RRT Path Planning using a Dynamic Vehicle Model

Path Planning using a Dynamic Vehicle Model
Romain Pepy, Alain Lambert, Hugues Mounier
Information and Communication Technologies, 2006. ICTTA ’06. 2nd

Ross Hatton, Siddharth Sanan

November 7, 2007
Path planning for a robot with constraints
  - Simple: kinematic constraints
  - Harder: dynamic constraints

RRT

Runge-Kutta (RK4) integration
Kinematic car

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \frac{v}{L} \tan \delta
\end{align*}
\]
Dynamic car 1

\[ \mathbf{x} = (x_g, y_g, \theta, v_y, r)^T \]

\[
m(\dot{v}_x - v_y r) = -F_{xf} \cos \delta_f - F_{yf} \sin \delta_f - F_{xr} \]
\[
m(\dot{v}_y + v_x r) = F_{yf} \cos \delta_f - F_{xf} \sin \delta_f + F_{yr} \]
\[
I_z \dot{r} = L_f (F_{yf} \cos \delta_f - F_{xf} \sin \delta_f) - L_r F_{yr} \]
Dynamic car 2

\[
\dot{v}_y = \frac{F_{yf}}{m} \cos \delta_f + \frac{F_{yr}}{m} - v_x r \\
\dot{r} = \frac{L_f}{I_z} F_{yf} \cos \delta_f - \frac{L_x}{I_z} F_{yr}.
\]
Dynamic car 3 - Tire slip

\[ F_{yf} = -C_{\alpha_f} \alpha_f \]
\[ F_{yr} = -C_{\alpha_r} \alpha_r \]

\[ \alpha_j \quad \nu_{roll} \]
\[ \alpha \quad \nu_{slip} \]
Dynamic car 4

\[
\begin{align*}
\dot{x}_g &= v_x \cos(\theta) - v_y \sin(\theta) \\
\dot{y}_g &= v_x \sin(\theta) + v_y \cos(\theta) \\
\dot{\theta} &= r
\end{align*}
\]

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

\[
A = \frac{-C_{\alpha f} \cos \delta_f + C_{\alpha r}}{mv_x},
B = \frac{-L_f C_{\alpha f} \cos \delta_f + L_r C_{\alpha r}}{mv_x},
C = \frac{-L_f C_{\alpha f} \cos \delta_f + L_r C_{\alpha r}}{I_z v_x},
D = \frac{-L_f^2 C_{\alpha f} \cos \delta_f + L_r^2 C_{\alpha r}}{I_z v_x},
E = \frac{C_{\alpha f} \cos \delta_f}{m},
F = \frac{L_f C_{\alpha f} \cos \delta_f}{I_z}.
\]
Rapidly Exploring Random Trees

- A sampling based planner
- Tree $G$ is started by adding an initial point $x_{init}$
- A random configuration $x_{rand} \in X_{free}$ is chosen
- The nearest neighbor function finds $x_{near}$, the node of $G$ that offers the best approach to $x_{rand}$
- An input $u$ is chosen to drive the robot from $x_{near}$ towards $x_{rand}$
- Equations of motion integrated for a single time step to bring robot to $x_{new}$
- If $x_{new} \in X_{free}$, add it to $G$
- Path is found when $x_{new} \in X_{free} \cap X_{goal}$

**Algorithm 1 RRT [10]**

Function: \( \text{RRT}(K \in \mathbb{N}, x_{init} \in X_{free}, \Delta t \in \mathbb{R}) \)

1. \( G.\text{init}(x_{init}) \)
2. \( \text{for } i = 0 \text{ to } K \text{ do} \)
3. \( x_{rand} \leftarrow \text{random.config}(X_{free}) \)
4. \( \text{Extend}(G, x_{rand}) \)
5. \( \text{end for} \)
6. \( \text{return } G \)

Function: \( \text{Extend}(G, x_{rand}) \)

1. \( x_{near} \leftarrow \text{nearest.neighbor}(G, x_{rand}) \)
2. \( u \leftarrow \text{select.input}(x_{rand}, x_{near}) \)
3. \( x_{new} \leftarrow \text{new.state}(x_{near}, u, \Delta t) \)
4. \( \text{if collision.free.path}(x_{near}, x_{new}) \text{ then} \)
5. \( G.\text{add.node}(x_{new}) \)
6. \( G.\text{add.edge}(x_{near}, x_{new}, u) \)
7. \( \text{end if} \)
8. \( \text{return } G \)
RRTs with kinematic constraints

\[
\begin{align*}
\dot{x} &= v \cos \theta \\
\dot{y} &= v \sin \theta \\
\dot{\theta} &= \frac{v}{L} \tan \delta
\end{align*}
\]

- \( \delta \) is the controlled variable
- Kinematic equations restrict allowable paths
- new_state function respects restriction
- Any path found by the planner will respect the nonholonomic constraints on the car
RRT with dynamic constraints

- Greater lower bound on turning radius
- Smoother paths
- Tighter turns at speed cause slip
- Fewer open paths in $X_{\text{free}}$
Relative performance

Kinematic

Dynamic

Figure 3. RRT exploring a circle with an infinite number of vertices. This is much more difficult when using a dynamic car model with a higher speed.

4. Path planning in a larger environment is needed.

5. Obstacle avoidance using dynamic model.


7. Path planned using RRT and dynamic model.

5.2 Results... compared to

0-7803-9521-2/06/$20.00 ©2006 IEEE. 784

Computation time comparison

- Approximately equal iteration time for both models.
- Slightly higher integration time on dynamic model to deal with extra degrees of freedom.
- Significant increase in overall computation time for dynamic model.

<table>
<thead>
<tr>
<th></th>
<th>Kinematic</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (s)</td>
<td>4.4</td>
<td>11.8</td>
</tr>
</tbody>
</table>

- Not all iterations produce a node in $X_{free}$.
- Computation time lost each time a node is thrown out.
- Dynamic vehicle can’t turn as fast, so more bad nodes.
- 30k nodes needed for kinematic model, 70k for dynamic model.
Runge-Kutta integration 1

- Numerical integration
- Euler’s method doesn’t account for changes in force during time step
  - Error $O(\Delta t)$
- Runge-Kutta methods add extra terms
  - Weighted average of integrand at $t_n$, $t_n + \frac{\Delta t}{2}$ and $t_n + \Delta t$
  - More computational cost per step than Euler
  - Error $O(\Delta t^4)$
Runge-Kutta integration 2

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

\[ k_1 = f(x_n, u_n)_{t_n}, \]
\[ k_2 = f \left( x_n + \frac{k_1}{2}, u_n \right)_{t_n + \frac{\Delta t}{2}}, \]
\[ k_3 = f \left( x_n + \frac{k_2}{2}, u_n \right)_{t_n + \frac{\Delta t}{2}}, \]
\[ k_4 = f \left( x_n + k_3, u_n \right)_{t_n + \Delta t}, \]
Conclusions

- Path planning for a robot with constraints
  - Simple: kinematic constraints
  - Harder: dynamic constraints
- RRT
- Runge-Kutta (RK4) integration