RRT algorithm with vehicle dynamics

Arunachalam Venkatachalam
Department of Mechanical Engineering
Texas A&M University
College Station, TX
arun23venkat@tamu.edu

I INTRODUCTION

Traditional path planning algorithms that rely on simple kinematic models are ineffective for high-speed navigation. These models generate paths that are only feasible at extremely low speeds. At higher speeds, vehicles deviate significantly from the planned trajectory, rendering the path unusable for the intended mission. To address this limitation, this project proposes a path planning algorithm that incorporates a dynamic vehicle model. This model accounts for the vehicle's dynamics, enabling the generation of realistic paths that are achievable at higher speeds. The dynamic vehicle model-based path planner effectively bridges the gap between planning and navigation. It generates paths that are not only feasible but also executable in real-world scenarios, ensuring successful mission completion.

Path planning for autonomous vehicles is a two-step process: generating a feasible path and executing it using a control law. Traditionally, path planning has used a simplified kinematic model, which is only accurate at low speeds. This model assumes the vehicle moves without skidding and neglects slip angle, which can significantly affect the vehicle's behavior at higher speeds. As a result, paths planned with the kinematic model may be impossible to follow in real-world conditions. To address these limitations, a dynamic vehicle model for path planning is more suitable. This model considers the vehicle's dynamics, including skidding and slip angle, resulting in more realistic and achievable paths. Rapidly exploring Random Tree (RRT) planner is utilized, which can incorporate the dynamic model into its path-finding algorithm. Unlike roadmap-based planners, RRT's incremental construction allows for seamless integration of dynamic constraints. And also, Dubin’s curve is used for kinematic models and thus it cannot be employed for a dynamic non-holonomic model. Due to these restrictions, the dynamic bicycle model is intended to be employed within an incremental planner such as the RRT.

This project is a simple implementation of a vehicle dynamics based RRT planner on a pre-determined map with obstacles. When a start point, and an end point are provided the algorithm must enable the vehicle to reach the destination with the shortest possible distance thereby avoiding any obstacles in the path. If an obstacle is present between a path joining two points, that branch of the tree would be neglected, and other paths which would result in a shorter distance are prioritized.

II DYNAMIC VEHICLE MODEL

The five-degrees-of-freedom dynamic car model is a mathematical representation of the motion of a car. The state of the car is defined by a vector $x = (x_g, y_g, \theta, v_y, r)^T$, where:

- x_g and y_g are the coordinates of the center of gravity of the car.
- θ is the orientation of the car.

- v_y is the lateral speed of the car.
- r is the yaw rate of the car.

The bicycle model is a simplified version of the five-degrees-of-freedom dynamic car model. It assumes that the car's front and rear wheels can be projected onto two virtual wheels located in the middle of the front and the rear axles of the car. This model is often used in path planning and control applications because it is simple to compute. The model is depicted in fig. 1.

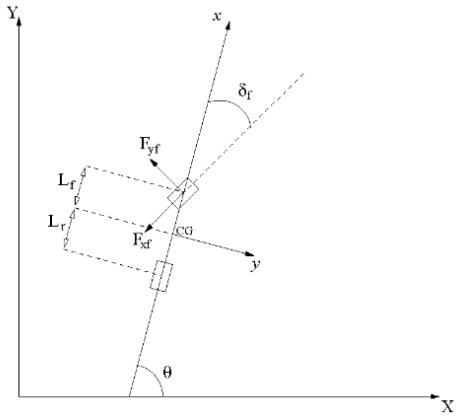

Fig. 1 Bicycle Model

The equations of motion for the five-degrees-of-freedom dynamic car model are derived using Newton's second law of motion.

$$m(\dot{v}_x - v_y r) = -F_{xf}cos\delta_f - F_{yf}sin\delta_f - F_{xr}$$
$$m(\dot{v}_y - v_x r) = F_{yf}cos\delta_f - F_{xf}sin\delta_f - F_{yr}$$
$$I_z\dot{r} = L_f(F_{yf}cos\delta_f - F_{xf}sin\delta_f) - L_r F_{yr}$$

where m is the mass of the car; v_x is the longitudinal velocity of car; I_z is the moment of inertia; L_f and L_r are the distance between front and rear wheels from the center of gravity of the car respectively; δ_f is the front steering angle; $F_{xf}, F_{yf}, F_{xr}, F_{yr}$ are the tire forces in x and y direction in the front and rear respectively, and $\dot{r}$ is the yaw rate.

Constant longitudinal speed is assumed. This implies that v_x is constant during the planning phase. Aerodynamic resistance is neglected, due to this the longitudinal tire force F_{xf} becomes zero. Applying these conditions to the above equations result:

$$\dot{v}_y = \frac{F_{yf}cos\delta_f}{m} - \frac{F_{yr}}{m} - v_x r$$
$$\dot{r} = \frac{L_f F_{yf}cos\delta_f}{I_z} - \frac{L_r F_{yr}}{I_z}$$

Linear tire models are employed. It could be represented as:

$$F_{yf} = -C_{\alpha f}\alpha_f$$
$$F_{yr} = -C_{\alpha r}\alpha_r$$

Where $C_{\alpha f}$ and $C_{\alpha r}$ are the cornering stiffness coefficients for the front and rear wheels, respectively. α_f and α_r are the slip angles for the front and rear wheels, respectively. For calculation purposes, the slip angles are assumed to be small, and they could be represented as

$$\alpha_f = \frac{v_y + L_f * radius\ of\ \ tire}{v_x} - \delta_f$$
$$\alpha_r = \frac{v_y + L_r * radius\ of\ tire}{v_x}$$

The full non-linear model could be represented as

$$\dot{x}_g = v_x cos\theta - v_y sin\theta$$
$$\dot{y}_g = v_x sin\theta + v_y cos\theta$$
$$\dot{\theta} = r$$
$$\dot{v}_x = -\left(\frac{C_{\alpha f}\ cos\delta_f + C_{\alpha r}}{m v_x}\right) v_y - \left(\frac{L_f C_{\alpha f}\ cos\delta_f + L_r C_{\alpha r}}{m v_x} - v_x\right) r + \frac{C_{\alpha f} cos\delta_f}{m}\ \delta_f$$
$$\dot{r} = -\left(\frac{L_f C_{\alpha f}\ cos\delta_f + L_r C_{\alpha r}}{I_z v_x}\right) v_y - \left(\frac{L_f^2 C_{\alpha f}\ cos\delta_f + L_r^2 C_{\alpha r}}{I_z v_x}\right) r + \frac{L_f C_{\alpha f} cos\delta_f}{I_z}\ \delta_f$$

The above are the five degrees of freedom non-linear model. This state transition model would be employed to compute the next state using the current state and the input.

III MAP GENERATION

The algorithm contains a class which is designed to process a map file in the Portable Graymap (PGM) format as its input. This map is generated using the gmapping feature within the Robot Operating System (ROS) as depicted in Fig.2. Specifically, the map is derived from the TurtleBot house environment. The algorithm utilizes the information encoded in the PGM file.

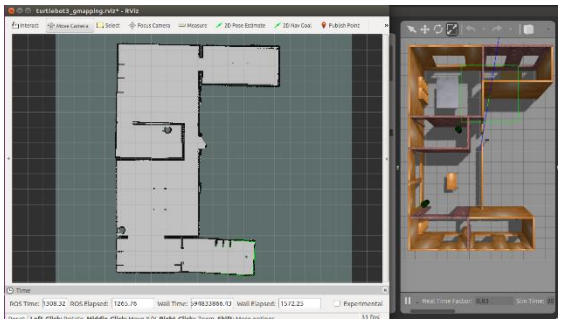

Fig 2. Gmapping in TurtleBot environment

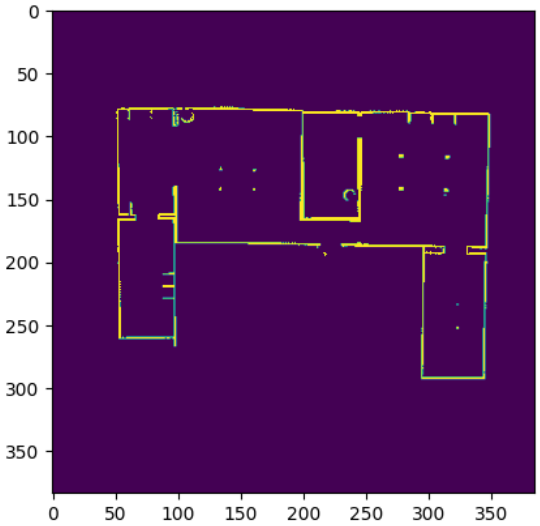

Fig 3. Occupancy grid view of the map

The input (.pgm) image is then converted into a binary occupancy grid map as shown in Fig.3, where pixel values above a certain threshold are considered obstacles, and those below or equal to the threshold represent free space. The map, its dimensions, and a default resolution value are stored and utilized throughout. When provided a custom start and an end point, the algorithm works its way through to find the shortest path between the points adhering to the dynamics.

IV RRT ALGORITHM

The Rapidly Exploring Random Tree (RRT) algorithm is a powerful and versatile tool for motion planning in high-dimensional configuration spaces. Its randomized approach iteratively expands a tree structure to explore unexplored regions, ensuring probabilistic completeness and efficient navigation in complex environments.

Generate Vanilla RRT ($\mathbf{X}_{init}$, K, Δt):

1. G.init($\mathbf{X}_{init}$)
2. **for** k=1 to K do:
3. $\mathbf{X}_{rand}$ ← random_config (X_{free});
4. $\mathbf{X}_{near}$ ← nearest_neigbor ($\mathbf{X}_{rand}$, T);
5. **u** ←select_input ($\mathbf{X}_{rand}$, $\mathbf{X}_{near}$);
6. $\mathbf{X}_{new}$ ← new_state ($\mathbf{X}_{near}$, **u**, Δt);
7. **if** collision_free_path($\mathbf{X}_{near}$, $\mathbf{X}_{new}$) **then**
8. G.add_node ($\mathbf{X}_{new}$);
9. G.add_edge ($\mathbf{X}_{near}$, $\mathbf{X}_{new}$, **u**);
10. **end if**
11. **return** G

The above represents a pseudo code of an RRT algorithm. Starting with an initial state, the algorithm iterates for a predefined number of times. In each iteration, a random state is sampled, followed by the identification of its nearest neighbor in the existing tree. A control input is then chosen, guiding the system towards the sampled state. By applying this control input, a new state is generated and added to the tree, along with an edge connecting it to its nearest neighbor. This process results in a progressively expanding tree that explores the configuration space, enabling the algorithm to efficiently navigate complex environments.

V INTEGRATION OF DYNAMICS TO THE RRT PLANNER

The custom RRT planner adhering to the vehicle dynamics is suitable for real-time

applications. The algorithm for the RRT with the vehicle dynamics is represented below.

The algorithm has three classes:

1. **PerceptionMapper,** which takes in a .pgm file and converts it to an occupancy grid map based on a threshold.
2. **TreeNode,** used for representing nodes in a tree structure, the tree holds nodes representing the five states, and two lists which hold the parent and child nodes of the current node.
3. **RandomlyExploringRandomTrees,** this is where all the functions pertaining to the RRT algorithm along with the function for vehicle dynamics are present. Instance of this class is used to call the functions iteratively as the RRT algorithm gets executed. The following functions are present within this class.
   - **(a) randomConfiguration:** This function is called whenever a random node is to be generated. This function returns a random array ([x, y, theta, v_y, ṙ]).
   - **(b) nearestneighbor:** This function takes in the tree(G) and the X_{rand} generated from previous step as inputs. It iterates over the nodes in the tree and the corresponding child nodes of each node present in the tree, thereby computing the distance between each node and X_{rand}. Whichever node has the shortest distance from X_{rand} will be returned.
   - **(c) selectInput:** This function determines the optimal steering input. It initializes the steering angle to the minimum allowable value and iterates over a specified range of steering angles (-pi/6 to pi/6), incrementing the angle in each iteration by (pi/60). For each steering angle, the method calculates a new state $\mathbf{X_{new}}$ using the **new_state** method and evaluates the distance between this new state and X_{rand}. The method keeps track of the steering angle that minimizes this distance and returns it.
   - **(d) new_state:** This function simulates the dynamics of a car by numerically integrating the rates of change calculated from the **dynamic_car_model** to calculate the new state of the car after a small-time increment. The integration is performed using the Runge-Kutta method, and the resulting new state is represented as a TreeNode instance. A fourth order Taylor's approximation is performed and used in Runge-Kutta integration. Runge-Kutta integration is depicted below.

$$x_{n+1} \approx x_n + \frac{\Delta t}{6} * (k_1 + 2k_2 + 2k_3 + k_4)$$

Where

$$k_1 = dynamic\_car\_model(x_n, u_n)_{t_n}$$
$$k_2 = dynamic\_car\_model\left(x_n + \frac{k_1}{2}, u_n\right)_{t_n + \frac{\Delta t}{2}}$$
$$k_3 = dynamic\_car\_model\left(x_n + \frac{k_2}{2}, u_n\right)_{t_n + \frac{\Delta t}{2}}$$
$$k_4 = dynamic\_car\_model(x_n + k_3, u_n)_{t_n}$$

x_n represents the state vector and u_n represents the input, i.e., δ_f. dynamic_car_model is the function responsible for state transition.

   - **(e) dynamic_car_model:** This function represents the mathematical model of the dynamics of the car, it embeds the vehicle dynamic equations discussed before. This function takes as input the current state of the car and the steering input. It computes the rates of change for each state variable based on the steering input. The resulting derivatives are used to update the state of the car, providing an estimate of how the car's position,

orientation, and velocities evolve over a small-time increment.

(f) stepsizeCover: This function takes two parameters, X_{near} and X_{new}. It calculates the unit vector pointing from Xnear to Xnew. It then multiplies this unit vector by the step size (variable) to obtain an offset vector. The resulting offset is scaled to determine the position of a new point in the configuration space. The stepsize tells how much the car could move in Δt.

(g) isObstacle: This function checks if the X_{near} and X_{new} are feasible to connect by checking if there are any obstacles in the path connecting these both.

Finally, the X_{new} generated is added to the child list. The functions are called iteratively until the goal is found. Once the goal node is reached the path is returned. The RRT planner terminates.

RRT with vehicle dynamics:

```
TreeNode ← empty tree
Add start_node to TreeNode
for i in num_iteration do
    generate X_rand
    iterate over TreeNode and find X_near
    select input u
    while select input u do
        Iterate over range of δ_f
        for each δ_f do
            compute new state using X_near
            while compute new state do
                compute state rate of change
            do Runge-Kutta integration
            return new state
            for each new state do
                compute distance to X_rand
                keep track of δ_f
    return δ_f that gives closest new state to X_rand
    using returned δ_f compute new state
    if dist (new state, X_near) > stepsize
      find the unit vector of X_near, new state
      offset ←multiply unit vector by stepsize
      new state ←add offset to X_near
    else
      return new state
    check for obstacle between X_near, new state
    if obstacle-free
      add new state as childnode of current node
    check if new state is goal state
        if true
            terminate RRT
            return path
        else
            continue with next iteration
```

The above algorithm is a pseudo code representation of the RRT planner with vehicle dynamics. Parameters like mass, v_x, tire stiffness, inertia, L_f, L_r are assumed and are hard coded in the program.

VI RESULTS AND CONCLUSIONS

The above algorithm was evaluated on a simulated map environment. The algorithm works well and was able to generate paths that were dynamically possible. A few snapshots of the results are attached below.

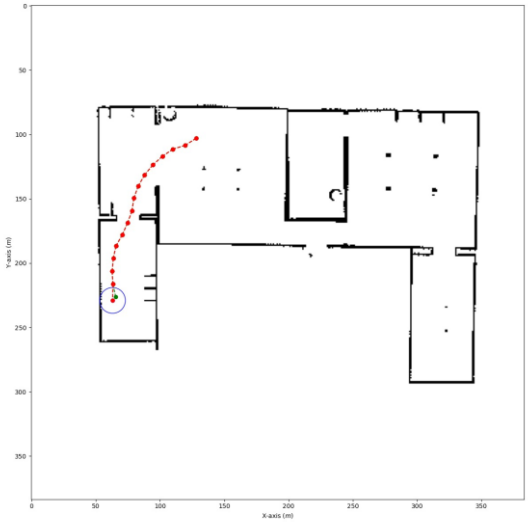

Fig 4. Output

The paths shown are possible for a real-time vehicle to follow. The time taken for the algorithm to find the trajectory is less than 4 seconds.

Instead of a goal point, a goal region is employed when collaborating with dynamic models. Considering a goal region instead of a single goal point is particularly pertinent due to the intricacies and constraints associated with the motion of vehicles. Defining a goal region allows the algorithm to find paths that consider

the vehicle's maneuverability constraints, enabling smoother and more feasible trajectories within the region. Also, vehicle dynamics involve considerations such as velocity profiles, tire friction, and other dynamic constraints. A goal region allows the algorithm to explore paths that adhere to these constraints while reaching the vicinity of the goal.

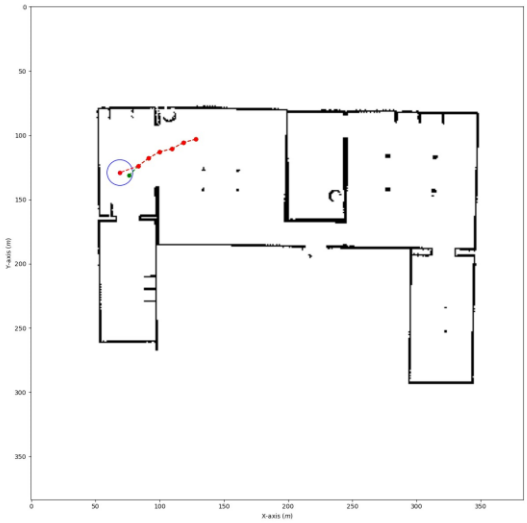

Fig 5. Output

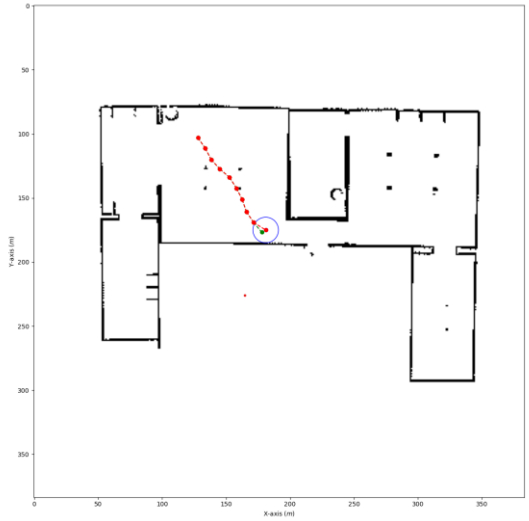

Fig 6. Output

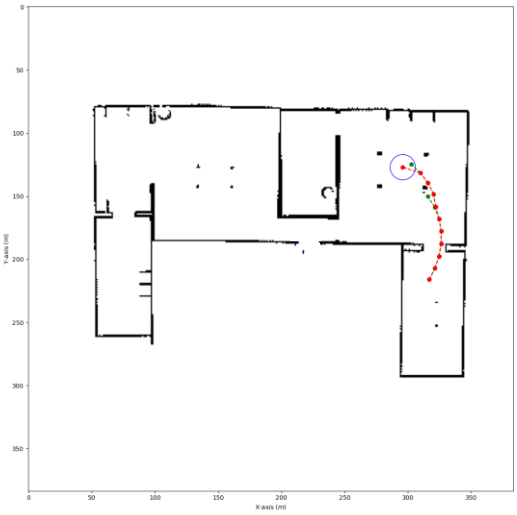

Fig 7. Output

Link to code: Code
Link to sample output: Sample Output

VII REFERENCES

[1] Dolgov, Dmitri, et al. "Practical search techniques in path planning for autonomous driving." *Ann Arbor* 1001.48105 (2008): 18-80. Chicago, IL. Menlo Park, CA, AAAI. (Visited on 10/27/2023)

[2] Dolgov, Dmitri, et al. "Path planning for autonomous vehicles in unknown semi-structured environments." *The international journal of robotics research* 29.5 (2010): 485-501. (Visited on 10/27/2023)

[3] Pepy, Romain, Alain Lambert, and Hugues Mounier. "Path planning using a dynamic vehicle model." *2006 2nd International Conference on Information & Communication Technologies*. Vol. 1. IEEE, 2006. (Visited on 10/27/2023)