Multi-Agent Mapping

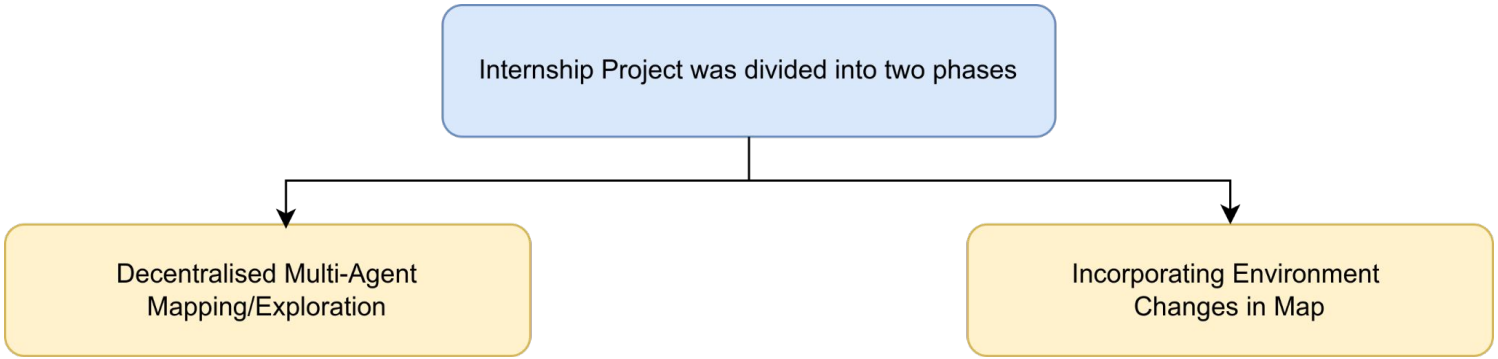
Dynamic Map Updates

ROBERT BOSCH CENTRE
for CYBER-PHYSICAL SYSTEMS
Indian Institute of Science, Bangalore-560012, India

ARTPARK
AI & Robotics Technology Park

- Aryaman Gupta
IIT (BHU) Varanasi

Task

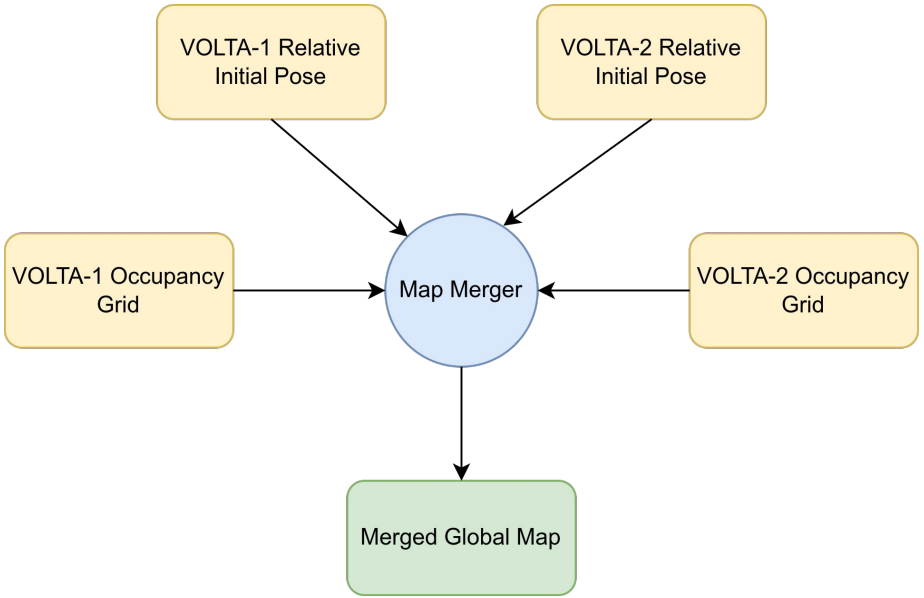


- Implemented map merging on maps created using Lidar-based SLAM by multiple VOLTA ground robots for manual collaborative mapping.
- Tried to make it autonomous using RRT-Based Exploration.

- Implemented 3D object detection to extract real-world position and dimensions of obstacles through infrastructure cameras.
- Built a ROS plugin for performing dynamic updates in global costmap and static map for enhanced navigation.

Manual Multi-Robot Mapping

- Each individual robot prepare its own map using LIDAR as the primary sensor with default ROS SLAM Gmapping package.
- The individual maps are fed to map merger which stitches them based on feature-matching in the global frame to prepare combined map with/without prior information of robot's initial positions.

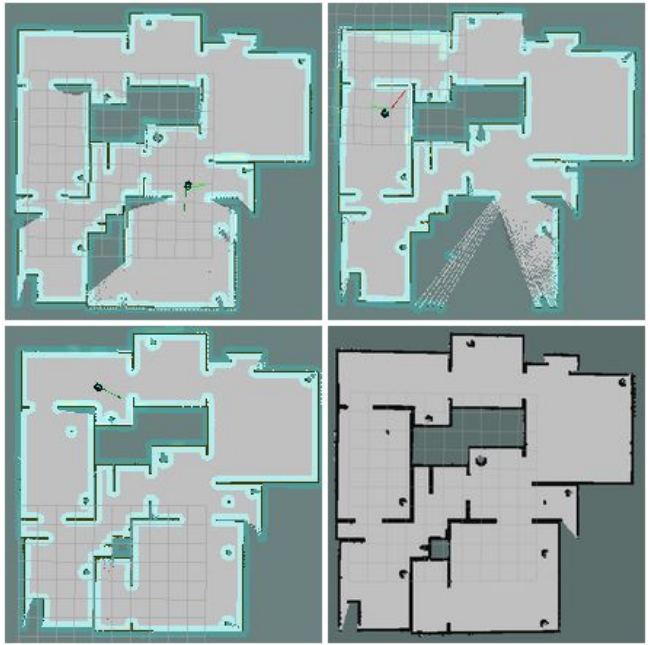


Map Merging

- An efficient algorithm which can work with heterogeneous multi-robot swarms and can be easily deployed with various SLAM algorithms.

- Main features of algorithm are:
  - Dynamic Robot Discovery
  - Initial Pose Estimation and Frame Transformation
  - Map Composition

- Work in both possible situations:
  - Known robots' relative initial position
  - Unknown robots' relative initial position

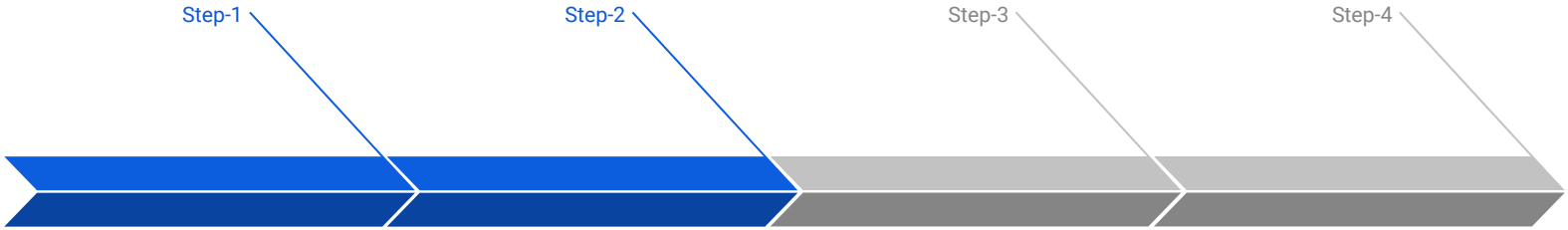


Map Merging Situations

Known Initial Positions	Unknown Initial Positions
• Transformations between grids are directly computed using known relative position of grid's origin in global frame. • No overlapping between grids is required. • Robots can be placed in decentralised manner. • Computationally efficient. • Produced map is close to perfect.	• Transformations between grids has to be computed using feature matching. • Overlapping is required between grids for efficient feature matching. • Robots can't be placed in decentralised manner. • Computationally expensive • Produced map may suffer from incorrect merging due to wrong transformations.

Demonstration on VOLTA Robots

Map Stitching Pipeline



Feature Detection

Oriented FAST and Rotated BRIEF (ORB) feature detection using OpenCV

Pairwise Matching

Pairwise feature matching for each grid

Finding Largest Detected Component

Filter out matches to get largest connected component of grids

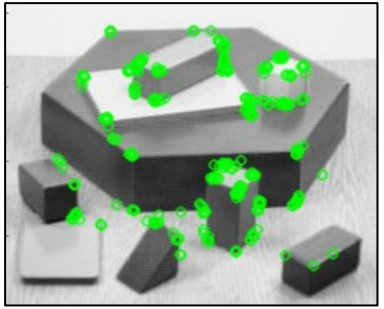
Final Transform Estimation

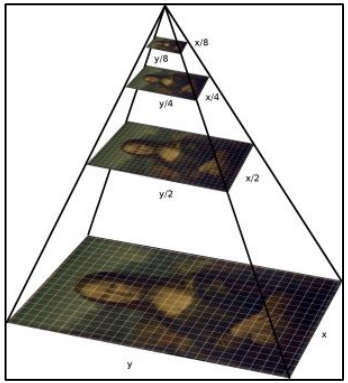
Calculating transform with global reference frame

Step-1 : Feature Detection

- **Oriented FAST and Rotated BRIEF (ORB) Feature Detection** algorithm is used because:
  - Patent free
  - Available in OpenCV
  - Already available implementations with occupancy grids

- Occupancy grids are not required to be downscaled (unlike images) because they are less complex and contains comparatively less features than multi-megapixel images.





Step-2 : Feature Matching

- Pairwise feature matching is the most computationally expensive part in this algorithm.
- Simple Brute-Force search is used for all possible grid pairs with $O(n^2)$ time complexity.
- Parallel Hierarchical Clustering Trees is used with binary ORB features to speed-up search on CPU using OpenCV.

- For each matching grid pair, 4 DOF reduced affine transformation is calculated as:

$$A'X = Y$$

$$\begin{pmatrix} \cos(\theta)s & -\sin(\theta)s & tx \\ \sin(\theta)s & \cos(\theta)s & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ 1 \end{pmatrix}$$

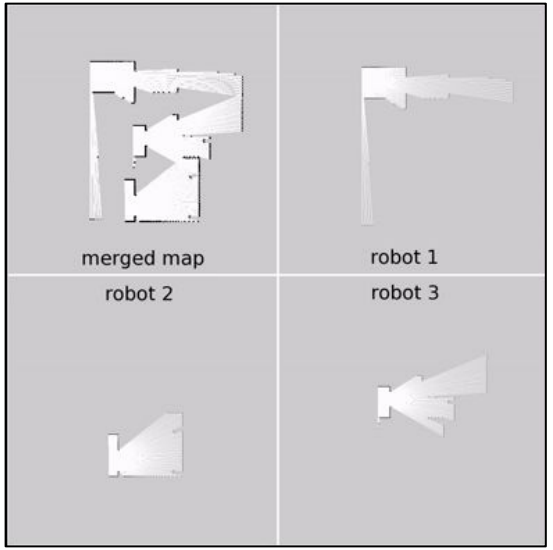
where X and Y are defined as

$$X = (x_1, x_2, 1)^\top,\ Y = (y_1, y_2, 1)^\top$$

- This A' matrix accounts for rotation, translation and scaling between grids.

Final Step : Estimating Final Transforms

- After feature matching, we filter out matches by putting a confidence threshold.
- Instead of taking all components, we only take largest connected component to preserve the topological accuracy of map.
- Largest connected component is generally the one which covers the largest area.
- At the end, we declare one grid as the global frame and estimate its transform with each other grid and start merging.
- The cost of each known cell lies in [0, 100] and arithmetic mean, median or extremal values are considered while merging.



Overall Algorithm

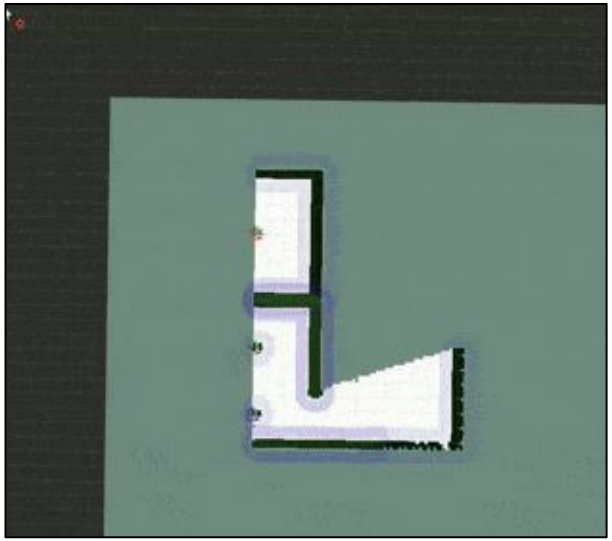
Input: k occupancy grids
Output: for each grid: transformation between grid and global reference frame, or value indicating transformation could not be be estimated for current grid

procedure ESTIMATEGRIDTRANSFORM($grids$)
  detect ORB features (keypoints) for each grid
  for all (i, j) pair of grids **do** ▷ compute transform for each pair
    match features
    $n \leftarrow$ number of matches
    if $n \leq$ matches threshold **then**
      confidence $\leftarrow 0$
    else
      try find restricted affine transformation for features with RANSAC
      $\psi \leftarrow$ number of inliers in RANSAC
      if transformation found **then**
        confidence $\leftarrow \frac{\psi}{8+0.3 \cdot n}$
        $P_{(i,j)} \leftarrow$ restricted affine transformation
      else
        confidence $\leftarrow 0$
      end if
    end if
  end for
  matches $\leftarrow (i, j)$ for matches with confidence ≥ 1.0
  $g \leftarrow (grids, matches)$
  $h \leftarrow$ largest connected component in g
  $t \leftarrow$ maximum spanning tree in h
  ESTIMATEFINALTRANSFORM(t, $P_{(i,j)} \forall e \in$ edges of t) ▷ walk t and compute transformations to global reference frame. See Algorithm 3
end procedure

Multi-Agent Exploration : RRT Exploration

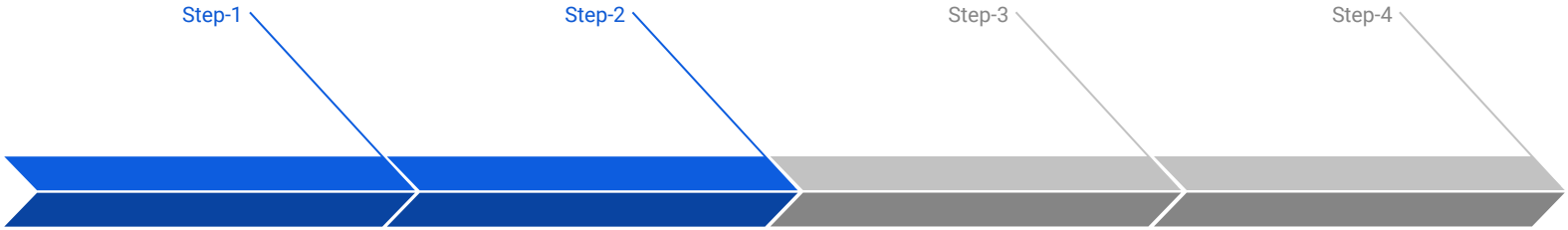
In **RRT (Rapidly Exploring Random Trees) Exploration,** RRT algorithm is used for detecting frontier points in the unmapped region. Here we run two different ROS nodes for exploration:

- Global Frontier Detector: Find frontier points in global map
- Local Frontier Detectors : Find frontier points in each robot's local map. (local node for each robot)



- This algorithm runs until the loop closure condition in global map is not satisfied.
- Here, we assume that the surface of the environment always forms a closed loop and the algorithm stops once it can't find any open frontiers in the current map, denoting that all the traversable parts of the terrain are mapped, and no more global frontiers are found.

Incorporating Dynamic Environment Changes in Map



3D Object Detection

Objectron module to extract 3D bounding box for each object

Coordinates Transformation

Transformation from image plane to real world

Pose and Size Estimation

Estimating object's position and dimensions

Static Map and Costmap Update

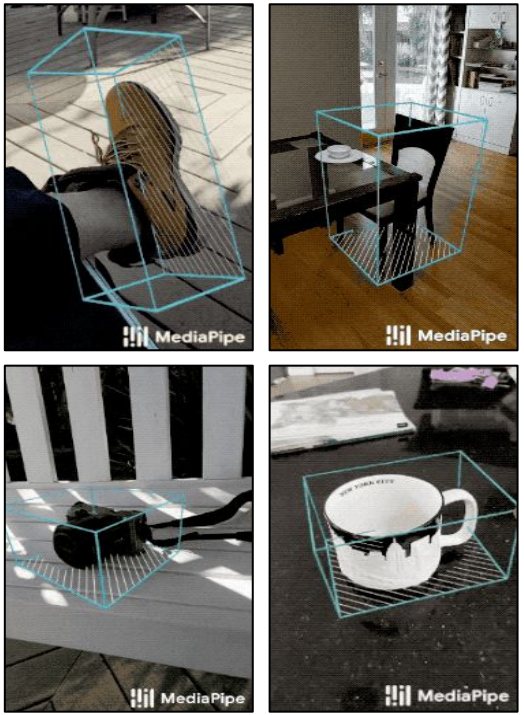
Update costmap through ROS plugin

Objectron : 3D Object Detection

- MediaPipe's Objectron is a mobile real-time 3D object detection API which can detect objects like chairs, shoes, coffee mugs, cameras, etc.

- The objectron dataset consists of 9 classes:
  - Bike
  - Book
  - Bottle
  - Camera
  - Cereal Box
  - Chair
  - Cup
  - Laptop
  - Shoe

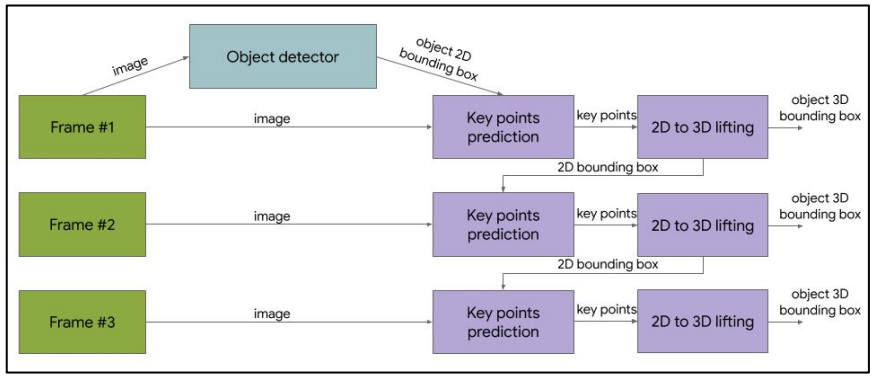
- It uses a **detection + tracking** pipeline to prevent jitter occurring from running detection on each frame and with this approach it can use complex detection models while maintaining decent output FPS.



Objectron Model

Objectron API consists of following 2 ML pipelines:

- Two Stage Pipeline



- Single Stage Pipeline

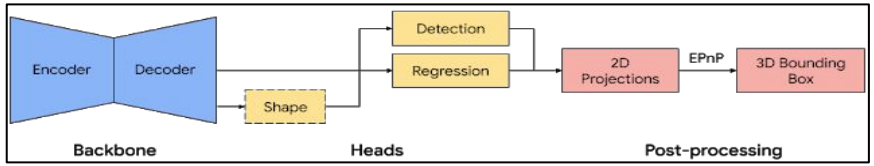
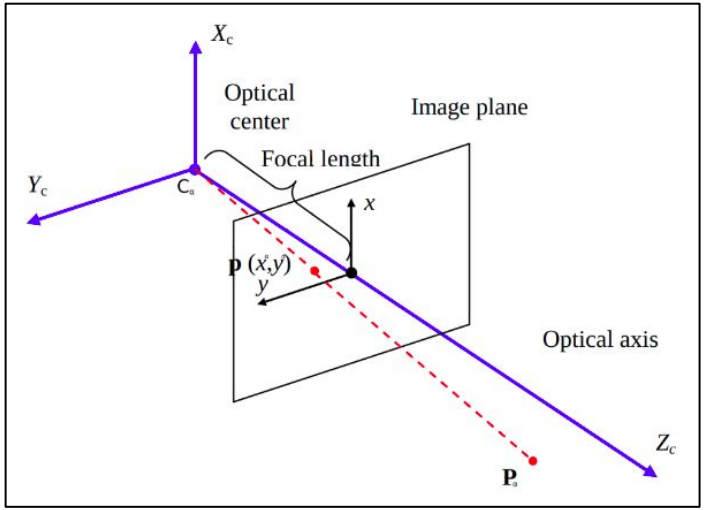


Image Plane to Camera View Frame Transformation

- Objectron returns the coordinates of all 9 points of a bounding box (corners & center) in image plane. We need to transform them to camera's view frame before transforming them to real world.
- In camera view frame, Z-axis is perpendicular to the image and X and Y axes lie in the image plane with origin at camera itself.
- So, we obtain the 3D projection of each point by following:

$$P_\alpha = [Z\frac{x_\alpha}{f_\alpha^x}, Z\frac{y_\alpha}{f_\alpha^y}, Z]$$

where, Z is an arbitrary value and focal length $f_\alpha = (f_\alpha^x, f_\alpha^y)$

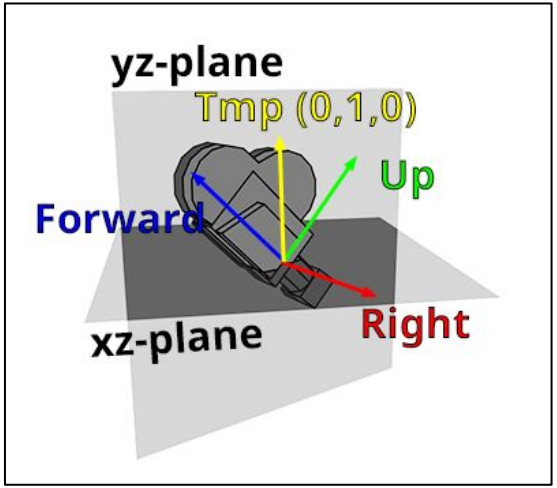


Transformation to Real World

- For transforming coordinates to real world, we use **Look-At Transformation** Method.
- Here we calculate the three camera axes in real world.
- From that, we compute our transformation matrix as follows:

$$\begin{pmatrix} Right_x & Right_y & Right_z & 0 \\ Up_x & Up_y & Up_z & 0 \\ Forward_x & Forward_y & Forward_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

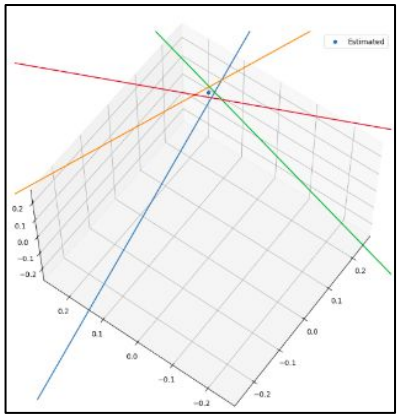
- Using this transformation matrix, we get a real world point corresponding to each of the 9 points.



Look-At Transformation

Optimization

- From the previous transform we get 3D line passing through projected point and camera on which object lies.
- Now, to get the exact location of object, we need to get at least one more line which intersects this line at the same point.
- So, for each obstacle, we use **Multiple Cameras** to get multiple such lines.
- Due to lens distortion, all lines will not intersect at same point and we need to estimate the point with shortest distance to each line.
- For this, we use first order iterative optimization algorithm, **Gradient Descent.**



$$L = \perp_1 + \perp_2 + \perp_3 + \perp_4$$

$$for\ i = 1 : k$$

$$X = X - lr\frac{\partial L}{\partial X}$$

$$Y = Y - lr\frac{\partial L}{\partial Y}$$

$$Z = Z - lr\frac{\partial L}{\partial Z}$$

Static Map Update & Global Costmap Update

- Now, we have extracted the real world coordinates of all 9 points of bounding box. Using this data, we can get the object's location and dimensions in real world.
- Current errors in estimation:
  - Max error in position : 5 cm
  - Max error in size : 40 cm *(only when objects are placed at boundaries of camera's FOV)*
- For updating the static map, we have created a ROS node which updates the cost in occupancy grid and publish it on map topic in real time.
- For updating the global costmap, we have added one more layer to it as a ROS plugin, which constantly receives object's previous and new location and updates both of them in global costmap while running navigation.
- Current thresholds for map update:
  - Distance Threshold : 50 cm
  - Time Threshold : 1 min

Lots of theory?? Got bored??
Its demo time...

Map Update Demo

VOLTA Navigation Test with Costmap Update

Assumptions/Limitations/Scope of Improvement

Assumptions/Limitations:

- We provided robots' initial positions because we didn't get expected results the other way.
- At least two cameras should detect each obstacle.

Scope of Improvements:

- Consider object's orientation while performing map update.
- Use better 3D object detection module which covers more number of objects.
- Optimize number of iterations in gradient descent.
- Debug RRT Exploration module for autonomous exploration with multiple robots.

Literature References

- Jiri Horner's Thesis for Map Merging from Charles University, Prague. [link]
- Hasan Umari's Paper on RRT Exploration published in IROS 2017. [link]
- Article on Look-At Transformation. [link]
- Paper on Dynamic Costmap Update published in CCDC 2017. [link]

Conclusion

I am incredibly grateful to **Dr. Amrutur Bharadwaj** and **Dr. Mukunda Bharatheesha** for giving me an opportunity to work at **Robert Bosch Centre for Cyber Physical Systems, IISc** and **ARTPARK** on such an interesting research-based task for summers of 2022. I want to thank for their continuous **support, guidance, and expertise**. I have witnessed steep growth in the past two months.

I have tried to invest significant time and energy to get the best possible results in all the tasks and have tried to achieve the primary objective of this internship project.