> By SunHaoOne 2022.3.8

Frenet坐标转换公式推导

一、为什么使用Frenet坐标系?

- Frenet可以将车辆某一时刻的轨迹投影到参考轨迹上，并分解成横向和纵向两个维度的运动，能简化后面的轨迹规划。
- Frenet是根据参考轨迹进行建立的，在这个参考系下，任意一点都可以看成是横向单位向量和纵向单位向量的线性组合。因此在Frenet下做轨迹规划有天然的优势。

二、Frenet坐标系的基本定义

1. 定义弧长

令$\vec{r}(t)$是欧式空间内随t改变的一条非退化曲线，非退化曲线是一条不会退化成直线的曲线，即曲率不为零。令$s(t)$是t时刻曲线的累计弧长，定义为：

$$S(t)=\int_0^t ||r'(\sigma)||d\sigma \tag{1}$$

> 这么定义的好处是，如果两边同时求导，那么有$s'(t)=||r'(t)||$

如果$r\prime(t)\neq 0$，那么说明$s(t)$是单调递增函数，因此$s=s(t)\rightarrow t=t(s)$，也就是说$\vec{r}(t)=\vec{r}(t(s))$，通过这个复合函数可以看出，$r=\vec{r}(s)$表示曲线是弧长的函数。

2. 推导Frenet-Serret公式的矩阵形式

由于曲线是弧长的函数，我们定义三个向量，分别是切向量$\vec{T}$、法向量$\vec{N}$和副法向量$\vec{B}$。注意，这三个方向的向量都是单位向量。

$$\begin{aligned}\vec{T} &= \frac{\frac{d\vec{r}}{ds}}{||\frac{d\vec{r}}{ds}||}\\ \vec{N} &= \frac{\frac{d\vec{T}}{ds}}{||\frac{d\vec{T}}{ds}||}\\ \vec{B} &= \vec{T}\times\vec{N}\end{aligned} \tag{2}$$

如果我们定义κ是曲线的曲率，定义τ是曲线的挠率，那么基于上述定义的Frenet-Serret公式可以表示为

$$\begin{aligned}\frac{d\vec{T}}{ds} &= \kappa\vec{N}\\ \frac{d\vec{N}}{ds} &= -\kappa\vec{T}+\tau\vec{B}\\ \frac{d\vec{B}}{ds} &= -\tau\vec{N}\end{aligned} \tag{3}$$

这里的第一个和第三个容易推导。以第一个为例：

回忆一下曲率的定义：

$$\kappa = lim_{\Delta s\to 0}\frac{\Delta\theta}{\Delta s} \tag{4}$$

$$\begin{aligned}\frac{d\vec{T}}{ds} &= lim_{\Delta s\to 0}\frac{\vec{T}_1-\vec{T}_2}{\Delta s}\\ &= lim_{\Delta s\to 0}\frac{\Delta\theta\cdot\vec{N}}{\Delta s}\\ &= \kappa\vec{N}\end{aligned} \tag{5}$$

第二个公式如何推导呢？

容易看出，这个是两个向量的线性叠加

$$\frac{d\vec{N}}{ds} = a\vec{T}+b\vec{B} \tag{6}$$

而根据定义，又有两组垂直的关系。

$$\vec{N}\cdot\vec{T} = 0$$

$$
\vec{N} \cdot \vec{T} = 0 \\
\vec{N} \cdot \vec{B} = 0 \tag{7}
$$

由于是单位向量，还有如下的已知条件

$$
\begin{aligned}
\vec{T} \cdot \vec{T} = ||\vec{T}|| = 1 \\
\vec{N} \cdot \vec{N} = ||\vec{N}|| = 1 \\
\vec{B} \cdot \vec{B} = ||\vec{B}|| = 1
\end{aligned} \tag{8}
$$

因此可以推得：

$$
\begin{aligned}
\frac{d(\vec{N} \cdot \vec{T})}{ds} &= \frac{d\vec{N}}{ds} \cdot \vec{T} + \vec{N} \cdot \frac{d\vec{T}}{ds} \\
&= (a\vec{T} + b\vec{B}) \cdot \vec{T} + \vec{N} \cdot \kappa \vec{N} \\
&= a\vec{T} \cdot \vec{T} + \kappa \vec{N} \cdot \vec{N} \\
&= a + \kappa \\
&= 0
\end{aligned} \tag{9}
$$

因此$a = -\kappa$,同理可以推得$b = \tau$

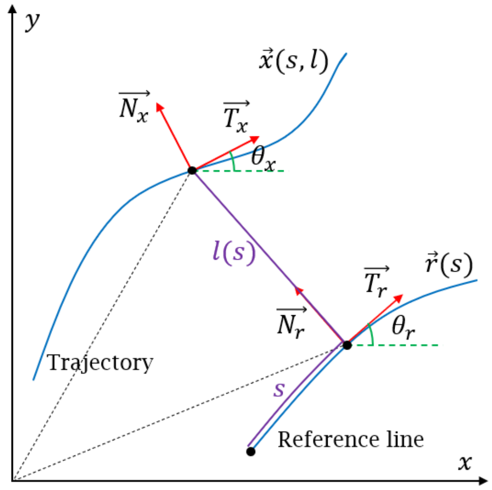
$$
\frac{d\vec{N}}{ds} = -\kappa \vec{T} + \tau \vec{B} \tag{10}
$$

$$
\begin{bmatrix} \frac{d\vec{T}}{ds} \\ \frac{d\vec{N}}{ds} \\ \frac{d\vec{B}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \begin{bmatrix} T \\ N \\ B \end{bmatrix} \tag{11}
$$

而对于无人驾驶车辆来说，对高度信息不感兴趣，因此$\tau = 0$时，上式可以简化为

$$
\begin{bmatrix} \frac{d\vec{T}}{ds} \\ \frac{d\vec{N}}{ds} \end{bmatrix} = \begin{bmatrix} 0 & \kappa \\ -\kappa & 0 \end{bmatrix} \begin{bmatrix} T \\ N \end{bmatrix} \tag{12}
$$

三、Frenet坐标系与笛卡尔坐标系的转换



A. 为了方便表示，定义两组导数的定义：

$$\frac{dx}{dt} = \dot{x}, \frac{dx}{ds} = x' \tag{13}$$

B. 将切向量和法向量与角度相对应，得到四组公式：

$$\begin{aligned} \vec{T_r} &= [cos\theta_r, sin\theta_r] \\ \vec{N_r} &= [-sin\theta_r, cos\theta_r] \\ \vec{T_x} &= [cos\theta_x, sin\theta_x] \\ \vec{N_x} &= [-sin\theta_x, cos\theta_x] \end{aligned} \tag{14}$$

1. 推导x、r和l的导数

根据向量的运算有

$$\vec{x}(s,l) = \vec{r}(s) + l(s)\vec{N_r} \tag{15}$$

在后续的推导中，省略自变量继续推导

$$\begin{aligned} \vec{x} &= \vec{r} + l\vec{N_r} \\ l\vec{N_r} &= [\vec{x} - \vec{r}] \\ l\vec{N_r}^T\vec{N_r} &= \vec{N_r}^T[\vec{x} - \vec{r}] \\ l &= \vec{N_r}^T[\vec{x} - \vec{r}] = [\vec{x} - \vec{r}]^T\vec{N_r} \end{aligned} \tag{16}$$

两边同时对时间求导，则有

$$\dot{l} = [\dot{\vec{x}} - \dot{\vec{r}}]^T\vec{N_r} + \dot{\vec{N_r}}^T[\vec{x} - \vec{r}] \tag{17}$$

$$= [\dot{\vec{x}} - \dot{\vec{r}}]^T \vec{N}_r + [\vec{x} - \vec{r}]^T \dot{\vec{N}}_r$$

回顾单位切向量和单位法向量的定义：

$$\begin{aligned} \dot{\vec{x}} = \frac{d\vec{x}}{dt}, \vec{x} = ||\vec{x}||\vec{T}_x \\ \dot{\vec{r}} = \frac{d\vec{r}}{dt}, \vec{r} = ||\vec{r}||\vec{T}_r \end{aligned} \tag{18}$$

以及根据弧长的定义公式得到的结论$s'(t) = ||r'(t)||$

由于单位向量是和时间没有关系的，因此可以拿到外面来。

$$\begin{aligned} \dot{\vec{x}} = \frac{d\vec{x}}{dt} = \frac{d(||\vec{x}||\vec{T}_x)}{dt} = \frac{d||\vec{x}||}{dt}\vec{T}_x = v_x\vec{T}_x \\ \dot{\vec{r}} = \frac{d\vec{r}}{dt} = \frac{d(||\vec{r}||\vec{T}_r)}{dt} = \frac{d||\vec{r}||}{dt}\vec{T}_r = \dot{s}\vec{T}_r \end{aligned} \tag{19}$$

2. 推导l和T、N的关系

$$\dot{\vec{N}}_r = \frac{d\dot{N}_r}{dt} = \frac{d\dot{N}_r}{ds}\frac{ds}{dt} \tag{20}$$

由Frenet-Serret的定义可知

$$\frac{d\vec{N}}{ds} = -\kappa\vec{T} \tag{21}$$

那么有

$$\dot{\vec{N}}_r = \frac{d\dot{N}_r}{ds} = \frac{d\dot{N}_r}{dt}\frac{ds}{dt} = -\kappa_r\dot{s}\vec{T}_r \tag{22}$$

那么代入可得

这里的推导主要应用到了向量正交来化简，即

$$\vec{T}_r^{\,T}\vec{N}_r = 0 \tag{23}$$

$$
\begin{aligned}
\dot{l} &= [\dot{\vec{x}} - \dot{\vec{r}}]^T \vec{N}_r + [\vec{x} - \vec{r}]^T \dot{\vec{N}}_r \\
&= [v_x \vec{T}_x - \dot{s}\vec{T}_r]^T \vec{N}_r + l{\vec{N}_r}^T \dot{\vec{N}}_r \\
&= [v_x \vec{T}_x - \dot{s}\vec{T}_r]^T \vec{N}_r - l{\vec{N}_r}^T \kappa \dot{s} \vec{T}_r \\
&= v_x {\vec{T}_x}^T \vec{N}_r
\end{aligned} \tag{24}
$$

3. 求解l的导数和Vx

回顾一下向量与角度的其中两组公式

$$
\begin{aligned}
\vec{N}_r &= [-sin\theta_r, cos\theta_r] \\
\vec{T}_x &= [cos\theta_x, sin\theta_x]
\end{aligned} \tag{25}
$$

那么继续计算

$$
{\vec{T}_x}^T \vec{N}_r = [cos\theta_x, sin\theta_x]^T [-sin\theta_r, cos\theta_r] = sin\theta_x cos\theta_r - cos\theta_x sin\theta_r = sin(\theta_x - \theta_r) \tag{26}
$$

令$\Delta\theta = \theta_x - \theta_r$，则有

$$
\dot{l} = v_x sin\Delta\theta \tag{27}
$$

接下来计算v_x，

$$
v_x = \frac{d||\vec{x}||}{dt} = ||\frac{d\vec{x}}{dt}|| = ||\dot{x}|| \tag{28}
$$

回顾几何关系公式

$$
\vec{x} = \vec{r} + l\vec{N}_r \tag{29}
$$

再回顾两组公式，第一个公式由弧长的定义得到了导数关系，因此再乘以方向单位向量即可。

$$
\begin{aligned}
\dot{\vec{r}} &= \dot{s}\vec{T}_r \\
\dot{\vec{N}}_r &= -\kappa_r \dot{s} \vec{T}_r
\end{aligned} \tag{30}
$$

$$
\dot{\vec{x}} = \frac{d(\vec{r} + l\vec{N}_r)}{dt} = \dot{\vec{r}} + \dot{l}\vec{N}_r + l\dot{\vec{N}}_r \tag{31}
$$

$$
\begin{aligned}
&= \dot{s}\vec{T}_r + \dot{l}\vec{N}_r - \kappa l \dot{s}\vec{T}_r \\
&= \dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r
\end{aligned}
$$

而

$$
\begin{aligned}
v_x = ||\dot{x}|| &= \sqrt{\dot{x}^T\dot{x}} \\
\dot{x}^T\dot{x} &= [\dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r]^T[\dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r] \\
&= [\dot{s}(1-\kappa_r l)\vec{T}_r{}^T + \dot{l}\vec{N}_r{}^T][\dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r] \\
&= \dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2
\end{aligned} \tag{32}
$$

因此可以推得

$$
v_x = \sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \tag{33}
$$

4. 求解s的导数

由于

$$
\begin{aligned}
\dot{\vec{x}} &= \dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r \\
\dot{\vec{x}}\vec{T}_r{}^T &= \dot{s}(1-\kappa_r l)\vec{T}_r\vec{T}_r{}^T + \dot{l}\vec{N}_r\vec{T}_r{}^T \\
\dot{\vec{x}}\vec{T}_r{}^T &= \dot{s}(1-\kappa_r l)
\end{aligned} \tag{34}
$$

而由前文的导数推导可以知道

$$
\dot{\vec{x}} = v_x\vec{T}_x \tag{35}
$$

因此

$$
\begin{aligned}
\dot{s} = \frac{\dot{\vec{x}}\vec{T}_r{}^T}{1-\kappa_r l} = \frac{v_x\vec{T}_x\vec{T}_r{}^T}{1-\kappa_r l} = \frac{v_x cos\Delta\theta}{1-\kappa_r l} \\
\vec{T}_x\vec{T}_r = [cos\theta_r, sin\theta_r][cos\theta_x, sin\theta_x]^T = cos\theta_r cos\theta_x - sin\theta_r sin\theta_x = cos(\theta_x - \theta_r) = cos\Delta\theta
\end{aligned} \tag{36}
$$

中场休息

目前推导了三个公式及其转换关系

$$
\dot{l} = v_x sin\Delta\theta
$$

$$
\begin{gathered}
v_x = \sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \\
\dot{s} = \frac{v_x cos\Delta\theta}{1-\kappa_r l}
\end{gathered} \tag{37}
$$

5. 求解l对s的导数

对于$\dot{s}$

定义为：$\dot{s} = \frac{d\|\vec{r}\|}{dt} = \left\|\frac{d\vec{r}(t)}{dt}\right\|$

可以看出式曲线r(t)得导数向量的模，理论上

如果参考轨迹r(t)已知，可以求出$\dot{s}$

这部分主要将前文对时间的导数转换成对弧长的导数。

$$
\begin{gathered}
l' = \frac{dl}{ds} = \frac{dl}{dt}\frac{dt}{ds} = \frac{\dot{l}}{\dot{s}} \\
\dot{l} = l'\dot{s}
\end{gathered} \tag{38}
$$

$$
\begin{aligned}
l' &= \frac{v_x sin\Delta\theta}{\dot{s}} \\
&= \frac{\sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \cdot sin\Delta\theta}{\dot{s}} \\
&= \frac{\sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \cdot sin\Delta\theta}{\dot{s}} \\
&= \frac{\sqrt{\dot{s}^2(1-\kappa_r l)^2 + (l'\dot{s})^2} \cdot sin\Delta\theta}{\dot{s}} \\
&= \sqrt{(1-\kappa_r l)^2 + l'^2} \cdot sin\Delta\theta
\end{aligned} \tag{39}
$$

进一步地，两边同时平方

$$
\begin{gathered}
l'^2 = [(1-\kappa_r l)^2 + l'^2]sin^2\Delta\theta \\
l' = (1-\kappa_r l)tan\Delta\theta
\end{gathered} \tag{40}
$$

类似地，定义s_x为轨迹x的弧长

$$
v_x = \frac{ds_x}{dt} \tag{41}
$$

$$\frac{d}{ds} = \frac{d}{ds_x}\frac{ds_x}{ds} = \frac{ds_x}{dt}\frac{dt}{ds}\frac{d}{ds_x} = \frac{v_x}{\dot{s}}\frac{l}{ds_x} = \frac{1-\kappa_r l}{cos\Delta\theta}\frac{d}{ds_x} \tag{42}$$

6. 求解l对s的二阶导数

由于$l' = (1-\kappa l)tan\Delta\theta$，对其求导数则有

$$\begin{aligned} l'' &= \frac{dl'}{ds} \\ &= (1-\kappa_r l)' tan\Delta\theta + (1-\kappa_r l)\frac{(\Delta\theta)'}{cos^2\Delta\theta} \\ &= -(\kappa_r' l + \kappa_r l')tan\Delta\theta + (1-\kappa_r l)\frac{(\Delta\theta)'}{cos^2\Delta\theta} \end{aligned} \tag{43}$$

$$(\Delta\theta)' = \frac{d(\theta_x - \theta_r)}{ds} = \frac{d}{ds}\theta_x - \frac{d}{ds}\theta_r \tag{44}$$

> 根据曲率的定义，则有
>
> $$\begin{aligned} \kappa_r &= \frac{d\theta_r}{ds} \\ \kappa_x &= \frac{d\theta_x}{ds_x} \end{aligned} \tag{45}$$

因此有

$$\begin{aligned} \frac{d}{ds}\theta_x &= \frac{1-\kappa_r l}{cos\Delta\theta}\frac{d}{ds_x}\theta_x \\ &= \frac{1-\kappa_r l}{cos\Delta\theta}\frac{d\theta_x}{ds_x} \\ &= \frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x \end{aligned} \tag{46}$$

$$(\Delta\theta)' = \frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r \tag{47}$$

则代入得到l''为

$$\begin{aligned} l'' &= -(\kappa_r' l + \kappa_r l')tan\Delta\theta + (1-\kappa_r l)\frac{(\Delta\theta)'}{cos^2\Delta\theta} \\ &= -(\kappa_r' l + \kappa_r l')tan\Delta\theta + \frac{1-\kappa_r l}{cos^2\Delta\theta}[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r] \end{aligned} \tag{48}$$

7. 求解加速度

$$a_x = \dot{v_x} = \frac{dv_x}{dt} \tag{49}$$

回顾前面推导的三组公式

$$\dot{s} = \frac{v_x cos\Delta\theta}{1-\kappa_r l} \tag{50}$$

微分公式

$$\frac{d}{ds} = \frac{1-\kappa_r l}{cos\Delta\theta}\frac{d}{ds_x} \tag{51}$$

$$(\Delta\theta)' = \frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r \tag{52}$$

$$\begin{aligned} v_x &= \frac{\dot{s}(1-\kappa_r l)}{cos\Delta\theta} \\ a_x &= \frac{dv_x}{dt} \\ &= \ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta} + \dot{s}\frac{d\frac{(1-\kappa_r l)}{cos\Delta\theta}}{ds}\frac{ds}{dt} \\ &= \ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta} + \dot{s}\frac{d\frac{(1-\kappa_r l)}{cos\Delta\theta}}{ds}\dot{s} \end{aligned} \tag{53}$$

接下来去求解右半部分的导数

$$\begin{aligned} \frac{d\frac{(1-\kappa l)}{cos\Delta\theta}}{ds} &= \frac{-(\kappa_r' l + \kappa_r l')cos\Delta\theta + (1-\kappa_r l)sin\Delta\theta}{(cos\Delta\theta)^2} \\ &= \frac{-(\kappa_r' l + \kappa_r l') + (1-\kappa_r l)tan\Delta\theta(\Delta\theta)'}{cos\Delta\theta} \\ &= \frac{1}{cos\Delta\theta}[-(\kappa_r' l + \kappa_r l') + (1-\kappa_r l)tan\Delta\theta[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r]] \end{aligned} \tag{54}$$

最终我们得到了加速度的公式

$$a_x = \ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta} + \dot{s}\frac{d\frac{(1-\kappa_r l)}{cos\Delta\theta}}{ds}\dot{s} \tag{55}$$

$$= \ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta} + \dot{s}^2\frac{1}{cos\Delta\theta}[-(\kappa_r' l + \kappa_r l') + (1-\kappa_r l)tan\Delta\theta[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r]]$$

四、坐标转换总结

https://fjp.at/posts/optimal-frenet/

1. 定义

A. 笛卡尔坐标系下的变量

$$[\vec{x}, v_x, a_x, \theta_x, \kappa_x] \tag{56}$$

- $\vec{x}$为笛卡尔坐标系下的坐标用向量表示
- θ_x为$\vec{x}$与X轴的夹角
- $v_x = ||\dot{\vec{x}}||$为笛卡尔坐标系下的速度
- $\kappa_x = \frac{d\theta_x}{d\theta s}$为曲率
- $a_x = \frac{dv_x}{dt}$为笛卡尔坐标系下的加速度

B. Frenet坐标系下的变量

$$[s, \dot{s}, \ddot{s}, l, \dot{l}, l', l''] \tag{57}$$

> With Frenet coordinates, we use the variables s and d to describe a vehicle's position on the road or a reference path. The s coordinate represents distance along the road (also known as longitudinal displacement) and the d coordinate represents side-to-side position on the road (relative to the reference path), and is also known as lateral displacement.
>
> - 即沿着道路的方向 `s` 为纵坐标，表示道路的左右位置为横向位移 `l` 横坐标
> - 在代码中用 `d` 来表示 `l`

- s为Frenet坐标系下的纵坐标
  - $\dot{s} = \frac{ds}{dt}$为Frenet纵坐标对时间的导数，即沿着base frame的速度
  - $\ddot{s} = \frac{d\dot{s}}{dt}$为沿着base fame的加速度
- l为Frenet坐标系下的横坐标
  - $\dot{l} = \frac{dl}{dt}$为Frenet纵坐标对时间的导数，为横向速度
  - $\ddot{l} = \frac{dl}{dt}$为横向加速度
  - $l' = \frac{dl}{ds}$为Frenet横坐标对纵向坐标的导数
  - $l'' = \frac{dl'}{ds}$为Frenet横向坐标对纵向坐标的二阶导数

2. 如何由笛卡尔坐标系转换为Frenet坐标系?

$$[\vec{x}, v_x, a_x, \theta_x, \kappa_x] \rightarrow [s, \dot{s}, \ddot{s}, l, \dot{l}, l', l''] \tag{58}$$

A. 求解s和l

> 问题描述如下：已知$\vec{x} = [x, y]$，求解s和l

在点$[x, y]$处做切线和参考线的焦点为$\vec{r} = [x_r, y_r]$，那么该参考点处的s即为Frenet坐标系下的

在笛卡尔坐标系中，$\vec{l} = \vec{x} - \vec{r}$，那么将该向量$\vec{l}$投影到Frenet坐标系下，而$\vec{l}$和$\vec{N}_r$共线，那么将$l$投影到$N_r$上即可。

$$\vec{l_{N_r}} = ||\vec{l}||cos\theta_{lN} \cdot \frac{\vec{N}_r}{||\vec{N}_r||} \tag{59}$$

而又已知

$$\begin{gathered} ||\vec{l}|| = \sqrt{(x - x_r)^2 + (y - y_r)^2} \\ ||\vec{N}_r|| = 1 \end{gathered} \tag{60}$$

那么有

$$\vec{l_{N_r}} = \sqrt{(x - x_r)^2 + (y - y_r)^2} cos\theta_{lN} \vec{N}_r \tag{61}$$

接下来推导角度$\cos\theta_l N$，利用余弦定理可得

$$\begin{aligned} \vec{l} &= (x - x_r, y - y_r) \\ \vec{N}_r &= [-sin\theta_r, cos\theta_R] \\ cos\theta_{lN} &= \frac{\vec{l} \cdot \vec{N}_r}{||\vec{l}||||\vec{N}_r||} \\ &= \frac{-(x - x_r)sin\theta_r + (y - y_r)cos\theta_r}{\sqrt{(x - x_r)^2 + (y - y_r)^2}} \end{aligned} \tag{62}$$

由于两个向量共线，因此其角度为0°或180°,代入后求得l

$$\begin{gathered} cos\theta_{lN} = -1, l = -\sqrt{(x - x_r)^2 + (y - y_r)^2} \\ cos\theta_{lN} = 1, l = \sqrt{(x - x_r)^2 + (y - y_r)^2} \end{gathered} \tag{63}$$

B. 求解s和l及对时间和弧长的导数

> 问题描述如下：已知$\vec{x}=[x,y],\theta_x,v_x,a_x,\kappa_x$，推导$s,l,\dot{s},\ddot{s},l',l''$

回顾前文的公式，可以得到

$$
\begin{aligned}
\dot{l} &= v_x sin\Delta\theta, \Delta\theta=\theta_x-\theta_r \\
l' &= (1-\kappa_r l)tan\Delta\theta \\
l'' &= -(\kappa_r' l+\kappa_r l')tan\Delta\theta+\frac{1-\kappa_r l}{cos^2\Delta\theta}[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x-\kappa_r] \\
\dot{s} &= \frac{v_x cos\Delta\theta}{1-\kappa_r l}
\end{aligned} \tag{64}
$$

而$\ddot{s}$的推导在前文有表示，即

$$
a_x=\ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta}+\dot{s}^2\frac{1}{cos\Delta\theta}[-(\kappa_r' l+\kappa_r l')+(1-\kappa_r l)tan\Delta\theta[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x-\kappa_r]] \tag{67}
$$

那么有

$$
\ddot{s}=\frac{a_x cos\Delta\theta-\dot{s}^2[-(\kappa_r' l+\kappa_r l')+(1-\kappa_r l)tan\Delta\theta[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x-\kappa_r]]}{1-\kappa_r l} \tag{68}
$$

3. 如何由Frenet坐标系转换为笛卡尔坐标系?

$$
[s,\dot{s},\ddot{s},l,\dot{l},l',l'']\rightarrow[\vec{x},v_x,a_x,\theta_x,\kappa_x] \tag{69}
$$

> 问题描述如下：已知$s,l,\dot{s},\ddot{s},l',l''$，推导$\vec{x}=[x,y],\theta_x,v_x,a_x,\kappa_x$，

由向量关系可得

$$
\begin{gathered}
\vec{x}=\vec{r}+l\vec{N_r} \\
(x,y)=(x_r,y_r)+(-lsin\theta_r,lcos\theta_R)
\end{gathered} \tag{70}
$$

因此，

$$
\begin{gathered}
x=x_r-lsin\theta_r \\
y=y_r-lcos\theta_r
\end{gathered} \tag{71}
$$

$$v_x = \sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \tag{72}$$

接下来推导θ_x

> 回顾前文的推导结论：
>
> $$\begin{aligned} \dot{\vec{x}} &= \dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r \\ \vec{T}_r &= [cos\theta_r, sin\theta_r] \\ \vec{N}_r &= [-sin\theta_r, cos\theta_r] \end{aligned} \tag{73}$$

$$\begin{aligned} \dot{\vec{x}} &= \dot{s}(1-\kappa_r l)\vec{T}_r + \dot{l}\vec{N}_r \\ &= \dot{s}(1-\kappa_r l)[cos\theta_r, sin\theta_r] + \dot{l}[-sin\theta_r, cos\theta_r] \\ &= [\dot{s}(1-\kappa_r l)cos\theta_r - \dot{l}sin\theta_r, \dot{s}(1-\kappa_r l)sin\theta_r + \dot{l}cos\theta_r] \end{aligned} \tag{76}$$

因此，可以得到(这个公式可能暂时用不到？后续研究一下答疑)

$$tan\theta_x = \frac{\dot{s}(1-\kappa_r l)sin\theta_r + \dot{l}cos\theta_r}{\dot{s}(1-\kappa_r l)cos\theta_r - \dot{l}sin\theta_r} \tag{77}$$

另一种表示方法为

$$\begin{aligned} l' &= (1-\kappa_r l)tan\Delta\theta \\ \theta_x &= \theta_r + arctan(\frac{l'}{1-\kappa_r l}) \end{aligned} \tag{78}$$

而a_x可以参考前文的结论

$$a_x = \ddot{s}\frac{(1-\kappa_r l)}{cos\Delta\theta} + \dot{s}^2\frac{1}{cos\Delta\theta}[-(\kappa_r' l + \kappa_r l') + (1-\kappa_r l)tan\Delta\theta[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r]] \tag{81}$$

最后推导κ_x

由前文的公式可知

$$l'' = -(\kappa_r' l + \kappa_r l')tan\Delta\theta + \frac{1-\kappa_r l}{cos^2\Delta\theta}[\frac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r] \tag{82}$$

那么反解可以直接得到κ_x

$$\kappa_x = \{[l'' + (\kappa_r' l + \kappa_r l')tan\Delta\theta]\frac{cos^2\Delta\theta}{1-\kappa_r l} + \kappa_r\}\frac{cos\Delta\theta}{1-\kappa_r l} \tag{83}$$

A. Cartesian2Frenet

> $[\vec{x}, v_x, a_x, \theta_x, \kappa_x] \rightarrow [s, \dot{s}, \ddot{s}, l, l', l'']$

$$\begin{cases} s = s_r \\ \dot{s} = \dfrac{v_x cos\Delta\theta}{1-\kappa l} \\ \ddot{s} = \dfrac{a_x cos\Delta\theta - \dot{s}^2[-(\kappa_r' l + \kappa_r l') + (1-\kappa l)tan\Delta\theta[\frac{1-\kappa l}{cos\Delta\theta}\kappa_x - \kappa_r]]}{1-\kappa_r l} \\ l = sign((y-y_r)cos\theta_r - (x-x_r)sin\theta_r)\sqrt{(x-x_r)^2+(y-y_r)^2} \\ l' = (1-\kappa_r l)tan\Delta\theta \\ l'' = -(\kappa_r' l + \kappa_r l')tan\Delta\theta + \dfrac{1-\kappa_r l}{cos^2\Delta\theta}[\dfrac{1-\kappa_r l}{cos\Delta\theta}\kappa_x - \kappa_r] \end{cases} \tag{84}$$

B. Frenet2Cartesian

> $[s, \dot{s}, \ddot{s}, l, l', l''] \rightarrow [\vec{x}, v_x, a_x, \theta_x, \kappa_x]$

$$\begin{cases} x = x_r - l sin\theta_r \\ y = y_r - l cos\theta_r \\ \theta_x = \theta_r + arctan(\dfrac{l'}{1-\kappa_r l}) \\ v_x = \sqrt{\dot{s}^2(1-\kappa_r l)^2 + \dot{l}^2} \\ a_x = \ddot{s}\dfrac{(1-\kappa l)}{cos\Delta\theta} + \dot{s}^2\dfrac{1}{cos\Delta\theta}[-(\kappa_r' l + \kappa_r l') + (1-\kappa l)tan\Delta\theta[\dfrac{1-\kappa l}{cos\Delta\theta}\kappa_x - \kappa_r]] \\ \kappa_x = \{[l'' + (\kappa_r' l + \kappa_r l')tan\Delta\theta]\dfrac{cos^2\Delta\theta}{1-\kappa_r l} + \kappa_r\}\dfrac{cos\Delta\theta}{1-\kappa_r l} \end{cases} \tag{85}$$

五、代码理解

首先看一下各个变量的定义方法

> 这里 `dot` 表示的是对时间 `t` 的导数，而 `prime` 表示的是对弧长 `s` 的导数。
>
> - 在 `s_condition` 中，均为对时间 `t` 的导数
> - 在 `d_condition` 中，均为对弧长 `s` 的导数
>
> 实际上，在真正的函数中，还有一些额外的 `6` 个变量作为输入条件，他们是$s_r, x_r, y_r, \kappa_r, \theta_r, \kappa_r'$
>
> 另外， `l` 通常用 `d` 来表示
>
> `w.r.t.` : with respect to，关于的意思
>
> $$s\_condition = [s, s\_dot, s\_dddot]$$

$$
\begin{aligned}
s\_dot &= ds/dt \\
s\_ddot &= d(s\_dot)/dt \\
d\_condition &= [d, d\_prime, d\_pprime] \\
d\_prime &= dd/ds \\
d\_pprime &= d(d\_prime)/ds
\end{aligned} \tag{86}
$$

```cpp
#ifndef MODULES_PLANNING_MATH_FRAME_CONVERSION_CARTESIAN_FRENET_CONVERSION_H_
#define MODULES_PLANNING_MATH_FRAME_CONVERSION_CARTESIAN_FRENET_CONVERSION_H_

#include <array>

#include "modules/common/math/vec2d.h"

namespace apollo {
namespace planning {

// Notations:
// s_condition = [s, s_dot, s_ddot]
// s: longitudinal coordinate w.r.t reference line.
// s_dot: ds / dt
// s_ddot: d(s_dot) / dt
// d_condition = [d, d_prime, d_pprime]
// d: lateral coordinate w.r.t. reference line
// d_prime: dd / ds
// d_pprime: d(d_prime) / ds
// l: the same as d.
class CartesianFrenetConverter {
 public:
  CartesianFrenetConverter() = delete;
  /**
   * Convert a vehicle state in Cartesian frame to Frenet frame.
   * Decouple a 2d movement to two independent 1d movement w.r.t. reference
   * line.
   * The lateral movement is a function of longitudinal accumulated distance s
   * to achieve better satisfaction of nonholonomic constraints.
   */
  static void cartesian_to_frenet(const double rs, const double rx,
                                  const double ry, const double rtheta,
                                  const double rkappa, const double rdkappa,
                                  const double x, const double y,
                                  const double v, const double a,
                                  const double theta, const double kappa,
                                  std::array<double, 3>* const ptr_s_condition,
                                  std::array<double, 3>* const ptr_d_condition);
  /**
   * Convert a vehicle state in Frenet frame to Cartesian frame.
   * Combine two independent 1d movement w.r.t. reference line to a 2d movement.
   */
  static void frenet_to_cartesian(const double rs, const double rx,
                                  const double ry, const double rtheta,
                                  const double rkappa, const double rdkappa,
                                  const std::array<double, 3>& s_condition,
                                  const std::array<double, 3>& d_condition,
                                  double* const ptr_x, double* const ptr_y,
                                  double* const ptr_theta,
                                  double* const ptr_kappa, double* const ptr_v,
                                  double* const ptr_a);
```

```
  // given sl point extract x, y, theta, kappa
  static double CalculateTheta(const double rtheta, const double rkappa,
                               const double l, const double dl);

  static double CalculateKappa(const double rkappa, const double rdkappa,
                               const double l, const double dl,
                               const double ddl);

  static common::math::Vec2d CalculateCartesianPoint(
      const double rtheta, const common::math::Vec2d& rpoint, const double l);
  /**
   * @brief: given sl, theta, and road's theta, kappa, extract derivative l,
   *second order derivative l:
   */
  static double CalculateLateralDerivative(const double theta_ref,
                                           const double theta, const double l,
                                           const double kappa_ref);

  // given sl, theta, and road's theta, kappa, extract second order derivative
  static double CalculateSecondOrderLateralDerivative(
      const double theta_ref, const double theta, const double kappa_ref,
      const double kappa, const double dkappa_ref, const double l);
};

}  // namespace planning
}  // namespace apollo
```

函数实现文件

```
#include "modules/planning/math/frame_conversion/cartesian_frenet_conversion.h"

#include <cmath>

#include "modules/common/log.h"
#include "modules/common/math/math_utils.h"

namespace apollo {
namespace planning {

using apollo::common::math::Vec2d;

void CartesianFrenetConverter::cartesian_to_frenet(
    const double rs, const double rx, const double ry, const double rtheta,
    const double rkappa, const double rdkappa, const double x, const double y,
    const double v, const double a, const double theta, const double kappa,
    std::array<double, 3>* const ptr_s_condition,
    std::array<double, 3>* const ptr_d_condition) {
  const double dx = x - rx;
  const double dy = y - ry;

  const double cos_theta_r = std::cos(rtheta);
  const double sin_theta_r = std::sin(rtheta);

  const double cross_rd_nd = cos_theta_r * dy - sin_theta_r * dx;
  // 求解d
  ptr_d_condition->at(0) =
      std::copysign(std::sqrt(dx * dx + dy * dy), cross_rd_nd);
```

```
  const double delta_theta = theta - rtheta;
  const double tan_delta_theta = std::tan(delta_theta);
  const double cos_delta_theta = std::cos(delta_theta);

  const double one_minus_kappa_r_d = 1 - rkappa * ptr_d_condition->at(0);
  // 求解d' = dd / ds
  ptr_d_condition->at(1) = one_minus_kappa_r_d * tan_delta_theta;

  const double kappa_r_d_prime =
      rdkappa * ptr_d_condition->at(0) + rkappa * ptr_d_condition->at(1);

  // 求解d'' = dd' / ds
  ptr_d_condition->at(2) =
      -kappa_r_d_prime * tan_delta_theta +
      one_minus_kappa_r_d / cos_delta_theta / cos_delta_theta *
          (kappa * one_minus_kappa_r_d / cos_delta_theta - rkappa);

  // 求解s
  ptr_s_condition->at(0) = rs;
  // 求解ds / dt
  ptr_s_condition->at(1) = v * cos_delta_theta / one_minus_kappa_r_d;

  const double delta_theta_prime =
      one_minus_kappa_r_d / cos_delta_theta * kappa - rkappa;
  // 求解d(ds) / dt
  ptr_s_condition->at(2) =
      (a * cos_delta_theta -
       ptr_s_condition->at(1) * ptr_s_condition->at(1) *
           (ptr_d_condition->at(1) * delta_theta_prime - kappa_r_d_prime))
           / one_minus_kappa_r_d;
  return;
}

void CartesianFrenetConverter::frenet_to_cartesian(
    const double rs, const double rx, const double ry, const double rtheta,
    const double rkappa, const double rdkappa,
    const std::array<double, 3>& s_condition,
    const std::array<double, 3>& d_condition, double* const ptr_x,
    double* const ptr_y, double* const ptr_theta, double* const ptr_kappa,
    double* const ptr_v, double* const ptr_a) {
  CHECK(std::abs(rs - s_condition[0]) < 1.0e-6)
      << "The reference point s and
  const double cos_theta_r = std::cos(rtheta);
  const double sin_theta_r = std::sin(rtheta);

  *ptr_x = rx - sin_theta_r * d_condition[0];
  *ptr_y = ry + cos_theta_r * d_condition[0];

  const double one_minus_kappa_r_d = 1 - rkappa * d_condition[0];

  const double tan_delta_theta = d_condition[1] / one_minus_kappa_r_d;
  const double delta_theta = std::atan2(d_condition[1], one_minus_kappa_r_d);
  const double cos_delta_theta = std::cos(delta_theta);

  *ptr_theta = common::math::NormalizeAngle(delta_theta + rtheta);

  const double kappa_r_d_prime =
      rdkappa * d_condition[0] + rkappa * d_condition[1];
  *ptr_kappa = (((d_condition[2] + kappa_r_d_prime * tan_delta_theta) *
                 cos_delta_theta * cos_delta_theta) /
                    (one_minus_kappa_r_d) +
```

```
                rkappa) *
               cos_delta_theta / (one_minus_kappa_r_d);

  const double d_dot = d_condition[1] * s_condition[1];
  *ptr_v = std::sqrt(one_minus_kappa_r_d * one_minus_kappa_r_d *
                         s_condition[1] * s_condition[1] +
                     d_dot * d_dot);

  const double delta_theta_prime =
      one_minus_kappa_r_d / cos_delta_theta * (*ptr_kappa) - rkappa;

  *ptr_a = s_condition[2] * one_minus_kappa_r_d / cos_delta_theta +
           s_condition[1] * s_condition[1] / cos_delta_theta *
               (d_condition[1] * delta_theta_prime - kappa_r_d_prime);
}

double CartesianFrenetConverter::CalculateTheta(const double rtheta,
                                                const double rkappa,
                                                const double l,
                                                const double dl) {
  return common::math::NormalizeAngle(rtheta + std::atan2(dl, 1 - l * rkappa));
}

double CartesianFrenetConverter::CalculateKappa(const double rkappa,
                                                const double rdkappa,
                                                const double l, const double dl,
                                                const double ddl) {
  double denominator = (dl * dl + (1 - l * rkappa) * (1 - l * rkappa));
  if (std::fabs(denominator) < 1e-8) {
    return 0.0;
  }
  denominator = std::pow(denominator, 1.5);
  const double numerator = rkappa + ddl - 2 * l * rkappa * rkappa -
                           l * ddl * rkappa + l * l * rkappa * rkappa * rkappa +
                           l * dl * rdkappa + 2 * dl * dl * rkappa;
  return numerator / denominator;
}

Vec2d CartesianFrenetConverter::CalculateCartesianPoint(const double rtheta,
                                                        const Vec2d& rpoint,
                                                        const double l) {
  const double x = rpoint.x() - l * std::sin(rtheta);
  const double y = rpoint.y() + l * std::cos(rtheta);
  return Vec2d(x, y);
}

double CartesianFrenetConverter::CalculateLateralDerivative(
    const double rtheta, const double theta, const double l,
    const double rkappa) {
  return (1 - rkappa * l) * std::tan(theta - rtheta);
}

double CartesianFrenetConverter::CalculateSecondOrderLateralDerivative(
    const double rtheta, const double theta, const double rkappa,
    const double kappa, const double rdkappa, const double l) {
  const double dl = CalculateLateralDerivative(rtheta, theta, l, rkappa);
  const double theta_diff = theta - rtheta;
  const double cos_theta_diff = std::cos(theta_diff);
  const double res = -(rdkappa * l + rkappa * dl) * std::tan(theta - rtheta) +
                     (1 - rkappa * l) / (cos_theta_diff * cos_theta_diff) *
                         (kappa * (1 - rkappa * l) / cos_theta_diff - rkappa);
```

```
  if (std::isinf(res)) {
    AWARN << "result is inf when calculate second order lateral "
             "derivative. input values are rtheta:"
          << rtheta << " theta: " << theta << ", rkappa: " << rkappa
          << ", kappa: " << kappa << ", rdkappa: " << rdkappa << ", l: " << l
          << std::endl;
  }
  return res;
}
```