{x}$ is the mean number of cells assigned to each subarea. The implemented Python function is directly derived from [19].

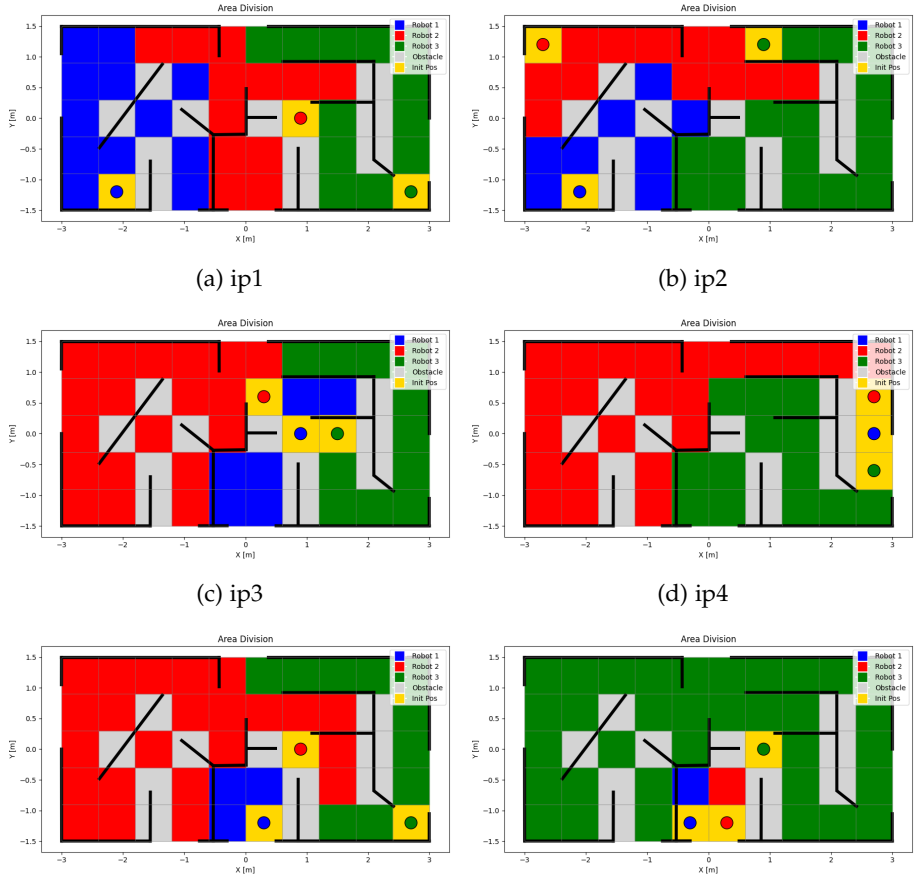
#### Standard deviation

The standard deviation of the number of cells assigned to each subarea is calculated alongside the Gini coefficient to evaluate the performance of DARP in the given environments. While the Gini coefficient measures inequality, the standard deviation highlights the variability in cell assignments, with lower values indicating a more balanced workload distribution among robots.

The standard deviation (std) and the average (avg) of cells assigned to each subarea are given in number of cells.

## 3.4 Results

### 3.4.1 Influence of the Initial Positions




(a) ip1 (b) ip2

(c) ip3 (d) ip4

(e) ip5 (f) ip6

Figure 3.3: Results: Influence of the initial positions

| Settings | Results | | | |
|---|---|---|---|---|
| Experiment | nb iter | avg | std | gini |
| ip1 | 70 | 13.00 | 0.00 | 0.0000 |
| ip2 | 6000 | 13.00 | 3.26 | 0.1368 |
| ip3 | 6000 | 13.00 | 5.35 | 0.2222 |
| ip4 | 6000 | 13.00 | 9.42 | 0.3932 |
| ip5 | 6000 | 13.00 | 8.29 | 0.3419 |
| ip6 | 9000 | 13.00 | 15.56 | 0.5641 |

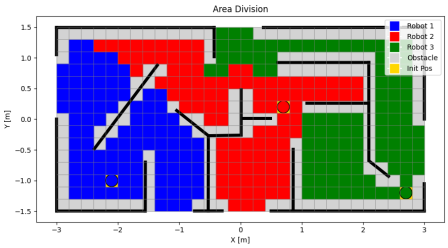
Table 3.3: Results: Influence of the initial positions

The experiments *ip1* present a scenario where a perfectly fair work division is achievable. Experiment *ip2* presents scenarios where a fair work division is not possible, even if no robot's initial position blocks another. Specifically, the central region of the map, where all three subareas are adjacent to each other, is a bottleneck, which is large enough for only one subarea to expand.

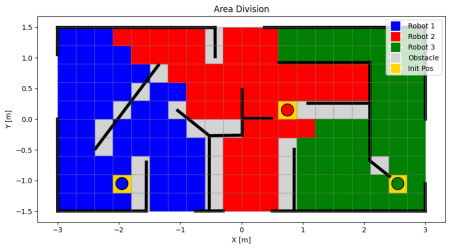
Experiments *ip3* to *ip6* present typical scenarios where fair work distribution is not possible due to a blocking robot's initial position. However, the solutions found by the algorithm are nearly optimal, as the best solution within the given maximum number of iterations is selected and with the condition that each cell can be assigned only to one robot.

### 3.4.2 Influence of the Cell Size

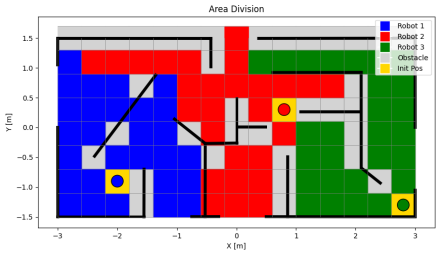
With fixed number of robots and initial positions, the cell size is modified between 0.2 and 0.6. The robot's initial positions are well distributed over the area, allowing fair solutions, as no initial position is blocking any other robot. The results for the cell size 0.6 can be found in experiment *ip1*.



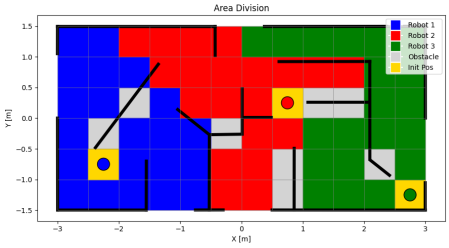

(a) cs1: grid size 0.2




(b) cs2: grid size 0.3




(c) cs3: grid size 0.4




(d) cs4: grid size 0.5

Figure 3.4: Results: Influence of the cell size

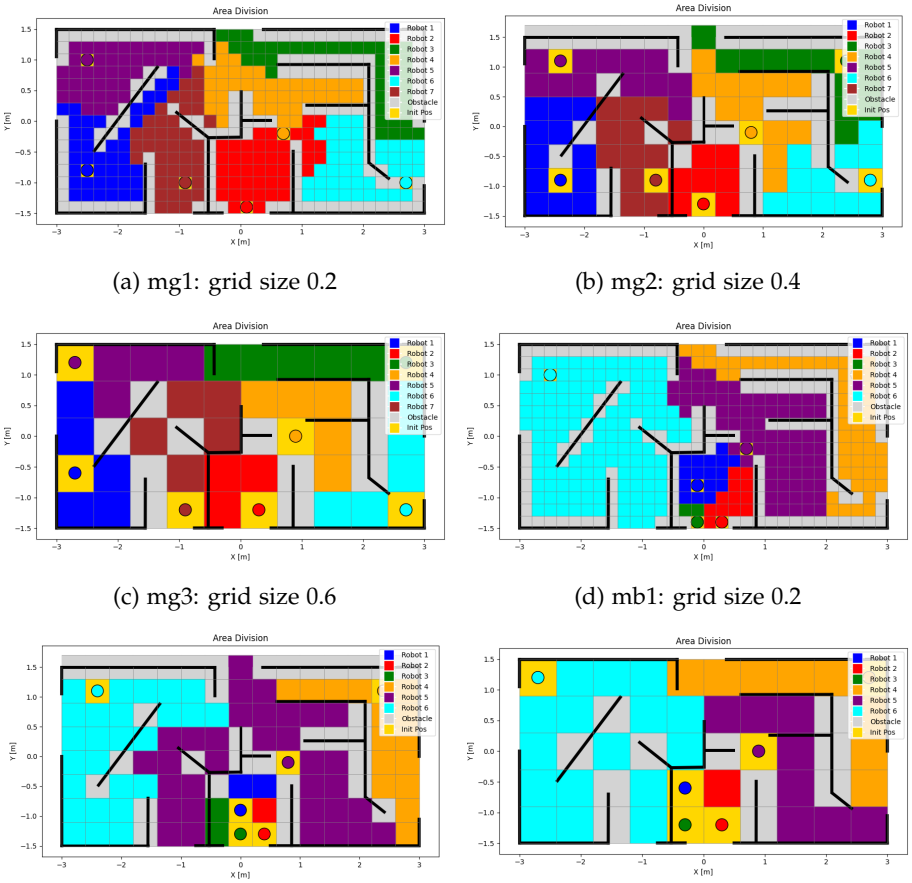
| **Settings** | **Results** | | | |
|---|---|---|---|---|
| **Experiment** | **nb iter** | **avg** | **std** | **gini** |
| cs1 | 215 | 192.0 | 0.00 | 0.0000 |
| cs2 | 83 | 57.67 | 0.47 | 0.0039 |
| cs3 | 134 | 28.33 | 0.47 | 0.0078 |
| cs4 | 78 | 21.33 | 0.47 | 0.0104 |

Table 3.4: Results: Influence of the cell size

These experiments show that the algorithm can scale to larger grids. A perfectly fair work division is possible in experiment *cs1*, while it is not possible in experiments *cs2* to *cs4*, because an even number of cells to cover has to be distributed to an odd number of robots.

### 3.4.3 Influence of the Number of Robots

The number of robots is increased to seven with well distributed initial positions and to six with complex initial positions, for different cell sizes.




(a) mg1: grid size 0.2

(b) mg2: grid size 0.4

(c) mg3: grid size 0.6

(d) mb1: grid size 0.2

(e) mb2: grid size 0.4

(f) mb3: grid size 0.6

Figure 3.5: Results: Influence of the number of robots

| Settings | Results | | | |
|---|---|---|---|---|
| **Experiment** | **nb iter** | **avg** | **std** | **gini** |
| mg1 | 575 | 43.71 | 0.45 | 0.0047 |
| mg2 | 7000 | 12.14 | 1.7 | 0.0571 |
| mg3 | 2000 | 5.571 | 0.73 | 0.0586 |
| mb1 | 18000 | 51.00 | 46 | 0.4913 |
| mb2 | 18000 | 14.17 | 13 | 0.4804 |
| mb3 | 18000 | 6.500 | 5.8 | 0.4744 |

Table 3.5: Results: Influence of the number of robots

These experiments show that the algorithm can scale to multiple robots, even in a small and cluttered environment. In the experiments *mb1* to *mb3* the robot *5*'s initial position blocks the robots *1*, *2* and *3*. A fair solution is therefore not possible and the solution found is the best one during the $12,000$ iterations described in Table 3.2.

# Chapter 4

# Coverage Path Planning

The objective of the Coverage Path Planning (CPP) is to determine the order in which each robot visits the cells of its subarea. The CPP should be computed with the objectives described in section 1.1, focusing on complete coverage, no path overlapping, and simple motion.

## 4.1 Spanning Tree Coverage

The original DARP algorithm in [6], as well as the improvements in [7, 8, 20], utilize a STC-based path planning approach after the area division. As described in section 1.2, STC ensures complete coverage without path overlapping. To minimize the number of turns, the implementation in [13] generates four spanning trees for each subarea, each following a heuristic that encourages the tree to span upwards, leftwards, downwards, or rightwards. For each subarea, the spanning tree that results in the fewest turns is selected.

The necessity to divide each cell in four subcells to perform the path planning is not neglectable in cluttered environment. In fact, corridor or passage tighter than two times the desired coverage cell size between to obstacles can become not reachable. These passages would be deemed as obstacles when building the spanning tree, before the cell subdivision. This leads to incomplete coverage, which violates the first objective.

The DARP approach followed by STC extension in [7] uses UF-STC to address that challenge. The idea is to cover subcells which are contained in cells which were marked as obstacles initially but are free once the grid is subdivided. To cover these subcells, a back and forth motion is performed in addition to the initial STC. This allows for a complete coverage. The Figure 4.1 shows an example of UF-STC. In fact, Figure 4.1a shows the coverage performed by STC in this environment. The obstacle is discretized using the initial cells, while the path planning is performed on the subcells. Some cells are therefore uncovered using standard STC. The UF-STC is able to perform complete coverage as seen on Figure 4.1b.

Nevertheless, this solution introduces path overlapping and complex turns. In cluttered environment, tight corridor or passages with a width smaller than the initial cell size would be considered as obstacles. They could be covered using the UF extension to the STC but the resulting path would perform

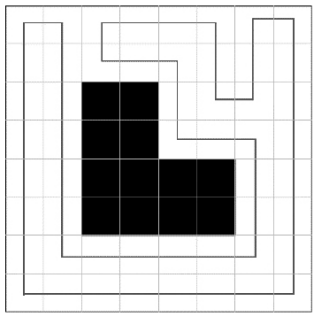


Fig. 3. Example of a path generated by STC

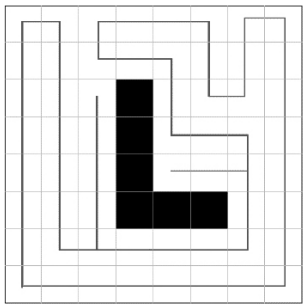


Fig. 4. Example of a path generated by UF-STC

(a) STC path example

(b) UF-STC path example

Figure 4.1: Comparison of STC and UF-STC path examples [7]

poorly, if the environment is mainly made of such narrow passages.

## 4.2 $\epsilon$* - based coverage path planning

To overcome the subdivision required by STC, an alternative approach is explored. [2] references multiple methods that do not require cell subdivision, such as learning-based techniques (e.g., Reinforcement Learning), optimization-based methods (e.g., Ant Colony Optimization or Particle Swarm Optimization), and Greedy Graph Search approaches. For simplicity and to avoid high computational costs, a Greedy Graph Search approach will be investigated.

The explored algorithm is an $\epsilon$*-based coverage path planning, which builds upon the implementation developed in [21], itself based on the works presented in [22–25].

### 4.2.1 Greedy Coverage with Dead-End Escape

The $\epsilon$*-based CPP implementation developed by [21] computes a path for a single robot. This is achieved in a greedy manner by locally determining the best next uncovered neighbor cell to visit based on a cost function (see Equation 4.1). If no neighbor cell remains uncovered (dead-end), an A* search is performed to find the closest uncovered cell, which is then set as the next to visit. The CPP process is complete once all cells have been visited at least once, achieving full coverage.

The cost function for the coverage is the sum of an action cost with an arbitrary heuristic:

$$\text{cost} = \text{cost}_{\text{action}} + \text{heuristic} \tag{4.1}$$

This reduces the number of turns while encouraging a specific coverage direction, resulting in an efficient coverage path guided by the heuristic. The action cost is given by:

$$\text{cost}_{\text{action}} = \frac{\Delta\theta}{\pi/2} \tag{4.2}$$

With $\Delta\theta$ the angle difference between the current orientation and the angle of the next movement.

To achieve the best solution using this approach, the CPP is executed multiple times with different heuristics, as the optimal heuristic depends on the environment and the initial robots position. In [21], four heuristics are implemented (Vertical, Horizontal, Manhattan, and Chebyshev) yielding a total of four solutions for each robot. The best solution is selected based on the lowest path overlapping and minimal action cost.

### 4.2.2 Extensions

The implementation in [21] is extended to better suit cluttered environments. As in the DARP implementation, the Movement Map is incorporated into the coverage and A* closest uncovered searches to account for obstacles.

Diagonal movements are added to the path planner, enabling more efficient navigation in cluttered environments. Consequently, the action cost is extended to include both turn and distance costs, as diagonal motions are longer by a factor of $\sqrt{2}$ compared to horizontal and vertical motions. The action cost is given by:

$$\text{cost}_{\text{action}} = \frac{\text{distance}}{\text{cell size}} + \frac{\Delta\theta}{\pi/4} \tag{4.3}$$

Figure 4.2 illustrates the cost for all eight possible movements given a current orientation. This cost is independent of the cell size and could be adjusted to prioritize penalizing distance over turns, or vice versa.

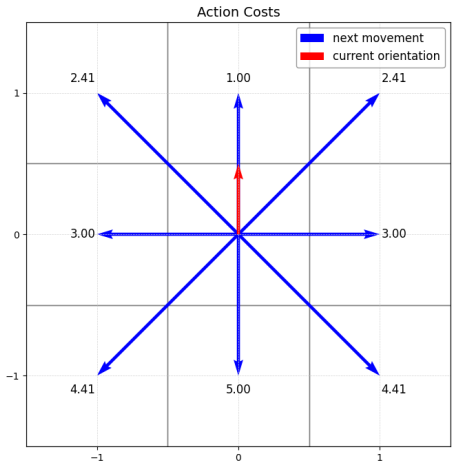



Figure 4.2: Action cost

Six additional heuristics are introduced alongside the four heuristics implemented in [21] (see Figure 4.3). The Euclidean and Diagonal heuristics

leverage diagonal movements, while the Null heuristic removes directional influence by making the total cost solely dependent on the action cost.

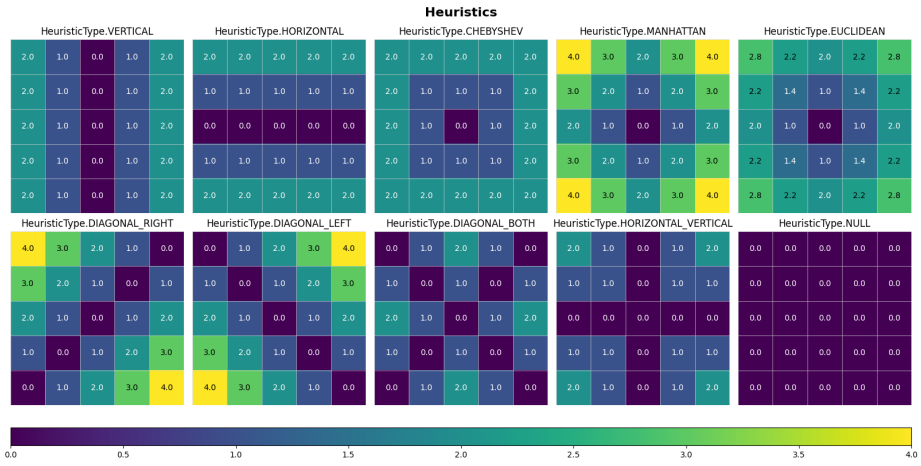



Figure 4.3: Heuristics

As described in subsection 4.2.1, the greedy coverage search selects the next cell to visit based on the lowest cost, which is the weighted sum of the action cost (see Equation 4.3) and an arbitrary heuristic (see Figure 4.3):

$$\text{cost} = w \cdot \text{cost}_{\text{action}} + \text{heuristic} \tag{4.4}$$

With $w$ an  to avoid biases arising from the scale differences between action cost and heuristic. The path planning is computed using the following weights $w$: 0.10, 0.25, 0.50, 0.75, 1.0, and 1.5, which were chosen experimentally. The heuristic has a higher impact with a lower action cost weight, while it has a lower impact with a higher action cost weight. This weight has no impact with the Null heuristic, since it assigns to each cell an heuristic of 0.

With ten heuristics and six action cost weights (only one for the Null heuristic), the number of CPP computations increases from 4 to 55, significantly improving the chances of finding a path with minimal overlap and lowest cost.

### 4.2.3 Explored Extensions

A path is planned for each robot independently, and the length of the path is not guaranteed to be similar from one robot to another. In fact, one robot's path could have significantly more path overlapping than another's, which reduces the fairness of the work division. To address this, one could divide the area again using DARP, but with weighted areas to allocate more space to robots with less path overlapping. However, experimental results did not show an improvement in overall performance. In fact, the best heuristic for path planning is not necessarily the same before and after the second area division. This leads to significantly different paths, which does not necessarily result in a fairer work division. As a result, it was not implemented in the final algorithm.

Another possible extension is to use different heuristics on different regions (group of cells) of the area to cover. In fact, paths planned with different heuristics have path overlapping in different spatial regions. One could argue that for each group of spatially close cells, the heuristic that gave the best results in that region should be used. Nevertheless, we did some experiments that showed that this method did not improve the overall performance. As a result, it was not implemented neither in the final algorithm.

## 4.3 Metrics

Overlapping is measured as the percentage of waypoints that overlap with others, relative to the total number of waypoints assigned to each robot.

The fairness of the work division is addressed and assessed during the area division process. However, the fairness of path lengths or complexity is not directly assessed, as the path planning is independent for each robot.

The simplicity of the motion is assessed with the action cost (see Equation 4.3. Nevertheless, it will only be used to compare two solutions in case both have the same overlapping.

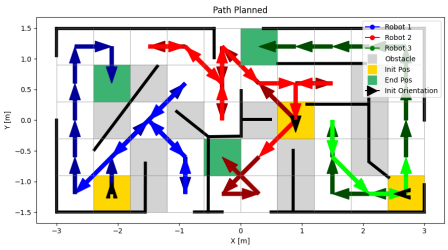
## 4.4 Results

The results presented are based on the area division results presented in section 3.4.

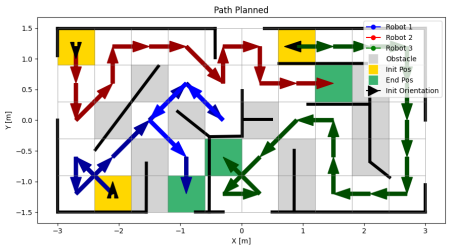
Heuristics or action cost weights in parentheses indicate cases where they have no impact on the planned path. This may occur either because the heuristic is the null heuristic, where the action cost weight is irrelevant, or because the planned path is straightforward, with each cell to be covered having only one neighboring cell available during the path planning (see for example robots 1 and 2 in experiment *ip6*).

All heuristics and action cost weights have been used to find the best solution in at least one experiment, showing their relevance in the path planning.

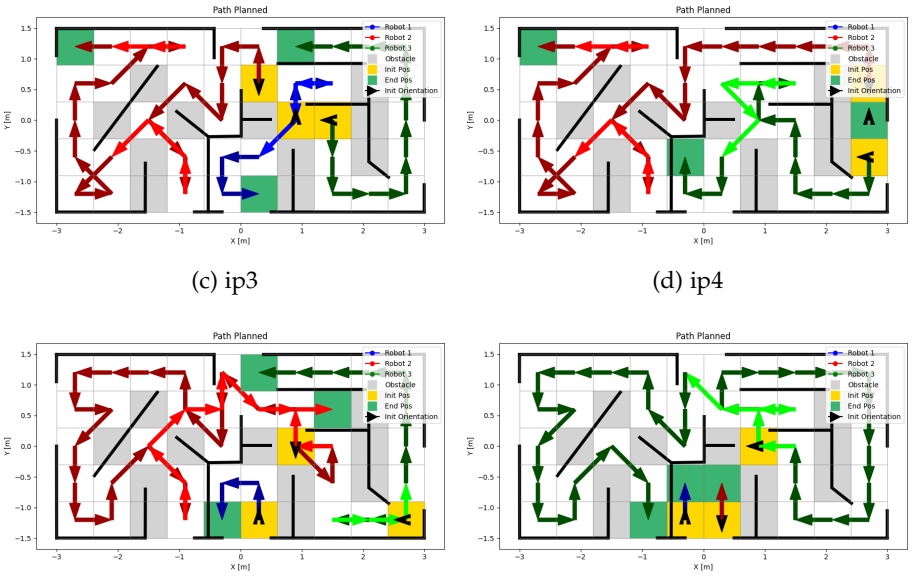
### 4.4.1 Influence of the Initial Positions




(a) ip1




(b) ip2




(c) ip3 (d) ip4

(e) ip5 (f) ip6

Figure 4.4: Results: Influence of the initial positions

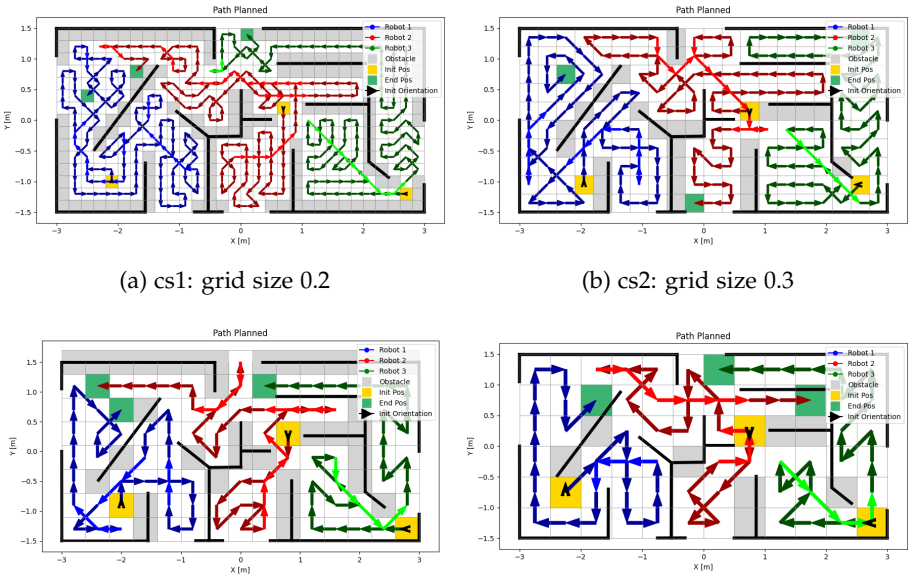
| Settings | Solution | | Results |
|---|---|---|---|
| **Experiment** | **Heuristic** | **Cost weight** | **Overlapping %** |
| ip1 | Horizontal | 0.25 | 25.0 |
| | Diagonal Right | 1.0 | 36.8 |
| | Vertical | 0.25 | 20.0 |
| ip2 | Diagonal Both | 0.1 | 20.0 |
| | Horizontal | 0.1 | 0.0 |
| | Null | (0.1) | 0.0 |
| ip3 | Vertical | 0.25 | 25.0 |
| | Diagonal Left | 0.75 | 13.6 |
| | Vertical | 0.25 | 0.0 |
| ip4 | - | - | - |
| | Diagonal Left | 0.75 | 11.5 |
| | Horizontal | 0.25 | 13.3 |
| ip5 | Vertical | 0.75 | 0.0 |
| | Diagonal Right | 0.75 | 25.8 |
| | Vertical | 0.25 | 16.7 |
| ip6 | (Vertical) | (0.1) | 0.0 |
| | (Vertical) | (0.1) | 0.0 |
| | Manhattan | 0.75 | 10.5 |

Table 4.1: Results: Influence of the initial positions

The planned paths exhibit significant overlapping, often caused by dead-ends resulting from the environment or the area division process. While DARP en-

sures coverage completeness, work division fairness, and collision avoidance, it does not address the minimization of path overlapping. As a result, the work is fairly distributed, where possible, in terms of the number of cells to visit but not in terms of the number of waypoints in the path or the action cost.

### 4.4.2 Influence of the Cell Size




(a) cs1: grid size 0.2

(b) cs2: grid size 0.3

(c) cs3: grid size 0.4

(d) cs4: grid size 0.5

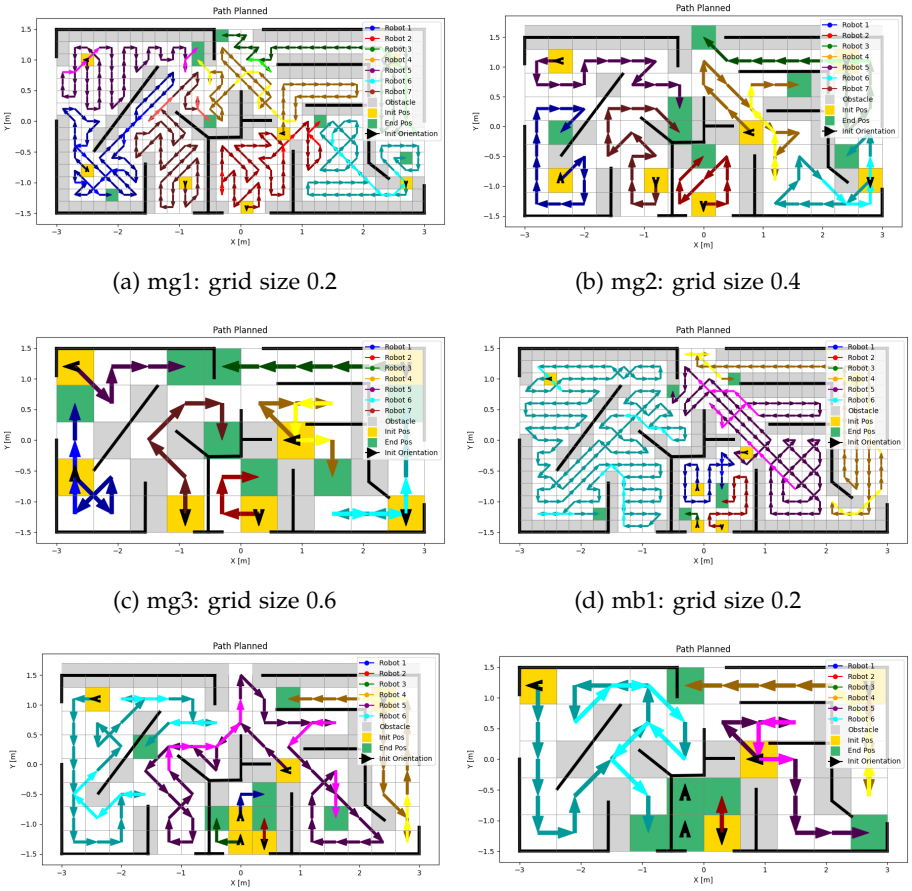
Figure 4.5: Results: Influence of the cell size

| Settings | Solution | | Results |
|---|---|---|---|
| Experiment | Heuristic | Cost weight | Overlapping % |
| cs1 | Diagonal Both | 0.75 | 9.8 |
| | Horizontal Vertical | 0.5 | 14.4 |
| | Manhattan | 0.75 | 7.3 |
| cs2 | Null | (0.1) | 11.1 |
| | Vertical | 0.25 | 10.9 |
| | Vertical | 0.5 | 5.0 |
| cs3 | Chebyshev | 0.75 | 9.7 |
| | Diagonal Left | 0.5 | 20.6 |
| | Vertical | 0.5 | 10.0 |
| cs4 | Diagonal Both | 0.75 | 8.7 |
| | Null | (0.1) | 20.0 |
| | Manhattan | 0.1 | 13.0 |

Table 4.2: Results: Influence of the cell size

With a smaller cell size, obstacle-free regions contain more cells, enabling the generation of sweeping paths aligned with an arbitrary heuristic (see Figure 4.5 and Figure 4.8).

### 4.4.3 Influence of the Number of Robots




(a) mg1: grid size 0.2

(b) mg2: grid size 0.4

(c) mg3: grid size 0.6

(d) mb1: grid size 0.2

(e) mb2: grid size 0.4

(f) mb3: grid size 0.6

Figure 4.6: Results: Influence of the number of robots

The area division for multiple robots in this environment results in complex-shaped subareas, making it challenging for any single heuristic to perform optimally, as observed for robot 4 in experiment *mb1*. In this subarea, the Diagonal Left heuristic, combined with an action cost weight of 1.5, produces the best solution. This approach effectively generates long diagonal paths, reducing the number of turns. However, certain unnatural "X"-shaped paths are created, increasing the turn count, whereas a "U"-shaped path would have yielded a more efficient solution. This is due to the high action cost weight, which minimizes the number of turns. Nevertheless, as the algorithm is greedy, the "X"-shaped paths is generated to avoid a turn in the first place, instead of having a longer term cost minimization.

| Settings | Solution | | Results |
|---|---|---|---|
| **Experiment** | **Heuristic** | **Cost weight** | **Overlapping %** |
| mg1 | Diagonal Both | 0.5 | 10.6 |
| | Chebyshev | 0.5 | 6.5 |
| | Diagonal Left | 0.5 | 14.0 |
| | Diagonal Left | 1.5 | 8.7 |
| | Horizontal | 0.1 | 6.5 |
| | Vertical | 1.0 | 14.0 |
| | Chebyshev | 0.5 | 4.4 |
| mg2 | Diagonal Left | 1.0 | 0.0 |
| | Vertical | 0.5 | 0.0 |
| | Vertical | 0.75 | 7.7 |
| | Vertical | 1.0 | 20.0 |
| | Chebyshev | 0.5 | 0.0 |
| | Diagonal Both | 0.5 | 15.4 |
| | Manhattan | 0.75 | 0.0 |
| mg3 | Diagonal Right | 0.25 | 16.7 |
| | (Vertical) | (0.1) | 0.0 |
| | Vertical | 0.1 | 0.0 |
| | Vertical | 0.5 | 28.6 |
| | Horizontal | 0.25 | 0.0 |
| | Vertical | 0.25 | 28.6 |
| | Horizontal | 0.1 | 0.0 |
| mb1 | Diagonal Both | 1.0 | 5.9 |
| | Vertical | 0.75 | 0.0 |
| | (Vertical) | (0.1) | 0.0 |
| | Diagonal Left | 1.0 | 17.1 |
| | Euclidean | 1.0 | 9.4 |
| | Vertical | 0.25 | 7.5 |
| mb2 | Vertical | 0.1 | 0.0 |
| | Vertical | 0.1 | 0.0 |
| | Vertical | 0.25 | 0.0 |
| | Horizontal | 0.25 | 11.8 |
| | Vertical | 1.5 | 15.4 |
| | Diagonal Right | 1.0 | 13.3 |
| mb3 | - | - | - |
| | (Vertical) | (0.1) | 0.0 |
| | - | - | - |
| | Vertical | 1.0 | 20.0 |
| | Vertical | 0.5 | 20.0 |
| | Vertical | 1.0 | 20.0 |

Table 4.3: Results: Influence of the number of robots

## 4.5 Discussion

The impact of the heuristic and action cost weight on the greedy search-based coverage path planning is analyzed using a single robot in an environment with a cell size of 0.2 m.

In this scenario, the optimal solution is obtained with the Vertical heuristic and the action cost weights 0.1 and 0.25 which gave the same result. The solution found has an overlapping of 11.85% and an action cost of 679.36.

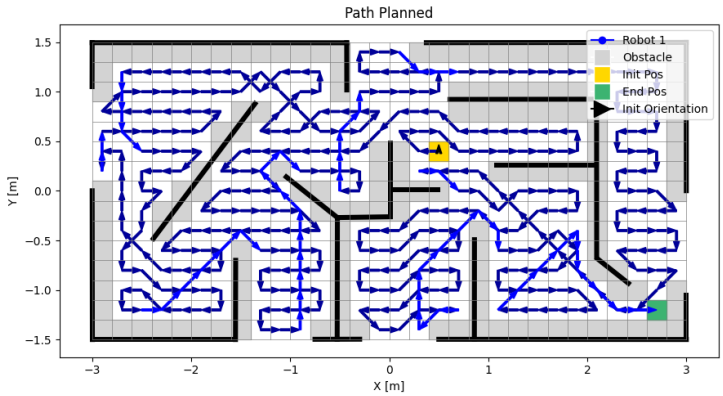
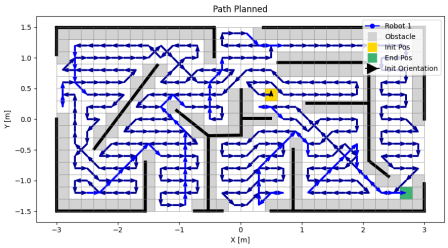


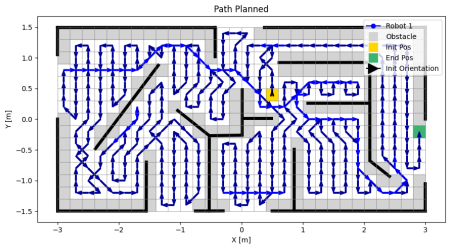

Figure 4.7: Discussion scenario

To observe the effects of each hyperparameter, the others will be fixed to the value found as the best one in the current scenario (see Figure 4.7).

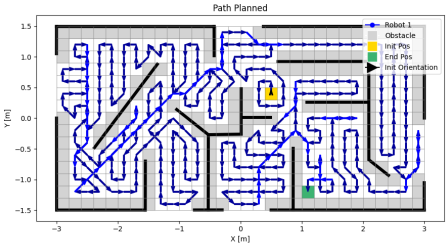
### 4.5.1 Influence of the Heuristic



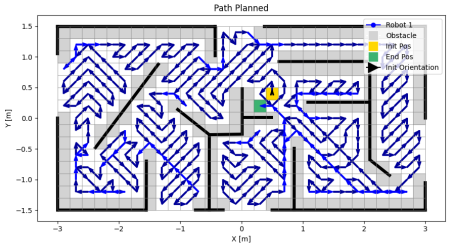

(a) Heuristic: Vertical



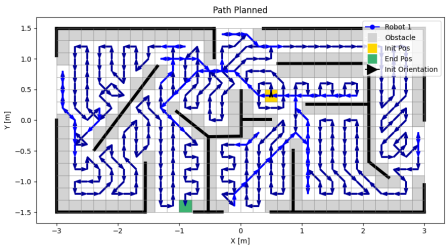

(b) Heuristic: Horizontal



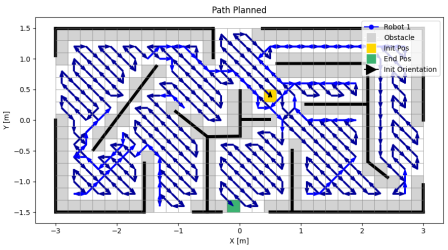

(c) Heuristic: Chebyshev



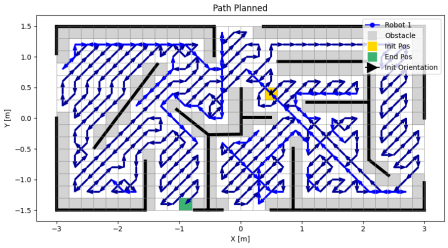

(d) Heuristic: Manhattan



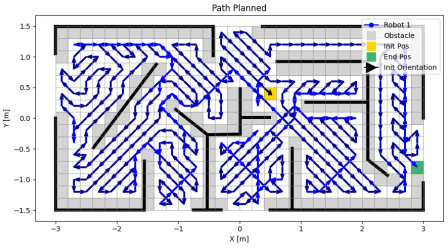

(e) Heuristic: Euclidean



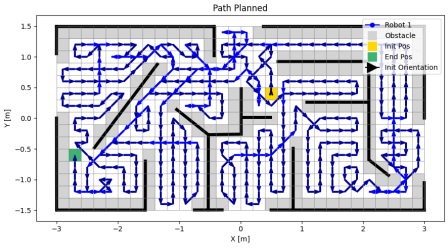

(f) Heuristic: Diagonal Right



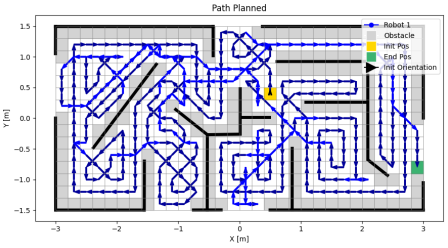

(g) Heuristic: Diagonal Left




(h) Heuristic: Diagonal Both




(i) Heuristic: Horizontal Vertical




(j) Heuristic: Null

Figure 4.8: Results: Influence of the Heuristic

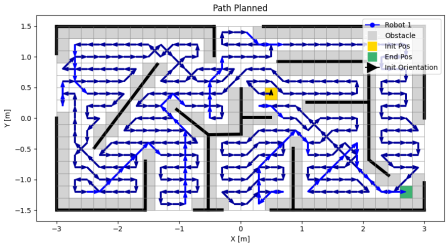
| **Settings** | | **Results** | |
|---|---|---|---|
| **Heuristic** | **Action Cost Weight** | **Overlapping %** | **Action Cost** |
| Vertical | 0.1 | 11.85 | 679.36 |
| Horizontal | 0.1 | 12.86 | 669.36 |
| Chebyshev | 0.1 | 12.61 | 693.80 |
| Manhattan | 0.1 | 12.61 | 861.52 |
| Euclidean | 0.1 | 15.04 | 775.28 |
| Diagonal Right | 0.1 | 16.21 | 803.83 |
| Diagonal Left | 0.1 | 17.57 | 844.75 |
| Diagonal Both | 0.1 | 18.88 | 844.29 |
| Horizontal Vertical | 0.1 | 15.75 | 675.52 |
| Null | (0.1) | 19.95 | 654.62 |

Table 4.4: Discussion: Influence of the Heuristic

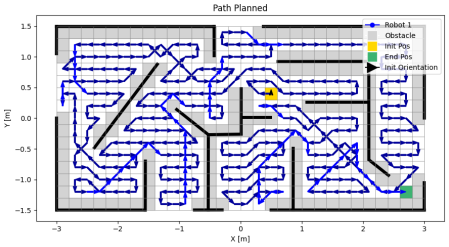
The influence of the heuristic can be observed in Figure 4.8 as the path tries to follow the direction of the heuristic as much as possible.

The Null heuristic, while giving the highest overlapping percentage, gives the lowest action cost, as it tries to minimize it at each waypoint during the planning. Nevertheless, this path planned with the Null heuristic is not kept as the best one in this scenario, as the best solution is selected by lowest overlapping.

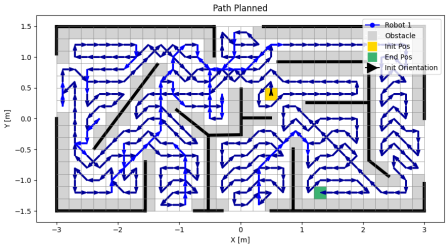
### 4.5.2 Influence of the Action Cost Weight



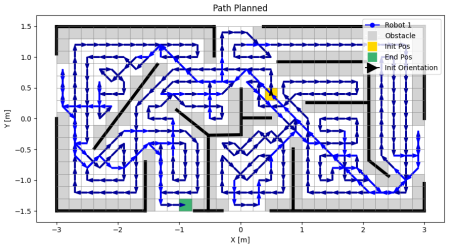

(a) Action Cost Weight: 0.10



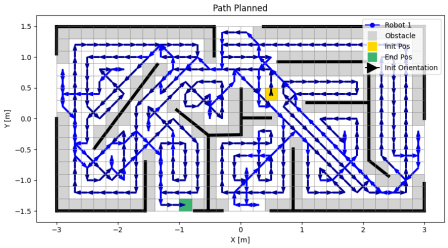

(b) Action Cost Weight: 0.25



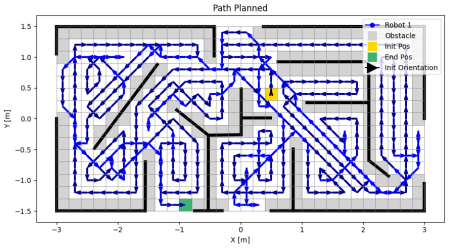

(c) Action Cost Weight: 0.50




(d) Action Cost Weight: 0.75




(e) Action Cost Weight: 1.0




(f) Action Cost Weight: 1.5

Figure 4.9: Results: Influence of the Action Cost Weight

| Settings | | Results | |
|---|---|---|---|
| **Heuristic** | **Action Cost Weight** | **Overlapping %** | **Action Cost** |
| Vertical | 0.10 | 11.85 | 679.36 |
| Vertical | 0.25 | 11.85 | 679.36 |
| Vertical | 0.50 | 12.36 | 781.96 |
| Vertical | 0.75 | 19.10 | 662.78 |
| Vertical | 1.0 | 21.99 | 688.88 |
| Vertical | 1.5 | 21.19 | 670.88 |

Table 4.5: Discussion: Influence of the Action Cost Weight

In this scenario, the action cost weights of 0.1 and 0.25 result in the same planned path. However, this outcome does not generalize to all initial conditions and heuristics.

Notably, as the action cost weight increases, the planned path increasingly prioritizes minimizing the action cost, which leads to minimizing turns.

# Chapter 5

# Robot Integration and Experimentation

## 5.1 Trajectory Tracking

The objective of the Trajectory Tracking for each robot is to follow a reference trajectory closely, which follows the waypoints planned in the CPP.

### 5.1.1 Reference Trajectory

The waypoints provided by the CPP include the position of the cells to visit in the correct order. The reference trajectory includes setpoints, which correspond to the target position and orientation of the robot at each timestep. The reference trajectory is computed to have equidistant setpoints distributed along a straight line between each waypoint. For each setpoint, the orientation is determined as the direction pointing towards the next setpoint. The timestep duration and distance between setpoints can be tuned to adapt the speed of the robots.

### 5.1.2 CCILQGames

An implementation of CCILQGames [26] (Chance-Constrained Iterative Linear - Quadratic Stochastic Games), was first extended to track the reference trajectory.

This algorithm iteratively solves a multi-robot trajectory planning problem while avoiding collisions under uncertainty. The system is linearized around a reference trajectory, and an optimal solution is computed with respect to quadratic costs.

The implementation described in [27] incorporates the following quadratic costs:

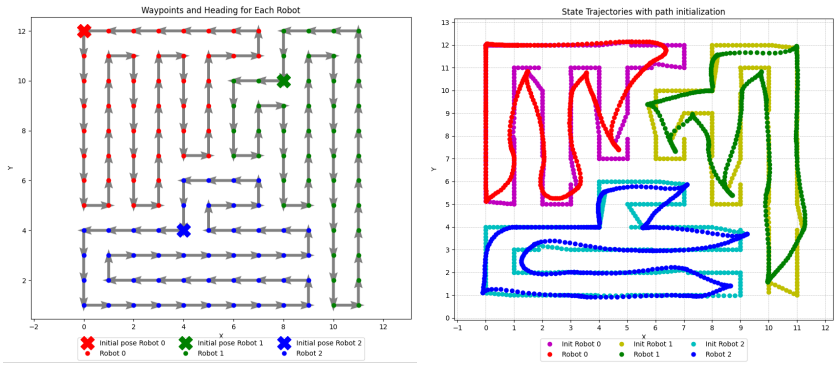
- Proximity Cost: Ensures collision avoidance with other agents. The chance constraint replaces hand-tuned weights to guarantee safety.
- Reference Cost: Rewards the robots to get close to their designated goals.

- Input Cost: Rewards smooth control inputs.
- Wall Cost: Keeps the robots within the bounds of a rectangular-shaped area.

Additionally, an Initial Trajectory Cost is introduced to these costs to reward robots to stay close to a given reference trajectory.

However, this extended implementation does not converge under all initial conditions. This is due to contradictory Initial Trajectory and Input Costs, stemming from the fact that the reference trajectory is not inherently smooth.

An example of a simulated trajectory is shown in Figure 5.1. In this scenario, multiple games are solved sequentially. Each game follows a reference trajectory comprising four waypoints provided by the CPP. The reference trajectory is computed at the beginning of each game to account for the robot's real position, rather than its desired position. The final simulated trajectory represents a balance between smooth control inputs and close tracking of the reference trajectory. However, the result exhibits some undesired behavior, such as sharp turns and significant deviations from the reference trajectory. This outcome is obtained after tuning the cost weights, which the trajectory tracking is highly sensitive to.




(a) CPP (b) Reference and tracked trajectories

Figure 5.1: CCILQGames simulated trajectory tracking

The CCILQGames approach primarily solves trajectory planning to prevent collisions between robots. Since the subareas assigned to robots do not overlap, their reference trajectories do not intersect, inherently preventing collisions. As a result, this extended implementation is not utilized in the final pipeline.

### 5.1.3 MPC

Instead of solving the CCILQGames, a Model Predictive Control (MPC) implementation directly on the robots is used to follow closely the reference trajectories. This allows a robust trajectory tracking.

## 5.2 Hardware Setup

Nvidia JetBots [9] are used to test the full coverage pipeline. An Opti'Track motion capture system [28] localizes the robots in real time. A ROS2 framework is used for communication between the motion capture system, the central computer, which computes the reference trajectory and the robots.

The coordinate system of the motion capture system may require a 90-degree rotation about the y-axis to align with the coordinate system depicted in Figure 2.1.

## 5.3 Algorithm Pipeline

The entire offline algorithm pipeline is described in algorithm 2.

**Algorithm 2:** Entire Pipeline

**Data:** Robots positions from Motion Capture System
**Result:** Reference Trajectory for each robot

——————————Initialization——————————
Initialize Robots initial positions with Motion Capture System;
Build Movement Map from environment; `/* section 2.3 */`

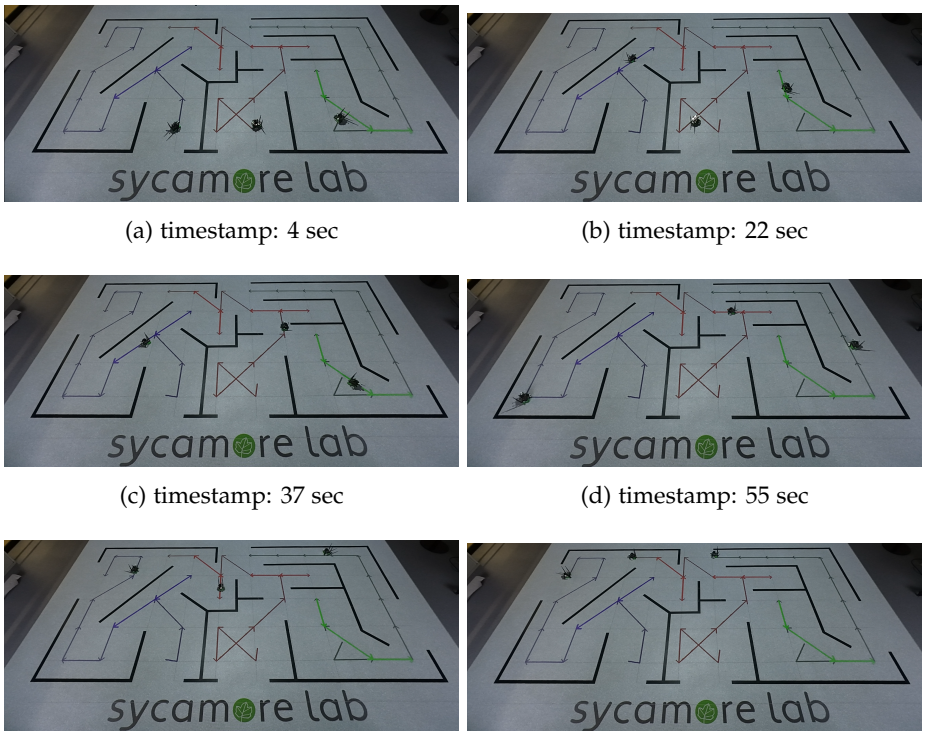
——————————Area Division——————————-
subareas ← A* DARP BFS Connect(*first DARP run* = **True**);
`/* algorithm 1 */`
**if** *subareas is* ***None*** **then**
  subareas ← A* DARP BFS Connect(*first DARP run* = **False**)

——————————-Coverage Path Planning——————————
**for** *Each subarea (robot) i* **do**
  **for** *Each Heuristic h* **do** `/* Figure 4.3 */`
    **for** *Each Action Cost Multiplier a* **do** `/* subsection 4.2.2 */`
      path_planned$_{i|h|a}$ ← Coverage Path Planning(h,a);
      `/* subsection 4.2.1 */`
  path_planned$_i$ ← shortest(path_planned$_{i|h|a}$);

——————————-Reference Trajectory——————————-
**for** *Each path planned* **do**
  trajectory$_i$ ← reference trajectory from path_planned$_i$;
**return** trajectories; `/* send to each corresponding robot */`

## 5.4 Results

The images above show three robots following the reference trajectory with their online MPC. The cell size is 0.6 m and the initial position are distributed along the area to have a fair area distribution.




(a) timestamp: 4 sec

(b) timestamp: 22 sec

(c) timestamp: 37 sec

(d) timestamp: 55 sec

(e) timestamp: 76 sec

(f) timestamp: 99 sec

Figure 5.2: Test on Nvidia Jetbots

The implementation on the Nvidia Jetbots shows the feasibility of the trajectory planning.

At low speeds (approx. 0.1 [m/s] in the example given in Figure 5.2), the controller can accurately follow the reference trajectory's sharp turns (see the blue robot in Figure 5.2d). However, at higher speeds, the robot's inertia may prevent the controller from handling sharp turns effectively. A possible solution is to smooth the trajectory, for instance, by using Bézier curves [29]. Nevertheless, many applications (see section 1.1 require low speed for the sensing (such as surveillance or rescue) or action (such as cleaning or demining) performed during the coverage.

# Chapter 6

# Conclusion

In this work, DARP and a greedy $\epsilon$*-based coverage path planner implementations where extended and assembled in a pipeline to perform path planning in a cluttered environment. A reference trajectory was computed from the path planning to test the coverage on Nvidia Jetbots. The experiments provided some insight into the potential of using DARP in a cluttered environment, as it is originally designed to pair with STC-based coverage.

## 6.1 Future Work

### 6.1.1 Area Division

Since DARP assigns each cell exclusively to one robot, the initial position of a robot can confine others to a small region, resulting in highly unequal area distributions. Future research could explore relaxing this constraint by extending DARP to allow blocked robots to escape, permitting cells to be assigned to multiple robots. To avoid collisions on these shared cells, different options could be considered:

- Once the path is planned, if two robots reach the shared cell at the same timestamp, one of the robot could be stopped and wait until the other robot leaves the shared cell. This solution is very light, but introduces stopping in the robots motions, which could be an undesired behavior in some applications.
- The selection of the best solution could be extended to select the path with the lowest overlapping rate and the lowest action cost and without any collisions. This would be limited, as no solution would be found if all solutions present a collision on a shared cell. A solution could also present a very high overlapping rate if it is the only one left without collision.
- CCILQGames could be used in the shared cells to avoid collisions. Nevertheless, this solution is heavy computationally and only works if the cell size is large enough for two robots to cross it simultaneously (at least twice the robot's radius). A cost to avoid obstacles should also be introduced.

The DARP algorithm does not guarantee a solution, even with relaxed work distribution fairness conditions and increased Connectivity Matrix penalties. However, no scenario was identified during experimentation where DARP failed to provide a solution, even if it was suboptimal.

### 6.1.2 Path Planning

Even when area division achieves equal distribution in terms of the number of cells, path planning can still introduce disparities due to overlapping, leading to unequal numbers of waypoints for each robot. To mitigate overlapping, metaheuristic approaches like Ant Colony Optimization (ACO) based coverage path planning [30] could be employed. While this approach optimizes paths within subareas, it comes at the cost of increased computational complexity, and entirely eliminating overlapping in subareas remains infeasible.

An alternative approach could involve integrating path planning directly into the DARP algorithm, recalculating paths at each iteration to ensure connected subareas. This would allow fairness to be assessed based on the number of waypoints rather than the number of assigned cells. However, this approach would require careful investigation, as it might affect convergence and significantly increase computational costs.

## 6.2 Acknowledgment

I want to thank Kai Ren for his precious advices and supervision along the entire project, redirecting the research direction when needed and encouraging each progress.

# Bibliography


[1] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

[2] Chee Sheng Tan, Rosmiwati Mohd-Mokhtar, and Mohd Rizal Arshad. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. *IEEE Access*, 9:119310–119342, 2021.

[3] Yaniv Gabriely and Elon Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. *Annals of Mathematics and Artificial Intelligence*, 31:77–98, 2001.

[4] Jingtao Tang, Chun Sun, and Xinyu Zhang. MSTC∗:multi-robot coverage path planning under physical constrain. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2518–2524, 2021.

[5] Junjie Lu, Bi Zeng, Jingtao Tang, Tin Lun Lam, and Junbin Wen. TMSTC∗: A path planning algorithm for minimizing turns in multi-robot coverage. *IEEE Robotics and Automation Letters*, 8(8):5275–5282, 2023.

[6] A. C. Kapoutsis, S. A. Chatzichristofis, and E. B. Kosmatopoulos. DARP: Divide areas algorithm for optimal multi-robot coverage path planning. *Journal of Intelligent and Robotic Systems*, 86:663–680, 2017.

[7] Yufan Huang, Man Li, and Tao Zhao. A multi-robot coverage path planning algorithm based on improved DARP algorithm, 2023. Available at `https://arxiv.org/abs/2304.09741`.

[8] Manish Kumar, Arindam Ghosh, and Muneendra Ojha. Multi-robot path planning for comprehensive area coverage in complex environments. In *2024 28th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 562–567, 2024.

[9] Waveshare. Jetbot ai kit b, ai robot based on jetson nano, comes with waveshare jetson nano dev kit. `https://www.waveshare.com/jetbot-ai-kit.htm`. Accessed: 2024-12-21.

[10] Tauã M. Cabreira, Lisane B. Brisolara, and Paulo R. Ferreira Jr. Survey on coverage path planning with unmanned aerial vehicles. *Drones*, 3(1), 2019.

[11] Shapely. *shapely.intersects - Shapely 2.0.6 documentation*, no date. Available at: `https://shapely.readthedocs.io/en/2.0.6/reference/shapely.intersects.html` (Accessed: 23 December 2024).

[12] Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[13] Alice-St. Alice-st/darp. `https://github.com/alice-st/DARP`, n.d. Accessed: 21 December 2024.

[14] OpenCV team. *Structural analysis and shape descriptors*, n.d. Available at: `https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html` (Accessed: 24 December 2024).

[15] OpenCV team. *Image segmentation with distance transform and watershed algorithm*, n.d. Available at: `https://docs.opencv.org/3.4/d2/dbd/tutorial_distance_transform.html` (Accessed: 24 December 2024).

[16] Anaconda, Inc. and others. *Compiling Python code with @jit*, n.d. Numba documentation. Available at: `https://numba.readthedocs.io/en/stable/user/jit.html` (Accessed: 24 December 2024).

[17] StatisticalHelp. Gini coefficient of inequality, 2024. Accessed: 28 December 2024.

[18] Joe Hasell. Measuring inequality: what is the gini coefficient? *Our World in Data*, 2023. https://ourworldindata.org/what-is-the-gini-coefficient.

[19] Olivia Guest. oliviaguest/gini. `https://github.com/oliviaguest/gini`, n.d. Accessed: 28 December 2024.

[20] Olivier Idir and Alessandro Renzaglia. Multi-robot weighted coverage path planning: a solution based on the darp algorithm. In *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 98–104, 2022.

[21] Rodriguesrenato. Rodriguesrenato/coverage-path-planning: A coverage path planning algorithm that combines multiple search algorithms to find a full coverage trajectory with the lowest cost. `https://github.com/rodriguesrenato/coverage-path-planning`, n.d. Accessed: 21 December 2024.

[22] Junnan Song and Shalabh Gupta. $\varepsilon^{\star}$: An online coverage path planning algorithm. *IEEE Transactions on Robotics*, 34(2):526–533, 2018.

[23] Zongyuan Shen, James P. Wilson, and Shalabh Gupta. $\epsilon^{\star}+$: An online coverage path planning algorithm for energy-constrained autonomous vehicles. In *Global Oceans 2020: Singapore – U.S. Gulf Coast*, pages 1–6, 2020.

[24] Marija Dakulović, Sanja Horvatić, and Ivan Petrović. Complete coverage d* algorithm for path planning of a floor-cleaning mobile robot. *IFAC Proceedings Volumes*, 44(1):5950–5955, 2011. 18th IFAC World Congress.

[25] Karthik Karur, Nitin Sharma, Chinmay Dharmatti, and Joshua E. Siegel. A survey of path planning algorithms for mobile robots. *Vehicles*, 3(3):448–468, 2021.

[26] Hai Zhong, Yutaka Shimizu, and Jianyu Chen. Chance-constrained iterative linear-quadratic stochastic games. *IEEE Robotics and Automation Letters*, 8(1):440–447, 2023.

[27] Emre Gursoy. Multi-robot trajectory planning under traffic interactions. Semester project, Sycamore Lab, EPFL, 2024. Supervised by Kai Ren and Prof. Maryam Kamgarpour.

[28] Optitrack. Optitrack - motion capture systems, 2024. Accessed: 2024-12-27.

[29] Fengyu Zhou, Baoye Song, and Guohui Tian. Bézier curve based smooth path planning for mobile robot. *Journal of Information & Computational Science*, 8(12):2441–2450, 2011.

[30] Mohammad Hasan Jalili Bahabadi, Amir Mahdavi, and Saeed Khankalantary. Heterogeneous coverage path planning for multi-agent systems with aco and ga. In *2024 32nd International Conference on Electrical Engineering (ICEE)*, pages 1–6, 2024.