深蓝学院
shenlanxueyuan.com

第四章思路提示

主讲人 夏韵凯

标题

深蓝学院
shenlanxueyuan.com

- 轨迹跟踪LQR可描述为

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}_1\delta + \boldsymbol{B}_2 r_{des}, \text{ where } \boldsymbol{x} = \left(e_{cg}\ \ \dot{e}_{cg}\ \ e_\theta\ \ \dot{e}_\theta\right)^T.$$

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\dfrac{(c_f + c_r)}{mv} & \dfrac{c_f + c_r}{m} & \dfrac{l_r c_r - l_f c_f}{mv} \\ 0 & 0 & 0 & 1 \\ 0 & \dfrac{l_r c_r - l_f c_f}{I_z v} & \dfrac{l_r c_r - l_f c_f}{I_z} & -\dfrac{l_f^2 c_f + l_r^2 c_r}{I_z v} \end{bmatrix}, \boldsymbol{B}_1 = \begin{bmatrix} 0 \\ \dfrac{c_f}{m} \\ 0 \\ \dfrac{l_f c_f}{I_z} \end{bmatrix}, \boldsymbol{B}_2 = \begin{bmatrix} 0 \\ \dfrac{l_r c_r - l_f c_f}{mv} - v \\ 0 \\ -\dfrac{l_f^2 c_f + l_r^2 c_r}{I_z v} \end{bmatrix}$$

标题

深蓝学院
shenlanxueyuan.com

●状态矩阵线性化

$$A=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{(c_f+c_r)}{mv} & \frac{c_f+c_r}{m} & \frac{l_r c_r - l_f c_f}{mv} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{l_r c_r - l_f c_f}{I_z v} & \frac{l_r c_r - l_f c_f}{I_z} & -\frac{l_f^2 c_f + l_r^2 c_r}{I_z v} \end{bmatrix},$$

离散化方法：

向前欧拉法：$x(t+dt)=(I+Adt)x(t)$

向后欧拉法：$x(t+dt)=(I-Adt)^{-1}x(t)$

中点欧拉法：$x(t+dt)=(I-\frac{Adt}{2})^{-1}(I+\frac{Adt}{2})x(t)$

参考视频

```cpp
// 配置状态矩阵A
matrix_a_(0, 1) = 1.0;
matrix_a_coeff_(0, 2) = 0.0;
matrix_a_(1, 2) = (cf_ + cr_) / mass_;
matrix_a_(3, 2) = (lf_ * cf_ - lr_ * cr_) / iz_;
matrix_a_coeff_(1, 1) = -(cf_ + cr_) / mass_;
matrix_a_coeff_(1, 3) = (lr_ * cr_ - lf_ * cf_) / mass_;
matrix_a_coeff_(3, 1) = (lr_ * cr_ - lf_ * cf_) / iz_;
matrix_a_coeff_(3, 3) = -1.0 * (lf_ * lf_ * cf_ + lr_ * lr_ * cr_) / iz_;
```

```cpp
// 更新状态矩阵A并将状态矩阵A离散化
void LqrController::UpdateMatrix(const VehicleState &vehicle_state) {
  double v;
  v = std::max(vehicle_state.velocity, minimum_speed_protection_);
  matrix_a_(1, 1) = matrix_a_coeff_(1, 1) / v;
  matrix_a_(1, 3) = matrix_a_coeff_(1, 3) / v;
  matrix_a_(3, 1) = matrix_a_coeff_(3, 1) / v;
  matrix_a_(3, 3) = matrix_a_coeff_(3, 3) / v;
  Matrix matrix_i = Matrix::Identity(matrix_a_.cols(), matrix_a_.cols());
  matrix_ad_ = (matrix_i - ts_ * 0.5 * matrix_a_).inverse() *
               (matrix_i + ts_ * 0.5 * matrix_a_);
}
```

```cpp
// 动力矩阵B
matrix_b_(1, 0) = cf_ / mass_;
matrix_b_(3, 0) = lf_ * cf_ / iz_;
matrix_bd_ = matrix_b_ * ts_;
```

标题

深蓝学院
shenlanxueyuan.com

- LQR系统状态（误差）的计算

e_{cg}: Orthogonal distance of the C.G. to the nearest path waypoint;

$$\dot{e}_{cg} = v_y + v_x \tan(\theta - \theta_p(s))$$
$$= v_y + v_x \tan(e_\theta)$$

$$e_\theta = \theta - \theta_p(s)$$

$$\dot{e}_\theta = r - r(s)$$

```cpp
void LqrController::ComputeLateralErrors(const double x, const double y,
                                         const double theta,
                                         const double linear_v,
                                         const double angular_v,
                                         const double linear_a,
                                         LateralControlErrorPtr &lat_con_err) {
  TrajectoryPoint target_point;
  target_point = QueryNearestPointByPosition(x, y);
  const double dx = target_point.x - x;  // x轴误差
  const double dy = target_point.y - y;  // y轴误差
  const double cos_target_heading = cos(target_point.heading);
  const double sin_target_heading = sin(target_point.heading);
  double lateral_error =
      cos_target_heading * dy - sin_target_heading * dx;  //横向误差
  lat_con_err->lateral_error = lateral_error;
  double heading_error = NormalizeAngle(target_point.heading - theta);
  lat_con_err->heading_error = heading_error;
  auto lateral_error_dot = linear_v * tan(heading_error);
  lat_con_err->lateral_error_rate = lateral_error_dot;
  double ref_heading_rate = target_point.kappa * target_point.v;
  lat_con_err->heading_error_rate = angular_v - ref_heading_rate;
}
```

标题

● 前馈的计算

1/R=c

$$\delta_{ff} = \frac{L}{R} + K_v a_y - k_3 \left(\frac{\ell_r}{R} - \frac{\ell_f}{2c_{\alpha r}} \frac{m v_x^2}{RL} \right),$$

$$K_v = \frac{\ell_r m}{2c_{\alpha f}(\ell_f + \ell_r)} - \frac{\ell_f m}{2c_{\alpha r}(\ell_f + \ell_r)}$$

$$a_y = \frac{v_x^2}{R}.$$

```
// 前馈控制，计算横向转角的反馈量
double LqrController::ComputeFeedForward(const VehicleState &localization,
                                         const double ref_curvature) {
  const double kv =
    lr_ * mass_ / 2 / cf_ / wheelbase_ - lf_ * mass_ / 2 / cr_ / wheelbase_;

  const double v = localization.velocity;
  double steer_angle_feedforwardterm;

  steer_angle_feedforwardterm =
      (wheelbase_ * ref_curvature + kv * v * v * ref_curvature -
       matrix_k_(0, 2) *
           (lr_ * ref_curvature -
            lf_ * mass_ * v * v * ref_curvature / 2 / cr_ / wheelbase_));

  return steer_angle_feedforwardterm;
}
```

参考文档：Automatic Steering Methods for Autonomous Automobile Path Tracking

标题

●求解Riccati方程

Summary of LQR solution via DP

1. set $P_N := Q_f$

2. for $t = N, \ldots, 1,$

$$P_{t-1} := Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A$$

3. for $t = 0, \ldots, N-1$, define $K_t := -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$

4. for $t = 0, \ldots, N-1$, optimal u is given by $u_t^{\mathrm{lqr}} = K_t x_t$

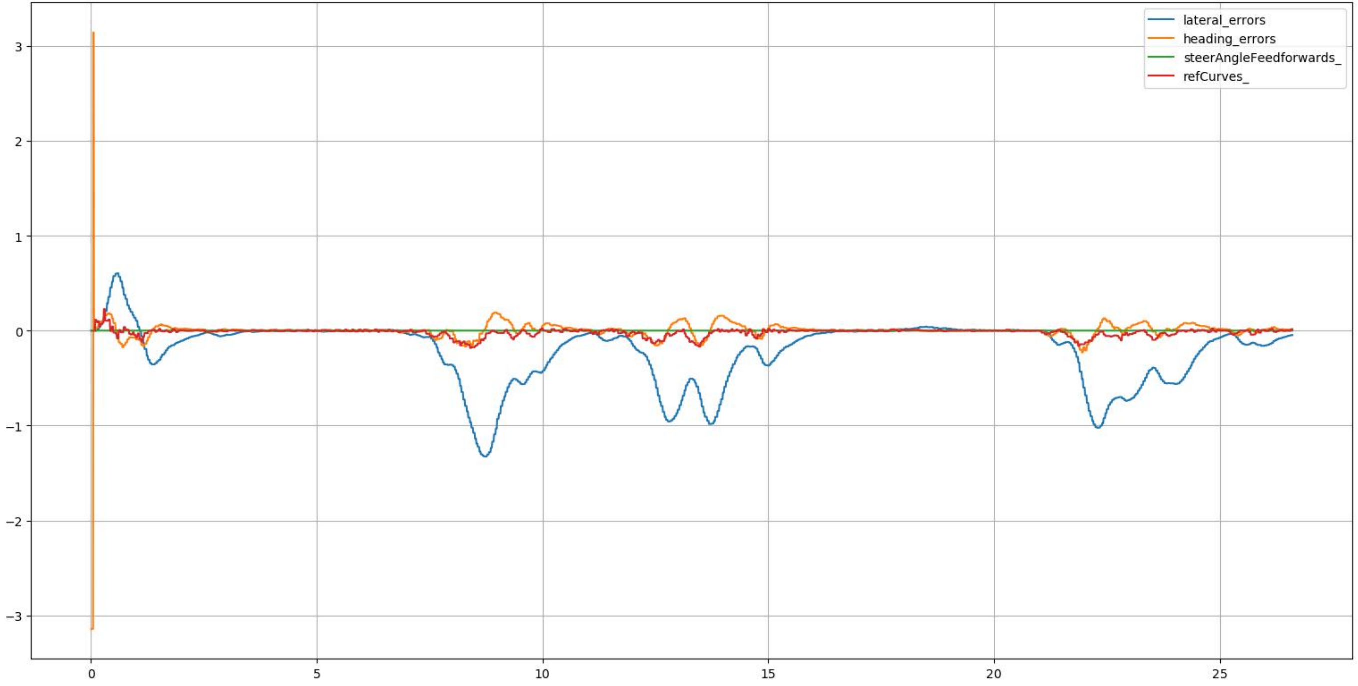
参考文档

```cpp
// 求解LQR方程
void LqrController::SolveLQRProblem(const Matrix &A, const Matrix &B,
                                    const Matrix &Q, const Matrix &R,
                                    const double tolerance,
                                    const uint max_num_iteration,
                                    Matrix *ptr_K) {
  if (A.rows() != A.cols() || B.rows() != A.rows() || Q.rows() != Q.cols() ||
      Q.rows() != A.rows() || R.rows() != R.cols() || R.rows() != B.cols()) {
    std::cout
        << "LQR solver: one or more matrices have incompatible dimensions."
        << std::endl;
    return;
  }
  Matrix AT = A.transpose();  // 状态矩阵A的转置
  Matrix BT = B.transpose();  // 状态矩阵B的转置
  Matrix P = Q;
  uint num_iteration = 0;  // 迭代次数
  double diff = std::numeric_limits<double>::max();

  while (num_iteration++ < max_num_iteration && diff > tolerance) {
    Matrix P_next = AT * P * A -
                    (AT * P * B) * (R + BT * P * B).inverse() * (BT * P * A) +
                    Q;
    diff = fabs((P_next - P).maxCoeff());
    P = P_next;
  }
  *ptr_K = (R + BT * P * B).inverse() * (BT * P * A);
}
```

标题

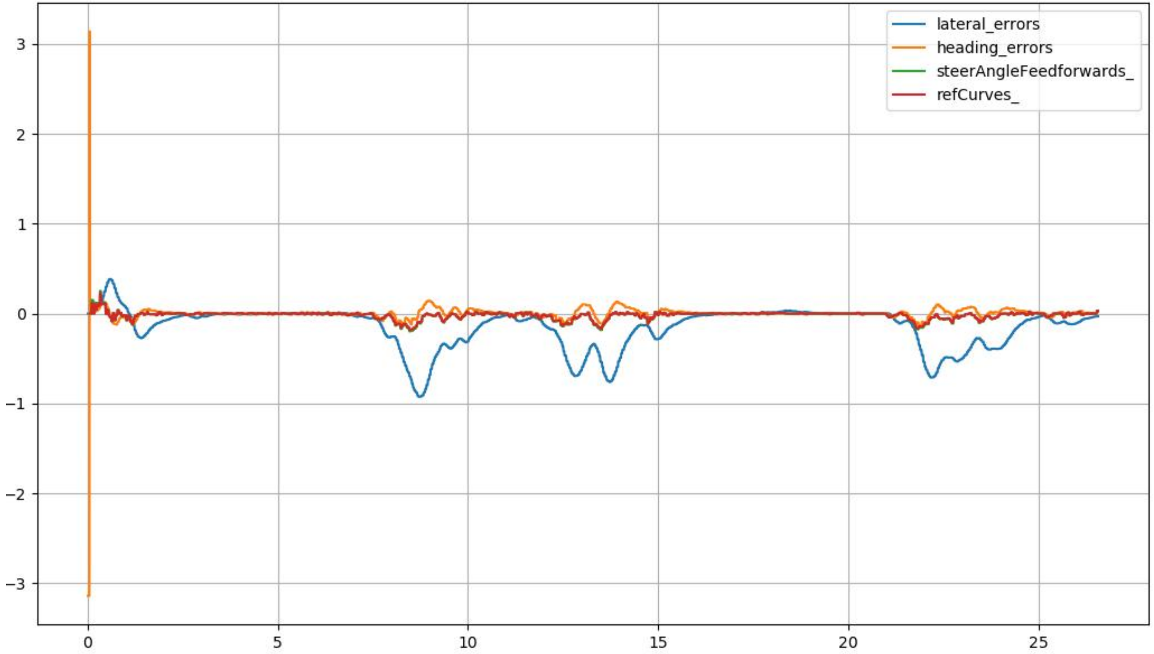
● 不同前馈的跟踪效果：无前馈



标题

●不同前馈的跟踪效果

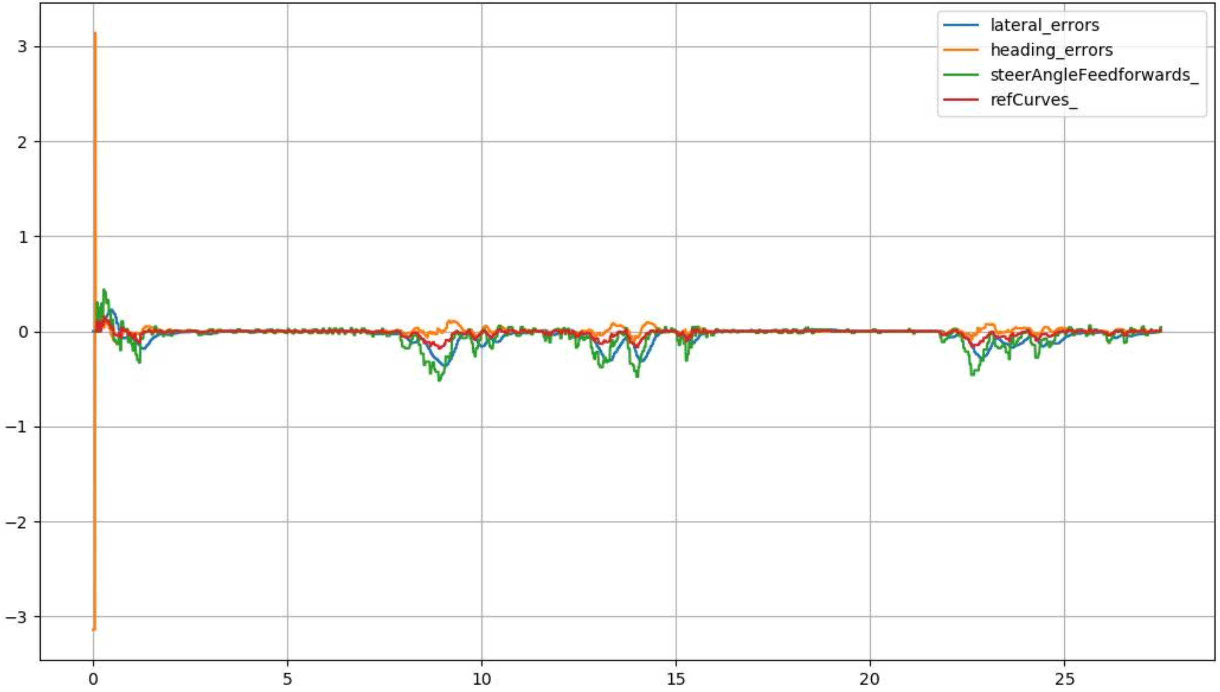
$$\delta_{ff} = \frac{L}{R} + K_v a_y - k_3 \left(\frac{\ell_r}{R} - \frac{\ell_f}{2c_{\alpha r}} \frac{m v_x^2}{RL} \right),$$



标题

- 不同前馈的跟踪效果

$$\delta_{ff} = \frac{L}{R}.$$



在线问答

Q&A

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening