A Study of PRM* Algorithm for a Restaurant Environment

Stan Zwinkels, Jelmer de Wolde, Chinmay Polyaramesh
4630726, 4705041, 5135125

***Abstract*— A path planning and control method is developed for the scenario of a non-holonomic robot serving food in a restaurant. The path planning algorithm used is PRM*, where a semi-random point sampling algorithm is used. Graph search is performed by the Dijkstra algorithm, after which b-splines is used to smoothen the path. By using a pure pursuit model in combination with a PID-controller, the serving robot can follow the trajectory within predefined limits. Finally, the results are discussed, commenting on the performance and limitations of the system, followed by future recommendations.**

I. Introduction

Ground robots are being implemented for a wide range of indoor and outdoor applications varying from logistic, self-driving vehicles to leisure applications [3]. Due to Covid19, restaurant owners have begun to look towards robots to aid in serving the customers to reduce human contact and adhere to social distancing regulations in the face of a global pandemic that continues unabated in many parts of the world. The goal of this paper is to find an appropriate motion planner that can be used for a serving robot within a typical restaurant environment. The serving robot should be able to find a collision-free path within the restaurant, without the use of any external assistance.

Many indoor robots and restaurant serving-robots are centered differential driven [5][6]. These robots are holonomic and can manoeuvre easily, which makes them ideal for a restaurant environment. However, to increase the complexity of this study, the designed robot will be based on a non-holonomic bicycle model. In this study only static obstacles within the restaurant are considered.

Because the floor plan of a restaurant does not change much over time, a multi-query algorithm like PRM could reduce the computational time to find a path by pre-computing its roadmap. Another common algorithm is RRT, however this is a single-query planner, which means that it has to re-sample the roadmap for each new location that is given. The other alternative for a non-holonomic model, is a path planning algorithm that is based on motion primitives [15][16]. Just as RRT, motion primitives are also single-query. Therefore, in this report, a path planning design is chosen that is based on PRM.

Many variations of the PRM exist that takes into account different sampling methods, connection methods, and collision-checking methods [8]. A variation on the standard form of PRM is Lazy PRM. This variant is also probabilistic complete and uses a faster method for checking obstacles, but is particularly suited for situations with a few queries [8], [9], [10]. Another variant of PRM is PRM*, which uses a varying radius to connect nodes depending on the number of nodes sampled. Also, any node that falls within the threshold values becomes connected [17]. Although this study assumes a static environment, a real restaurant will be located in a dynamic environment. However, the standard PRM method is not suited for dynamic environments. There have been variants of PRM developed that are suitable for dynamic environments, such as Dynamic PRM [11]. This method can reroute its path, when for example a human is blocking its trajectory. Other methods to handle dynamic environments are by making use of dynamic obstacles avoidance methods, such as VO, ORCA or NH-ORCA [14].

For this paper, it was decided to use the PRM* method. PRM* has an advantage over the standard PRM method by using an optimum distance threshold for a given map and sampling set [17]. PRM* has the same advantage as PRM, that it is probabilistic complete. To prevent PRM* from being incomplete, a different sampling strategy has been implemented and explained in section III-A.

The rest of this paper describes in detail how the path planning and control mechanism for a restaurant robot is developed. The next section II first gives an overview of the dynamic model that is used. Section III describes the implemented planning and control algorithms. Finally, the results are presented in section IV and discussed in section V.

II. Robot Model

As discussed in the introduction, the model will be simulated as a non-holonomic bicycle model. The dynamic equations are made using the no-slip assumption.

A. Dynamic Model

Figure 1 shows the FBD of the bicycle model, where F_f and F_r denote the lateral tire forces at the front and rear wheels respectively. F_a is the propulsive force. l_r and l_f are the distance at the front and rear wheels to the center of gravity of the vehicle. The front wheel has a steering angle δ, while the whole vehicle has an inertial heading of θ.

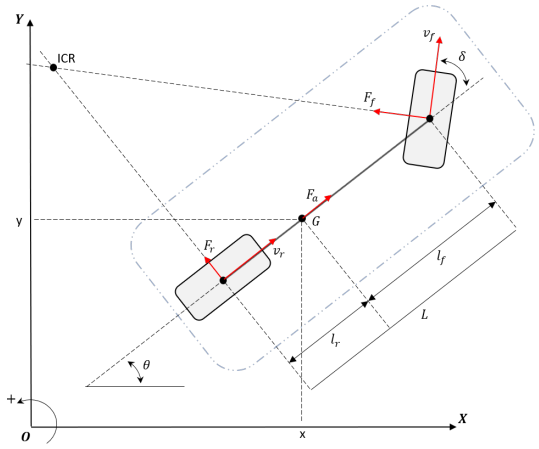

Fig. 1. Dynamic Bicycle Model

By using the Newton-Euler method, the equations for ΣF_x, ΣF_y and ΣM_z, can be respectively can be given as,

$$m\ddot{x} = F_a cos(\theta) - F_r sin(\theta) - F_f sin(\delta + \theta) \quad (1)$$

$$m\ddot{y} = F_a sin(\theta) + F_r cos(\theta) - F_f cos(\delta + \theta) \quad (2)$$

$$I_z\ddot{\theta} = -F_r l_r + F_f l_f cos(\delta) \quad (3)$$

The non-holonomic constraints can be derived as:

$$\dot{x}sin(\theta) - \dot{y}cos(\theta) + l_r\dot{\theta} = 0 \quad (4)$$

$$\dot{x}sin(\theta + \delta) - \dot{y}cos(\theta + \delta) - \dot{\theta}l_f cos(\delta) = 0 \quad (5)$$

Finally, eq (1)-(5) can be combined resulting in the matrix equation shown in eq (6).

B. Workspace and Configuration Space

The degree of maneuverability for the bicycle is defined as. $\delta_M = \delta_m + \delta_s$, where the mobility and steerability are both 1. Because the Maneuverability is 2, the ICR is constrained to lie on a line [5]. The workspace of the robot can be described as $\mathcal{W} = \mathbb{R}^2$. To solve a motion planning problem, the algorithm must conduct a search in the Configuration space C. The configuration space of the robot has the dimension: $C = \mathbb{R}^2 \times \mathbb{S}^1$, and can be represented by three parameters (x, y, θ).

III. Planning & Control

In this section both the planning and control algorithms are being discussed in order to make the vehicle smoothly follow the optimal path to the given goal location.

A. Sampling Method

As mentioned before, the motion planning algorithm implemented is PRM*. Instead of using an uniform sampling approach, a slightly different strategy is chosen to sample the nodes. First, the workspace is divided into small grids. Next, the algorithm picks a random point within each grid after which it checks whether the chosen point is in free space. When the point is in obstacle space, the algorithm will pick another random point in the same grid, with a maximum of 100 attempts. If a grid is not able to find a point in free space, within the 100 attempts, the grid is considered as an object. Using this method of sampling, the sampled points are more likely to be found in narrow passages between objects.

Next, the algorithm will sample vertices between all the sampled points. PRM* uses a threshold for connecting the nodes with each other, defined by the amount of nodes to be sampled (n) and a constant (γ), shown in eq (7) [22].

$$\text{Threshold} = \gamma(\frac{\log(n)}{n})^{\frac{1}{3}} \quad (7)$$

Whenever an edge is created it will be checked if the complete edge is in free space. All the sampling algorithms are developed in Python, completely from scratch.

B. Graph Search

By section III-A, a complete graph $G = (V, E)$ with all the desired vertices and edges is created. To find the shortest route between two edges, Dijkstra and A* algorithms were considered. Since both algorithms find the shortest path, but A* only in less steps when using the right heuristic function, it was decided to implement Dijkstra. It is possible to add an heuristic function to Dijkstra in the future, to turn it into A*. Even as the sampling methods, is the Dijkstra algorithm implemented in Python, completely from scratch.

C. Path Smoothing

To smoothen the path found by the Dijkstra algorithm, the use of b-splines was chosen since it ensures C^2 (parametric) and G^2 (geometric) continuity [17]. Fifth order splines are used so that a continuous jerk (third-order derivative of path) is obtained [18]. This is crucial for our restaurant scenario because sudden accelerations could result in the spilling of drinks. The smoothening algorithm is developed in Python using the Scipy library. The interpolate function parameterizes x and y on parameter 't' using splprep and extrapolated back into Cartesian coordinates using splev functions. First-order and second-order derivatives are obtained to verify curvature continuity and radius of curvature at each point.

$$\begin{bmatrix} m & 0 & 0 & s(\theta) & s(\theta+\delta) \\ 0 & m & 0 & -c(\theta) & -c(\theta+\delta) \\ 0 & 0 & I_z & 0 & -l_f c(\delta) \\ s(\theta) & -c(\theta) & l_r & 0 & 0 \\ s(\theta+\delta) & -c(\theta+\delta) & -l_f c(\delta) & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \\ F_r \\ F_f \end{bmatrix} = \begin{bmatrix} -F_a c(\theta) \\ -F_a s(\theta) \\ 0 \\ -c(\theta)\dot{x}\dot{\theta} - s(\dot{\theta})\dot{y}\dot{\theta} \\ -c(\theta+\delta)\dot{x}\dot{\theta} - s(\theta+\delta)\dot{y}\dot{\theta} \end{bmatrix} \quad (6)$$

D. Vehicle Control

Although path smoothing is performed by the use of splines, it does not solve the problem of having a curvature that exceeds the turning capabilities of our robot. This can be ensured by setting a maximum steering limit δ on the steering controller. The pure pursuit model is employed for it's simple and effective method of tracking a path which is widely employed for planned navigation of non-holonomic ground vehicles [12]. It uses a look-ahead point which is at a fixed distance L_d from the reference path ahead of the vehicle as it follows, as shown in Figure 4.

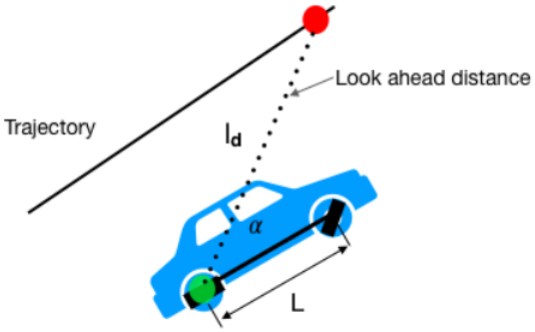

Fig. 4. Pure pursuit geometric relationship, [21]

The vehicle proceeds to that point using a steering angle δ, whose formula is given in eq (8), where L and α refer to the wheelbase and error in heading angle respectively.

$$\delta = tan^{-1}\left(\frac{2Lsin(\alpha)}{Ld}\right) \tag{8}$$

A PID-controller is employed to obtain accelerations by penalizing the error between the current velocity and the target velocity. The resulting propulsive force F_a is fed into the dynamic equations along with steering input δ. The output of the dynamic equations are the second order state derivatives $[\ddot{x}, \ddot{y}, \ddot{\theta}]^T$, which upon integration gives the required updated state variables $[x, y, \theta]^T$. The PID-controller is developed in Python from scratch and for the Pure Pursuit an example code from Github is used [23]. However, an improvement was made by using a dynamic model for our case instead of the kinematic model used in the example code.

IV. Results

This section will discuss the results and the performance of the system. The model is tested in three different restaurant settings using a simulator, which is created in Python from scratch using the so-called PyGame library.

A. Performance

The created simulation is interactive and makes it therefore possible to select a desired place on the map as the next location for the robot. The Dijkstra algorithm can find the shortest route between that node and the current node, when at least one route exists. If there is no connection possible, the simulation prints an error and the user is able to select a new location. Using the combination of pure pursuit and the PID-controller, the robot can follow this smoothed path.

Figure 2 shows an example of the simulation in which the robot is travelling inside the restaurant. The cyan lines are all the vertices of the map, the blue line is the path found by Dijkstra, the red line is the smoothed path and the black line behind the robot shows the path travelled so far. An interesting thing to mention is the fact that the robot travels around table B instead of between table B and C. This is because no vertex is sampled between table B and C in this situation. Increasing the number of nodes to sample would increase the chance of sampling a vertex in such small areas.

The system is not optimal yet. The biggest problem is that the robot cannot rotate in place, because of the fact that a bicycle model is used. The robot has no parking turn mechanism, so when the robot is faced in the opposite direction than the route, it will drive a circle as shown in Figure 3. Since there is no check implemented whether this circular part is in free space, it happens that the robot in the simulation drives through objects.

To evaluate the performance of the PID-controller, the target velocity, real velocity, acceleration, and acceleration

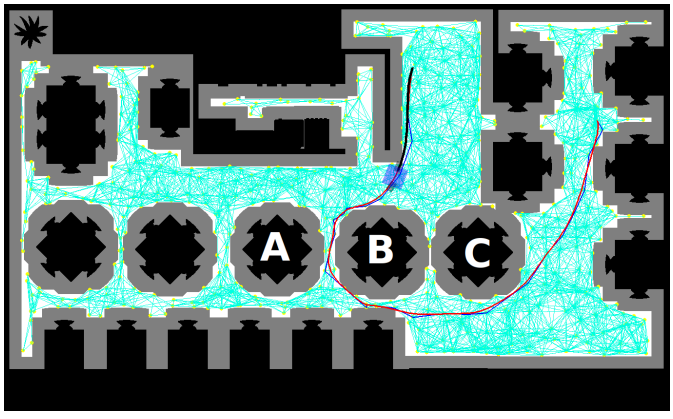

Fig. 2. Good performance of the system

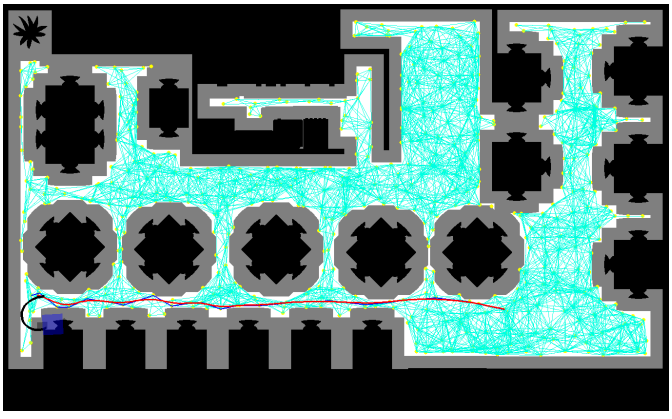

Fig. 3. Problem of rotation of the vehicle

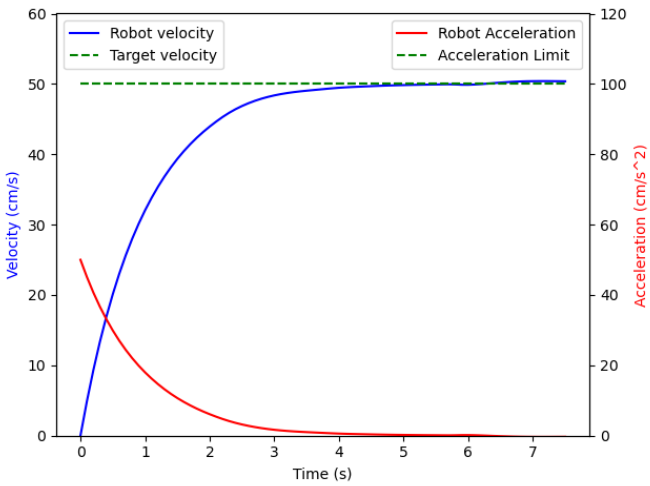

Fig. 5. Velocity and acceleration of the robot

limit for a random trajectory are plotted and shown in Figure 5. The controller uses a high proportional gain Kp for a quick response. The integral and derivative terms Ki and Kd ensure that there is only a minor steady-state error and overshoot. It can be observed that the final velocity does not come back to zero, meaning that the robot does not come to a rest at the target position. This happens because currently there is no such constraint applied in the controller implementation. The acceleration plot shows that the accelerations generated by the controller are well within the limit, which ensures that the generated force does not topple the Robot.

This report only shows a few screenshots from the performance of the system. A complete video of the performance, for three different restaurant setting can be seen at: https://youtu.be/tswPvcibv8E.

V. Discussion & Further Work

In this section, the important aspects of the path planner with its limitations and advantages are discussed and compared with other methods.

A. Theoretical Aspects

As discussed in the introduction, PRM* is a probabilistic complete algorithm. In real situations it could be possible that the algorithm is incomplete. This is because of obstacles that prevent the serving robot from reaching the location. However, the simulation shown in Figure 2, shows that with enough space between the tables and a reasonable amount of nodes, the robot is capable of navigating through the whole restaurant. The found solution is not optimal. However, by selecting enough nodes, the found path would go to an optimal solution. It is, however, not important if the created path is an optimal solution, because it is even better if the serving robot takes wider corners around the tables, instead of passing each table closely.

The planning model is fully off-line, meaning that the whole plan is constructed before execution and only takes a longer computation time for the first run. The time complexity of the algorithm is measured by the number of elementary operations. The code is written as blocks that ultimately form the whole motion planner. These blocks consist of the following steps: Sampling points, Neighbor Searching, Collision Checking, Dijkstra Method, Path Smoothing, Pure Pursuit + PID controller. The time complexity of the Dijkstra algorithm can be expressed as $O(|E| + V \log |V|)$.

B. Practical Aspects

The PRM* motion planner performed well, and the nodes that were sampled covered the whole workspace. This made it possible that the motion planner was capable of driving to the location without colliding with any obstacles.

Even though a bicycle model has been implemented, the downside of the model is the maximum steering angle. As discussed in the introduction, common serving robots are based on holonomic differential driven methods. This allows the robot to turn in place, and makes it able to manoeuvre in small places. The bicycle model that is implemented does not have a parking turn mechanism, and therefore drives a circle to turn, possibly through an object, as shown in Figure 3. Another method that could have made the bicycle model more manoeuvrable, is by implementing the Reeds-Shepp path [20]. This model enables the possibility to drive forward and backward, so it can manoeuvre its way out and avoid hitting any obstacles.

For the static environment the PRM* performed well and was always capable of finding a solution when a new goal was marked. The problem with PRM* is that it does not work well for dynamic environments. This means that for a variant, like DPRM should be used when adding dynamic objects.

C. Other Recommendations

An acceleration limit has been set to prevent the robot from spilling food while driving. Another method to prevent this is presented in a study that has been conducted for serving robots, which adds a S-velocity profile with a limit in the amount of jerk [7]. It would be of interest to explore if that method could be applied to improve the productivity of the robot while keeping the accelerations within limits.

The pure pursuit model that is implemented is an easy to use control method. However, an even more stable method is the so-called Stanley controller [21] used by MIT teams for the DARPA challenge. The Stanley controller uses the cross-track error in addition to the heading error which makes the generated path steadier, especially in turns. Another approach could be to use Model Predictive Control, which can take multiple objectives and constraints on final velocity and angle into account.

REFERENCES

[1] Example, Example bibliography, The bibliography: A great example, 1st ed. vol. 1, Delft, 2019, pp. 1-2

[2] Jaillet, Léonard & Siméon, Thierry. (2004). A PRM-based motion planner for dynamically changing environments. 1606 - 1611 vol.2. 10.1109/IROS.2004.1389625.

[3] RTN Editors. (2020, July 12). As the Restaurant Robotics Space Heats Up, Pudu Technology Secures 15 Million for Its Server and Dish Return Robots. Restaurant Technology News. https://restauranttechnologynews.com/2020/07/as-the-restaurant-robotics-space-heats-up-pudu-technology-secures-15-million-for-its-server-and-dish-return-robots/

[4] A. Knobloch, N. Vahrenkamp, M. Wächter, and T. Asfour, "Distanceaware dynamically weighted roadmaps for motion planning in unknown environments," IEEE Robotics and Automation Letters (RA-L), pp. 0–0, 2018

[5] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2004). Introduction to Autonomous Mobile Robots (Intelligent Robotics and Autonomous Agents series) (first edition). The MIT Press.

[6] Pudu Robotics. (n.d.). Smart delivery robot-Pudu robotics. Pudu Products. Retrieved January 5, 2021, from https://pudurobotics.com/

[7] Wan, A. Y. S., Soong, Y. D., Foo, E., Wong, W. L. E., & Lau, W. S. M. (2020). Waiter Robots Conveying Drinks. Technologies, 8(3), 44.

[8] Geraerts, R., Overmars, M. H. (2002), "A comparative study of probabilistic roadmap planners", Proc. Workshop on the Algorithmic Foundations of Robotics (WAFR'02)

[9] Bohlin, R., & Kavraki, L. E. (2000, April). Path planning using lazy PRM. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065) (Vol. 1, pp. 521-528). IEEE.

[10] Hauser, K. (2015, May). Lazy collision checking in asymptotically-optimal motion planning. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2951-2957). IEEE.

[11] Jaillet, L., & Siméon, T. (2004, September). A PRM-based motion planner for dynamically changing environments. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566) (Vol. 2, pp. 1606-1611). IEEE.

[12] Morales, J., Martínez, J. L., Martínez, M. A., & Mandow, A. (2009). Pure-pursuit reactive path tracking for nonholonomic mobile robots with a 2D laser scanner. EURASIP Journal on Advances in Signal Processing, 2009(1), 935237.

[13] Suh, H. J., Deacon, J., & Wang, Q. A Fast PRM Planner for Car-like Vehicles.

[14] Visual Paradigm. (n.d.). Restaurant Seating Plan Seating Chart Example. Visual Paradigm Online Diagrams. Retrieved January 6, 2021, from https://online.visual-paradigm.com/de/diagrams/templates/floor-plan/seating-chart/restaurant-seating-plan/

[15] Wang, S. (2015). State lattice-based motion planning for autonomous on-road driving (Doctoral dissertation).

[16] Pivtoraiko, M., Knepper, R. A., & Kelly, A. (2007). Optimal, smooth, nonholonomic mobile robot motion planning in state lattices. Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-15.

[17] Chin, T. (2019, February 19). Robotic Path Planning: PRM and PRM* - Tim Chin. Medium. https://theclassytim.medium.com/robotic-path-planning-prm-prm-b4c64b1f5acb

[18] Ildik ó Papp et all (2007,Journal for Geometry and Graphics 11(2):1-7), C2 and G2continuous spline curves with shape parameters.

[19] A. Al Jawahiri(TU delft Master thesis 2018), Spline-based Trajectory Generation for Autonomous Truck-Trailer Vehicles in Low Speed Highway Scenarios

[20] LaValle, S. M. (2006). Planning algorithms. Cambridge university press.

[21] Ding, Y. (2020, August 20). Three Methods of Vehicle Lateral Control: Pure Pursuit, Stanley and MPC. Medium. https://dingyan89.medium.com/three-methods-of-vehicle-lateral-control-pure-pursuit-stanley-and-mpc-db8cc1d32081

[22] Karaman, S., and E. Frazzoli. "Sampling-based algorithms for optimal motion planning." The International Journal of Robotics Research 30, no. 7 (June 22, 2011): 846-894.

[23] https://github.com/AtsushiSakai/PythonRobotics