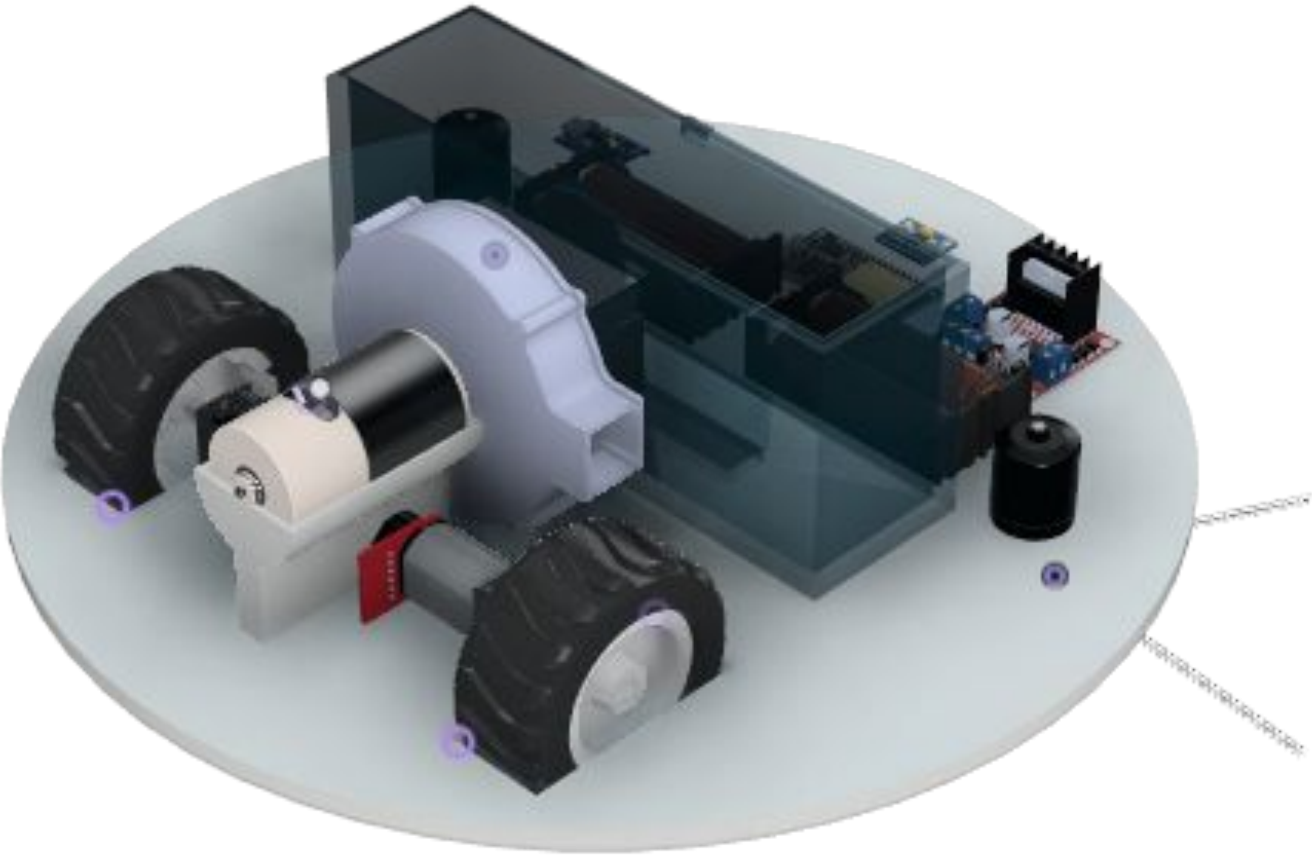
Multi-agent Coverage Planning System

TEAM
TECHKNIGHT'S ALLIANCE
IIT MADRAS



KIRTAN PATEL
KAHAN LAKHANI
NISHARG MANVAR
HARISH R
DHRUV MAROO

TABLE OF CONTENTS

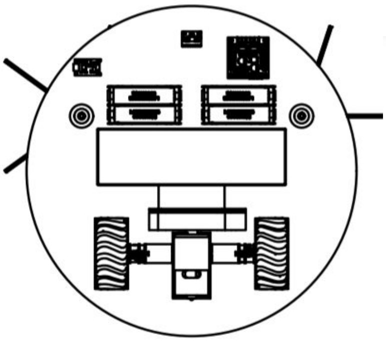
ROBOT DESIGN

Mechanical Module

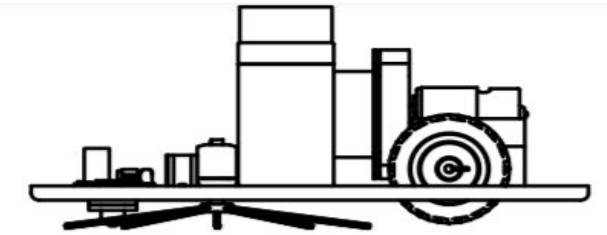
The key factors taken into consideration while design the robot were:

- Dimension Constraints
    - The design had to be made in accordance with the dimensions provided in the problem statement. This mainly influenced the base geometry of the robot.
- Cleaning Efficiency
    - The robot had to be designed in such a way that optimal cleaning efficiency could be achieved.
- Equipment Onboard
    - The design had to take into account the electrical as well as the cleaning equipment needed to be installed into the robot. It had to make sure that there was not only enough space for the equipment to be installed but also be easily accessible and have enough space for them to perform optimally.
- Mobility
    - The design had to take care that the robot was mobile enough for it to cover the area quickly but at the same time not compromise on its cleaning abilities.

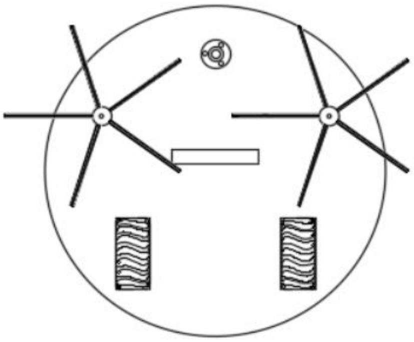
Taking into consideration the above factors, the robot was designed to be of a disk shape for better maneuvering with two rotating brushes on the sides for increasing cleaning area and efficiency. A three wheel system was carefully and systematically laid out to ensure low center of gravity and accurate control of the robot. The wheel placement was decided by triangulation method to provide more room at the front for a higher sweeping area for the brushes. The two motors provide the necessary mobility and the front caster provides the necessary support and stability for the movement.



Top View

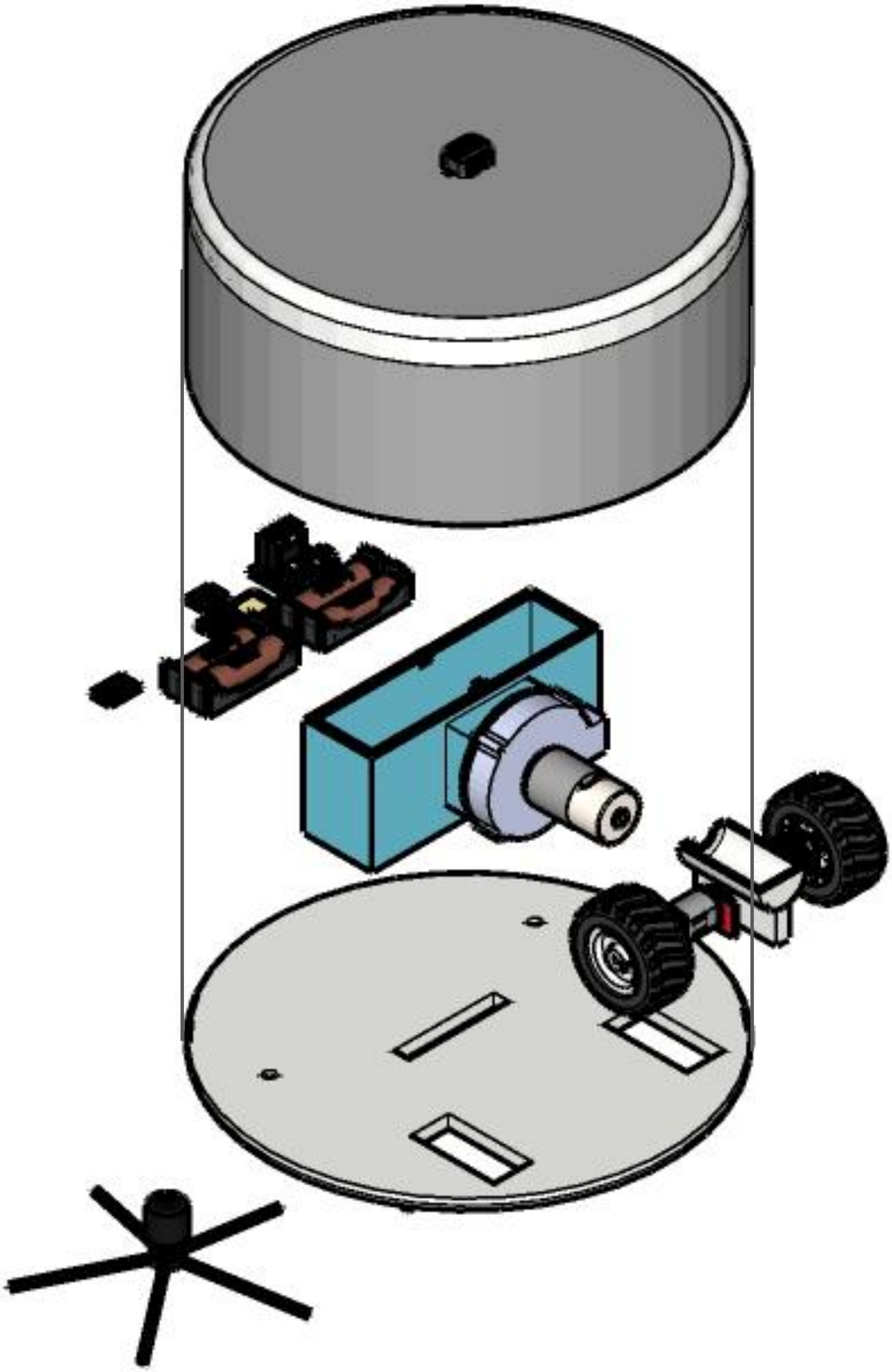


Front View



Bottom View

EXPLODED VIEW



Manufacturability

The material being used for the case and most of the robot is ABS because it is easily

manufactured, cheap and it can be **injection molded**. Acrylonitrile butadiene styrene, or ABS, is a common thermoplastic used to make light, rigid, molded products such as pipes, golf club heads. The styrene gives the plastic a shiny, impervious surface. The butadiene, a rubbery substance, provides resilience even at low temperatures. ABS can be used between −25 °C and 60 °C.

ABS is relatively safe to handle as it cools down and hardens, making it one of the easiest plastics to handle, machine, paint, sand, glue, or otherwise manipulate.

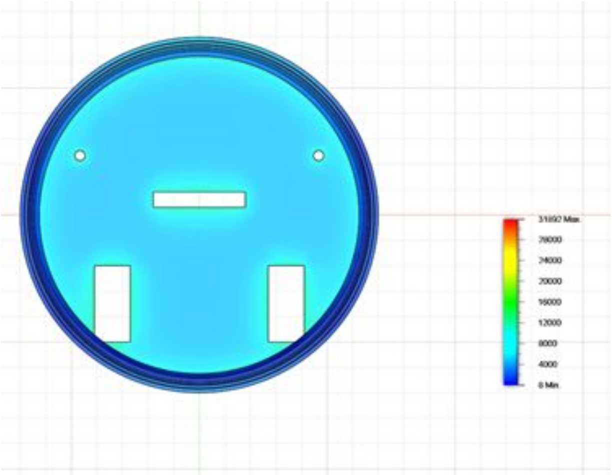
The brushes are arranged in a circular pattern to provide complete all around coverage. The radius of the brush is taken to be **100mm** to provide efficient coverage. The brushes used are of **nylon** to provide necessary **strength** and **flexibility.**

Serviceability

The top is attached to the side body by push joints and can be easily uncovered to give access to the internal parts. As a result all the internal parts can be directly accessed and be cleaned, repaired or replaced. The side body is connected to the base via screws at the bottom to give unhindered access to the internal components.

Since we expect the robot to function in a dirty environment, the mechanical parts used need regular maintenance for optimum performance and need to be replaced after a certain period of time.

PART	CLEANING FREQUENCY	REPLACEMENT FREQUENCY
BIN	AFTER EVERY USE	–
BRUSHES	AFTER EVERY 5 USES	AFTER 180 USES
FILTER	AFTER EVERY 5 USES	AFTER 50 USES
FRONT CASTER WHEEL	AFTER EVERY 10 USES	–



Failure Cases

The body has been designed with an appropriate quantity of **ABS plastic** to handle the weight of the components and the **high forces and torques** developed due the motors. An approximate simulation for load design was done with an even distribution of 3 kgs over the base along with the outer case and the result gave a von Mises stress of **0.13 MPa**. So the base can handle a lot larger loads as compared to the one currently applied. Even half the thickness of the base can be sufficient for the load currently applied.

The major cause of malfunction of mechanical elements can be seen is **contamination** i.e. dust entering the parts. Dust entering the vacuum fan can lead to decrease in the vacuum power of the cleaner. Similar malfunctions can be caused by dirt entering the brush motors causing a **reduction in the RPM** of the brushes. Proper and timely **maintenance** of the filter can help in reducing these problems.

A good design practice is to provide a separate compartment for the electrical components, preventing them from being contaminated by the debris collected. So that they can operate optimally in an **isolated environment**, also providing proper ventilation to the batteries is necessary to stop them from overheating.

Redeposition can be a major problem leading to inefficient cleaning, placing a hinged cover on the mouth of the bin so that it closes once the vacuum process stops and no debris is allowed to escape back outside. Also making sure that the walls of the bin above the hole are at an angle to prevent the debris coming in at high speed to bounce right back out of the bin.

Electronic Components

Manufacturability

Component	Manufacturer	Procurement in India	Component Testing
MPU 6050	InvenSense	Mouser Electronics	Basic Calibration (using Adafruit Library)
TF-LUNA Micro LiDAR	Benewake	Robu.in	Elementary software testing (using the Serial library)
ESP-32 WROOM board	Espressif	Robu.in	Unit tests (using the ESP-IDF)
LG HG2 18650 Cells	LG	Battery Bro	Capacity tests and CCD tests
DC Motors	Kysan Electronics	Kysan Electronics	Direct testing to find temperature and efficiency

Serviceability

Battery Pack

- The life of the battery pack is expected to be around 400 cycles till a significant drop in performance is noticed.
- In case of depreciated capacity, the batteries can easily be replaced with new ones.

Failure Mode Effect Analysis

Scope

- **Resolution -** The analysis will limit itself to the electrical and electronic components of the Robot vacuum cleaner. It will also consider the implementation of the path planning algorithms.
- **Focus -** Accuracy of working, Safety of user

Potential Failure modes

1. **Damaged Sensors**
    - **Cause :** The IMU and Lidar might get damaged due to constant exposure to water and other fluids.
    - **Effect :** Decrement in accuracy and total non-working in case of severe damage
    - **Detection :** If one sensor is damaged, the sensor fusion algorithm detects a significant amount of discrepancy in the values obtained from two sensors.
    - **Action :** Alert the user to replacement of the sensor.
2. **Erroneous Navigation**
    - **Cause :** The cumulative errors in the position of the bot leads to a significant deviation from the intended path
    - **Effect :** Fallacious values being fed into the path planning algorithm further reducing the efficiency of bot
    - **Action :** The following solutions are proposed :

        i. Implementation of an Extended Kalman filter algorithm to fuse the sensor data obtained from inertial measurement unit and LIDAR. (Can be achieved via ROS Packages)

        ii. Setting up a Proportional-Integral-Derivative controller in the navigation ROS stack to fine tune the control.

3. Abnormal Temperature

- **Cause :** Due to overload and continuous operations, the Lithium batteries might get over-discharged
- **Effect :** Significant risk of abnormal temperatures and the batteries getting exploded
- **Detection :** The internal temperature sensors monitor the temperature of the batteries and the system continuously
- **Action :** The battery pack comes with a BMS which should handle this case forcing a shutdown of the power circuit. Further to prevent reaching abnormal temperatures, a proper ventilation system is considered while designing the model.

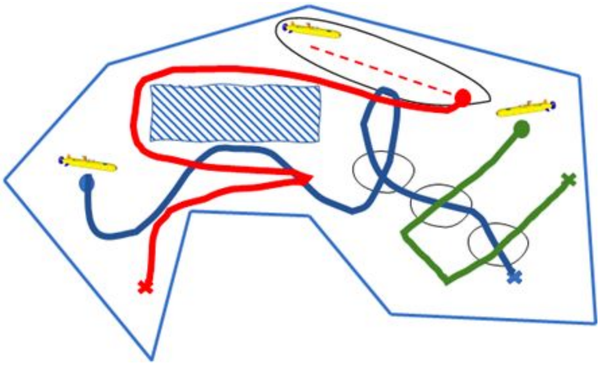
DARP Algorithm Analysis

OBJECTIVE:

The most fundamental problem in path planning is determining the path each robot would take to optimise time as well as avoid obstacles. Of course, this can even be achieved by a single robot moving randomly in the space after a (probably) long period of time. However, the limited battery capabilities of autonomous robots, in addition to the constantly increased areas that need to be covered/monitored, have motivated the deployment of several autonomous devices with advanced path planning mechanisms.

So our objective is to: <u>Design robot paths that completely cover this area of interest in the minimum possible time called coverage path planning problem (CPP).</u>

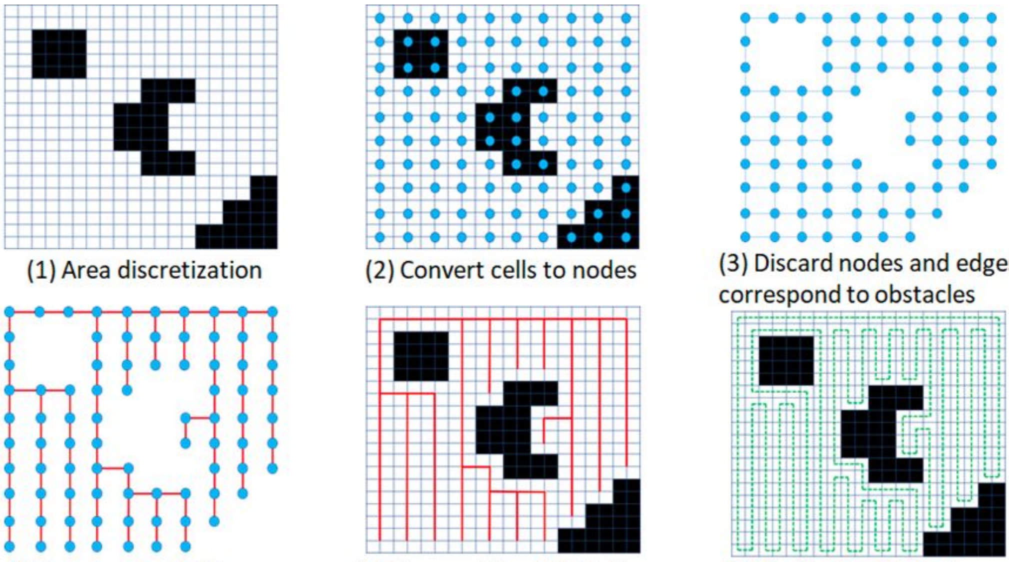
- ***Eliminate backtracking***
- ***Leave no area uncovered***
- ***Equal robots' trajectories***
- ***Avoid mismatch between current position and starting location***



BASIC BUILDING BLOCKS:

Firstly we need to represent our area of interest into small blocks. Therefore, one of the most common areas representation techniques is to separate the field into identical cells (e.g. in the size of the robot), such that the coverage of each cell can be easily achieved. Apparently, for any arbitrarily shaped area, the union of the cells only approximates the target region. Thus this technique, which is also adopted in our approach, is termed **cellular decomposition**.

Now, let us assume we have only one robot to cover the whole area, and we need to find the optimal path for that one robot for the already given map. This is a relatively simple problem, i.e. “single robot coverage planning problem (inside an already known terrain)”. There are, of course, multiple methods to solve this problem. However, one of the dominant approaches is the **spanning tree coverage (STC)** algorithm, which is able to guarantee an optimal covering path in linear time. The term *optimal* encapsulates that the generated path does not revisit the same cell (non-backtracking property), completely covers the area of interest and achieves all the above without any preparatory effort (the robot can be initiated at any non-occupied cell). It is based on constructing a minimum spanning tree on the already cellular decomposed region, as discussed previously.



(1) Area discretization

(2) Convert cells to nodes

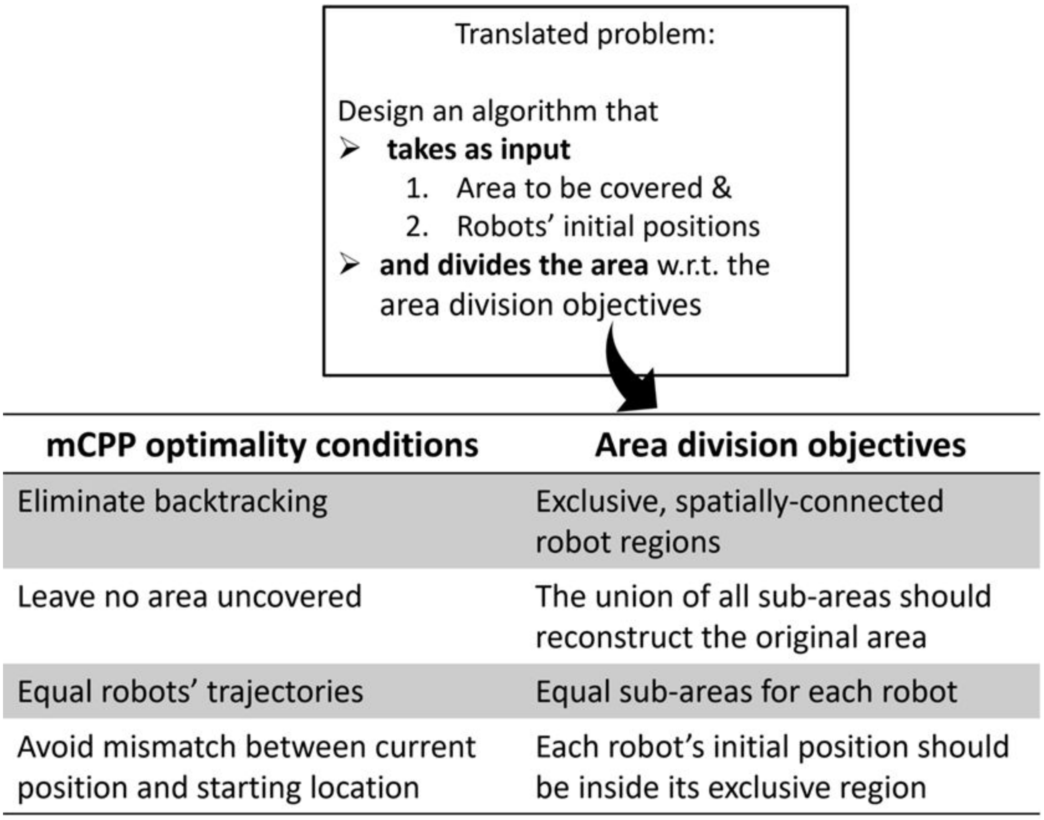
(3) Discard nodes and edges that correspond to obstacles

(4) Construct a Minimum Spanning Tree

(5) The resulting MST is "our guide" for the navigation

(6) The final path that circumnavigates the MST

However, as mentioned above, we are concerned with finding the solution for multiple robots called mCPP (multiple CPP). Unfortunately, the mCPP problem was proven to be extremely difficult to be adequately addressed. As a matter of fact, solving mCPP with minimal covering time has been proven to be NP-hard. We shall now use the above-mentioned building blocks to try and tackle this problem.



Translated problem:

Design an algorithm that

- **takes as input**
  1. Area to be covered &
  2. Robots' initial positions
- **and divides the area** w.r.t. the area division objectives

mCPP optimality conditions	**Area division objectives**
Eliminate backtracking	Exclusive, spatially-connected robot regions
Leave no area uncovered	The union of all sub-areas should reconstruct the original area
Equal robots' trajectories	Equal sub-areas for each robot
Avoid mismatch between current position and starting location	Each robot's initial position should be inside its exclusive region

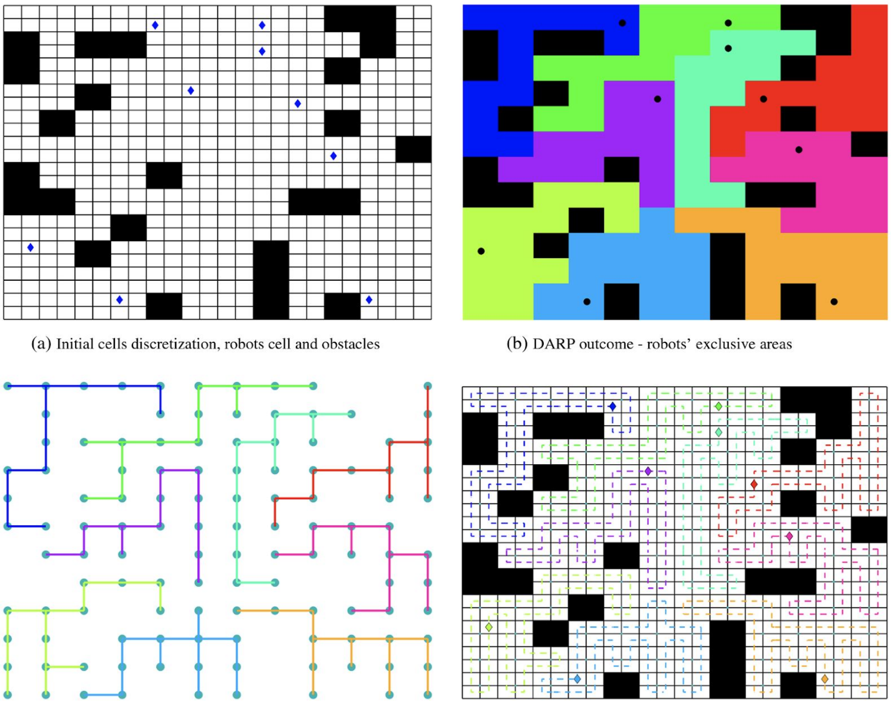
BUILDING TOWARDS DARP:

Now, the first thought that comes to mind is, can we use the STC method for mCPP. The answer is Yes, we can by

1. Extract a single path using the STC algorithm
2. Each robot follows the segment that starts from its initial position till it reaches the starting position of any other robot.

However, this results in Uneven robot trajectories, and the final solution is highly dependent on the initial positions of the robots.

So, the main idea of DARP (Divide Areas based on Robot's Initial Positions) is to apply STC individually for each robot by first assigning a certain area of the map to each robot. Thus here, we can manually control how much of the area do we want each robot to cover (usually equal areas for all robots).



(a) Initial cells discretization, robots cell and obstacles

(b) DARP outcome - robots' exclusive areas

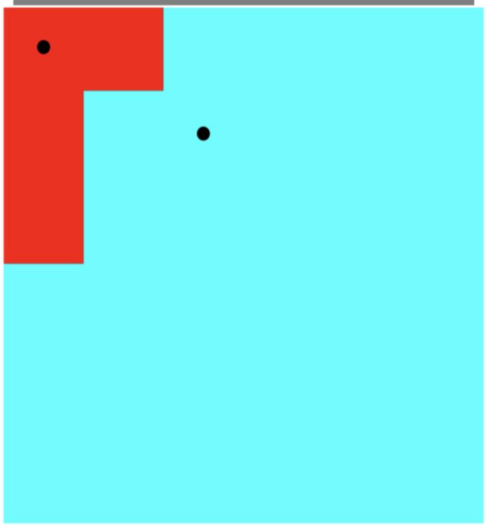
(c) Constructing Minimum Spanning Trees for each one of the robots sets

(d) Final Paths, designed to circumnavigate the MSTs

Now the question is how do we divide the map and give individual regions to each robot optimally.

Again the most natural way seems to use the idea of Voronoi partitioning, i.e. assign a block to the robot with the closest initial position.

Voronoi partition based on the distance matrices



However, if we use this method, we again run into the problem where the area that each bot covers becomes dependent on the initial position of each bot.

Therefore we need to tune our process such that eventually, all robots have equal areas. We can basically use the concept of error control, i.e. in case the area of a bot is lesser than what it is supposed to be, we reduce its ‘distance’ to all cells so that more are now assigned to that robot, and similarly, for robots which have higher area, we increase their ‘distance’.

Let us now formally define this method,

DARP 0.5:

For every ith operational robot, an evaluation matrix E$_i$ is maintained. This evaluation matrix E$_i$ expresses the level of reachability (e.g. distance) between the cells of L (the map) and ith robot's initial position x$_i$(t$_0$). During each iteration, the assignment matrix A is constructed as follows:

$$A_{x,y} = argmin_i(E_{i|x,y}), \forall\, i$$

Let the values of the matrix Eibe initialised with some distance metric (e.g., Euclidean distance). Now, as mentioned earlier, the DARP algorithm's core idea is that each evaluation matrix E$_i$ can be appropriately "corrected" by a term mi as follows:

$$E_i = m_i E_i$$

where m$_i$ is a scalar correction factor for the ith robot.

Now, let us define an error function for each robot J,

$$J = \; \sum_{i=1}^{N} (k_i - f_i)^2$$

Where k$_i$ denotes the number of assigned cells to the ith robot, while f$_i$ denotes the number of cells that the ith robot has to cover.

Now, depending on the value of J for each robot, we can change the value of m$_i$ appropriately for all robots by using a standard gradient descent method,

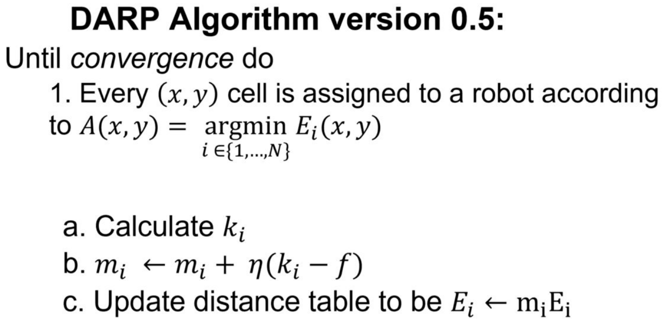
$$m_i = m_i - \mathrm{\eta} \frac{\partial J}{\partial m_i}$$

Which can eventually be written as,

$$m_i = m_i + c(k_i - f_i)$$

where c denotes a positive tunable parameter.

Thus, to summarise the algorithm so far, the pseudo code of the algorithm is



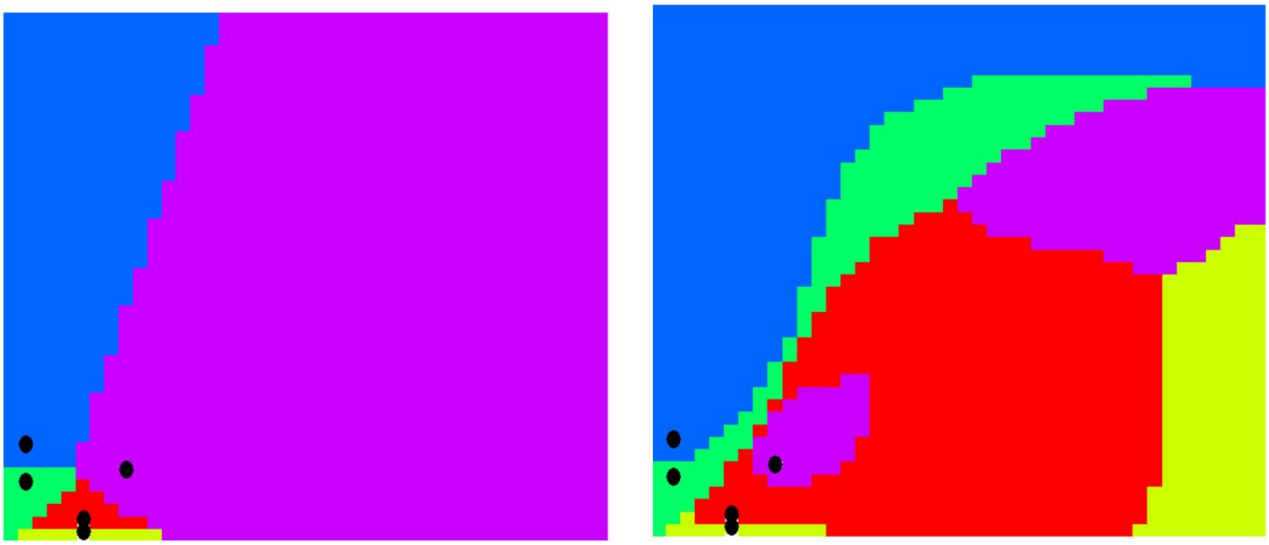
DARP Algorithm version 0.5:

```
Until *convergence* do
    1. Every $(x, y)$ cell is assigned to a robot according
    to $A(x, y) = \underset{i\,\in\{1,\dots,N\}}{\operatorname{argmin}}\, E_i(x, y)$

      a. Calculate $k_i$
      b. $m_i \leftarrow m_i + \eta(k_i - f)$
      c. Update distance table to be $E_i \leftarrow \mathrm{m_i E_i}$
```

DARP 1.0

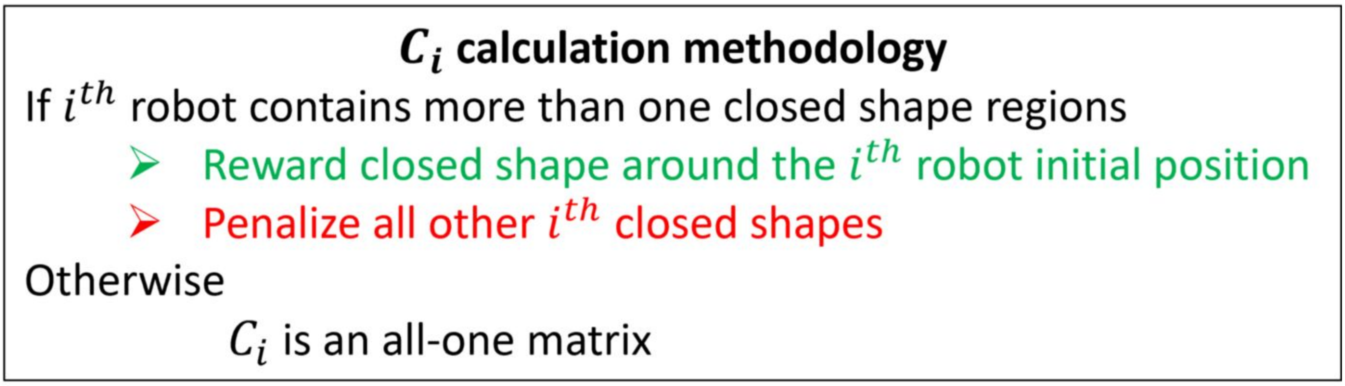
Although in the original Voronoi partitioning, spatial connectivity was guaranteed by design, now, in some cases is violated. This phenomenon is due to the fact that now the metric function for each robot, over which the assignment process is implemented, is no longer a distance function. For example, the following figure illustrates the initial partitioning (left-hand side) based on the Euclidean distance, and the subfigure on the right-hand side illustrates the assignment after some iterations.



Initial Euclidean partitioning

Spatially disconnected partition after applying $E_i \leftarrow \mathrm{m_i E_i}, \forall\, i$ several times

It is evident that the purple region is disconnected, and therefore with such an assignment, the non-backtracking guarantee is now at risk. To tackle this issue, an extra connectivity matrix is introduced. The rationale behind the calculation of this connectivity matrix is as follows:



C_i calculation methodology

If i^{th} robot contains more than one closed shape regions

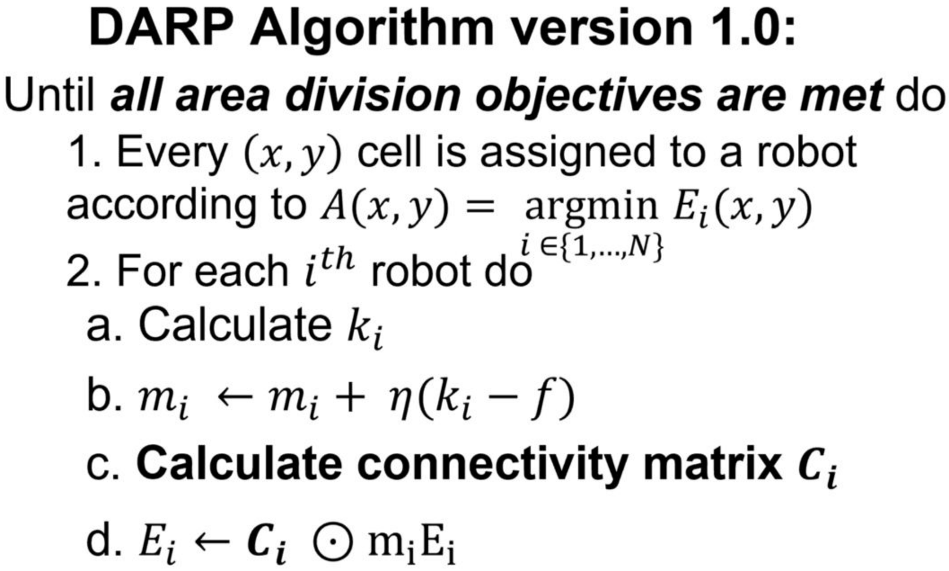
➢ Reward closed shape around the i^{th} robot initial position

➢ Penalize all other i^{th} closed shapes

Otherwise

C_i is an all-one matrix

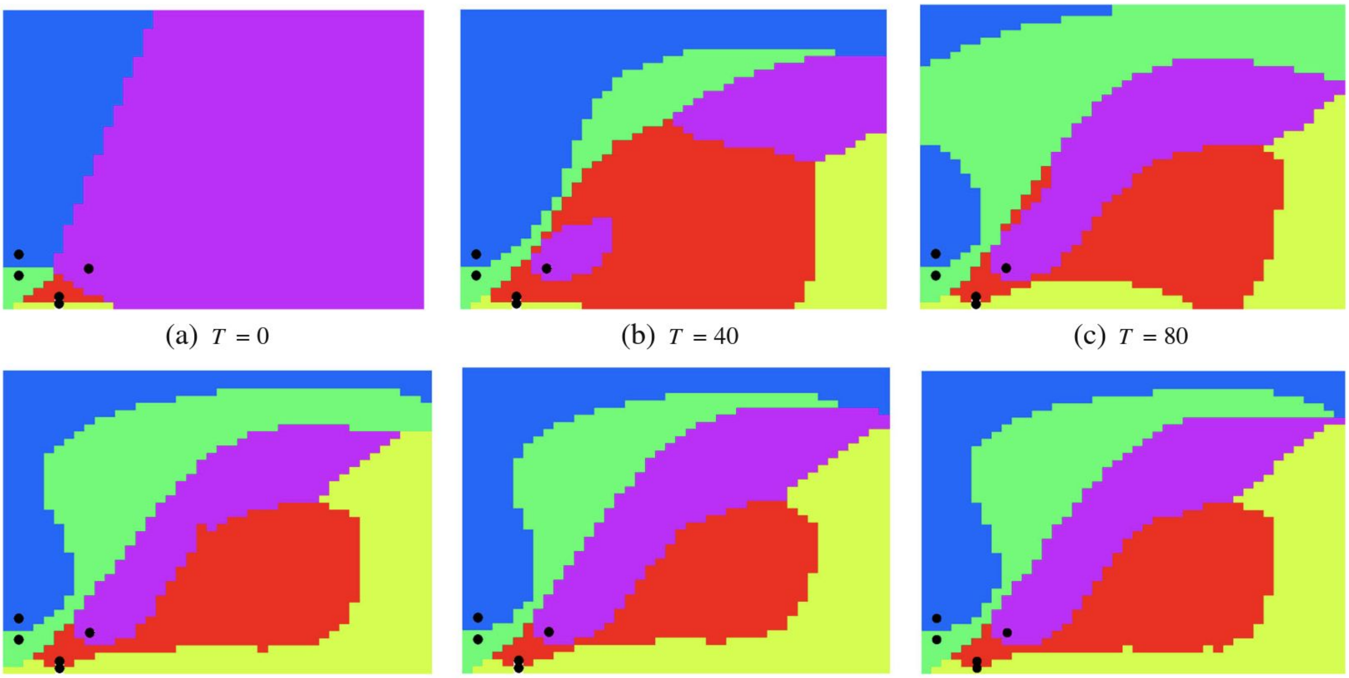
Thus, The final update in the ith evaluation matrices is calculated as,



DARP Algorithm version 1.0:

```
Until all area division objectives are met do
  1. Every $(x, y)$ cell is assigned to a robot
  according to $A(x, y) = \underset{i \in \{1,\ldots,N\}}{\operatorname{argmin}} E_i(x, y)$
  2. For each $i^{th}$ robot do
    a. Calculate $k_i$
    b. $m_i \leftarrow m_i + \eta(k_i - f)$
    c. Calculate connectivity matrix $\boldsymbol{C_i}$
    d. $E_i \leftarrow \boldsymbol{C_i} \odot \mathrm{m_i E_i}$
```

Even after this update, the DARP algorithm might give unconnected regions for some iterations. However, the algorithm claims that in most cases, this will eventually help find the optimal solution. More precisely, the DARP algorithm is capable of escaping the local minima by temporarily violating the condition about the connectivity of each ith robot assignment matrix. Afterwards, the algorithm gradually eliminates unconnected areas by reinforcing the robot’s evaluation Ei around the original (the one that the robot lies in) subarea. By the time the connectivity inside the exclusive robot sets Li is restored, the evaluation matrices Ei will have completely changed their forms, and ideally towards the optimal cells assignment.



(a) $T = 0$

(b) $T = 40$

(c) $T = 80$

(d) $T = 120$

(e) $T = 200$

(f) $T = 260$

DEALING WITH SPATIAL DISCONNECTIVITY:

When we used DARP in practice, we came across multiple corner cases where the algorithm failed to converge, one of the cases being when the initial positions of the robots were too close to each other. We are of the belief that the spatial disconnectivity problem of DARP is responsible for this, and once the regions become disconnected, either it stays that way or it takes too many interactions to join back. To tackle this problem we introduced a filter in the sense that only if all pairs of robots were separated a constant minimum distance will we try and find the optimal path using DARP.

In some other cases, even though the bots were far spaced, the algorithm was unable to converge. We found out using trial and error that if we give some leeway in terms of area covered by each bot (e.g., for two bots instead of exactly 50%, we use 45% and 55% areas), the algorithm is able to converge. Basically, sometimes the algorithm trying to assign exactly equal areas to all bots fails to converge. Thus, using slightly unequal areas seems to be the way forward for those cases.

Mapping and Planning

Online Path Planning Algorithms

With Mapping being introduced in the problem, the base problem changes from an offline path planning problem to an online path planning problem.

In offline mode, the path planning task starts from a point where the agent has a complete knowledge about: its environment with obstacles (The complete map is given), its initial position, and its final goal within this environment. The offline path planning task simply connects the initial position to the final position, then the agent will execute the generated path.

In online mode, path planning is carried out in parallel while (a) moving towards the goal, and (b) perceiving the environment including its changes.

However, Considering the interactions among (i) Agent, (ii) Environment, and (iii) goal, the difference between online and offline path planning can be seen from many points of view:

Offline path planning is generally used for static environments (or slowly changing). and only when a global map of the environment is initially available (given or built). And thus, this approach can ensure global optimum path (in terms of safety, shortest path, time, energy,).

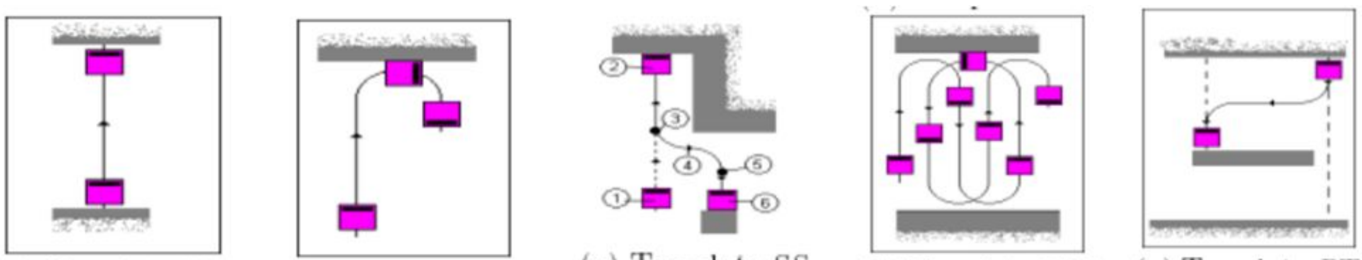
For dynamic environments, online path planning must be used since the path must be updated according to environment changes. This approach is also necessary for motion in an incompletely known environment map, since the agent is discovering its map while moving and needs to update the path according to any new knowledge. However, with this partial knowledge, this approach is time consuming, requires extra sensors, and it doesn't generate globally optimum paths, it can even fail to reach the target. but again, it is still a necessity.

1.Template Based Methodology

This system, when implemented on a commercially available mobile platform, exclusively relies on odometry and ultrasonic data, aiming at providing low-cost navigation modules that could be implemented.

Unlike the previously used DARP algorithm, for a satisfactory coverage of the entire space, neighbour paths may have overlapping areas. Sequences of maneuvers consisting of a predefined number of line segments and arcs will be employed to generate the total coverage path. Each elementary path type is denoted as a template, the total path being a sequence of templates. All templates take into consideration the parameters of the robot such as the minimum turning radius, the robot width and the cleaning area.To account for vehicle dimensions and simplify the path planning, the robot will be considered as a point object.

The complete coverage trajectory is planned as a sequence of predefined trajectories denoted as templates along the lines proposed. A set of 5 templates is proposed as the minimum number to achieve a satisfactory coverage of an area which contains obstacles in its interior.



(a) Template TM (b) Template UT (c) Template SS (d) Template UTI (e) Template BT

The use of the BT template is to cope with obstacles in the middle of the environment.

- **Template Towards Marker, TM**

Line segment that links two given points. This template is the backbone of nearly all the other templates

- **Template Uturn, UT**

Line segment followed by a U turn. With the exception of TM the execution of a UT template is fast when compared with all the other templates due to the simplicity of the included maneuvers The successive application of UT templates creates a snake trail pattern as represented above. The platform minimum turning radius and the required overlap between adjacent trails are the main geometric parameters associated with UT

- **Template Side Shift, SS**

This template provides a tool for changing between adjacent tracks when the mobile robot has a minimum turning radius that prevents the use of a UT template to achieve that change A Side Shift template can also be used when an environmental barrier such as a wall prevents the use of a UT template, this being the situation displayed. The main drawback of this template is that high coverage is redundant.

- **Template U turn Interlaced, UTI**

It is well suited for robots with a large minimum turning radius and its application reduces the wheel slippage when compared to the UT template This fact was experimentally observed.

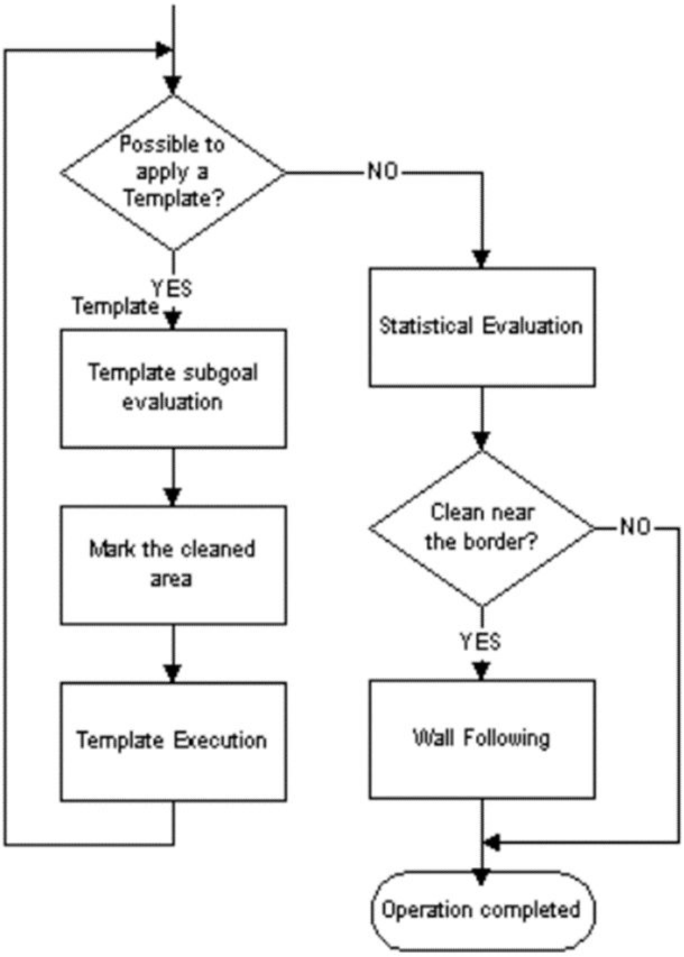
- **Template Backtracker BT**

This template is extremely useful when there are obstacles lying in the middle of the area to be cleaned The backtracking movements supported by template BT allow the mobile robot to clean areas not yet covered that would otherwise be left behind uncleaned, this being one of the novelties of the proposed method.

The complete coverage path planning methodology is implemented iteratively after an initial localization procedure. After a human operator drives the platform to its initial location and an estimated location is evaluated based on ultrasonic data. At each iteration and based on the previously cleaned area and the a priori map the algorithm chooses the template to be applied as explained later in this section.

Before execution the trajectory that will be generated by the chosen template has to be further decomposed in such way that simple motion commands.such as Move(x,y,forward) or Turn(angle,left,radius) can be dispatched to the mobile robot This decomposition achieved by the path tracking module breaks up the templates into subgoals The subgoals of each template are the boundary points that separate line segments from turns thus decomposing the template in a set of subtemplates see for details Following template subgoals evaluation a prediction of the new cleaned area is carried out followed by the template execution which is achieved by the execution of each of its sub templates.

If no template can be applied this means that all the area has already been cleaned or there is a deadlock situation. In both cases a further procedure for wall following is implemented This procedure fully supported on ultrasonic data.



For obstacles like the ones mentioned in Round-1, (All those that can be shows as blocks on a grid),A template will always be applicable, except at the end, or if the robot is not aligned properly. In cases with slat edged obstacles, the wall following is implemented.

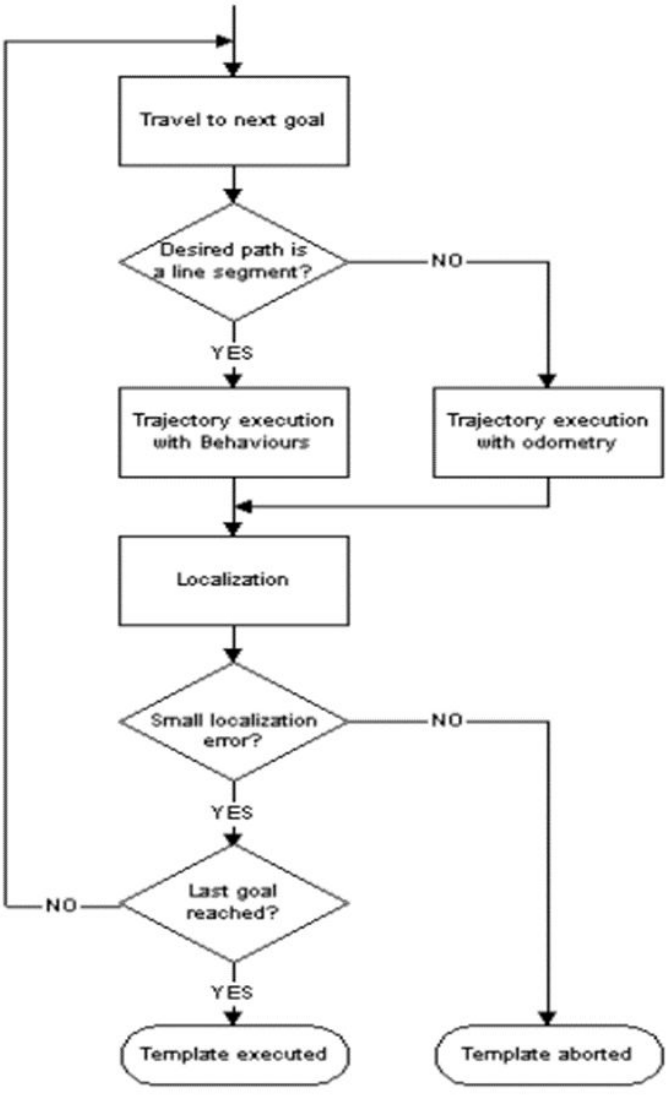
A heuristic formula is used for choosing the best template to apply at each stage aiming at maximizing the total covered area The template BT is the 1st one whose application is checked by the planner because if any uncleaned space is left behind it must be covered before moving on otherwise it will be left uncleaned.

Next the choice of an interlaced template UTI is trade aiming at reducing wheel slippage and consequently reducing odometry errors Due to a fast execution time the use of a UT template is checked next The Side Shift template is the last one to be considered due to its high coverage redundancy The TM template is used as a major component of all the other templates given that all line segments are executed by TM templates

Thus, the order or checking of templates is as follows: BT →UTI →UT →SS

Path Tracking

For Path Tracking and Path Correction using Odometry, the following algorithm is used to track the Path of the robot.


Travel to next goal
Desired path is a line segment?
NO
YES
Trajectory execution with Behaviours
Trajectory execution with odometry
Localization
Small localization error?
NO
YES
Last goal reached?
NO
YES
Template executed
Template aborted

Localization

Localization is essential to correct the cumulative errors inherent to the odometry system. Using the a priori map and the data from the LiDAR attached on the front of the bot's body, a test is carried out at each subgoal so as to determine if the real data acquired by the sensors matches the bot's position and orientation estimate given by odometry. If the deviation between the predicted and the real measurements is more than a certain threshold the real data is used to recalibrate the overall location of the mobile robot. Also, to make localization more robust, we intend to fuse both the aforementioned data sources using Kalman filters, allowing to reduce the effect of noise and errors in the values obtained from these sources.

This localization approach, based on low cost sensors and on the a priori map, requires no special modification on the environment (e.g. installation of bar codes). However, the implemented methodology introduces an overhead on path execution duration because localization has to be done with the vehicle stopped near an obstacle.

2. Epsilon Star

The Epsilon Star Algorithm works on a grid based approach.

The algorithm is built upon the concept of an Exploratory Turing Machine (ETM) which acts as a supervisor to the autonomous vehicle to guide it with adaptive navigation commands. The ETM generates a coverage path online using Multiscale Adaptive Potential Surfaces (MAPS) (hierarchical potential surfaces) which are hierarchically structured and dynamically updated based on sensor information. The ε ⋆ -algorithm is computationally efficient, guarantees complete coverage, and does not suffer from the local extrema problem.

Vehicle Requirements

1. Localization System
   Provides vehicle location (e.g., GPS), and heading (e.g.,Compass)
2. Range Detector with Sensing Radius R_s
   Allows the vehicle to detect obstacles in the local neighborhood (e.g., laser)
3. Tasking Sensor with Radius r_t
   Allows the vehicle to carry out certain tasks (e.g., cleaning, target detection, crops cutting) while it operates in the field

Step 1 : Dividing Coverage Area into Tiles

The tiling formed by square tiles of side ε is called an ε -cell tiling. It is recommended that an ε -cell should be atleast big enough 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 ε depends on the following factors.

- A smaller ε provides a better approximation of the search area and its obstacles.
- On the other hand, a larger ε reduces the computational complexity by requiring less number of ε -cells to cover the area and it also provides improved robustness to uncertainties for localization within a cell.

The tiling is partitioned into three subsets:

- Obstacle cells (T^o) : they are detected online.
- Forbidden cells (T^f) : create buffer around obstacles
- Allowed cells (T^a) : these are the target cells to cover

Step 2 : Dynamically Constructed Multi-scale Potential Surfaces (MAPS)

With the Movement of Bot and detection of obstacles, the potential allotment of the maps is dynamic and thus this is useful in online path planning.These are Multi-level to avoid getting the robot into a place from which it has to back track it's way out to reach other uncleaned cells.

The Level 0 of the Map is where the boundaries of the map are set initially and the location of the known obstacles. Higher Level of Mapping is used to avoid the bot from getting cornered between cleaned areas. Thus, higher the Level of mapping, better the tendency of the robot to avoid back-tracking. This is termed as reaching a local extrema in the Potential Surface language.

Higher levels of MAPS are used to prevent the local extrema problem. Local Extrema implies no unexplored cells are available in the local neighborhood on Level 0. The potentials of unexplored blocks are adjusted as per the inputs and feedback given the detection system of the robot.

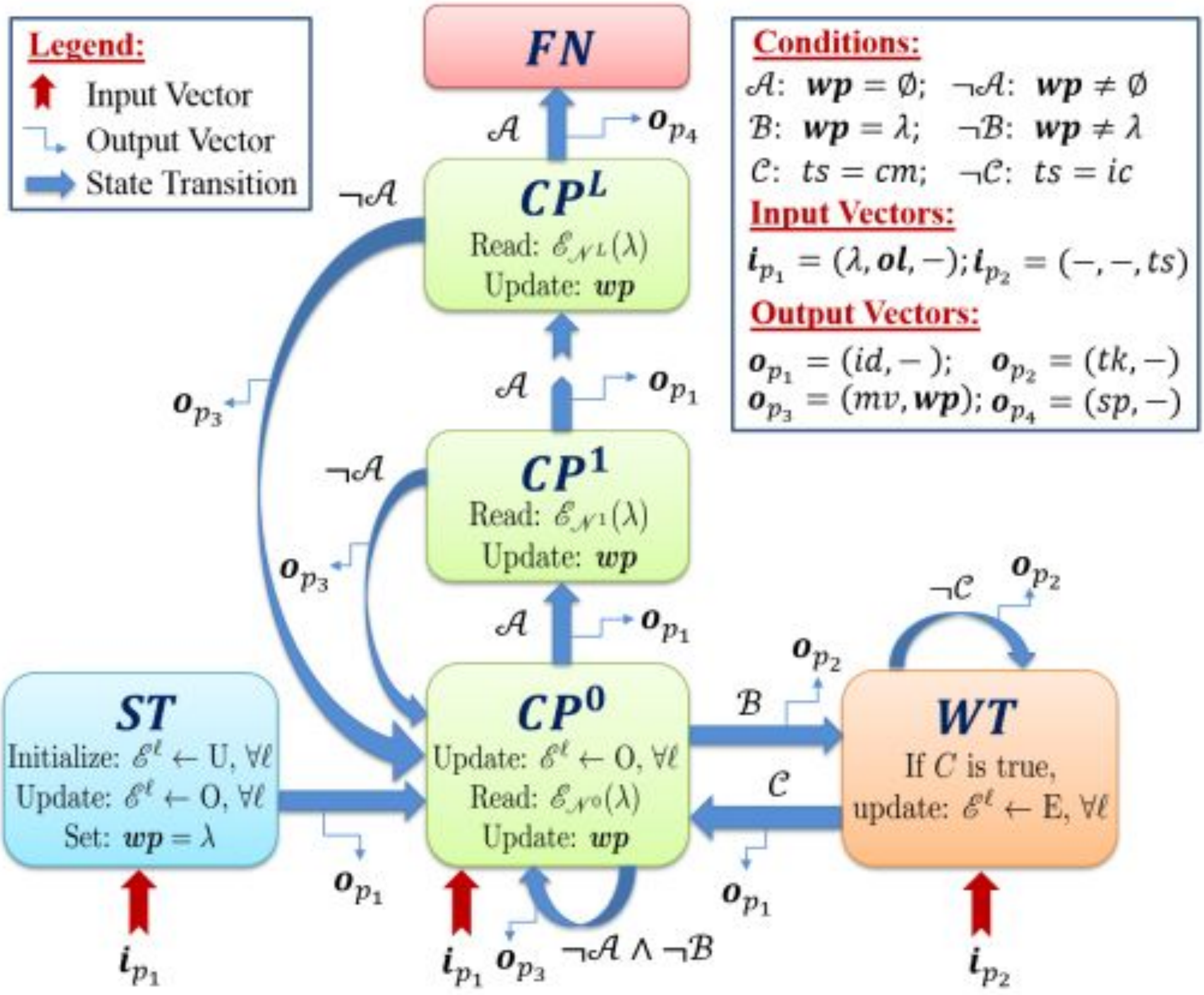
Step 3 : Supervisory Control Structure

The Exploratory Turing Machine (ETM) is used to control the usage of Machine States.

Machine States

- **The Start State (ST)**: start the machine and initialize the MAPS with all e-cells as unexplored.
- **The Computing States (CP):**
  - **CP^0**: compute waypoint **wp** using Level 0 of MAPS, and send navigation command cd.
  - **CP^1, CP^2….CP^L** : sequentially used to compute **wp** in case of a local extremum. Local extremum is when no waypoint is found.
- **The Waiting State (WT)**: wait for the vehicle to complete specific task (e.g., cleaning) in the current
  cell, until the status turns to complete.
- **The Finished State (FT):** terminate the operation upon complete coverage

Epsilon Star Algorithm Flow Chart


Legend:
Input Vector
Output Vector
State Transition
FN
$\mathcal{A}$
$\boldsymbol{o}_{p_4}$
$\neg\mathcal{A}$
CP^L
Read: $\mathscr{E}_{\mathcal{N}^L}(\lambda)$
Update: $\boldsymbol{wp}$
$\mathcal{A}$
$\boldsymbol{o}_{p_1}$
$\boldsymbol{o}_{p_3}$
$\neg\mathcal{A}$
CP^1
Read: $\mathscr{E}_{\mathcal{N}^1}(\lambda)$
Update: $\boldsymbol{wp}$
$\boldsymbol{o}_{p_3}$
$\mathcal{A}$
$\boldsymbol{o}_{p_1}$
$\neg\mathcal{C}$
$\boldsymbol{o}_{p_2}$
$\boldsymbol{o}_{p_2}$
ST
Initialize: $\mathscr{E}^\ell \leftarrow$ U, $\forall\ell$
Update: $\mathscr{E}^\ell \leftarrow$ O, $\forall\ell$
Set: $\boldsymbol{wp} = \lambda$
CP^0
Update: $\mathscr{E}^\ell \leftarrow$ O, $\forall\ell$
Read: $\mathscr{E}_{\mathcal{N}^0}(\lambda)$
Update: $\boldsymbol{wp}$
$\mathcal{B}$
$\mathcal{C}$
WT
If C is true,
update: $\mathscr{E}^\ell \leftarrow$ E, $\forall\ell$
$\boldsymbol{o}_{p_1}$
$\boldsymbol{o}_{p_1}$
$\boldsymbol{i}_{p_1}$
$\boldsymbol{i}_{p_1}$
$\boldsymbol{o}_{p_3}$
$\neg\mathcal{A} \wedge \neg\mathcal{B}$
$\boldsymbol{i}_{p_2}$
Conditions:
$\mathcal{A}$: $\boldsymbol{wp} = \emptyset$; $\neg\mathcal{A}$: $\boldsymbol{wp} \neq \emptyset$
$\mathcal{B}$: $\boldsymbol{wp} = \lambda$; $\neg\mathcal{B}$: $\boldsymbol{wp} \neq \lambda$
$\mathcal{C}$: $ts = cm$; $\neg\mathcal{C}$: $ts = ic$
Input Vectors:
$\boldsymbol{i}_{p_1} = (\lambda, \boldsymbol{ol}, -); \boldsymbol{i}_{p_2} = (-, -, ts)$
Output Vectors:
$\boldsymbol{o}_{p_1} = (id, -)$; $\boldsymbol{o}_{p_2} = (tk, -)$
$\boldsymbol{o}_{p_3} = (mv, \boldsymbol{wp})$; $\boldsymbol{o}_{p_4} = (sp, -)$

Operation of the ETM

Machine State	When is it reached?	What does it do?
The ST State	As soon as the autonomous vehicle is turned on.	Initialization of the system.
The CP^0 State	Either after system initialization, or when the current cell has just been tasked and needs a new 01.	Default state to compute for 01 on Level 0 of MAPS.
The CP^1, CP^2….CP^L States	When waypoint wp cannot be found in CP^0 state.	Sequentially switches to higher levels of MAPS, until wp can be found at some Level l ≤ L
The WT State	When the autonomous vehicle reaches the computed wp.	Command the vehicle to perform tasking (e.g., cleaning) in the current cell.
The FT State	When wp cannot be found in CP^L state.	Terminate operation since no unexplored cells are left.

That is,in state ST , the ETM initializes the MAPS. Since the whole area is initially unexplored, all ε -cells are assigned the state U, thus MAPS are constructed using only the potential field B. Then, the ETM cycles on and between the states CP^0 and W T , as follows. In each iteration of state CP0 , the ETM takes input from the autonomous vehicle about the newly discovered obstacle locations and its current position (λ). Then, it moves the head on the tape to λ and updates the MAPS in accordance with the discovered obstacles, and performs the following operations:
i) reads the potentials from the local neighborhood N^0(λ) of λ to compute the new waypoint,
ii) changes the head state to W T 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.

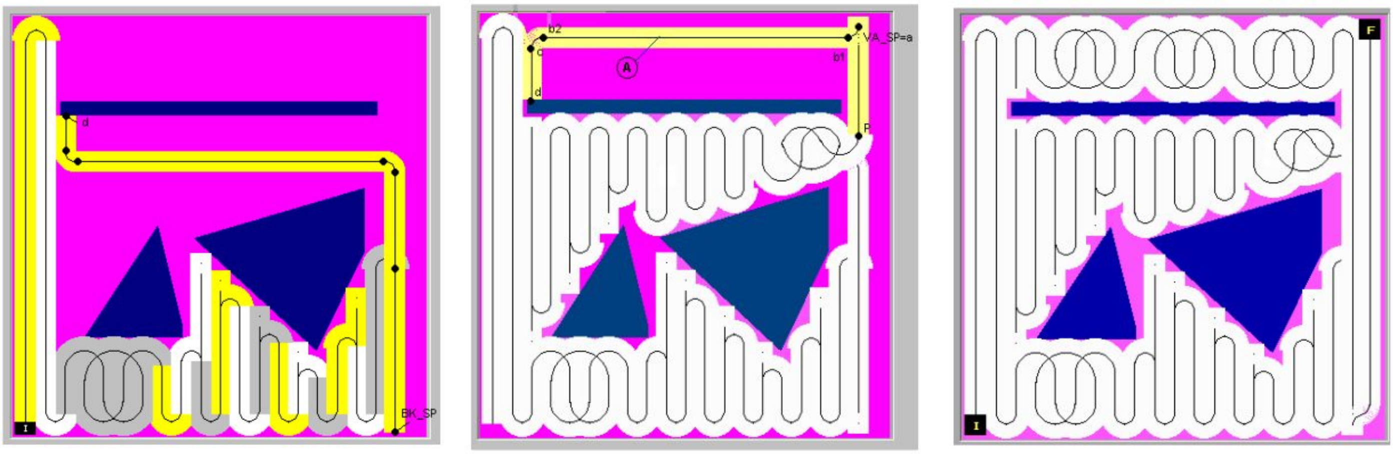
In each iteration of state WT, it receives the task status from the vehicle, and continues to send tasking
command until it is complete. Once the current cell is tasked, it updates the MAPS and returns to the state CP^0.
If the head gets stuck in a local extremum in state CP^0 , i.e. no waypoint could 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 N^1(λ) to find a waypoint. If no waypoint is found even at Level 1, then it switches to state CP^2 and so on until it finds one, then it comes down to state CP^0 and continues. If no waypoint is found even at the highest level then the ETM halts in state FN and the coverage is complete.

Here also backtracking occurs at extremas but it is something that can not be avoided in Online Complete Coverage Path Planning.

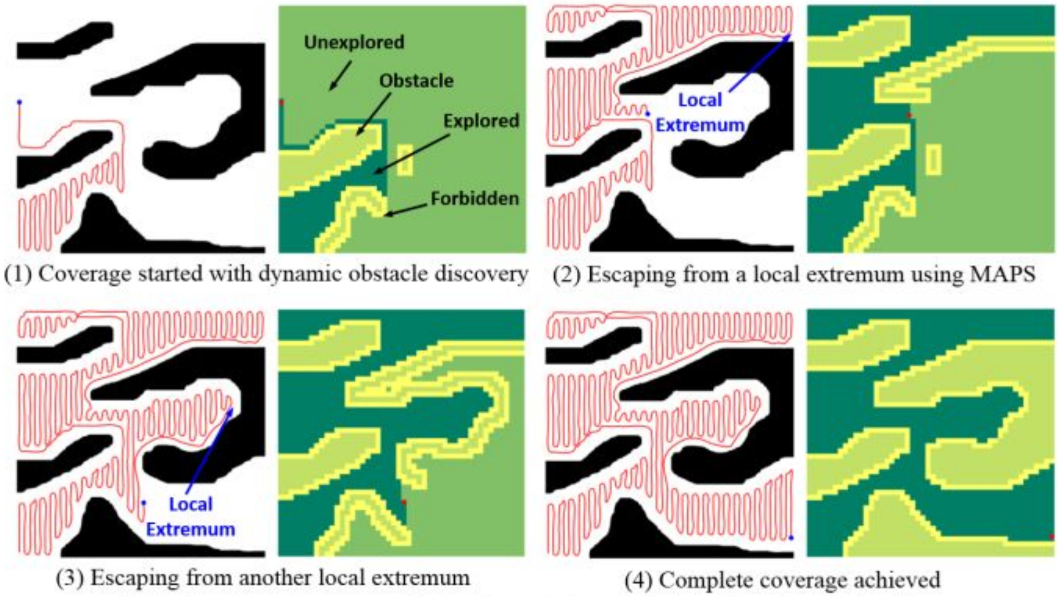
Final Robot Paths

Examples of Final Robot Paths :

1.Template Based Methodology

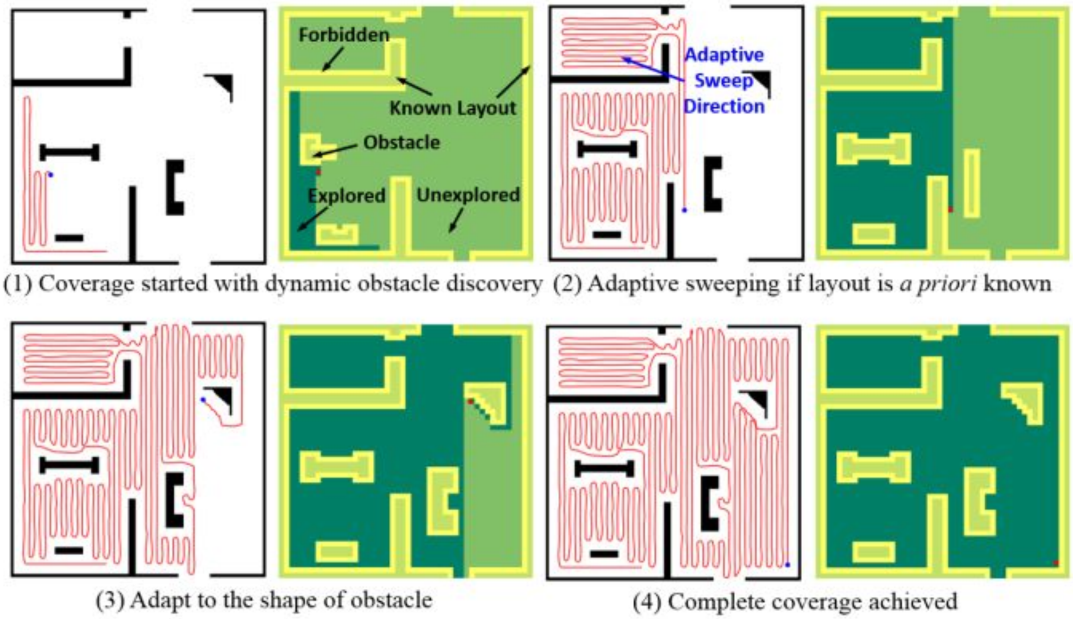


2. Epsilon Star



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

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



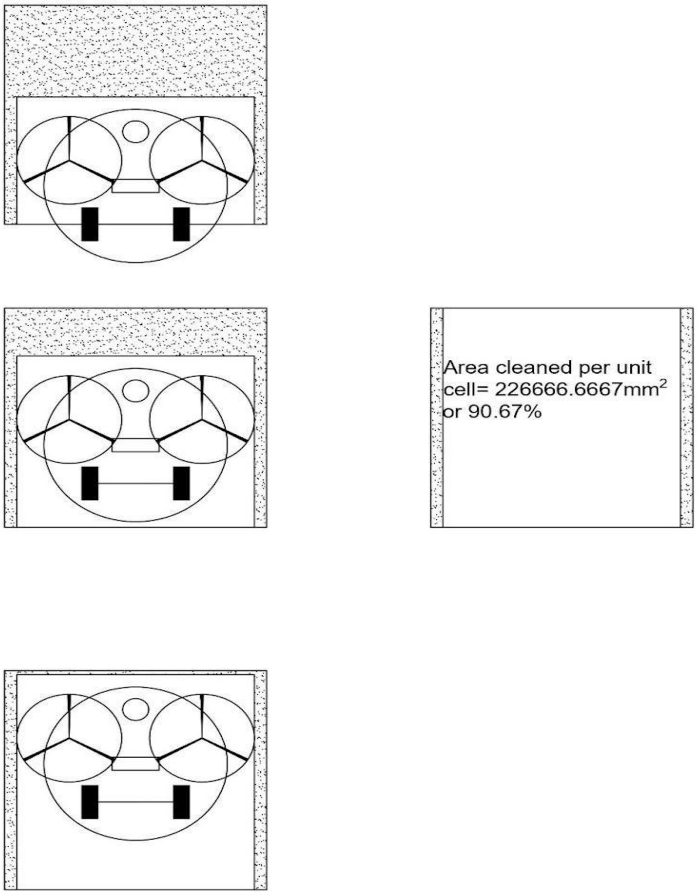
(1) Coverage started with dynamic obstacle discovery (2) Adaptive sweeping if layout is *a priori* known

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

Cleaning Efficiencies

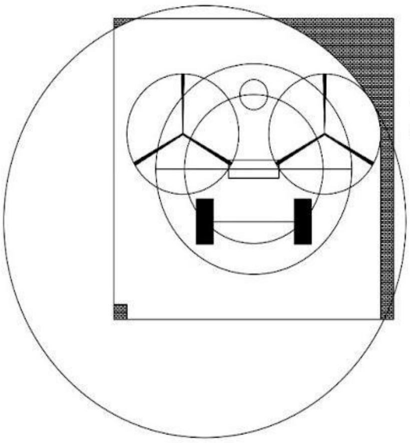
Coverage depends on the value of the orientation of the obstacle and epsilon. For grid obstacles and Smaller epsilon, we can achieve a cleaning efficiency of more than 90%. Having said that, since we are using Online Path Planning, the exact cleaning efficiency varies from map to map.

From Round 1 Report, it is clear that the bot covers the grid cell with more than 90% efficiency and hence any complete coverage algorithm used with the bot would give a cleaning efficiency of more than 90%.
Furthermore, with smaller and smaller values of epsilon and wall tracking, the cleaning efficiency is bound to go higher.



Area cleaned per unit cell= 226666.6667mm^2 or 90.67%

STRAIGHT PATH EFFICIENCY



Area covered =226950mm^2 or 90.78%

TURNING EFFICIENCY

Deployment Method

Having understood how the above mapping and planning algorithms work, we can now discuss their application to a system of robots.

1.Template Based Methodology

In any Template based Algorithm, there might be back-tracking and the need for the robot to go back to another location (for complete coverage).

Deploying multiple bots in such a scenario can save the repetition of cleaned areas and the travel time from one point to another.

- **Back-tracking**

  In case the robot finds itself in a local extrema, it needs to backtrack to get to another position. This backtracking may be brief or might be time consuming.

  We can evaluate the extent of backtracking and deploy a bot in order to avoid having to do that.
  When a situation of extreme backtracking occurs, we can deploy another bot at the new place instead of having to backtrack.

- **Traveling**

  As seen in the Final Robot Path Example, in the latter part of the path in image 1, we can see that in order to resume it’s cleaning after having dealt with the obstacles, the robot travels back to its intended position before encountering the obstacle.
  In case of large obstacles, this travel time can be significant. Thus as as soon as the obstacle crosses a certain threshold, a new bot can be instantly deployed at the intended position.

  Once the initial bot is done covering the obstacle , it can wait in order to reach another intended position if bot2 faces a large obstacle. This can be carried out efficiently with 5 bots by cyclic allocation of intended positions.

  For better efficiency, if more than one bots are idle, and an intended position opens up, the bot nearest to it (path-distance, not euclidean), will be assigned to travel to that position and continue the cleaning.

2. Epsilon Star

In any Template based Algorithm, there might be back-tracking and the need for the robot to go back to another location (for complete coverage).

Deploying multiple bots in such a scenario can save the repetition of cleaned areas and the travel time from one point to another.

Using the same approach as above, we can keep the idle robots on stand by and deploy them into the map when needed.