AUTONOMOUS ROBOT EXPLORATION IN ROUGH TERRAINS

Azad Yaşar

***Abstract*—This project aims to simulate a mobile robot's autonomous navigation in an environment containing walls, ditches, low height walls and ramps. The objective is avoiding the robot from paths through ditches and walls, let her move on continuous ramps and ground or calculate an optimal path through discontinuous ramps. All the contents of the project has been built on top of Robot Operating System [7]. To accomplish avoding the robot from obstacles; the 3D map of the environment which is stored as an OcTree has been projected onto a 2D plane. The height has been added to vertices to distinguish between obstacles. Then A* algorithm has been used to find paths from robot's position to the target.**

Keywords*—ROS, robotics, octomap, exploration, autonomous.

I. Introduction

The project is based on a section of a competition called RoboCup–Exploration 5. Within this section the environment contains ditches, walls, low height walls, ramps. Ditches have depths that robot can not climb back if she falls into. Some obstacles are in shape of continuous ramps, wheras the others are of discontinuous type to which climbing is only possible via certain parts. Sensory data from robot's RGB-D camera was the crucial part to cope these difficulties. OctoMap [6] is a map algorithm that stores the environment as an OcTree data structure. It takes *point cloud* from RGB-D camera and the odometry of the robot as inputs.

II. Materials and Methods

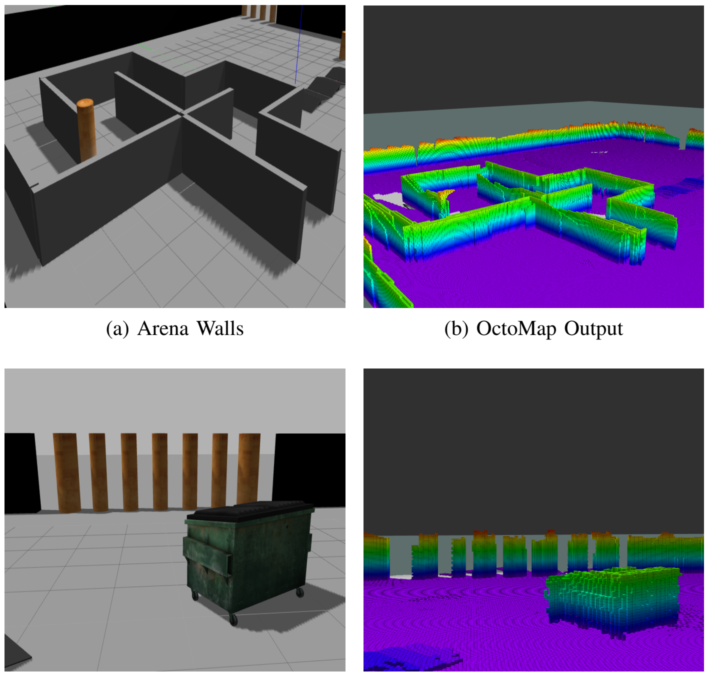
Several modules has been integrated into robot's software to solve this problem e.g., GMAPPING [5], OctoMap [6]. Simulation has been done by means of Gazebo. Visualization of the maps, robot's position, etc. has been done by means of Rviz.

A. Orientation of Sensors

When the robot is going upwards or downwards on a ramp, its roll and pitch values change. When this happens the orientation of the camera and laser sensor also changes which in turn disrupts construction of the OctoMap. A shelf has been mounted to the robot to keep the sensors parallel to the ground. Shelf's orientation depends on robot's shaft. If robot's orientation in Euler angles is $(x, y, z) = (0.2, -0.3, 1)$, then the orientation of the shelf must be $(-x, -y, z) = (-0.2, 0.3, 1)$ i.e., its x and y values are always the opposite of the shaft's.

B. OctoMap

OctoMap is the crucial module to avoid robot from passing through ditches, and discontinuous ramps. The OctoMap module has been integrated into the project. Some outputs of the OctoMap can be seen in Figure 1.


(a) Arena Walls (b) OctoMap Output

(c) Arena walls and obstacles (d) OctoMap Output

Fig. 1: OctoMap Results

C. Publisher–Subscriber Hierarchy in ROS

ROS framework regards processes as nodes and provides them a convenient way to communicate. It deploys a *ROS Master* which is responsible of keeping track of messages that one node might be publishing, whereas others might want to subscibe to. A hierarchy of nodes and messages, that they either publish or subscribe to, can be seen in Figure 2.

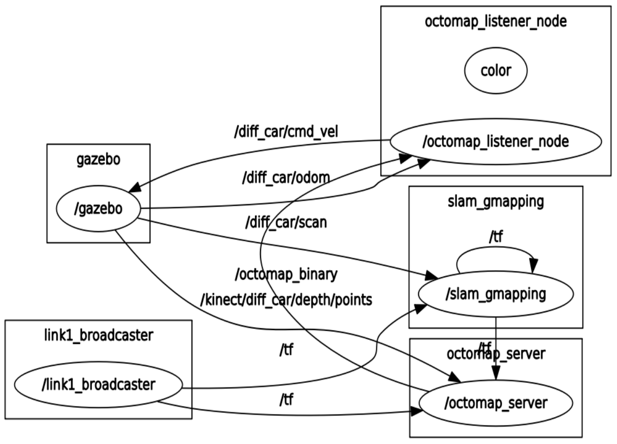


Fig. 2: Node Hieararchy

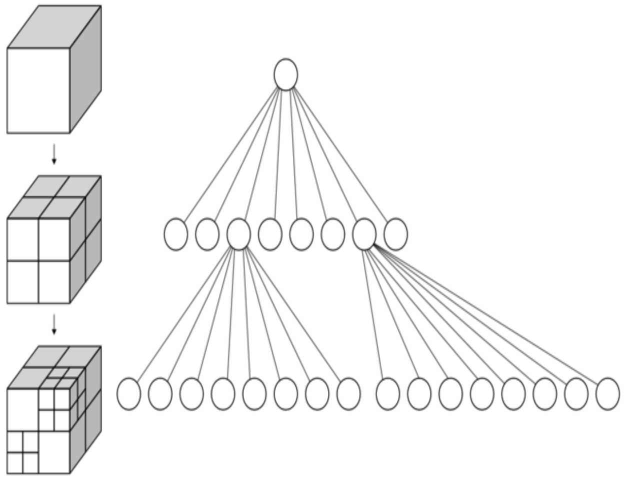


Fig. 3: OcTree Data Structure

D. Using OctoMap

It has been mentioned that OctoMap stores its data as an OcTree which can be seen in Figure 3. Within the OctoMap every cell has the information of the position (x, y, z), and the probability of the occupancy of the cell. To project this 3D structure to 2D plane; one needs to eliminate the third dimension which is z. Since the resolution of the OctoMap is higher than what is needed for robot to move safely, a model has been developed to map the cells to nearest vertex in the graph. The z value of the cells are necessary to distinguish between ramps, ditches and the groud. Therefore every vertex contains a height information.
The vertices will represent a tiny region of the environment. The size of the region is a parameter that can be changed from *.yaml* file. The method of mapping can be seen from Eq. 1.

$$(\textit{round}(0.2), \textit{round}(0.6)) \rightarrow (0, 1) \tag{1}$$

To give a concrete example how the mapping of cells is done, assume the size of the region is $s = 0.25$ and the position of the cell is $(x, y, z) = (0.2, -0.12)$. Then the mapping is done as the following equations..

$$\begin{aligned} k &= \frac{1}{s} \\ &= 4 \\ C &= c(0.2 \times k, -0.12 \times k) \\ &= (0.8, -0.48) \\ C(\textit{round}(0.8)/t, \textit{round}(-0.48/t) &\rightarrow (0.25, 0) \end{aligned} \tag{2}$$

An example of dividing the environment as vertices of a graph is demonstrated in Figure 4.

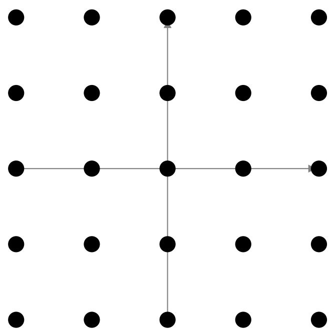


Fig. 4: Graph Vertices

E. Exploration

To explore the environment; robot has to be able to generate targets by itself. To do so, a rescaled version of mapping, which divides the environment to bigger regions, has been integrated (Refer to Figure 5). In the simulation environment, best results has been achieved by dividing the map as 4×4 areas. These vertices will be kept in a priority queue which will sort the vertices based on their distance to robot's position.

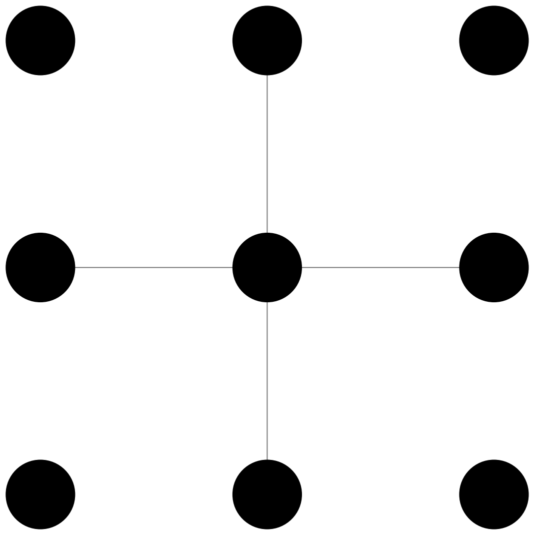


Fig. 5: Regions to explore

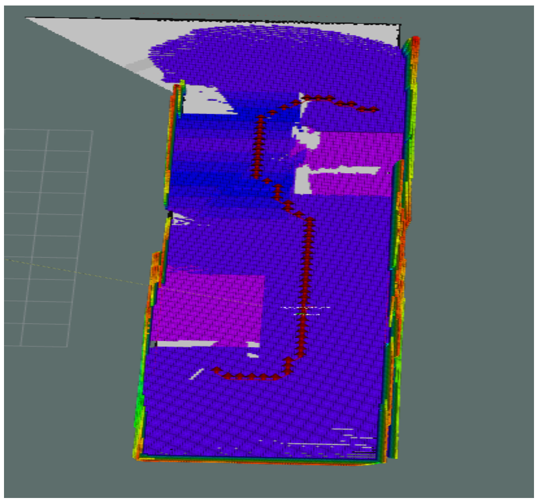
F. Path Planning

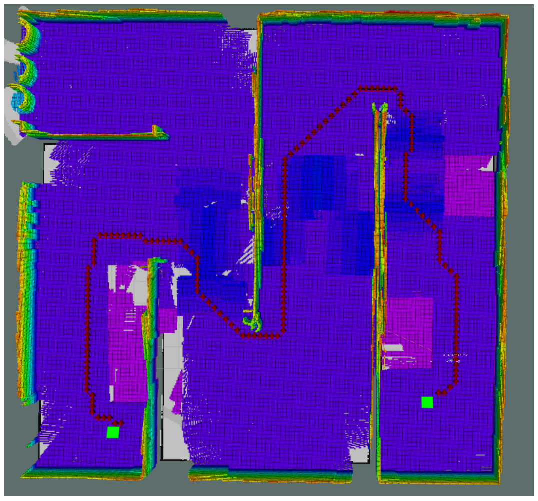
After the graph has been constructed and the robot has selected vertex as a target, it needs to find a safe path. To do so A* algorithm has been used [2], [3]. A* algorithm expects a graph $G = (V, E)$, where V represents vertices, and E represents edges, to find a path from current position to target position.
The secret to its success is that it combines the pieces of information that Dijkstra's algorithm uses (favoring vertices that are close to the starting point) and information that Greedy Best-First-Search uses (favoring vertices that are close to the goal). In the standard terminology used when talking about A*, $g(n)$ represents the exact cost of the path from the starting point to any vertex n, and $h(n)$ represents the heuristic estimated cost from vertex n to the goal. Within this project the height difference $(diff)$ has been added to cost function of A* as follows.

$$f = \ g\_cost \ + \ h\_cost \ + \ a \times diff \qquad \{a | a > 0\} \tag{3}$$

Some results of path planning algorithm can be found at Figure 6.

- **At Fig. (a)**, the path is produced in a way that robot moves safely away from ditches.
- **At Fig. (a)** and **(b)**, also walls are taken into consideration. More importantly robot produces a path that goes through the intersection of two discontinuous ramps.


(a) Between two ditches


(b) From start to end - I

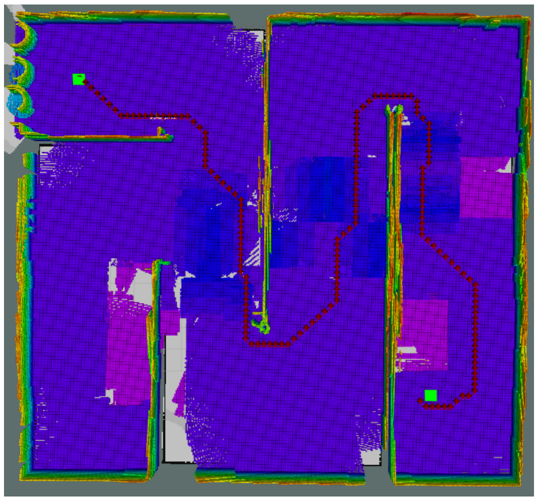

(c) From start to end - II

Fig. 6: Path Planning Results

G. Navigation

ROS framework contains a Navigation library called *move_base* which can be used to navigate the robot to the targer. But in this project, bare-bones version of navigation packet has been designed.
The differential driven robot can be moved by triggering its linear velocity – x or angular velocity z.
To calculate the robot's linear velocity, the distance between the target and the robot's position has been used with scaling.
To calculate the robot's angular velocity, one firstly needs to find the angle between robot's direction and the vector from robot's position to the target position, then the position of the target with respect to robot's position has to be figure out i.e., whether the target lies on the left or right of the robot.

Assume the robot's position and yaw angle are $pos_r = (x, y)$, θ respecively and the target position is $pos_t = (x_t, y_t)$. Note that the yaw angle can be calculated from the robot's quaternion.
Assume that the robot's direction is $dir_r = (x_d, y_d)$ which can be calculated as follows.

$$x_d = \cos(\theta), \quad y_d = \sin(\theta) \tag{4}$$

To find out the angle between the direction of the robot and the vector (dir_t) from robot's position to target, cosine law can be used as in Equation 5.

$$\begin{aligned} \cos(\phi) &= \frac{\vec{dir}_t \cdot \vec{dir}_r}{\|\vec{dir}_t\| \, \|\vec{dir}_r\|} \\ \phi &= \arccos\left(\frac{\vec{dir}_t \cdot \vec{dir}_r}{\|\vec{dir}_t\| \, \|\vec{dir}_r\|}\right) \end{aligned} \tag{5}$$

After finding the angle, one needs to find out whether the robot should turn clockwise or counter-clockwise. To figure out where the target is lying with respect to robot's position. Cross product formulas from Equation 6 and 7 can be used. Considering the complexity and cost of the operations in Eq. 6 and 7, latter has been used.

$$\vec{v_1} \times \vec{v_2} = \|\vec{v_1}\|\|\vec{v_2}\| \sin(\alpha) \tag{6}$$

Cross product as a matrix determinant.

$$\begin{aligned} \vec{v_1} \times \vec{v_2} &= \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ x_1 & y_1 & 0 \\ x_2 & y_2 & 0 \end{vmatrix} \\ \vec{v_1} \times \vec{v_2} &= x_1\hat{i}x_2\hat{i} + x_1\hat{i}y_2\hat{j} + y_1\hat{j}x_1\hat{i} + y_1\hat{j}y_2\hat{j} \\ &= x_1y_2 - y_1x_2 \end{aligned} \tag{7}$$

To give a concrete example, consider following figures and the Equation 8 as calculation.

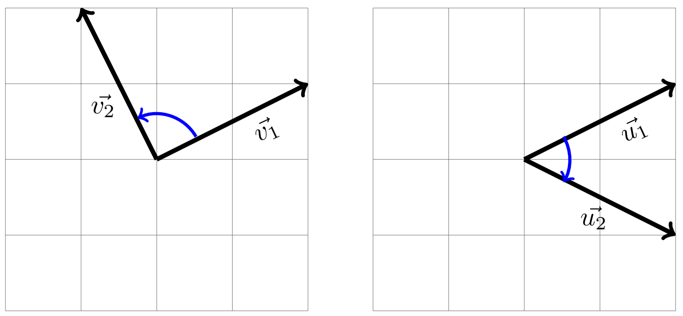


Fig. 7: Target position with respect to robot

$$\begin{aligned} \vec{v_1} \times \vec{v_2} &= 2 \times 2 - 1 \times (-1) \\ &= 5 > 0 \\ \vec{u_1} \times \vec{u_2} &= 2 \times (-1) - 2 \times 1 \\ &= -4 < 0 \end{aligned} \tag{8}$$

III. Conclusion

It can be seen that the 3D map of the environment is a requisite to avoid the robot from falling into ditches. The laser is fixed and it can only see the obstacles that lies on a horizontal plane. However the RGB-D camera is only capable of measuring obstacles that reside at most 5 meters far from robot, whereas the laser can see up to 30 meters. So a convenient and efficient way must be developed to alter this situation. Since after exploring the half of the arena, there are more than one million cells inside the OctoMap, it is crucial to cope the speed of mapping from cells to vertices.

References

[1] Open Source Robotics Foundations SDF Format. 2017, *http://sdformat.org/*

[2] A* Comparison, Stanford CS, 2015. *http://theory.stanford.edu/˜amitp/GameProgramming/AStarComparison.html*

[3] A* Searching, Rajiv Eranki, MIT 2002. *http://web.mit.edu/eranki/www/tutorials/search/*

[4] Van Hulse, Jason and Khoshgoftaar, Taghi M and Napolitano, Amri, "Experimental perspectives on learning from imbalanced data". Proceedings of the 24th international conference on Machine learning 935–942. 2007, ACM.

[5] W. B. Giorgio Grisetti Cyrill Stachniss, "Improved techniques for grid mapping with rao-blackwellized particle filters," IEEE Transactions on Robotics , vol. 23, no. 1, pp. 34–46, 2007.

[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," Autonomous Robots, 2013, Software available at http://octomap.github.com.DOI:10.1007/s10514-012-9321-0.[Online]. Available:http://octomap.github.com.

[7] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in ICRA Workshop on Open Source Software, 2009.

[8] Introduction to A*, Stanford. theory.stanford.edu/˜amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm