Development of a Navigation Stack for Turtlebot3 Robot using A* and Potential Fields path planners

Sai Pranav Adabala
12502523
Dept. of Mechatronic and Cyber-Physical Systems
Deggendorf Institute of Technology, Cham.
sai.adabala@stud.th-deg.de

***Abstract*—Abstract—This paper describes the development of a navigation stack for the turtlebot3 burger robot model with the help of A* global planner and Potential fields local planner. These two path planners are then used as logic to develop a Navigator node which integrates the kinematic controller with these path planners. This paper also discusses the comparison of A* and Dijkstra algorithms for different maps and the parameters tuning is also briefly discussed. Index Terms—A* Global path planner, Dijkstra path planner, Kinematic controller, Turtlebot3, Robot Operating System(ROS).**

***Index Terms*—A* Global path planner, Dijkstra path planner, Kinematic controller, Turtlebot3, ROS.**

I. Introduction

With the increase in demand for supply chain and management, the research in the field of mobile robots and autonomous systems has skyrocketed. Data shows that the global smart logistics market is estimated to increase to US 201.2 billion dollars by 2032, which is currently valued at US 30.6 billion dollars [1]. With such a demand, the navigation systems of the mobile robust must be robust in order to perform the specified task efficiently. With a proper combination of various path planning algorithms, sensor models, and uncertainty models, the navigation stack of these robots can be optimized, hence boosting industrial mobility solutions.

II. Objectives

The main aim of this project is to develop, implement, and validate a navigation stack for a Turtlebot3 robot using ROS and Gazebo environments. This study aims to achieve obstacle detection and navigation from a source point to a goal point. By building a custom map, converting it into an occupancy grid, the path planners are desired to be implemented. Further, a fully functional navigation node is desired to be built with the help of ROS by comparing two different global path planners and two different local path planners. The results are then planned to be visualized with the help of RViz (ROS Visualization) and the Gazebo environment.

III. Methodology

The project approach is divided into six main steps, including system map creation and map saving, development and comparison of global path planners, development and comparison of local path planners, development of the kinematic controller and tuning, and navigation node integrating global and local path planners with the kinematic controller and results.

A. Map Building and Saving using map_server

A robust 2D occupancy grid map is built in the Gazebo simulation environment, where the robot is placed in a workspace surrounded by obstacles. The map is scanned using SLAM (Simultaneous Localization And Mapping) with the help of gmapping technique. The map is then saved using the `map_saver` tool from the `map_server` ROS package, which outputs the map as a `.pgm` image and a corresponding `.yaml` metadata file. This saved map can later be loaded using the `map_server` node to provide static map data for localization and path planning.

B. Global path planning

Here two different global path planners are briefly discussed, tested for the same map and one among A* and Dijkstra is chosen. Further parameters are tuned to provide optimum solution.

C. Local path planning

Here the importance of local path planner is discussed and two path planning techniques are compared namely Potential fields and Dynamic Window Approach(DWA). Further the parameters are tuned in order to ensure smooth robot movement.

D. Design of a kinematic controller

Here the kinematic controller is designed by subscribing to the odometry topic and choosing the controller for Differentail Drive Mobile Robot (DDMR). The rho, alpha, beta parameters are tuned for smooth movement and to avoid overshoot and oscillating behavior of the DDMR.

E. Navigation node to integrate all processes

The Navigation node is built in order to integrate all three functions. The robot initially moves with the help of global planner and kinematic controller. When the robot is at an epsilon ϵ distance away from the obstacle, the navigation node switches global path planner to local i.e Potential fields path planner in order to avoid obstacle.

F. Results

This section shows the comparison of various path planner methods, impact of tuning in the robot motion and path visualization using RViz and Gazebo simulation images.

IV. Project Flowchart

Here the pipeline of this project is explained with he help of a flow chart.

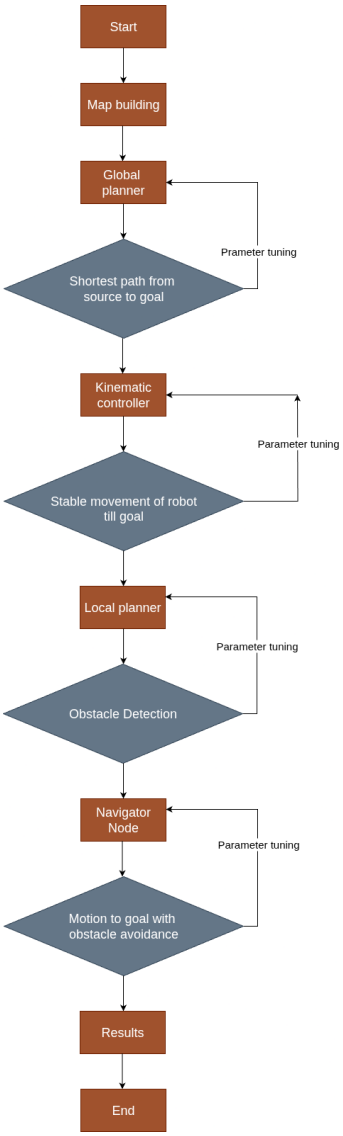

Fig. 1. Project flowchart

V. Conversion of Map into 2D Occupancy Grid

In this section, the conversion of a map in gazebo into a 2D occupancy grid is explained. After building a custom map in Gazebo world, the file is saved as a SDF file in the workspace. After that a launch file is created gicing the directory to the SDF file.

```
<arg name="sdf_file" default="$(find asgn)/map1.sdf"
    />)
```

Later the turtlebot3 SLAM node is ran by giving the following commands which saves the created map as 2D occupancy grid named as map1.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch
    slam_methods:=gmapping
rosrun map_server map_saver -f ~/map1
```

VI. Global Path Planning

In this section the main logic behind two global planning methods are discussed and one among the two was selected for the further progress.

A. Dijkstra's Algorithm

Dijkstra's algorithm is a graph-based shortest path algorithm used to find the minimum cost path between nodes in a weighted graph with non-negative edge weights. It was proposed by Edsger W. Dijkstra in 1956.

- **Input:** A graph with nodes and weighted edges, and a starting node.
- **Output:** The shortest path from the starting node to all other nodes (or to a specific target node).
- **Constraint:** All edge weights must be non-negative.

The algorithm maintains a set of unvisited nodes and assigns tentative distances to all nodes. Initially, the distance to the starting node is set to zero and to infinity for all others. At each step, the node with the smallest tentative distance is selected, and distances to its neighbors are updated accordingly. The process continues until all nodes are visited or the destination is reached.

Dijkstra's algorithm guarantees the optimal shortest path solution in graphs with non-negative weights. Unlike the A* algorithm, it does not use a heuristic function and therefore explores all possible paths uniformly.

B. A Algorithm

A (A-star) is a heuristic-based graph traversal and path search algorithm used to find the shortest path between nodes. It combines features of Dijkstra's algorithm and Best-First Search by using both the actual cost from the start node and an estimated cost to the goal.

- **Input:** A graph with nodes and weighted edges, a start node, a goal node, and a heuristic function $h(n)$.
- **Output:** The shortest path from the start node to the goal node.
- **Constraint:** Edge weights must be non-negative; the heuristic must be admissible (i.e., never overestimates the true cost to the goal).

A maintains a priority queue of nodes to be explored, where the priority is given by the function:

$$f(n) = g(n) + h(n)$$

where:

- $g(n)$ is the actual cost from the start node to the current node n.
- $h(n)$ is the estimated cost from n to the goal (heuristic).
- $f(n)$ is the total estimated cost of the path through n.

A is optimal and complete when the heuristic function is admissible and consistent. It is widely used in robotics and game AI due to its efficiency and accuracy in path finding.

a) Global Path Planner Selection: As we can see in Fig.(3), as it is a simpler map with several square boxes, A* is chosen as the global planner because it uses a heuristic unless like Dijkstra which h no heuristics and hence it tries exploring all unnecessary fields. Here in this case Manhattan heuristic is used as it is a grid based map. Also to reduce computational complexity, A* is selected.

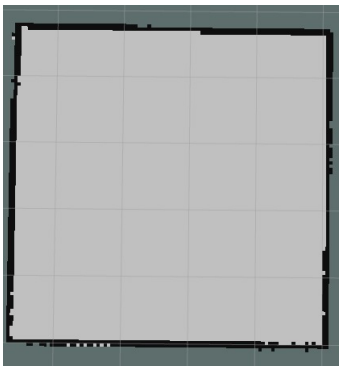

Fig. 2. No path published when there are no obstacles

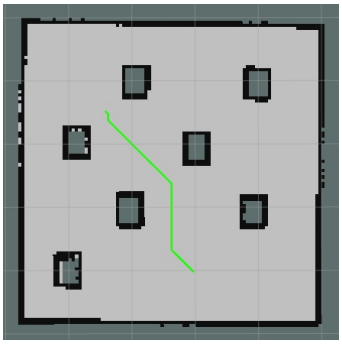

Fig. 3. Path published when obstacles are present

```
<arg name="sdf_file" default="$(find asgn)/map1.sdf"
    />)
```

VII. Kinematic Controller

To orient the robot towards the goal, a controller is required which governs the motion. Here a simple kinematic controller is chosen with Euclidean distance and the parameters are tuned to achieve the desired motion and avoid overshoot.

$$\rho = \sqrt{(x_g - x)^2 + (y_g - y)^2}$$
$$\alpha = \arctan 2(y_g - y, x_g - x) - \theta$$
$$\beta = -\theta - \alpha$$
$$v = k_\rho \cdot \rho$$
$$\omega = k_\alpha \cdot \alpha + k_\beta \cdot \beta$$

Fig. 4. Enter Caption

The robots behavior with the values of rho, alpha and beta as 1.0, 3.0 , 2.0 was not as desired as it has overshoot as shown below, where red dot is the goal pose:

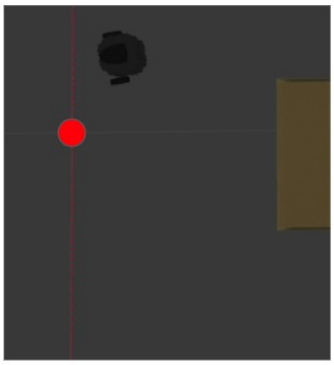

Fig. 5. Oscillatory motion before tuning

The robots behavior with the values of rho, alpha and beta as 0.1, 3.0 , -1.0 was not as desired as it has overshoot as shown below, where red dot is goal pose:

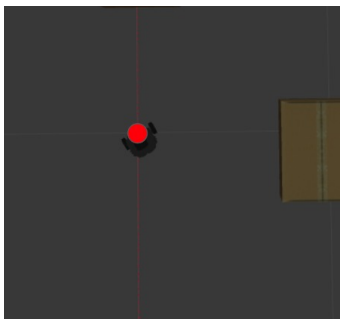

Fig. 6. Goal reached after tuning

VIII. Potential Fields

In this section local planners are briefly explained. Potential fields and Dynamic Window Approach(DWA) are the two local planners explained in this section.

A. Potential Fields

This is an approach where the goal and the obstacles are assigned attractive and repulsive forces respectively—similar to magnets, where the goal attracts the robot and obstacles repel it. The attractive and repulsive gains must be carefully chosen to avoid the local minima problem, where opposing forces cancel out, causing the robot to stall. After

several tuning trials, the following results were observed using: `self.influence_radius = 0.22`, `self.k_att = 0.3`, and `self.k_rep = 1.0`.

B. Dynamic Window Approach

As the map is mostly wide and open, using DWA resulted in sluggish motion. In many frames with no nearby obstacles, the robot often stopped unnecessarily due to the lack of velocity samples satisfying the cost function.

IX. Navigator Node

The Navigator node is a control logic unit that integrates the global planner, local planner, and kinematic controller. When a goal is given in RViz, the Navigator ensures that the robot begins its trajectory using the global planner and kinematic controller. When an obstacle is detected within a threshold of 0.4 m, the Potential Fields local planner activates and guides the robot until it clears the threshold. Each component script is modularly imported into `navigator.py`. For example, the global planner is imported as:

Listing 1. Global planner launched in thread

```
from global_planner import GlobalPlanner
# Launch each part of the navigation stack in
    separate threads
rospy.loginfo("Starting Global Planner...")
gp_thread = threading.Thread(target=GlobalPlanner)
gp_thread.daemon = True
gp_thread.start()
```

With the integration of all scripts, upon running the navigator node and giving inputs in the RViz terminal, the robot is able to successfully navigate to the goal while avoiding obstacles.

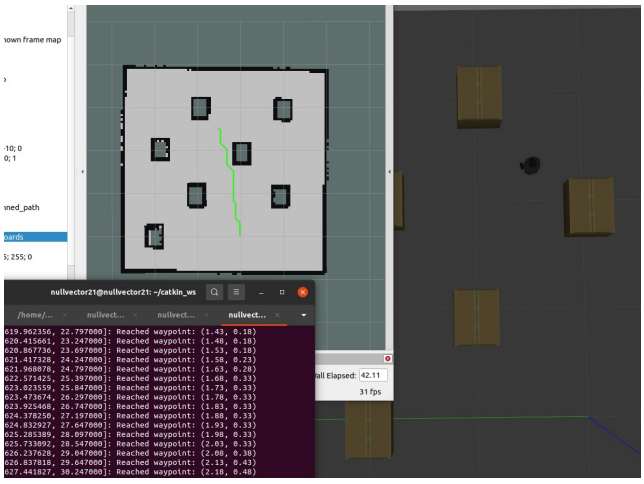

Fig. 7. Goal successfully reached

X. Results

Hence finally the navigation node is built with A* global planner and potential fields local planner with integraiton of kinematic controller. Below the paths for A with two different maps is shown:

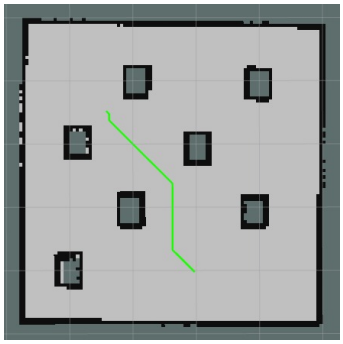

Fig. 8. Aimplementation on map1.yaml world

Fig. 9. Aimplementation on maze.yaml world

The robot is hence successfully navigating from source to goal when the inputs are given live in the RViz terminal. The kinematic gains used are: $K_\rho = 0.1$, $K_\alpha = 3.0$, and $K_\beta = -1.0$.

The gains used in potential fields are: $K_{\text{att}} = 0.3$ and $K_{\text{rep}} = 1.0$.

As the heuristic for A* path planning, the Manhattan distance is used, given by:

$$h = |x_1 - x_2| + |y_1 - y_2|$$

This is appropriate as the map is grid-based and primarily supports 4-connected motion.

XI. Conclusion

This study demonstrated the development of navigation stack for the turtlebot3 robot, selecting the suitable path planners as per the map and developing a kinematic controller and integrating all these with a navigator node.