曲线的曲率公式推导以及离散点曲率计算方式

曲率公式

圆的曲率公式

$$L = \theta * R$$

$$dL = d\theta \cdot R$$

$$k = \frac{1}{R} = \frac{d\theta}{dL}$$

曲线曲率公式

$$\begin{aligned} k = \frac{d\theta}{dl} &= \frac{\frac{d\theta}{dt}}{\frac{dl}{dt}} \\ &= \frac{\frac{d\theta}{dt}}{\sqrt{(\frac{dx}{dt})^2 + (\frac{dy}{dt})^2}} \\ &= \frac{\frac{d\theta}{dt}}{\sqrt{\dot{x}^2 + \dot{y}^2}} \end{aligned}$$

又因为$tan\theta = \frac{dy}{dx} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{\dot{y}}{\dot{x}}$

所以对$tan\theta$求导，可得

$$\frac{dtan\theta}{dt} = sec^2\theta\frac{d\theta}{dt} = \frac{d\frac{\dot{y}}{\dot{x}}}{dt} = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2}$$

又因为

$$
\begin{aligned}
\frac{d\theta}{dt} &= \frac{1}{\sec^2\theta} \cdot \frac{d(\tan\theta)}{dt} \\
&= \frac{1}{1+\tan^2(\theta)} \cdot \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2} \\
&= \frac{1}{1+\frac{\dot{y}^2}{\dot{x}^2}} \cdot \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2} \\
&= \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}
\end{aligned}
$$

因此曲率公式为

$$
\begin{aligned}
k = \frac{1}{R} = \frac{d\theta}{dl} &= \frac{\frac{d\theta}{dt}}{\frac{dl}{dt}} \\
&= \frac{\frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2}}{\sqrt{\dot{x}^2+\dot{y}^2}} \\
&= \frac{\dot{x}\ddot{y}-\dot{y}\ddot{x}}{(\dot{x}^2+\dot{y}^2)^{3/2}}
\end{aligned}
$$

函数的曲率公式计算

假设 $y = f(x)$,

则

$$
\begin{aligned}
\dot{x} &= 1 \\
\ddot{x} &= 0 \\
k &= \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \\
&= \frac{\ddot{y}}{(1 + \dot{y}^2)^{3/2}}
\end{aligned}
$$

计算离散点的曲率

计算离散点的曲率可以有多种方法，比较见到的就是通过构建三元二次方程，根据三个连续离散点，求解三元二次方程，之后求解中间那个点的曲率

但是这种方法计算出来的曲率精度较低

最好的方法是利用最小二乘优化方法，根据离散点坐标拟合离散点轨迹，之后通过拟合的曲线计算每个点的曲率，该方法计算精度较高，但是比较耗时

三个点计算曲率

假设三元二次多项式

$$
\begin{aligned}
x &= a_1 + a_2 \cdot t + a_3 \cdot t^2 \\
y &= b_1 + b_2 \cdot t + b_3 \cdot t^2
\end{aligned}
$$

设置两段曲线的长度作为t的取值范围

$$
\begin{aligned}
t_a &= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \\
t_b &= \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2}
\end{aligned}
$$

将t_a, t_b带入多项式中

$$
\begin{aligned}
(x, y)|_{t=-t_a} &= (x_1, y_1) \\
(x, y)|_{t=0} &= (x_2, y_2) \\
(x, y)|_{t=t_b} &= (x_3, y_3)
\end{aligned}
$$

则

$$
\begin{aligned}
x_1 &= a_1 - a_2 \cdot t_a + a_3 \cdot t_a^2 \\
x_2 &= a_1 \\
x_3 &= a_1 + a_2 \cdot t_b + a_3 \cdot t_b^2
\end{aligned}
$$

$$
\begin{aligned}
y_1 &= b_1 - b_2 \cdot t_a + b_3 \cdot t_a^2 \\
y_2 &= b_1 \\
y_3 &= b_1 + b_2 \cdot t_b + b_3 \cdot t_b^2
\end{aligned}
$$

将其写成矩阵形式

$$
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & -t_a & t_a^2 \\ 1 & 0 & 0 \\ 1 & t_b & t_b^2 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}
$$
$$
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & -t_a & t_a^2 \\ 1 & 0 & 0 \\ 1 & t_b & t_b^2 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}
$$

则

$$
\begin{aligned}
X = M \cdot A \\
Y = M \cdot B
\end{aligned}
$$

则

$$
\begin{aligned}
A = M^{-1} \cdot X \\
B = M^{-1} \cdot Y
\end{aligned}
$$

求解矩阵的逆，并将其带入多项式中，可以得到多项式系数，之后求解多项式的导数，根据曲率公式计算中间点的曲率

$$
\begin{aligned}
\dot{x} &= a_2 + 2a_3 \cdot t|_{t=0} = a_2 \\
\ddot{x} &= 2a_3|_{t=0} = 2a_3 \\
\dot{y} &= b_2 + 2b_3 \cdot t|_{t=0} = b_2 \\
\ddot{y} &= 2b_3|_{t=0} = 2b_3
\end{aligned}
$$

根据曲率公式计算中间点曲率

$$
\begin{aligned}
k = \frac{1}{R} &= \frac{\dot{x} \cdot \ddot{y} - \dot{y} \cdot \ddot{x}}{(\dot{x}^2 + \dot{y}^2)^{3/2}} \\
&= \frac{2(a_3 b_2 - a_2 b_3)}{(a_2^2 + b_2^2)^{3/2}}
\end{aligned}
$$

最小二乘拟合曲线计算曲率

首先对引导线的离散点进行优化，优化主要有三个方面:

利用矩阵运算求解曲率

```
bool ReferenceLineProvider::Fitting(
    const std::vector<MotionPathPoint> &point_vector,
    const std::size_t &point_number, LkaLane &fitting_lane) {
  static constexpr double kYDistanceError = 0.5;
  // NM_ERROR("====FakeCamera::CalculateLineParam====");  size_t
  size_t use_points_number = std::min(point_vector.size(), point_number);
  if (use_points_number <= 4) {
    return false;
  }
  int N = 2;
  Eigen::MatrixXd A(use_points_number, N + 1);
  for (size_t i = 0; i < use_points_number; ++i) {
    for (int n = N, dex = 0; n >= 1; --n, ++dex) {
      A(i, dex) = pow(point_vector[i].x(), n);
    }
    A(i, N) = 1;
  }
  Eigen::MatrixXd B(use_points_number, 1);
  for (size_t i = 0; i < use_points_number; ++i) {
    B(i, 0) = point_vector.at(i).y();
  }
  Eigen::MatrixXd W;
  W = A.bdcSvd(Eigen::ComputeThinU | Eigen::ComputeThinV).solve(B);
  NM_DEBUG_STREAM("W: " << W);
  if (W.size() >= 3) {
    fitting_lane.set_position_parameter_c0(static_cast<double>(-W(2)));
    fitting_lane.set_heading_angle_parameter_c1(static_cast<double>(-W(1)));
    fitting_lane.set_curvature_parameter_c2(static_cast<double>(-W(0)));
    fitting_lane.set_curvature_derivative_parameter_c3(0.);
    auto &start_pt = point_vector[0];
    auto start_y =
        common::cubic_curve::CalculateYOnLaneCurve(fitting_lane, start_pt.x());
    if (std::fabs(start_y - start_pt.y()) > kYDistanceError) {
      return false;
    }
  } else {
    return false;
  }
```

```
    return true;
}
```

参考文献

[公式推导资料]

[圆的曲率介绍]

[三点计算曲率]

[latex公式网页版]