AIITRA 2021 Challenge
Team Vox

Vox Bot

Index

Introduction

Problem Statement

Design of the Robot

- **Control and Dynamics**
  - Explain the omnidrive control
  - Explain the odometry equations briefly
  - Explain the terms of the base and world frame of reference
- **Vacuum suction**
  - Talk about the airflow
  - Demonstrate or quantify the quality of airflow
  - Describe the limitations of the design idea in a real-world situation
- **Onboard sensors**
  - Deployment details of the odometry and the sensors.

Multi-Agent algorithm

- Complete Outline
- RRT exploration
- Coverage control problem
  - Fortuners algorithm for Voronoi Diagram
  - Weighted Centroid shifts
  - pyBox2D based visualizer
- The full coverage path planner
- local planner using lidar
- Base Controller

Optimality of the multi-agent algorithm

- The test and predictions

Introduction

With the development and active research in multi-agent systems, we are able to achieve a high amount of efficiency in distributing tasks into multiple groups of tasks for executing them parallelly with the help of multiple agents. The major challenges in these systems are the increase in complexity for autonomously controlling them and cooperation between individual agents. These tasks need a high amount of planning and coordination between agents for generating feasible trajectories and motions.

Vacuum Cleaners are one of the most highly researched instruments throughout the 20th century. A tool with rather a simple principle has many indigenous design decisions for it to work with the desired efficiency, and control. The major difficulty in developing such a technology is the reduction of interference of such a high-speed air current with the motion of the vacuum cleaner and efficiently transport with the desired motion.

Problem Statement

In the given problem statement we are asked to design a vacuum cleaner-based mobile robot, and use a swarm of these robots to effectively cover more than 90 % of the terrain.
A few key points from the problem statement are as follows:

- There are different maps with obstacles and irregular walls for the robot to travel around.
- We have to keep in mind the efficiency of the vacuum cleaner and its interference with the robot's motion (like slipping, toppling, etc.)
- Leveraging the use of multiple robots efficiently as the robots with respect to the maps are 100 times smaller in size.

Hence our team came up with a solution for this problem statement which we would describe in this proposal.

After analyzing the Problem statement and making a few assumptions which we have enumerated below we come up with our Robot platform "*Vox-Bot*".

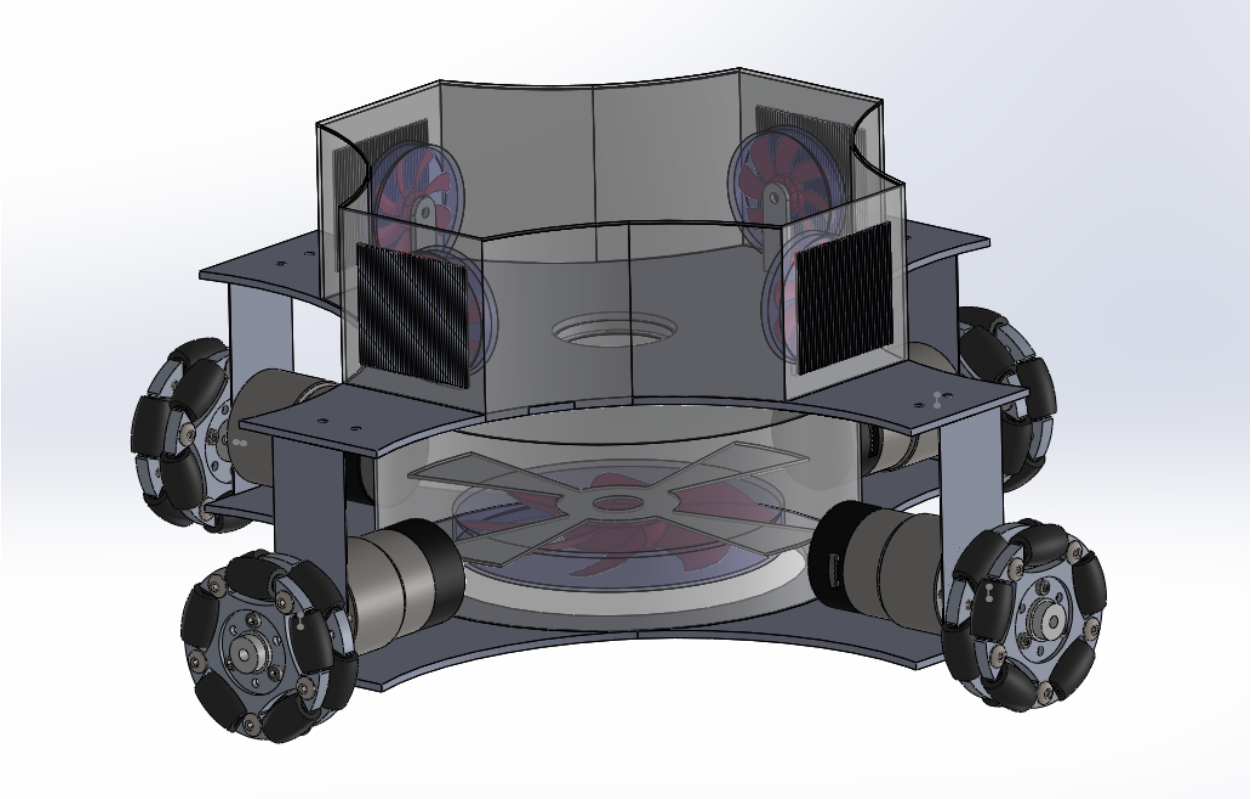
Assumptions:

- We are able to deploy the robots into the arena from different locations for maximizing the amount of exploration through multiple robots.

- We can’t use any prior knowledge about the maps, hence we would have to map the unknown terrain throughout all the arenas of the Problem statements.
- All the terrains we would test against would form a closed loop in terms of walls or any such boundaries.

Design of the Robot

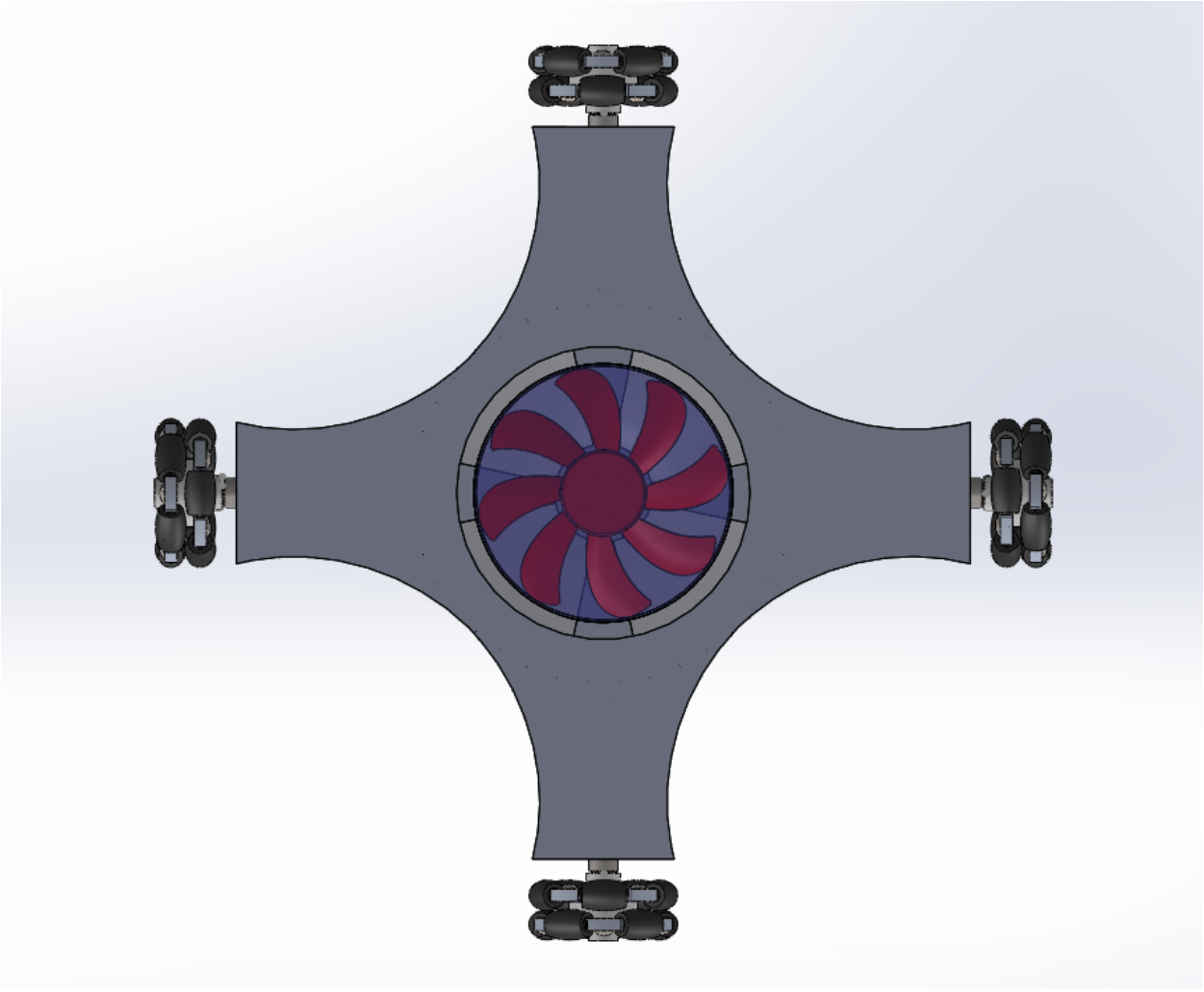
The Voxbot is a four-wheeled omnidrive bot with a built-in vacuum mechanism for cleaning purposes. The wheels are arranged in a plus configuration as is customary for a four-wheeled omnidrive robot.



The bot additionally consists of 5 fans. The bigger bottom one is used to create the suction for the vacuum while the top 4 act as exhausts to facilitate the creation of the vacuum. The four exhausts make sure that the vacuum consequently created is strong enough to pick-off dust particles. Some space is left in the top compartment for a dust bag.

Stability

The Voxbot has been designed to be extremely stable during sharp turns which it is completely capable of taking, owing to its Omni-wheel-based design. The much larger footprint of approx 0.35m*0.35m of the vox compared to its 0.15m approx height keeps its center of mass much closer to the ground thus eliminating the chance of toppling. Omni-wheels arranged in a plus configuration make sure that the slipping and skidding are next to negligible while traversing the map.
The exhausts have been placed symmetrically in 4 directions so that the same amount of air is released through each one, thus removing any effect of force or torque provided by the releasing air. This fact addresses an important problem in such systems of unequal force exertions on the bot making it unstable.

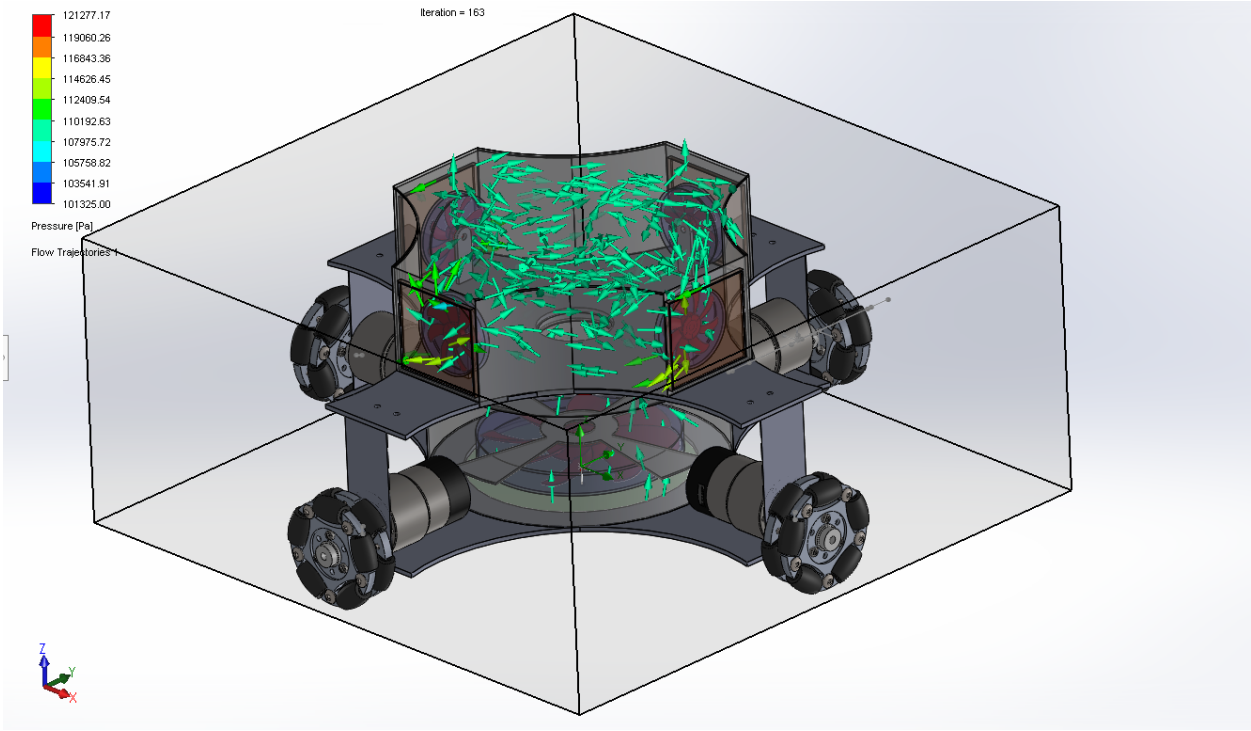


Suction System

The vacuum of the bot works on the principle of the lower fan creating pressure difference to suck in air while the exhaust pushes out the air from the above compartment for efficient vacuum generation.
The fan has been placed low for efficient cleaning, with a ground clearance measuring approximately less than half of the wheel radius.

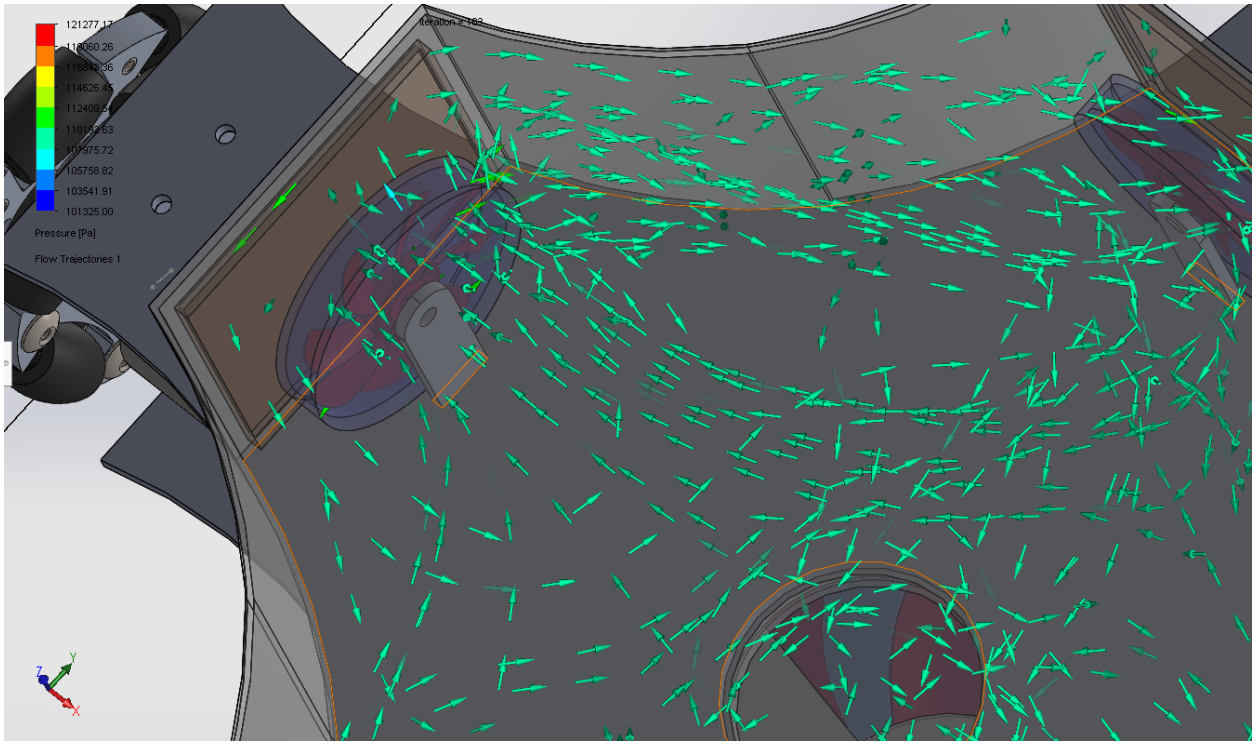
Analysis of the vacuum mechanism was done using the SolidWorks Flow Simulation tool to get outputs about the kind of behavior shown by our vacuum during actual implementation.

The simulation required us to cover our rotating regions with circular bounded bodies to define the rotation boundary. We also defined the **inlet** and the outlet velocities as 0.6m/s and 0.15m/s below the bot and at the exhausts respectively. As the simulation was an internal one, the image only shows the flow inside our bot but the fact is quite evident through the trajectory of the arrows that in real-world scenarios, vox would certainly be an efficient vacuum design.



The zoomed-in image of the flow clearly depicts that our exhausts are also working efficiently during closed space tests as the simulation required us to close the lower

open space of our bot with a lid. The fans are generating the expected and the required flows quite efficiently.



Velocity control for 4 wheel omnidrive

We have used a velocity controller directly from ROS control Package to make our Robot mobile. Ros control acts as a plugin for the wheel joint to receive the velocity commands as directed by the base controller. The controller has been implemented assisted by the following Kinematics model :

This is an image that contains all necessary geometric constraints to derive the kinematics equations:

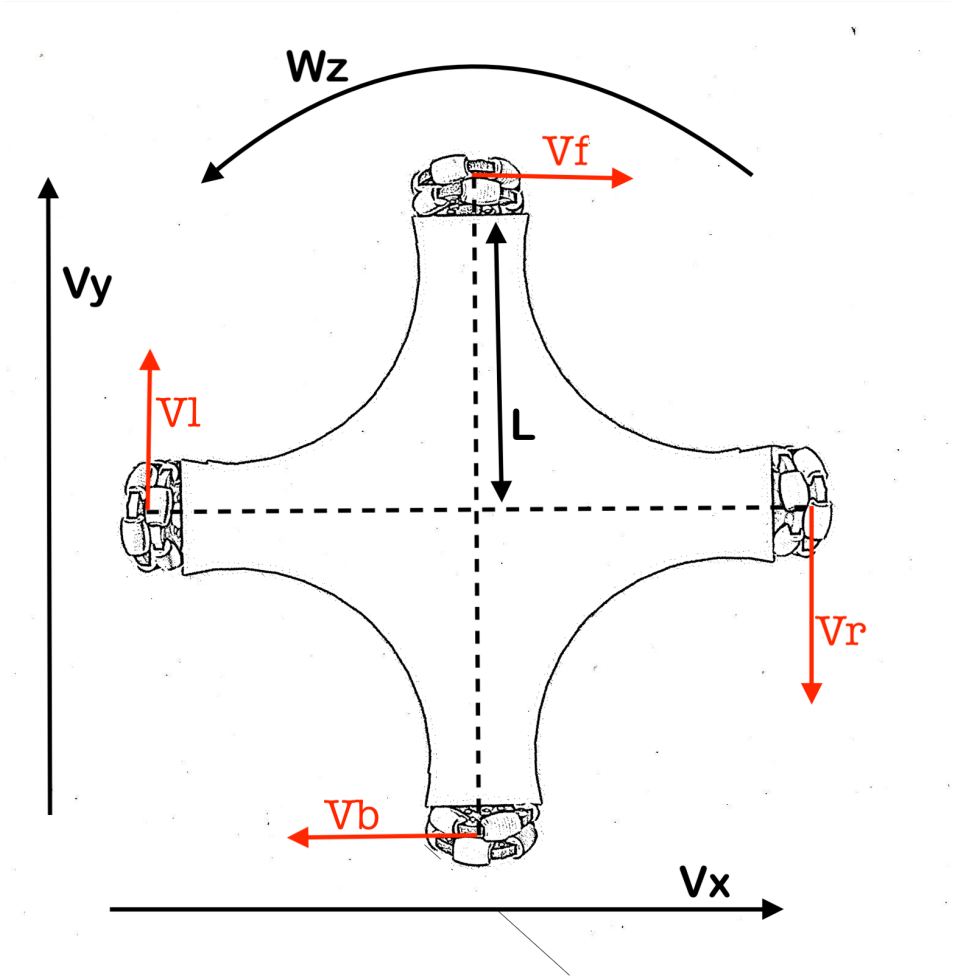


Figure 1: Base Coordinates and State inputs

Here Vx, Vy, Wz are the velocity in the x-axis, velocity in the y-axis, Angular Velocity in the z-axis respectively, and Vl, Vf, Vr, and Vb are the velocity of wheels. Here the Inverse kinematics equations relative to the robot's frame which is used for controls are given below:

$$\begin{pmatrix} V_l \\ V_f \\ V_r \\ V_b \end{pmatrix} = \begin{pmatrix} 1 & 0 & L \\ 0 & 1 & L \\ -1 & 0 & L \\ 0 & -1 & L \end{pmatrix} \begin{pmatrix} Vy \\ Vx \\ Wz \end{pmatrix}$$

Odometry

$$\begin{pmatrix} V_x \\ V_y \\ \omega_z \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ \frac{1}{L} & 0 & \frac{1}{L} & 0 \end{pmatrix} \begin{pmatrix} V_l \\ V_f \\ V_r \\ V_b \end{pmatrix}$$

Above are the Forward Kinematics equations, which represent the relation between the robot's linear and angular velocities and individual wheel velocities.

$$V_x^w = \cos(\theta)V_x - \sin(\theta)V_y$$
$$V_y^w = \sin(\theta)V_x + \cos(\theta)V_y$$

Here is the relation between velocities in the robot's frame and velocities in the world frame, where, V_x^w and V_y^w are the robot's velocities in x and y direction respectively, in the world frame.

Onboard Sensors

We have used GPU Laser LiDAR to determine the obstacles. LiDAR is deployed on sensor_link in such a way that LiDAR is parallel to the robot base_link. Specifications of LiDAR are:

- It covers Π angle around the robot with a minimum range of 0.1 meters to a maximum range of 30 meters.
- The range resolution of LiDAR is 0.1 m per nsec

We have used an IMU sensor to determine the orientation of the robot using the HECTOR_IMU_ROS plugin. This data is also used to calculate the odometry. IMU sensor is attached as the imu_link which is parallel to the base of the robot facing Wheel F.

We have used joint encoders in theory basically the ros state publisher, which publishes the position, velocity, and the effort of all the joints. These readings are used by the base controller and the odometry node.

Multi-Agent Algorithm

As we are instructed to use ROS we would be implementing the complete navigation stack as this would ease our process and help us solve the problem efficiently. Below we enumerate the parts of the navigation stack that we plan to implement and use.

- Global Planner
- *Local Planner**
- *Map Server**
- Odometry: For 4 wheeled Omni Drive
- *AMCL**
- Controller: Velocity controllers for 4 Wheeled Omni Drive

(* These are majorly pre implemented and only require fine-tuning for our platform)

The Navigation stack for an individual robot is independent of all other robots in the system, except for the Global Planner as its objective depends on Multi-Agent cooperation and motion planning to execute tasks. Hence we have discussed multiple phases in the global planner in detail, while the rest of the architecture and working of the navigation stack remains the same throughout all the phases.

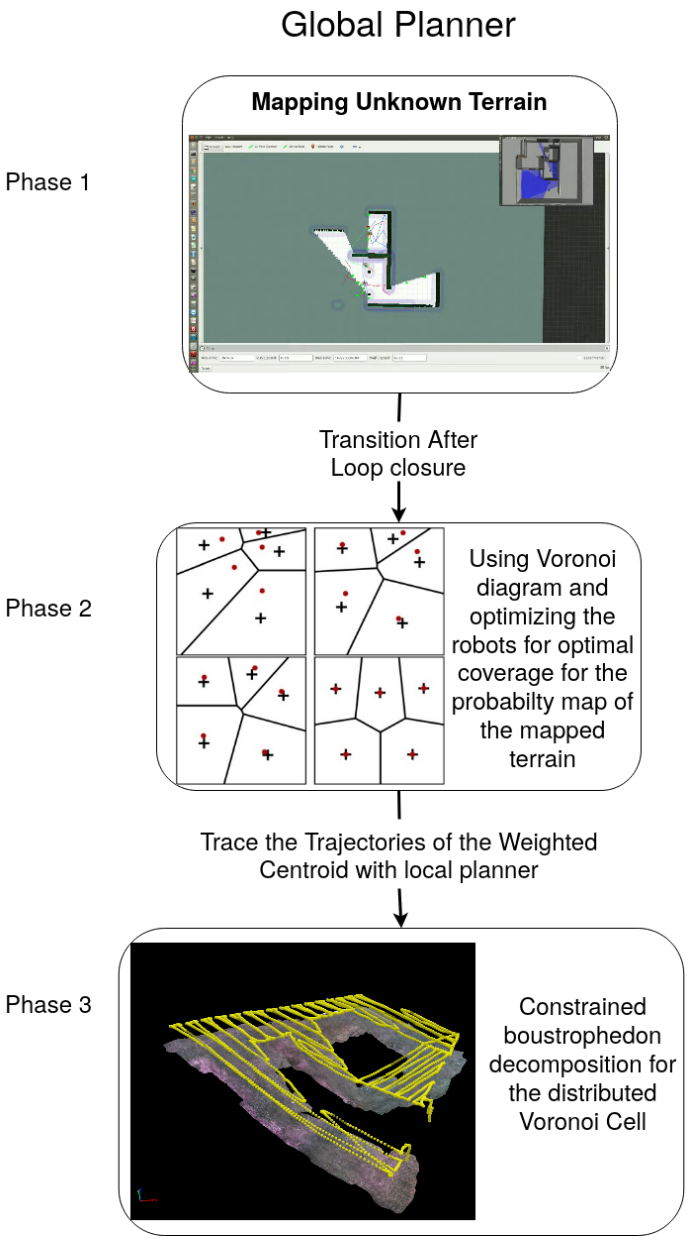


Figure 1: Phases in Global planner

Global Planner

The global planner is responsible for where the robots would move in accordance with other robots in the system. Hence we plan or optimize trajectories for all the robots to coordinate with each other. We split our problem statement into 3 phases, i.e.,

1. Mapping the unknown terrain
2. Splitting the terrain into equal Voronoi cells
3. Complete coverage (boustrophedon coverage) of individual cells.

For accomplishing these phases individually we have implemented/reused global planner packages. These different global planners can be seen in *figure 1*. Whose individual parts we would be discussing briefly

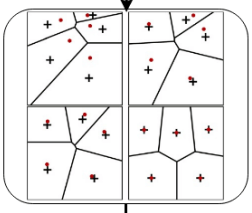
Mapping the Unknown Terrain

1.

GIF

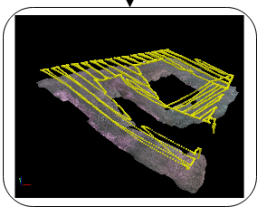
Note:-
We are mapping the world assuming we can't use a pre-mapped map or use any prior hardcoded definition of the maps in the given Problem statement. Rather if we could use the map this is an **unnecessary phase** and we could directly skip to the next phase saving a lot of exploration time.

2.



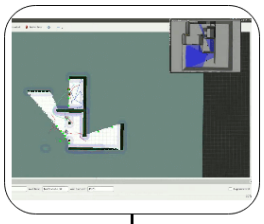
For mapping the unknown environment we are leveraging the fact that we can map it much faster using multiple robots in a given environment. We are using **RRT** (**Rapidly Exploring Random Tree**) for global path planning mainly for maximum exploration of the terrain. We are using a pre-implemented package named rrt_exploration, This package contains a modified RRT algorithm for exploring using multiple robots. You can see the output in the above gif where multiple robots are mapping the given terrain.

3.

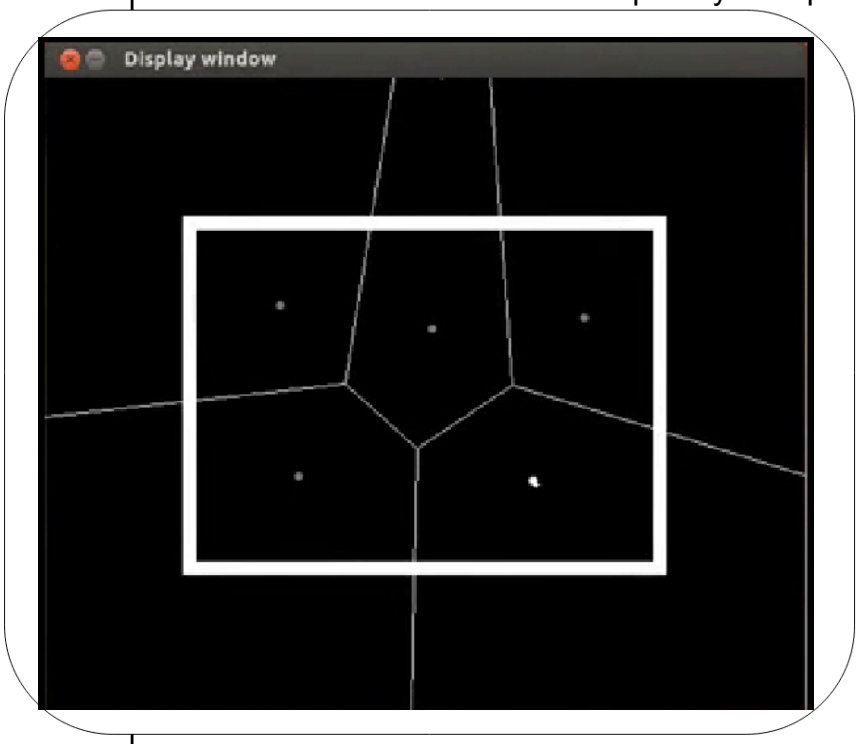


This phase runs till the complete map loop closure condition is not satisfied. This is assuming the surface of the environment always forms a closed loop, the algorithm stops once it can't find any open frontier in the current map denoting all the traversable parts of the arena are mapped, and no more global frontiers are found.

Optimal Coverage Control

1. 

Once we have the *occupancy map* of the current arena we convert it into a probabilistic model of uniform distribution in regions where the robots would have to cover the area and sweep through using the Vacuum Suction. This region with uniform probability distribution would be the continuous region in the occupancy map in which all the robots are enclosed, the remainder region would have a zero probability. Here it is a uniform distribution as all the area has equal probability else one could choose to model their priority as a poison distribution to use it.

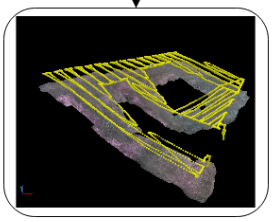
2. 

We can use this probability map to optimize the simple Coverage Control problem. As all the robots have equal ability to clean the Map it is logical to split the arena into convenient subsections for each robot to sweep through.

This is a one-time trajectory optimization algorithm and we would only be taking the trajectory for each agent from the agent's initial location to the final converged centroids of the Voronoi Cell. The above is Voronoi Diagram which we would compute using **Fortuner's Algorithm** and run the optimization till it converges.

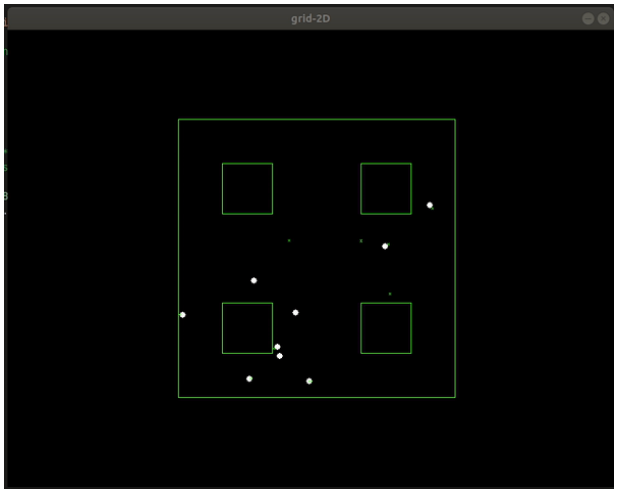
GIF

Gif

3. 

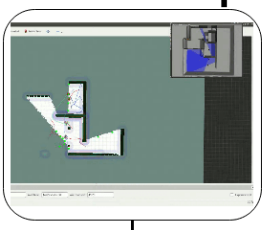
We would be updating the position of the agent using a weighted centroid method to reach the convergent point. These trajectories would be published to be traced using a local planner.

We would be implementing this from scratch so for ease of use we would be using pyBox2D testing and debugging the algorithm, whilst this would be a

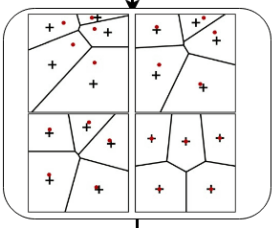


simple library for optimization. To the right, you can see the pyBox2D environment we have set up for the same.

Complete Coverage or boustrophedon Coverage

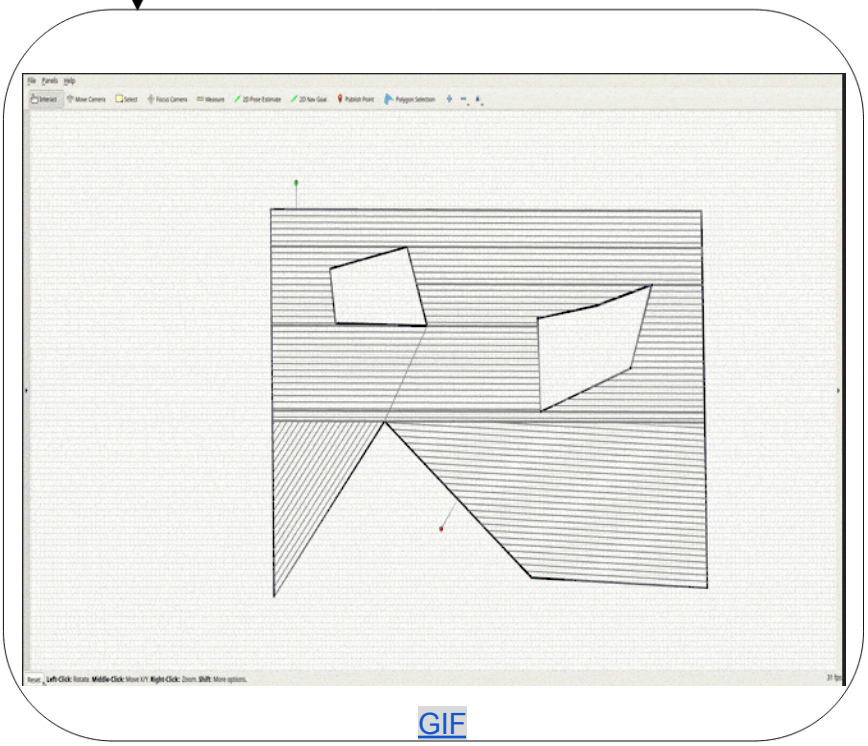
1. 

After equal distribution of the total area, we have to run each individual Robot on their respective Voronoi Cell which is a simple Complete coverage or Boustrophedon Coverage problem. A uniform coverage in a constrained segment is desired as we would not want our robots to get into segments of other robots.

2. 

We would be having a polygon of the Voronoi cell of each robot and we would have polygons of other obstacles enclosed within them. Having this information will help use the polygon_coverage_planning package for the complete coverage.

In the below illustration, we can see that the planner is able to traverse on a constrained plot of land. Which in our case would be the Voronoi Cell of each robot.

3. 

GIF

Local Planner

Throughout the complete run of our solution we are using teb_local_planner for each individual robot, as the density of robots per sq.ft. is much lower hence sharing a command local planner doesn't benefit any computation. We are using teb_local_planner, integrated with our navigation stack with the LiDAR for avoiding any dynamic obstacle like other robots and unmapped terrain.

Map Server

We are using the official gmapping map server and the use of multirobot_map_merge to merge maps from different robots, This map is later used in many formats like probability maps, occupancy grid, etc. for which gmapping servers had utility APIs pre-implemented.

Localization

For localization, we have used the joint states for odometry and AMCL with the laser scan for localization with very high accuracy. The description of our odometry is given in the previous section. We have implemented odometry and base controller in different nodes where their working and equations are given in the control section.

Optimality of the multi-agent algorithm

With a limited amount of setup and work done we were not able to complete our algorithm before the proposal as it was not expected. Though we have run a satisfactory amount of simulations on the 2D environment that we showed earlier, this doesn't model the kino-dynamics of individual robots but rather helps us understand the motion planning and the interaction of multiple robots in the system. Using this we got the robots to traverse the whole map and found out the optimal number of robots to cover maximal of the terrain vs time graph was a hockey stick curve reduced to change out after **7 robots**. There may be a lot of difference between the rough estimate of 7 robots to be optimal compared to the actual simulation run as there is no local planner nor any mapping using RRT exploration. We had only implemented a simple boustrophedon path planner for individual robots in their respective Voronoi Cells. We do not have any concrete time of completion yet.