Pure Pursuit using Adaptive Lookahead Distances

Rishabh Kumar, 2020A7PS1211P
Department of Computer Science and Information Systems
Birla Institute of Technology and Science, Pilani,
f20201211@pilani.bits-pilani.ac.in

***Abstract*—To autonomously navigate and plan interactions in real-world surroundings, robots require the ability to perceive and map complex, unstructured environments. In this paper, we propose a modification of the pure pursuit algorithm, which optimizes navigation speed and minimizes deviation from the path on tight corners. First, we calculate Voronoi graphs for the given map to receive the safest path traversal. Next, we employ a greedy algorithm [1] to compute and assign optimal lookahead distances for the pure-pursuit [2] controller for each waypoint on the Voronoi graph. Finally, we use the Firebird VI robot to evaluate the adaptive pure pursuit algorithm using calculated lookahead distances. Our framework can be found here: https://github.com/k-rishabh/tuw_voronoi_graph.**

I. Introduction

Robots operating autonomously in unstructured, real-world surroundings cannot rely on a preexisting description of the environment space for planning their interactions. They must be able to robustly perceive the complex environment around themselves and be able to derive task-relevant knowledge from it, to guide their subsequent actions. Specifically, robotic vision must be able to reconstruct the 3D objects around them, and dynamically find a path around it, to reach its destination. However, real-world objects and topographies exhibit large variability in appearance, shape, and placement, posing a direct challenge to robotic perception and path planning [3].

Autonomous vehicles (AVs) are responsible for performing complicated tasks, such as exploration, rescue, and surveillance. These tasks often involve driving in complex environments cluttered with numerous small-sized obstacles. When an AV drives in a highly cluttered environment, path planning is a crucial component that ensures traverse safety. In that case, the challenge of path planning is to quickly identify a way that avoids massive obstacles while ensuring that the path is kinematically feasible [4].

This study proposes a heuristic method to solve the above problems. The look-ahead point is selected by considering the geometric relationship between a vehicle and a path. This consists of two steps. In the first step, a look-ahead point is selected on a path by taking into account the position and the direction of the vehicle and the path. In the second step, the look-ahead point, which is obtained from the first step, is moved and located outside the path to address the cuttingcorner problem. This step is particularly effective when the vehicle is near the path and when the curvature abruptly increases, for example, when entering a corner [5].

In this paper, we focus on a robust framework for real-time object detection, dynamic path planning, and efficient navigation with the help of Voronoi graphs and Pure Pursuit.

II. Background

In this section, we introduce the main concepts of Voronoi graphs, tuw_multi_robot, and the Pure Pursuit Algorithm.

A. Voronoi Graphs

A Voronoi diagram is a partition of a plane into regions close to each of a given set of objects. It can be classified also as a tessellation. In the simplest case, these objects are just finitely many points in the plane. In an environment with walls, this collection of points can form a path that is furthest away from all edges – hence forming the safest path. To prioritize safety, we use Voronoi graphs for path planning.

B. tuw_multi_robot

tuw_multi_robot is a framework built by TU Wien robotics, and includes ROS packages to plan routes for multiple robots on a Voronoi search graph. It creates a search graph out of a pixel map and tries to find a path for multiple robots using an extended approach for prioritized planning. The inputs are the tuw_multi_robot_msgs/RobotInfo messages which include the robots pose, the map and the desired goal poses. The output are multiple synchronized routes given to the individual robots.

The figure below represents the current state and planned developments on the tuw_multi_robot framework. The framework is designed to cover all tools needed for an automated delivery system with autonomous vehicles. The current state of the system allows one to set goals for multiple vehicles using RViz, a configuration file, or an order management system which is capable to assign vehicles for specific deliveries and generates goals for the multi robot route planner. The green boxes show already existing modules while the red boxes are not yet implemented/released. The system provides a simple local motion controller for all robots, which allows a high number (greater than 100) of vehicles to be controlled in real time using stage. [6]

The tuw_multi_robot framework contains the tuw_multi_robot_router package which is responsible for routing and path planning. The framework uses a prioritized planning approach to find the robots' routes. Additionally, there are a Priority and a Speed Rescheduler as well as a Collision resolver integrated to solve special scenarios not solvable by standard prioritized planning approaches as shown in the figure below.

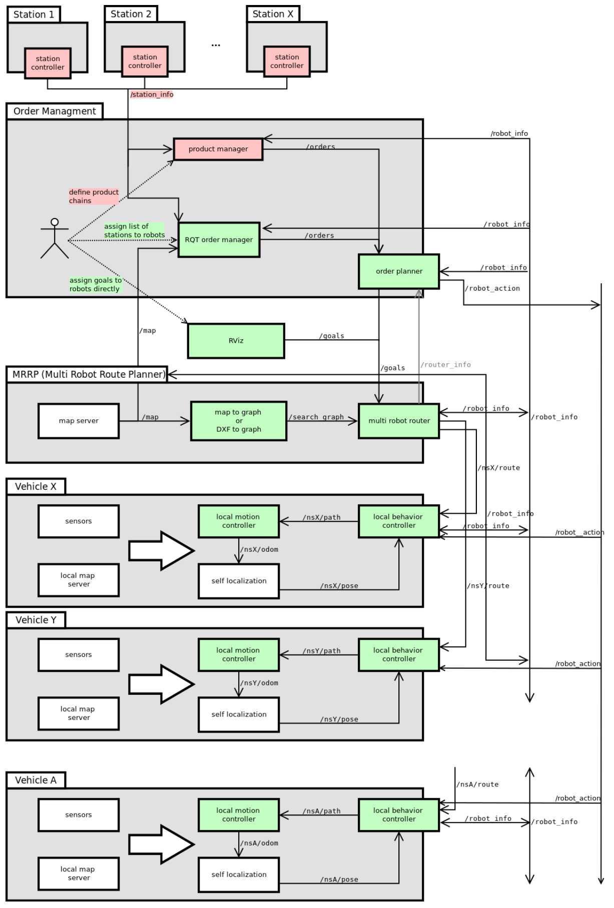
Fig. 1: System overview of the tuw_multi_robot framework.

C. Pure Pursuit

Pure pursuit [2] is a tracking algorithm that works by calculating the curvature that will move a vehicle from its current position to some goal position. The whole point of the algorithm is to choose a goal position that is some distance ahead of the vehicle on the path. We tend to think of the vehicle as chasing a point on the path some distance ahead of it – it is pursuing that moving point. This analogy is often used to compare this method to the way humans drive. This lookahead distance changes as we drive to reflect the twist of the road and vision occlusions.

The pure pursuit approach is a method of geometrically determining the curvature that will drive the vehicle to a chosen path point, termed the goal point. This goal point is a point on the path that is one lookahead distance from the current vehicle position. An arc that joins the current point and the goal point is constructed. The chord length of this arc is the lookahead distance, and acts as the third constraint in determining a unique arc that joins the two points.

III. System Setup

Our framework consisted of an Intel Realsense D435 mounted on the Firebird VI robot. We used ros-kinetic along with the tuw_multi_robot package [6].

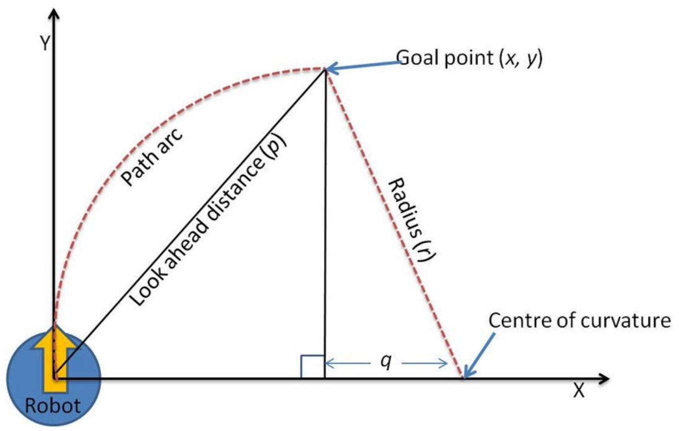

Fig. 2: Geometric derivation of the computation of lookahead distance in the pure pursuit algorithm.

A. Technologies Used

The Intel RealSense depth camera D435 is a stereo solution, offering quality depth for a variety of applications. With a range up to 10m, this small form factor camera can be integrated into any solution with ease, and comes complete with Intel RealSense SDK 2.0 and cross-platform support. The combination of a wide field of view and global shutter sensor on the D435 make it the preferred solution for applications such as robotic navigation and object recognition. The global shutter sensors provide great low-light sensitivity allowing robots to navigate spaces with the lights off.

Firebird VI is a reliable, four-wheeled versatile and rugged robot developed by Nex Robotics. The unique architecture of this robot allows it to be used in many areas of applications such as mapping, autonomous navigation, collaborative robotics, tele-presence and many more. The interface is designed to get a user started very quickly and is often used for advanced research in mobile robotics.

B. Setting up the Environment

The environment for running this program can be set up by installing the repository provided, which is a modification of the tuw_multi_robot framework [6], along with the rail_segmentation framework [7]; on ros-kinetic. The following launch files must be run:

- `roslaunch rail_segmentation rail_segmentation.launch`
- `roslaunch tuw_voronoi_graph corridor_voronoi.launch`
- `roslaunch pure_pursuit pp.launch`

The `rail_segmentation` launch file is responsible for object segmentation, and the command `rosservice call /rail_segmentaion/segment "{}"` must be used whenever there is a need to segment new obstacles. Alternatively, this command can be called at regular intervals for autonomous segmentation. The `corridor_voronoi` launch file will compute a Voronoi graph for the given map. A path will be computed as soon as the coordinates of the destination are provided with the `2D NavGoal` in Rviz.

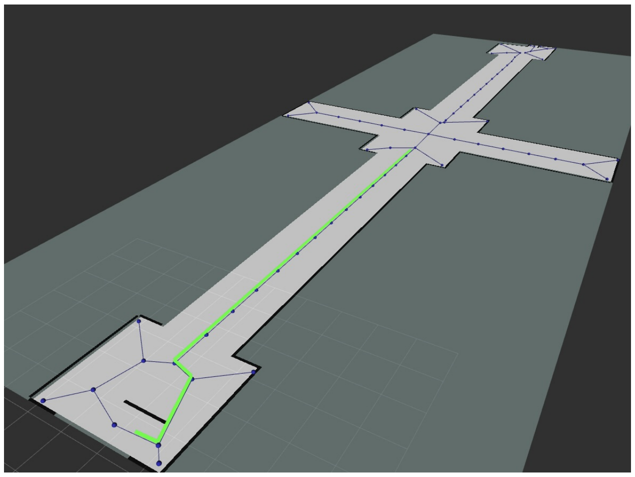
Fig. 3: Modified path and Voronoi graph of the corridor after addition of the obstacles.

Finally, the `pure_pursuit` launch file is responsible for calulating the lookahead distances, along with the steering angle and speed.

IV. Proposed Framework

Our proposed framework for navigating through a mapped area, with unknown obstacles consists of four steps: (i) detection of obstacles, (ii) real-time computation of path, (iii) computation of lookahead distances, and (iv) finding the optimal lookahead distances.

First, obstacles detected by the sensor are segmented and converted to a MarkerArray. These markers can be used to find the exact location of the obstacles with respect to our robot in the local map. Next, the Voronoi graph is recomputed for the new map, with added obstacles. Now, we must compute a new path to the destination, using the updated map. Finally, the path is converted to waypoints that connect lines, which can be sent to the robot.

A. Detection of Obstacles

The Intel's Realsense 2 depth sensor was used to detects obstacles in front of our robot. The `rail_segmentation` package for segmenting obstacles in front of the robot in real-time. The `rail_segmentation` package provides tabletop segmentation functionality given a point cloud. It also allows for segmentation within a robot's coordinate frame, so that objects stored on a robot's platform can be segmented.

The `/rail_segmentation/segment` service is invoked every time we wish to segment the object. The segmented obstacles are then published to the rostopic `/segmented_objects`, in the form of the SegmentedObjectList. This comprises of a list of objects from which we extract the centroid and the dimensions, and form a Bounding Volume of the cuboid enclosing the obstacle.

Once the objects are published on the rostopic `/segmented_objects`, the map region is modified and the pixels corresponding to the region of the obstacles are blacked out (by setting the probability of the cell to 100).

Algorithm 1: Lookahead label Assignment

```
Input: $T_{init}, v_{init} = 0, \mathcal{W}, \mathcal{L}$
Compute:
while $i < N$ do
    $R = SpawnCar(T_i)$
    for $l_j = 0$ to $K$ do
        PurePursuit($l_j$, $R$)
        if CrashDetected() then
            ResetCar($R$)
            $v_{exit\_i} = 0$
            $\delta_i = \infty$
        else
            until $R = l_j|\mathcal{W}$
            calculate $\{v_{current}, \delta_{current}\}$
            then:
            $v_{exit\_i} = v_{current}$
            $\delta_i = \delta_{current}$
            $v_{i+1} = v_{exit\_i}$
        end
    end
    $\pi(w_i) = \pi^*(\beta, v_{exit\_i}, \delta_i) \forall l_j \epsilon L$
end
Result: $\pi = l_i \forall w_i; l_i \epsilon L$
```

Fig. 4: Greedy algorithm [1] used for assigning optimal lookahead distances to each waypoint.

This makes the region of the obstacles inaccessible to the robot's path computation. The z-dimension was not taken into account (height). As soon obstacles are added to the map, the Voronoi graph is recomputed and consequently a new path is obtained. If no new path is found, the robot will stop.

B. Real-Time Computation of Path

The path from the source to the destination was calculated with the help of the Voronoi graph. Once the coordinates of the destination (or, goal) are known, our framework computed the closest point to the destination which lies on the graph. We also need the closest point on the graph from the robot's current position. Once the robot has knowledge of these two locations, our algorithm could swiftly calculate the shortest path between the two points, lying on the graph.

It can be noted here that the shortest path on the Voronoi graph may not be the actual shortest path. However, this was on purpose, since the Voronoi graph is calculated by taking the points that are furthest from all edges. This in turn, ensures that the path taken is the safest and hence, furthest away from all obstacles. Instead, we will optimise the path navigation with the help of our adaptive pure pursuit framework.

C. Computation of Lookahead Distances

Using the geometric derivation proposed in [2], we can compute the lookahead point for a given lookahead distance. This lookahead point is unique for a given path and starting position. Once this lookahead point is provided, we find a path point, called the lookahead point, that is in the direction of a goal point. Next, we use this lookahead point to calculate the curvature (steering angle in Ackermann format), and the speed to navigate to the lookahead point.

This lookahead point is calculated every second, and not after reaching the lookahead point. This would result in a curved and smooth path of the robot. To feed this instruction to the robot, the steering angle is converted to angular velocity.

D. Finding the Optimal Lookahead Distances

According to the findings in [1], we can conclude that longer straights require longer lookahead distances (and hence, more speed), whereas tight corners require shorter lookahead distances for safety and precision in navigation. If a longer lookahead distance is used at corners, the robot will tend to cut corners which will lead to collisions. Hence, to optimise the speed of navigation, larger lookahead distances are only used on straight parts of the computed path.

While [1] uses only three discrete values of lookahead distances, we propose a solution that uses distances from 1x the length of the robot all the way to 10x, for scenarios where the length of the robot is much shorter than the path provided. The algorithm tries to find the labelling policy π which assigns lookahead distances to different sections of the path based on the desired objectives.

The two desired objectives are: (i) Maximum Velocity Pure-Pursuit (vel*) and (ii) Minimum Deviation Pure-Pursuit (dev*). Depending on the objective, the trade-off factor β is adjusted, such that β = 0 produces minimum deviation and β = 1 produces maximum achievable velocity.

For each waypoint w_i, the robot is spawned there (starting location), and the navigation is simulated with different lookahead distances. If a crash is detected, the specific iteration is disregarded and the robot is reset.

V. Results

Our proposed framework was implemented and tested on a Firebird VI robot equipped with an Intel RealSense D435 depth sensor. The system demonstrated its effectiveness in real-time path planning and obstacle avoidance.

Upon successfully segmenting and detecting obstacles using the rail segmentation package, the framework was able to modify the obstacles in less than a second. As a result this modification, the Voronoi graph is recomputed to reflect the changes in the environment, which took about 1 second. Consequently, a new path is calculated in less than a second, from the robot's current position to the desired destination, taking into consideration the updated map and obstacle information. Now, a greedy algorithm is run to find the optimal lookahead distances for each waypoint. The resulting information of lookahead distance and speed is used for optimal path navigation.

Overall, the results confirmed that the system efficiently navigated through complex obstacles, ensuring safe and efficient path navigation. The combination of obstacle detection, and path computation, and optimisation enabled the robot to navigate through complex and dynamic environments with agility and obstacle avoidance capabilities.

VI. Conclusion

In this project, we developed an algorithm to optimize our pre-existing framework of using Voronoi graphs for navigating through a mapped area with unknown obstacles. Our proposed algorithm was a greedy algorithm that iterated through all the waypoints on the path to find the lookahead distance, and hence, speed of navigation. This was done to ensure minimum deviation from the path around tighter corners where there is a higher risk of collision.

The results of our experiments showed that the framework performed effectively in real-time scenarios. The computation of Voronoi graphs and lookahead distances was almost instantaneous. This demonstrated the feasibility of our approach for dynamic environments with changing obstacles.

VII. Future Work

We hope to extend this project to a 3D environment, where we need to take into account the terrain of the ground. In such a situation, we can use an attention based transformer to compute the optimal lookahead distances using deep learning techniques. Alternatively, other models may prove beneficial to finding the optimal navigation technique [8].

References

[1] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," *arXiv preprint arXiv:2111.08873*, 2021.

[2] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, January 1992.

[3] B. Binder, F. Beck, F. König, and M. Bader, "Multi robot route planning (mrrp): Extended spatial-temporal prioritized planning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4133–4139.

[4] B. Li, Y. Wang, S. Ma, X. Bian, H. Li, T. Zhang, X. Li, and Y. Zhang, "Adaptive pure pursuit: A real-time path planner using tracking controllers to plan safe and kinematically feasible paths," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 9, pp. 4155–4168, 2023.

[5] J. Ahn, S. Shin, M. Kim, and J. Park, "Accurate path tracking by adjusting look-ahead point in pure pursuit method," *International Journal of Automotive Technology*, vol. 22, pp. 119–129, 02 2021.

[6] TU Wien Robotics. (2023) TUW Multi Robot GitHub Repository. [Online]. Available: https://github.com/tuw-robotics/tuw_multi_robot

[7] ROS Wiki. (2023) ROS Wiki: Rail Segmentation. [Online]. Available: http://wiki.ros.org/rail_segmentation

[8] A. Joglekar, S. Sathe, N. Misurati, S. Srinivasan, M. Schmid, and V. Krovi, "Deep reinforcement learning based adaptation of pure-pursuit path-tracking control for skid-steered vehicles," *IFAC-PapersOnLine*, vol. 55, pp. 400–407, 11 2022.