Path Planning using a Dynamic Vehicle Model

Romain Pepy, Alain Lambert and Hugues Mounier
Institut d'Électronique Fondamentale
UMR CNRS 8622 - Université Paris-Sud XI
Bat. 220, 91405 Orsay, France
{pepy,lambert,mounier}@ief.u-psud.fr

Abstract

This paper addresses the problem of path planning using a dynamic vehicle model. Previous works which include a basic kinematic model generate paths that are only realistic at very low speed. By considering higher vehicle speed during navigation, the vehicle can significantly deviate from the planned trajectory. Consequently, the planned path becomes unusable for the mission achievement. So, to bridge a gap between planning and navigation, we propose a realistic path planner based on a dynamic vehicle model.

1 Introduction

Moving an autonomous vehicle is often divided in two phases. In the first one, a feasible path between two configurations is computed. Then, this path is followed by the vehicle, using the trajectory returned by the planner and a control law. Most of research works considered these two steps as independent and only focus on one of them. Unfortunately, the planning phase is strictly dependent on the model used during the navigation process.

The dependence between planning and navigation is always considered when planning for an holonomic or nonholonomic model [1] since one can imagine that a path planned for a differential drive could not be followed by a car as it is not differentiable and the derivative is not continuous. This dependence have to be preserved when working on path planning for car. Indeed, most of research works [2, 3] use the classical kinematic car-like model to plan path for every kind of car. However, each car does not react the same way (skidding, trajectory...) when applying the same input (the steering angle for example). So, when the path is planned, the control law have to hugely correct the car position to follow the path. Furthermore, the kinematic car-like model implies a moving without skidding assumption which is very limitative as it implies a really low speed during navigation. This model does not either consider the slip angle which can't be canceled on a moving car. On the contrary, these are considered when planning with a dynamic vehicle model.

Furthermore, some works [4–6] aim to provide a safe path using localization technique such as Kalman filtering. For example, in [6], uncertainties are represented by an ellipsoid centered on an estimated point. As the estimated point is computed during the planning phase using the kinematic model, ellipsoids are misplaced. Moreover, if the skids and slids are added to the Kalman prediction process, ellipsoids size would be increased. This implies an unsafe path during the navigation process. On the contrary, using a dynamic model will lead to a better placed estimated points which considers skidding and sliding. Thus, ellipsoid will be smaller and better placed. We do not use localization algorithm in this paper but it could be easily added [7].

To avoid these matters, we propose in this paper the use of a vehicle model identical to the car that will have to follow the planned path.

We choose to use a Rapidly-exploring Random Tree (RRT) planner since it can easily take the dynamic model between two configurations. As its construction is incremental, we only have to integrate the dynamic system to obtain the new configuration using the present one and the control input. Using model with dynamic constraints is more complicated on a planner that uses a roadmap. Indeed, it is for example impossible to generate Dubin's curves [8] that respect nonholonomic and dynamic constraints.

This paper is organized as follows. In a first part, we present the two models used; first, the classical kinematic model and then the dynamic model we use. In the second part, we focus on the RRT planner by looking at the algorithm and its properties. Then, the dynamic model is integrated in the planner. Finally, in section 5, we compare and contrast the results obtained using the two different models.

2 Problem Statement

2.1 Vehicle model

We present in this section two different kinds of vehicle model. The first one is a simple kinematic model which is used in many path planning works. The second one is a dynamic model usually called bicycle model.

2.1.1 Kinematic model

The robot moves in a configuration space $\mathbf{X}$. A configuration is given by (x, y, θ) where x and y represent the coordinates of the characteristic point located midway of the rear wheels. θ represents the orientation of the mobile.

The classical car-like model represented on figure 1 uses a kinematic based differential system

$$\begin{cases} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan\delta \end{cases} \quad (1)$$

This system implies the moving without skidding assumption. It is to restrictive and does not permit to

0-7803-9521-2/06/$20.00 ©2006 IEEE.

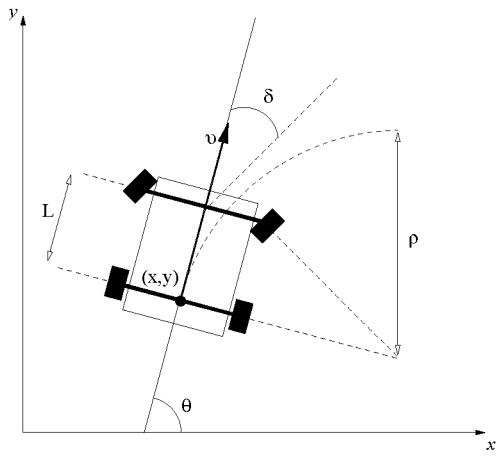


Figure 1.Kinematic model

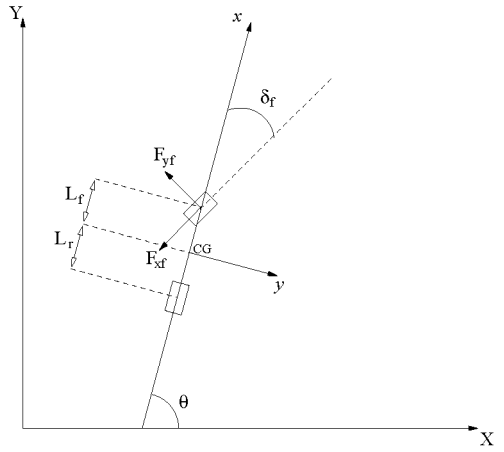


Figure 2.Simplified dynamic model

consider neither skidding when turning at high speed nor slip angle. So, using it in a planner implies lots of approximations. A path planned with this model could not be followed by a car without important corrections in the closed-loop control even if we use a good control law. This could lead to many collisions during the navigation process. As we want to reduce the correction during navigation, we use a more representative model of the car in the next section.

2.1.2 Dynamic model

In this paper, we use a five degrees of freedom dynamic car model. The state of the car is defined by $\mathbf{x} = (x_g, y_g, \theta, v_y, r)^T$. x_g and y_g represent the coordinates of the center of gravity of the robot. θ is still its orientation. v_y is the lateral speed and r is the yaw rate. This model is often simplified by projecting the front and the rear wheels on two virtual wheels located at the middle of the car. It is called the bicycle model (fig 2) based on [9].

The model we use in this paper is represented on figure 2. As it is often the case in commercialized vehicle, we only use one steering angle, the front one. Using the second Newton's law, the fundamental law of dynamics, we obtain :

$$\begin{cases} m(\dot{v}_x - v_y r) &= -F_{xf}\cos\delta_f - F_{yf}\sin\delta_f - F_{xr} \\ m(\dot{v}_y + v_x r) &= F_{yf}\cos\delta_f - F_{xf}\sin\delta_f + F_{yr} \\ I_z\dot{r} &= L_f\left(F_{yf}\cos\delta_f - F_{xf}\sin\delta_f\right) - L_r F_{yr} \end{cases} \tag{2}$$

In order to easily compare the results with those obtained with the kinematic model, we choose to use a constant longitudinal speed for both models. This implies a constant speed v during the whole planning phase for the differential system (1) and a constant longitudinal speed v_x for the dynamic model. Furthermore, as the aerodynamic resistance is neglected, the longitudinal tire force F_{xf} becomes zero.

Considering this, we obtain

$$\begin{cases} \dot{v}_y &= \frac{F_{yf}}{m}\cos\delta_f + \frac{F_{yr}}{m} - v_x r \\ \dot{r} &= \frac{L_f}{I_z}F_{yf}\cos\delta_f - \frac{L_r}{I_z}F_{yr}. \end{cases} \tag{3}$$

Thus, as a linear tire model is used, we can write

$$\begin{aligned} F_{yf} &= -C_{\alpha f}\alpha_f \\ F_{yr} &= -C_{\alpha r}\alpha_r \end{aligned}$$

where C_{xf} and C_{xr} are the cornering stiffness coefficients for front and rear tires. α_f and α_r are respectively the slips angles for the front and rear wheels. These angles are assumed to be small. So that, we obtain

$$\begin{aligned} \alpha_f &= \frac{v_y + L_f r}{v_x} - \delta_f \\ \alpha_r &= \frac{v_y - L_r r}{v_x}. \end{aligned}$$

L_f and L_r are the distance from the center of gravity to the front and rear wheel, respectively. r is the radius of the tire.

According to this, we obtain the full non-linear model

$$\begin{bmatrix} \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} A & C \\ B & D \end{bmatrix} \begin{bmatrix} v_y \\ r \end{bmatrix} + \begin{bmatrix} E \\ F \end{bmatrix} \delta_f \tag{4}$$

with

$$\begin{aligned} A &= -\frac{C_{\alpha f}\cos\delta_f + C_{\alpha r}}{m v_x}, \\ B &= \frac{-L_f C_{\alpha f}\cos\delta_f + L_r C_{\alpha r}}{m v_x} - v_x, \\ C &= \frac{-L_f C_{\alpha f}\cos\delta_f + L_r C_{\alpha r}}{I_z v_x}, \\ D &= -\frac{L_f^2 C_{\alpha f}\cos\delta_f + L_r^2 C_{\alpha r}}{I_z v_x}, \\ E &= \frac{C_{\alpha f}\cos\delta_f}{m}, \\ F &= \frac{L_f C_{\alpha f}\cos\delta_f}{I_z}. \end{aligned}$$

By integrating 4, we can compute the new coordinates (x_g, y_g, θ) of the mobile:

$$\begin{cases} \dot{x}_g &= v_x \cos(\theta) - v_y \sin(\theta) \\ \dot{y}_g &= v_x \sin(\theta) + v_y \cos(\theta) \\ \dot{\theta} &= r \end{cases} \quad (5)$$

We now have our five degrees of freedom non-linear model $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$. This state transition equation will be used in section 4 to compute $\mathbf{x}(t + \Delta t)$ using $\mathbf{x}(t)$ and the input $\mathbf{u}$.

2.2 Environment model

The planner uses an ideal 2D world map. Obstacles are represented by polygonal lines.

Set of n obstacles is represented by $\mathbf{X}_{obs}$.

$$\mathbf{X}_{obs} = \{\mathbf{X}^i_{obs}, i = 1...n\} \quad (6)$$

The part of $\mathbf{X}$ which is free of any obstacle is noted $\mathbf{X}_{free}$.

$$\mathbf{X}_{free} = \overline{\mathbf{X}_{obs}} \quad (7)$$

3 Rapidly-exploring Random Trees

3.1 Presentation of the algorithm

Rapidly-exploring Random Trees [10] is an incremental method to quickly explore the whole configuration space. The key idea is to visit the unexplored parts of the space by breaking the large Voronoi areas. The principle is simple (Alg. 1). First, $\mathbf{x}_{init}$ is added to the tree G. Then, a configuration $\mathbf{x}_{rand} \in \mathbf{X}_{free}$ is randomly chosen. The `nearest_neighbor` function searches in the tree the nearest node to $\mathbf{x}_{rand}$ according to a specified metric d. This node is called $\mathbf{x}_{near}$. A control input is then chosen, randomly or according to a specified criterion. Choosing the input that brings $\mathbf{x}_{new}$ as close as possible to $\mathbf{x}_{rand}$ could be a possible criterion. This input is then applied to $\mathbf{x}_{near}$. This is possible by integrating the equation 1 on a time increment Δt, using a numerical technique like Runge-Kutta. This creates a new point called $\mathbf{x}_{new}$. If the path between $\mathbf{x}_{near}$ and $\mathbf{x}_{new}$ lies in $\mathbf{X}_{free}$ (this is used to ensure that intermediate configurations lie in $\mathbf{X}_{free}$ when discrete jumps are made when integrating on Δt), $\mathbf{x}_{new}$ is added to G and the algorithm execution continues by choosing a new random configuration. Otherwise, $\mathbf{x}_{new}$ is not added to G.

A path is found when $\mathbf{x}_{new} = \mathbf{x}_{goal}$ or when

$$\mathbf{x}_{new} \in \mathbf{X}_{goal} \subset \mathbf{X}_{free} \quad (8)$$

where $\mathbf{X}_{goal}$ is the goal region.

Algorithm 1 RRT [10]

Function: RRT($K \in \mathbb{N}$, $\mathbf{x}_{init} \in \mathbf{X}_{free}$, $\Delta t \in \mathbb{R}$)
1: G.init($\mathbf{x}_{init}$)
2: **for** $i = 0$ to K **do**
3: $\mathbf{x}_{rand} \leftarrow$ random_config($\mathbf{X}_{free}$)
4: Extend(G, $\mathbf{x}_{rand}$)
5: **end for**
6: **return** G

Function: Extend(G, $\mathbf{x}_{rand}$)
1: $\mathbf{x}_{near} \leftarrow$ nearest_neighbor(G, $\mathbf{x}_{rand}$)
2: $\mathbf{u} \leftarrow$ select_input($\mathbf{x}_{rand}$, $\mathbf{x}_{near}$)
3: $\mathbf{x}_{new} \leftarrow$ new_state($\mathbf{x}_{near}$, $\mathbf{u}$, Δt)
4: **if** collision_free_path($\mathbf{x}_{near}$,$\mathbf{x}_{new}$) **then**
5: G.add_node($\mathbf{x}_{new}$)
6: G.add_edge($\mathbf{x}_{near}$, $\mathbf{x}_{new}$, $\mathbf{u}$)
7: **end if**
8: **return** G

3.2 Improvements

RRT is often slow to plan a path as it randomly reaches a defined $\mathbf{x}_{goal}$. To speed up the algorithm, we can bias it toward the goal by modifying the `random_config()` function (see algorithm 1, function RRT, line 3). Such modifications were proposed in [11] and called *goalbias* and *goalzoom*. Instead of choosing the random configuration in the whole configuration space, it sometimes (depending on a fixed probability p) picks the random configuration in a part of the space. *Goalbias* picks $\mathbf{x}_{goal}$ as random configuration $\mathbf{x}_{rand}$ with a probability p and *goalzoom* picks, with also a probability p, $\mathbf{x}_{rand}$ in a circle centered on $\mathbf{x}_{goal}$ with a radius of $\min_{\forall i} \mathrm{d}(\mathbf{x}_i, \mathbf{x}_{goal})$ where $\mathbf{x}_i$ represents the nodes of the tree.

Using the *connect* heuristic [12], the algorithm iterates the `extend()` function until a collision is detected. It allows to go quickly deeper to a given direction.

It is also possible to use two RRTs to plan paths faster. This bidirectional RRT [11] uses a RRT starting from $\mathbf{x}_{init}$ and another one starting from $\mathbf{x}_{goal}$. The path is planned when these two trees meet each other.

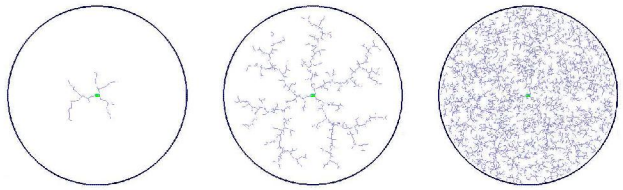
Observations in [11] provide comparisons between these improvements. The *connect* heuristic seems to explore more quickly the configuration space than *extend* in a holonomic case. On the contrary, *extend* seems to be quicker in a nonholonomic case. It also appears that *goalzoom* performances are better than *goalbias* ones as it gradually biases toward the goal.

3.3 Properties

RRT has really nice properties [10]. An interesting property for path planning problems is that a path planned with RRT does not need a local planner to find a way from a configuration to another. Indeed, the new point $\mathbf{x}_{new}$ is calculated to take nonholonomic constraints directly into account.

RRT expansion is biased toward unexplored parts of the configuration space by breaking large Voronoi regions. It allows the RRT to rapidly explore in the beginning, and then converge to a uniform coverage of the space.

The most interesting property, proved in [11, 12] is that the distribution of vertices is uniformly distributed in the configuration space. This involves the possibility of finding a path, if it exists, by generat-



(a) 100 points (b) 1000 points (c) 6000 points

Figure 3.RRT exploring a circle

ing an infinite number of vertices. That allows us to prove the probability completeness of the RRT algorithm. In figure 3, we show a RRT exploring a circle. As one can see, the distribution of vertices, after 6000 iterations of the algorithm, is uniform.

4 Integration of our model in the planner

The RRT is an incremental path planner. In this algorithm, the `new_state()` function allows us to choose which kind of model we want to use to represent the dynamic of our robot. Starting from a given configuration $\mathbf{x}_{near} = \mathbf{x}(t)$, we integrate our system (4) using a given input $\mathbf{u}$ to obtain $\mathbf{x}_{new} = \mathbf{x}(t + \Delta t)$, the configuration that will be added to the tree if it lies in $\mathbf{X}_{free}$.

The use of such a model would have been definitely more complicated with a non-incremental path planner like the Probabilistic Path Planner [2]. It implies the use of a local planner to compute a feasible path between two consecutive points of the path. It is indeed much more difficult to find a set of inputs that bring a vehicle with dynamic constraint from a configuration to an other than to integrate a system on a time increment Δt. Furthermore, dynamic constraints prevent the use of curves like Dubin's ones.

The integration of the systems in the `new_state` function is done using a numerical technique. In this paper, we do a fourth order Taylor's approximation and use it in a Runge-Kutta integration for both kinematic (1) and dynamic model (4):

$$\mathbf{x}_{n+1} \approx \mathbf{x}_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \qquad (9)$$

where

$$\begin{aligned} k_1 &= f(\mathbf{x}_n, \mathbf{u}_n)_{t_n}, \\ k_2 &= f\left(\mathbf{x}_n + \frac{k_1}{2}, \mathbf{u}_n\right)_{t_n + \frac{\Delta t}{2}}, \\ k_3 &= f\left(\mathbf{x}_n + \frac{k_2}{2}, \mathbf{u}_n\right)_{t_n + \frac{\Delta t}{2}}, \\ k_4 &= f\left(\mathbf{x}_n + k_3, \mathbf{u}_n\right)_{t_n + \Delta t}, \end{aligned}$$

and f is the state transition equation.

5 Results

To illustrate the efficiency of the RRT path planner with a kinematic and a dynamic model, we implemented it in C language on a 1.2GHz Pentium PC running Linux. We use $\Delta t = 200ms$ as time increment.

In a first part, we show classical results of a RRT planner using a Kinematic model.

Then, we show planning results using the dynamic model presented in section 2.1.2.

Finally, these results are compared.

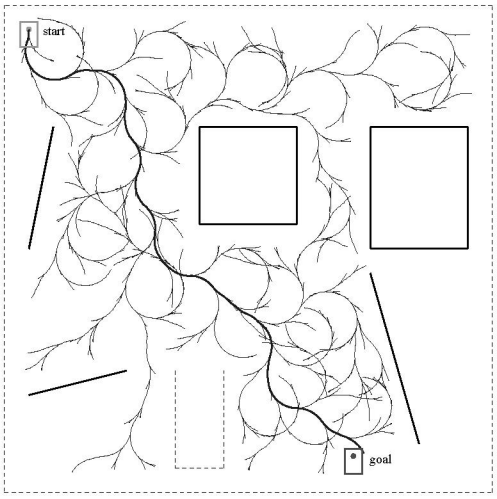


Figure 4.Path planned using RRT and kinematic model

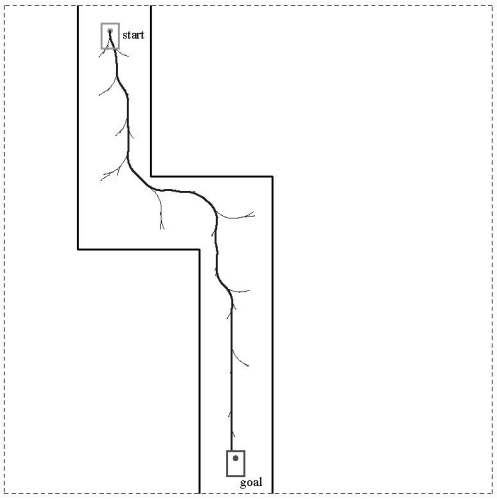


Figure 5.Obstacle avoidance using kinematic model

5.1 Results obtained with kinematic model

Some basic results using the kinematic model are shown in figures 4, 5 and 6.

In figure 4, we planned a path for a car that can only move forward (a Dubin's car [8]). In figure 5, we simulate an obstacle avoidance.

Figure 6 is computed in a more complicated environment which looks like a labyrinth. This example is interesting. It is indeed a kind of example where the RRT, using a kinematic car model with a low speed, can easily (quickly) find a path. As we are going to see in the next section, finding a path is much more difficult when using a dynamic car model with a higher speed.

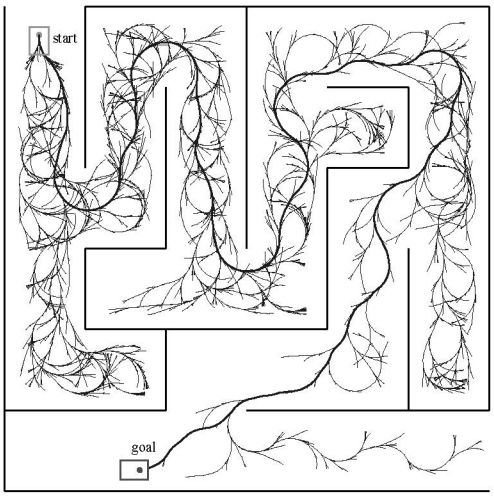


Figure 6.Kinematic model and labyrinth environment

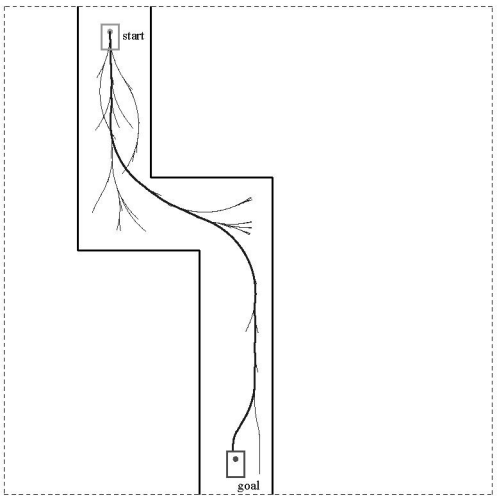


Figure 8.Obstacle avoidance using dynamic model

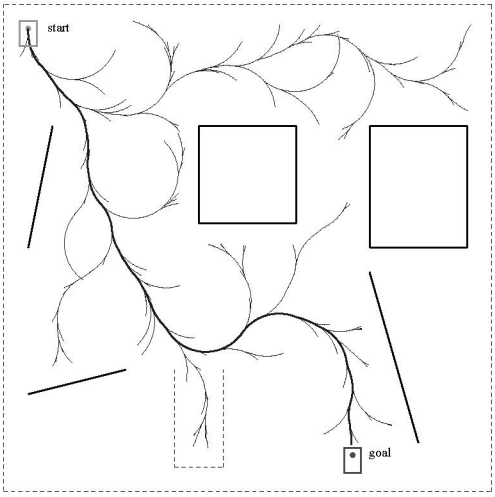


Figure 7.Path planned using RRT and dynamic model

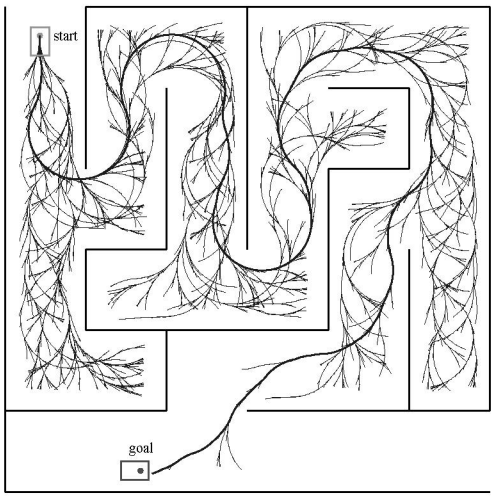


Figure 9.Dynamic model and labyrinth environment

5.2 Results obtained with dynamic model

In this section, we use the same environments as those used in the previous section. But, instead of planning using the kinematic model and the low speed assumption, we use a dynamic model with higher speed. The results can thus be compared between planning using these two models.

In figure 7, we use the same environment as in figure 4. In figure 8, we show another exemple using the same environment as in figure 5 with a really high speed.

The main visible difference is located on the turning radius value. Using our dynamic vehicle model, planned paths are smoother and the turning radius greatest lower bound is bigger than those obtained with kinematic model. This is a normal consequence of the high speed assumption since a car turning at high speed with a big steering angle will skid. This imply that the planner has more diffilculties to find a path that lies in $\mathbf{X}_{free}$.

Considering the obstacle avoidance situation, the path planned in figure 5 is not realistic. Indeed, during the navigation process, a car moving at a moderate or high speed could not follow such a trajectory when applying the commands returned by the planner. This would lead to skids and collisions. On the contrary, the path planned with a dynamic model is smooth and considers the skidding. It can be followed during navigation.

Figure 9 is also an explicit example as it can be directly compared to figure 6. Using the kinematic model, a path is found without any difficulty. On the contrary, using a dynamic model and a higher speed, it is quite hard to find a path which do not imply some collisions. High speed implies a bigger turning radius. Corridors are however narrows. That is why lots of paths could not be admissible without collisions.

5.3 Computation time comparison

The time spent to generate a node of the RRT is globally the same when using dynamic or kinematic models. We indeed use the same planning algorithm and integration technique. The integration is a little bit longer for the dynamic model since the number of degrees of freedom is bigger.

The computation time needed for an iteration of the RRT algorithm is however negligible compared to

Table 1.Computation time using kinematic and dynamic vehicle model; labyrinth example

	Kinematic model	Dynamic model
Time (s)	4.4	11.8

the time needed to compute all the iterations when exploring the state space between $\mathbf{x}_{init}$ and $\mathbf{x}_{goal}$.

In algorithm 1, if the `new_state()` function detects a collision, $\mathbf{x}_{new}$ is not created nor added to the tree. So, some iterations of the algorithm do not aim to the creation of a new node in the tree. This leads to a loss of time. As we can see in figure 9, the number of nodes in the tree is bigger than in figure 6. Resolving such a path planning problem needs to generate in mean 70000 nodes using the dynamic vehicle model and 30000 using the kinematic one. This is due, as explained in section 5.2, to the fact that the planner has difficulties to stay in $\mathbf{X}_{free}$ when considering a dynamic car that moves at high speed.

So, when planning in an environment where collisions could happened like the labyrinth in figures 6 and 9, the planning process could be more than 3 times slower when using a dynamic vehicle model (see table 1).

6 Conclusion

In this paper, we presented an efficient way to plan realistic paths using a RRT path planner which embedded a dynamic vehicle model. The methodology we propose is general. It can be easily extended to other kind of vehicle or models. We could imagine to extend our work to more efficient dynamic models using Pacejka's [13] or Dugoff's [14] formulas. We only have to modify the size of the vehicle and replace our dynamic model with another one in the `new_state()` function of algorithm.

Work about the quality of the path following as well as the realism of our dynamic model has not been done yet. This should form the subject of future studies. We will also study the computation time and try to reduce it in order to use it in a real-time application.

7 References

[1] J. P. Laumond, S. Sekhavat, and F. Lamiraux, *Robot Motion Planning and Control.* J.-P. Laumond, 1998, vol. Guidelines in nonholonomic motion planning for mobile robots.

[2] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *International Trans. Robotics and Automation*, vol. 12, no. 4, pp. 566–580, June 1996.

[3] E. Mazer, J. M. Ahuactzin, and P. Bessiere, "The ariadne's clew algorithm," *Journal of Artificial Intelligence Research*, vol. 9, pp. 295–316, November 1998.

[4] B. Bouilly, T. Simeon, and R. Alami, "A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty," in *Proc. IEEE International Conference on Robotics and Automation*, 1995, pp. 1327–1332.

[5] T. Fraichard and R. Mermond, "Path planning with uncertainty for car-like robots," in *Proc. IEEE International Conference on Robotics and Automation*, 1998, pp. 27–32.

[6] A. Lambert and D. Gruyer, "Safe path planning in an uncertain-configuration space," *IEEE International Conference on Robotics and Automation*, pp. 4185–4190, May 2003.

[7] A. Lambert and N. L. Fort-Piat, "Safe task planning integrating uncertainties and local map federation," *International Journal of robotics Research*, vol. 19, no. 6, pp. 587–611, 2000.

[8] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed and terminal position tangents." *American J. of Math.*, no. 79, pp. 497–516, 1957.

[9] S. Taheri, "An investigation and design of slip control braking systems integrated with four wheel steering," Ph.D. dissertation, Clemson University, 1990.

[10] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Dept., Iowa State University, Tech. Rep., November 1998.

[11] S. M. LaValle and J. J. Kuffner, *Algorithmic and Computational Robotics: New Directions.* B. R. Donald and K. M. Lynch and D. Rus, 2001, ch. Rapidly-exploring random trees: Progress and prospects, pp. 293–308.

[12] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," *IEEE International Conference on Robotics and Automation (ICRA'2000), San Francisco, CA*, pp. 995–1001, 2000.

[13] H. B. Pacejka and J. M. Besselink, "Magic formula tyre model with transient properties," *International Colloquium on Tyre Models for Vehicle Dynamic Analysis*, pp. 234–249, 1997.

[14] J. Dugoff, P. Fanches, and L. Segel, "An analysis of tire traction properties and their influence on vehicle dynamic performance," *SAE*, 1970.